Rapport projet Microcontrôleur

Killian HINARD
Rémi ODDON

Groupe 46
Printemps 2021

1 Introduction

Dans le cadre du cours de Microcontrôleur, il nous a été demandé de développer une application concrète en assembleur AVR. Pour ce faire, nous avons à notre disposition un kit STK-300 contenant un microcontrôleur Atmega128L et plusieurs périphériques supplémentaires comme le capteur de température 1-Wire ou encore un encodeur angulaire.

Notre application est un système permettant le contrôle de la température d'une pièce, serre de culture agricole, etc. Notre programme va tout d'abord mesurer la température de la pièce puis va contrôler l'ouverture d'une valve d'aération grâce à un moteur servo pour permettre à la température à l'intérieur de celle-ci d'augmenter ou de diminuer. Cette application permet à l'utilisateur de choisir plusieurs paramètres visant à un meilleur contrôle de la température de la pièce. En effet, grâce à une interface utilisateur via un écran LCD, l'usager a le choix de régler les températures minimum et maximum acceptables dans la pièce. De plus, il peut, si il le souhaite, être averti grâce à une alarme quand la température de la serre dépasse les limites précédemment imposées (plus ou moins une tolérance que celui-ci aura également réglé au préalable) afin de pouvoir venir régler le problème manuellement.

L'interface utilisateur est constituée de plusieurs modes consécutifs dans lesquels il peut réaliser différentes actions. En effet, il peut passer au mode suivant ou au mode précédent (SCROLL), il peut également choisir de modifier la valeur du paramètre lié au mode actuel (EDIT).

2 Mode d'emploi

Lorsque le programme se lance, il est par défaut sur le mode réglage de la valeur limite minimale de la température puisque nous estimons qu'il s'agit du point de départ de l'initialisation des paramètres par l'utilisateur. De plus, le programme est de base sur l'action de défilement (SCROLL) entre les modes. À présent, on peut en appuyant sur le bouton de l'encodeur angulaire changer d'action et passer en modification (EDIT). Dans le mode EDIT, la rotation de l'encodeur angulaire nous permet de changer la valeur minimale de la température acceptable (min_) mais en restant supérieur à la température minimale (temp_min) du capteur de température et inférieure à la température maximale acceptable (max_). Une rotation dans le sens horaire résulte en une incrémentation de min_ tandis qu'une rotation antihoraire décrémente min_.

Une fois que la température minimale (min_) réglée nous convient, on peut en appuyant sur le bouton de l'encodeur revenir à l'action SCROLL afin de changer de mode pour celui de la valeur maximale, en tournant l'encodeur dans le sens horaire. Les opérations pour modifier max_ sont identiques à celles pour min_, cependant l'intervalle valide pour max_ est compris entre min_ et temp_max étant la limite maximale que le capteur de température peut mesurer.

Ensuite, on peut passer au mode de la tolérance dans lequel nous avons le choix de passer au mode suivant ou précédent ou alors de modifier la valeur tol_ de la tolérance appliquée à l'intervalle [min_ ; max_] faisant sonner l'alarme si la température de la serre quitte l'intervalle [min_ − tol_ ; max_ + tol_]. La modification de tol_ suit le même principe que celui précédemment expliqué pour min_ et max_.

Puis, nous pouvons passer au mode de l'alarme où l'on peut activer ou désactiver l'alarme en changeant la valeur de alr_ ou alors poursuivre le défilement dans les modes.

L'avant dernier mode correspond au mode de réglage de la sonnerie de l'alarme. l'utilisateur peut, de la même manière que pour changer la valeur dans les autres modes, cliquer sur le bouton de l'encodeur angulaire de manière à entrer dans le mode EDIT. De là, il lui sera possible de changer la sonnerie qu'il souhaite pour notre alarme en tournant l'encodeur angulaire. Chaque fois que vous changez de sonnerie elle sera jouée une fois pour que l'utilisateur puisse savoir à quoi elle ressemble.

Enfin, nous arrivons en tournant l'encodeur encore de un cran dans le sens horaire dans le mode d'information. Pour ce mode, le fonctionnement est différent car il s'agit du stade final de l'édition des paramètres. En effet, le programme demande tout d'abord à l'utilisateur si il souhaite confirmer tous les paramètres choisis précédemment à savoir min_, max_, tol_, alr_ et son_. Pour confirmer, il faut appuyer sur le bouton de l'encodeur. Cela aura pour effet de changer le comportement du système qui va alors rentrer dans le mode de réglage de la température grâce au servo. L'affichage du LCD va également changer, celui-ci va à présent afficher si l'alarme et on ou off, quelle est la sonnerie actuelle et enfin la valeur actuelle de la température. Une fois le bouton pressé, il est impossible de changer de mode en tournant l'encodeur comme avant. Si l'on souhaite quitter le mode d'information, il faut presser à nouveau le bouton et le programme nous ramène dans le mode de la valeur minimale acceptable min_ avec pour action le défilement SCROLL. Notons qu'en quittant le mode d'information, les valeurs de min_, max_, tol_, alr_ et son_ sont conservées. L'utilisateur peut à présent choisir de nouveaux paramètres puis retourner dans le mode information.

3 Description technique

Notre application utilise plusieurs périphériques qui sont l'encoder angulaire, le capteur de température 1-Wire, le buzzer piezo-électrique, le moteur servo Futaba S3003 controlé en valeur angulaire et enfin l'écran Hitachi44780U 2x16 LCD. L'encodeur angulaire et le buzzer sont bran-

chés sur le port E configuré en entrée sur les pins 4 à 6 pour recevoir les données de l'encodeur et en sortie sur le pin 2 (SPEAKER) pour faire sonner le buzzer. Le capteur de température est placé sur le port B et communique selon le protocole 1-Wire sur le pin 5. Le moteur servo est situé sur le port D qui est entièrement configuré en sortie pour transmettre au moteur des pulses dont le rapport cyclique est contrôlé par le programme. Enfin l'écran LCD est branché sur les pins réservés à celui-ci.

Pour ce qui est des interruptions, nous n'en utilisons qu'une : l'interruption externe synchrone 6, située sur le pin 6 du port E, qui correspond au signal d'interruption utilisé par le bouton de l'encodeur angulaire. Cette interruption nous sert à repasser vers les modes des paramètres depuis le mode de fonctionnement principale de notre système (mode information).

4 Fonctionnement du programme

Le programme se décompose principalement en deux blocs, le premier gère les menus dans lesquels l'utilisateur choisit les paramètres et le second, pour lequel les paramètres sont définis, gère le calcul de la longueur des pulses à envoyer au servo-moteur pour avoir la bonne position angulaire. Chacun de ces deux blocs fait appel à des librairies externes pour les périphériques. En effet, le bloc des menus gère l'affichage sur le LCD, en utilisant les librairies lcd.asm, printf.asm et menu.asm, en fonction du mode et de l'action courante. Ce bloc permet aussi la modification des paramètres de l'application à l'aide de la librairie encoder.asm. Le second bloc, quant à lui, a pour tâche de récupérer la valeur de la température via la librairie du capteur wire1.asm et d'effectuer plusieurs opérations mathématiques sur cette valeur avec les valeurs minimale et maximale définies au préalable. Les routines et macros pour les opérations mathématiques proviennent de la librairie math.asm et nous en avons aussi créé de nouvelles.

5 Présentation détaillée des modules

Tout d'abord, on renomme les registres r6 et r7 en mode et action à l'aide d'un .def. De plus, on réalise de l'allocation de mémoire en mémoire donnée à l'aide de l'instruction .dseg pour les registres min_, max_, tol_, alr_ et son_.

Routine reset : Ensuite lors du lancement du programme, la routine reset située aux premières adresses de la mémoire programme est effectuée. Cette routine permet d'initialiser le Stack Pointer, les valeurs des registres Data Direction Register (DDR) pour le moteur et le buzzer et d'appeler les routines d'initialisation des périphériques comme le LCD, l'encoder et le capteur de température à travers les routines lcd_init, encoder_init et wire1_init, qui elles aussi configurent entre autres les registres DDR. Enfin, la routine reset permet d'initialiser les valeurs du mode courant à mode_min, de l'action courante à SCROLL mais aussi des registres min_ à temp_min, max_ à temp_max, tol_ et alr_ tous les deux à 0 et son_ à 1.

Module des menus de sélection des paramètres À la fin de reset, on effectue un saut à la routine affichage. Cette routine permet l'affichage correspondant au mode courant. Tout d'abord, on charge avec sts les valeurs des registres min_ et max_ dans a2 et a3, puis on appelle lcd_clear afin d'effacer l'écran et de ramener le curseur d'écriture au début de la première ligne. Ensuite, la routine effectue une série de test d'égalité dans lesquels on compare avec cpi la valeur du registre mode avec la valeur des différents modes comme défini au début du programme. Suivant le résultat de la comparaison, le fanion Z vaut 1 si il y avait égalité sinon 0 et ainsi avec des breq on peut faire un branchement vers la bonne routine d'affichage du mode courant.

Dans une routine d'affichage, on récupère dans un registre du banc de registre la valeur du paramètre correspondant au mode courant, tol_ par exemple si mode = mode_tol, puis on affiche avec PRINTF la valeur de ce paramètre. Ensuite, on charge dans a0 la valeur de action, afin d'utiliser la librairie menu.asm qui permet d'afficher le a0-ième élément dans un string avec | comme séparateur dans ce string. Ainsi, si action vaut SCROLL (= 0) alors dans le string "SCROLL|EDIT", la routine n'affichera que SCROLL. Ce fonctionnement de routine d'affichage n'est valable que pour les modes liés à un paramètre et non pour le mode d'information, pour lequel on affiche si l'on souhaite confirmer les paramètres choisis.

À la fin des routines d'affichages, on réalise un rjmp vers jump_action qui permet de faire des sauts vers les routines d'éditions des paramètres ou de défilement des modes suivant la valeur de action. Si action=SCROLL alors le PC saute vers action_scroll qui charge mode dans a0 puis appelle encoder qui permet d'incrémenter ou de décrémenter a0 et de vérifier si on a appuyé sur le bouton. Si celui-ci a été pressé alors le fanion T vaut 1 et il faut changer d'action donc on fait un branchement vers change_action avec un brts. Dans le cas contraire, le programme vérifie que a0 appartient toujours à l'intervalle [mode_min ; mode_info] grâce à la macro CYCLIC de encoder.asm qui permet de restreindre un registre dans un intervalle avec la possibilité de passer d'une extrémité à l'autre. Ainsi, la valeur de mode est bien bornée par des valeurs définies dans le code. Si action avait valu EDIT alors le PC aurait sauté vers action_edit qui redirige vers la bonne routine d'édition suivant la valeur de mode avec à nouveau des cpi et breq. Dans une routine d'édition, on charge avec sts la valeur du paramètre dans a0 puis le programme appelle encoder pour incrémenter/décrementer cette valeur. À nouveau si le bouton est pressé (T=1) alors on fait un branchement vers change_mode. Sinon on vérifie que la valeur respecte les bornes qui lui sont imposées avec CYCLIC ou BORDS, suivant la routine d'édition. BORDS est une macro équivalente à CYCLIC mais utilisant des registres et non des immediates et ne permettant pas de passer d'une extrémité à l'autre. À la fin des routines d'éditions, le PC retourne à l'adresse de la routine affichage.

Si pendant la routine action_scroll ou une routine d'édition le bouton est pressé alors le programme saute vers la routine change_mode dans laquelle si la valeur de mode est différente de mode_info alors action change de valeur entre SCROLL et EDIT. Sinon, on saute vers la routine main_init.

Le module main_init : Ce module nous permet d'exécuter le code nécessaire pour le bon fonctionnement du main. Ce code ne sera exécuté qu'une seule fois. On s'en sert donc pour rétablir les interruptions afin de pouvoir remodifier les paramètres par la suite, puis on clear le LCD et enfin on affiche si l'alarme est on ou off en lisant la valeur de alr_, puis on affiche quelle sonnerie est utilisé pour l'alarme en lisant la valeur de son_. A la fin de ce module on rentre directement dans le main.

Modules de lecture de la température Au début du main programme qui gère le positionnement du servo et le fonctionnement de notre programme principale on doit récupérer la température actuelle de la pièce et pour ce faire on utilise le capteur de température. On communique avec celui ci grâce au protocole 1-wire. Le fonctionnement de cette partie est mieux détaillé en partie 6 lors de la description de comment nous accédons au capteur de température. A la fin nous nous retrouvons avec la température stocké dans les registres d3 et d2.

Module de vérification de la température et déclenchement de l'alarme : On rentre ensuite dans un module nous permettant de savoir si la température actuelle est contenue ou non dans les températures limites fixé par l'utilisateur. Si oui, on jump directement au calcul de la durée des pulses correspondant à cette température. Sinon, on va soit à tooH si la température est trop haute, soit à tooL si la température est trop basse. Ces deux parties de codes sont très similaires à la différence que l'une fixe les pulses envoyées au moteur à 2000ms et compare la température à max_ + tol_ et l'autre fixe les pulses envoyées au moteur à 1000ms et compare la température à min_ - tol_. Si la température est hors de l'intervalle des températures limites en comptant la tolérance le programme va vers alarme (sinon on rejoins directement le module de positionnement du moteur servo). Le module alarme permet de jouer la mélodie de la sonnerie en utilisant le buzzer mais son fonctionnement est détaillé plus précisément en partie 6.

Le module de calcul des pulses pour le servo : Ce module nous permet de calculer la durée des pulses à imprimer sur la ligne de contrôle du servo moteur en fonction de la température qu'il fait dans la pièce. Pour ce faire nous effectuons une linéarisation de la correspondance entre pulse et température dans l'intervalle des températures limites choisies par l'utilisateur. En effet nous posons :

$$p(\text{la durée de pulse}) = \frac{1}{16}(16a \cdot T(\text{la température en degré}) + 16a \cdot min\_ + 16000)$$

(comme la capteur nous donne 16 fois la température on a adapté la formule au plus simple pour l'assembleur). Notre programme doit donc calculer lui même le coefficient "a" en fonction des températures limites fournies pour enfin calculer p. Dans la première partie, ce module il calcule a et le stocke dans b1 et b0, puis dans la deuxième il calcule p avec la formule donnée ci-dessus. On fait cela en utilisant les macros MUL2B4 pour multiplier 2 bits par 16, de la macro DIV4B4 qui permet de diviser 4 bits par 16 et enfin en utilisant les sous-routines "mul22" et "div22" de la librairie "math.asm". Au final la longueurs des pulses est stocké dans les bits a1 et a0. A la fin de ce module, on rejoint le module de positionnement du servo-moteur.

module de positionnement du servo : Ce module nous permet de fournir un signal avec des pulses de bonne durée au moteur. Son fonctionnement est plus détaillé en partie 6. A la fin de ce module le programme re-saute jusqu'au début du main pour recommencé.

6 Description de détail de l'accès au périphérique

Encodeur angulaire : Ce périphérique nous indique le sens de rotation de l'encodeur en créant un décalage entre deux signaux A et B. Ce décalage est créé par la connexion entre une des broches A ou B avant l'autre broche sur des zones de contact équiréparties sur un disque à l'intérieur de l'encodeur. La ligne I fonctionne comme un bouton de la carte : 1 au repos et 0 si le bouton est pressé. La routine encoder de encoder.asm permet de détecter des changements sur ces 3 lignes, en stockant l'état précédent des lignes en mémoire donnée. Tout d'abord, cette routine met T à 0 car on utilise le fanion T pour indiquer que le bouton a été pressé, il faut donc le clear au préalable. Ensuite, encoder récupère l'état des 3 lignes dans _w et l'état précédent dans _u et les compare. Si il n'y a pas de différence alors la routine s'arrête puisque l'état de l'encodeur est toujours le même. Sinon, _u stocke à présent les changements des lignes avec un eor entre _u et _w. Si il y a une transition sur la ligne I, alors la routine encoder ne traitera que ce cas et mettra à 1 le fanion T si il s'agit un flanc descendant de la ligne I sinon elle ne fera rien. Si la ligne I n'a pas changé d'état, mais que A a changé alors suivant le flanc de la ligne A, il faut soit incrémenter soit décrémenter a0. a0 doit être incrémenté si A est en avance sur B donc si A a un flanc montant avec B=0 ou un flanc descendant avec B=1. Pour ce faire, la routine regarde la valeur actuelle de A dans _w et en déduit le sens du flanc. Ensuite, dans les routines a_rise et a_fall, a0 est incrémenté puis on lui soustrait 2 si l'état de la ligne B dans _w correspond à un scénario ou a0 devait être décrémenté. Le raisonnement est le même lorsque le bouton I est enfoncée sauf qu'il s'agit du registre b0 qui est modifié, cependant ce cas n'est pas utilisé dans ce projet.

Buzzer : Ce périphérique nous permet de faire sonner l'alarme car celui ci peut produire du son. Pour ce faire il faut lui appliquer un tension alternative à la fréquence du son souhaité (par exemple on veut faire un La 440 on doit lui appliquer un signal avec un fréquence de 440 Hz). Pour créer ce signal il nous suffit d'imprimer un 1 logique ,sur le bit relié au speaker du port relié à son module, pendant la demi-période de la note que l'on veut jouer, puis on imprime un 0 logique sur ce même bit de manière à créer un signal carré de la bonne fréquence. On fait durer ce signal le temps que l'on désire pour la note. Pour ce faire dans notre programme on se sert de la sous-routine sound de la librairie "sound.asm" qui nous permet de créer ce signal en indiquant dans a0, la valeur de la période de la note que l'on souhaite jouer (en dizaine de μs) et en indiquant la durée de celle-ci dans b0 (en 2.5 ms). La routine produit ensuite simplement un signal normalisé de la façon que j'ai décrite ci-dessus en insérant des délais à certains endroits.

Capteur de température : Ce périphérique nous permet de capter la température actuelle de notre pièce. Pour ce faire, il nous communique celle-ci sous le format d'un nombre en degré celsius sur 12 bits qui correspond à 16 fois la température dans la pièce (la vraie valeur de la

température est donc ce nombre sur 12 bits divisé par $16 = 2^4$). Il nous transmet ces 12 bits par le procédé de communication 1-wire. Pour communiquer avec le capteur dans notre programme nous nous servons dans la librairie "wire1.asm" qui contient plusieurs macro et sous-routine utiles pour établir une telle communication.

Pour débuter la communication avec le capteur, notre programme commence par envoyer une impulsion de reset (c'est le signal qui permet en 1-wire d'indiquer le début d'une communication : on tire la ligne à zéro). Ensuite comme il n'y a qu'un périphérique qui communique sur la ligne on saute l'identification de celui-ci car on sait que le périphérique qui nous répond ne peut être que notre capteur. Ensuite, on lance la lecture de la température par le capteur et on attend 750 ms pour que le processus puisse se faire. Enfin les lignes de codes suivantes permettent d'acquérir la température et de la stocker sur 2 registres (d3 et d2). Pour cela, on commence par ré-indiquer le début de la communication par un reset de la ligne, on re-saute l'identification des périphériques, ensuite on indique que l'on veut lire ce qui est stocké dans la mémoire temporaire (à savoir la température convertie juste avant) et enfin on récupère ses valeurs dans les deux registres d2 et d3 en commençant pas stocker le LSB, puis le MSB.

Servo-moteur : Le servo-moteur que nous utilisons est le S3003. Il peut se positionner à un angle précis, défini par le duty cycle des impulsions que nous lui envoyons, et maintenir sa position (méthode de PWM). Pour le piloter notre programme créer des impulsions d'un durée calculée au préalable, proportionelle à la température lue par le capteur. On commence la création de ces durée au début de notre programme en mettant le pin de notre moteur à zéro et en attendant 20 ms afin de nous assurer que le programme va au moins maintenir à zéro la ligne pendant ce temps qui est un temps minimal à respecter pour la période de nos impulsions. Ensuite, à la fin de notre programme, nous allons mettre à 1 le bit de contrôle du moteur pendant la durée calculée plus tôt dans le programme et stockée dans a1 et a0. Cela va avoir pour effet de créer un signal compréhensible par le servo, avec des pulses comprises entre 1ms et 2ms sur une période totale supérieur à 20ms.

Écran LCD : Ce périphérique est notre périphérique de communication avec l'utilisateur. Il permet au programme de nous transmettre des données compréhensibles. Pour pouvoir écrire sur cette écran il faut d'abord le formater en remplissant convenablement un registre nommé registre d'instruction de l'écran LCD. C'est ce que fait la sous-routine lcd_init, de la librairie "lcd.asm", appelée dans le reset du programme. Ensuite pour afficher des caractère sur l'écran il faut gérer deux paramètres principaux : le curseur de l'écran LCD (c'est l'endroit où nous sommes placé sur l'écran) et ce qu'il faut écrire qui sera placé dans le data-register. On envoit alors en code ASCII le caractère dans le data-register qui sera par la suite écrit sur l'écran là où se trouve le curseur. Cependant, pour écrire des textes complexes c'est une méthode très fastidieuse et trop longue. Heureusement, on a à notre disposition la librairie "printf.asm" qui nous fournit la macro PRINTF permettant d'afficher des phrases complètes ou des nombres contenus dans des registres directement avec un formatage que l'on définit.

```
.include "macros.asm"       ; include macro definitions
.include "definitions.asm"  ; include register/constant definitions


;========== définitions ==========
.equ    SCROLL=0
.equ    EDIT=1
.equ    mode_min=0
.equ    mode_max=1
.equ    mode_tol=2
.equ    mode_alr=3
.equ    mode_son=4
.equ    mode_info=5
.equ    temp_min=0
.equ    temp_max=125
.equ    tol_max=5
.equ    portMot = PORTD

.def    mode=r6
.def    action=r7

;============ Allocations de mémoire ===========
.dseg
min_:       .byte   1       ; allocation de mémoire pour la variable min_ stockant la limite
  min
max_:       .byte   1       ; allocation de mémoire pour la variable max_ stockant la limite
  max
int_:       .byte   1       ; allocation de mémoire pour la variable int_ indiquant si une
  interruption à déjà été effectué
tol_:       .byte   1       ; allocation de mémoire pour la variable tol_ stockant la tolérance
alr_:       .byte   1       ; allocation de mémoire pour la variable alr_ stockant si l'alarme
  est on ou off
son_:       .byte   1       ; allocation de mémoire pour la variable son_ stockant la sonnerie
.cseg

;========== definitions des macros ==========
.macro  DIV4B4  ; permet de diviser par 16 un nombre à 4 bits
    LSR4    @0,@1,@2,@3
    LSR4    @0,@1,@2,@3
    LSR4    @0,@1,@2,@3
    LSR4    @0,@1,@2,@3
.endmacro

.macro MUL2B4   ; permet de multiplier par 16 un nombre à 2 bits
    LSL2    @0,@1
    LSL2    @0,@1
    LSL2    @0,@1
    LSL2    @0,@1
.endmacro

.macro BORDS ;reg, reg (low value), reg (high value) ; permet de s'assurer qu'un variable est
  contenue dans les bords definis
    cp      @0,@1
    brne    PC+2
    inc     @0
    cp      @0,@2
    brne    PC+2
    dec     @0
.endmacro
```

```
;========== Entré dans la partie d'initialisation de notre programme ==========
;========== tables d'interruptions ==========
.org 0
    rjmp    reset
.org INT6addr ; interruption du bouton de l'encodeur angulaire
    rjmp    ext_int6

;========== routines de service d'interruptions ==========
ext_int6:
    push    a3              ; sauvegarde de a3
    lds     a3,int_         ; récuperation de la valeur de int_
    tst     a3
    breq    prm_int         ; si int_ est égale à 0 (interuption pas déjà effectué avant)
      branch sur prm_int

    ldi     _w,mode_min     ; les 4 lignes suivantes remettent le mode d'édition par défaut
    mov     mode,_w
    ldi     w,SCROLL
    mov     action,_w
    ldi     a3,0
    sts     int_,a3         ; remise de int_ à 0
    pop     a3              ; remise de a3 à son état initiale
    rjmp    affichage       ; jmp jusqu'au mode d'affichage

prm_int:    ; permet d'indiquer au programme que la première interuption (qui se déclenche
  intempestivement) a été traitée
    ldi     a3,1
    sts     int_,a3         ; met int_ à 1
    pop     a3              ; remise de a3 à sont état initiale
    reti                    ; reprise du programme à l'endroit ou il s'était arrêté


;========== initialisation (reset) ==========
reset:
    LDSP    RAMEND
    OUTI    DDRD,0xff       ; configure le portD(moteur) en sortie
    OUTI    DDRE,1<<SPEAKER ; configure le pin Speaker en sortie

    OUTI    EIMSK,0b01000000    ; autorise l'interrupton 6 <=> bouton encoder
    in      _w,EICRB
    ori     _w,0b00100000   ; met l'interruption 6 en flanc descendant sans changer les autres
    out     EICRB,_w
    cli                     ; désactive les interruptions par défaut

    rcall   wire1_init      ; initialise l'interface 1-wire
    rcall   lcd_init        ; initialise l'écran LCD
    rcall   encoder_init    ; initialise l'encodeur

    ldi     _w,mode_min
    mov     mode,_w         ; choisi la valeur minimum comme la valeur à changer de base
    ldi     _w,SCROLL
    mov     action,_w       ; choisi l'action par défaut comme étant SCROLL
    ldi     _w,temp_min
    sts     min_,_w         ; met temp_min et temp_max dans min_ et max_
    ldi     _w,temp_max
    sts     max_,_w
    ldi     _w,1
    sts     son_,_w         ; met la sonnerie de base à zéro
    ldi     _w,0
```

```
    sts     tol_,_w         ; met la tolérance de base à zéro
    sts     alr_,_w         ; met l'alarme de base sur off
    sts     int_,_w         ; mise de int_ à 0 par défaut
    rjmp    affichage

;========== include des différentes routines ==========
.include "lcd.asm"          ; include LCD driver routines
.include "printf.asm"       ; include formatted printing routines
.include "wire1.asm"        ; include Dallas 1-wire(R) routines
.include "math.asm"         ; include Math routines
.include "encoder.asm"      ; include encoder routines
.include "menu.asm"         ; include menus routines
.include "sound.asm"        ; include les notes de musiques

;========== Les partitions ==========
son1:
.db     si2,so2,si2,0
son2:
.db     la2,do3,mi3,0
son3:
.db     dom2,fam2,som2,la2,0

;========== Les sous-routines ==========
quelle_sonnerie:    ; sous-routines permettant de positionner le pointeur z sur la bonne sonnerie
    lds     w,son_
    cpi     w,1
    brne    PC+3
    ldi     zl,low(2*son1)
    ldi     zh,high(2*son1)
    cpi     w,2
    brne    PC+3
    ldi     zl,low(2*son2)
    ldi     zh,high(2*son2)
    cpi     w,3
    brne    PC+3
    ldi     zl,low(2*son3)
    ldi     zh,high(2*son3)
    ret

;========== Entrée dans la partie de configuration des paramètres ==========
; === affichage du mode actuel ===
affichage:
    rcall   lcd_clear
    mov     _w,mode             ; met la valeur contenue dans mode dans _w pour pouvoir effectuer des op avec des imediate
    cpi     _w,mode_tol
    brne    PC+2
    rjmp    affichage_mode_tol  ; si le mode est le mode_tol on va a affichage_mode_tol
    cpi     _w,mode_alr
    brne    PC+2
    rjmp    affichage_mode_alr  ; si le mode est le mode_alr on va a affichage_mode_alr
    cpi     _w,mode_son
    brne    PC+2
    rjmp    affichage_mode_son  ; si le mode est le mode_son on va a affichage_mode_son
    cpi     _w,mode_info
    brne    PC+2
    rjmp    affichage_mode_info ; si le mode est le mode_info on va a affichage_mode_info
```

```
affichage_mode_temp:
    lds     a2,min_         ; récupère les valeurs de min_ et max_ stockées en SRAM
    lds     a3,max_
    PRINTF  LCD
.db "MIN=", FDEC, 20, " , MAX=", FDEC, 21, LF, 0
    mov     a0,action
    rcall   menui           ; affiche SCROLL ou EDIT suivant la valeur de action
.db "SCROLL  |EDIT    ",0
    mov     a0,mode
    rcall   menui           ; affiche MIN ou MAX suivant la valeur de mode
.db "MIN|MAX",0
    rjmp    jump_action


affichage_mode_tol:
    lds     b0,tol_         ; récupère la valeur de la tolérance
    PRINTF  LCD
.db "TOLERANCE : ",FDEC,b,LF,0
    mov     a0,action
    rcall   menui           ; affiche SCROLL ou EDIT suivant la valeur de action
.db "SCROLL  TOL|EDIT    TOL",0
    rjmp    jump_action


affichage_mode_alr:
    PRINTF  LCD
.db "ALARME :  ",0
    lds     a0,alr_         ; récupère la valeur de l'alarme
    rcall   menui           ; affiche ON ou OFF suivant la valeur de alr_
.db "OFF|ON",0
    PRINTF LCD
.db LF,0
    mov     a0,action
    rcall   menui           ; affiche SCROLL ou EDIT suivant la valeur de action
.db "SCROLL ALARME|EDIT   ALARME",0
    rjmp    jump_action


affichage_mode_son:
    PRINTF  LCD
.db "SONNERIE :  ",0
    lds     a0,son_         ; récupère la valeur de la sonnerie
    PRINTF LCD
.db FDEC,a,LF,0
    mov     a0,action
    rcall   menui           ; affiche SCROLL ou EDIT suivant la valeur de action
.db "SCROLL SONNERIE|EDIT   SONNERIE",0
    rjmp    jump_action


affichage_mode_info:
    PRINTF  LCD
.db " CONFIRMER LES",LF,"  PARAMETRES",0

jump_action:
    WAIT_MS 5               ; attente de 5ms pour etre sur que l'affichage sur le LCD ait le
      temps de se faire
    mov     _w,action
    cpi     _w,EDIT
    breq    action_edit     ; branchement au code de action_edit si la valeur de action est
```

```
      EDIT


action_scroll:
    mov     a0,mode
    rcall   encoder         ; vérifie l'état de l'encodeur angulaire
    brtc    PC+2            ; si le bouton n'est pas appuyé, on skip le rjmp
    rjmp    change_action

    CYCLIC  a0,mode_min,mode_info   ; nous assure que la valeur de mode appartient à [mode_min, ↪
       mode_info]
    mov     mode,a0                 ; change le mode
    rjmp    affichage               ; retourne à l'affichage


action_edit:
    mov     _w,mode
    cpi     _w,mode_max
    breq    edit_max        ; si on est en mode mode_max on edite le max
    cpi     _w,mode_tol
    breq    edit_tol        ; si on est en mode_tol on edite la tolérance
    cpi     _w,mode_alr
    breq    edit_alr        ; si on est en mode_alr on édite l'alarme
    cpi     _w,mode_son
    brne    PC+2            ; si on est pas en mode_son on skip le rjmp (car on peux pas branch ↪
       direct sur edit_son)
    rjmp    edit_son


edit_min:   ; vérifie et met à jour la valeur du min
    mov     a0,a2
    rcall   encoder
    brtc    PC+2            ; si le bouton de l'encodeur n'est pas pressé, on skip le rjmp
    rjmp    change_action

    ldi     _w,temp_min-1   ; permet de nous assurer que min_ peut etre égal à temp_min
    BORDS   a0,_w,a3        ; assure que temp_min <= min_ < max_
    sts     min_,a0         ; met la valeur de a0 dans min_
    rjmp    affichage


edit_max:   ; vérifie et met à jour la valeur du max
    mov     a0,a3
    rcall   encoder
    brtc    PC+2            ; si le bouton de l'encodeur n'est pas pressé, on skip le rjmp
    rjmp    change_action

    ldi     _w,temp_max+1   ; permet de nous assurer que max_ peut etre égal à temp_max
    BORDS   a0,a2,_w        ; assure que min_ < max_ <= temp_max
    sts     max_,a0         ; met la valeur de a0 dans max_
    rjmp    affichage

edit_tol:   ; vérifie et met à jour la valeur de tol_
    mov     a0,b0
    rcall   encoder
    brtc    PC+2            ; si le bouton de l'encodeur n'est pas pressé, on skip le rjmp
    rjmp    change_action

    ldi     b1,-1           ; permet de définir les limites des bords
    ldi     _w,tol_max+1
```

```
    lds     a2,min_         ; charge min_ dans a2
    addi    a2,1            ; permet que la tolérance ne dépasse pas la valeur min
    BORDS   a0,b1,a2        ; on s'assure que a0 <= tol_max et tol <= min_
    BORDS   a0,b1,_w
    sts     tol_,a0         ; met la valeur de a0 dans tol_
    rjmp    affichage


edit_alr:   ; vérifie et met à jour la valeur de alr_
    lds     a0,alr_         ; récupère la valeur de alr_ dans a0
    rcall   encoder
    brts    change_action   ; si le bouton de l'encodeur est pressé, branche à change_action

    CYCLIC  a0,0,1          ; nous assure que la valeur de a0 appartient à [0, 1] (soit on ou
      off)
    sts     alr_,a0         ; met la valeur de a0 dans alr_
    rjmp    affichage


edit_son:   ; vérifie et met à jour la valeur de son_
    lds     a0,son_         ; récupère la valeur de son_
    mov     d0,a0           ; copie la valeur de a0 dans d0
    rcall   encoder
    brts    change_action   ; si le bouton de l'encodeur est pressé, branche à change_action

    CYCLIC  a0,1,3          ; nous assure que la valeur de a0 appartient à [1, 3] (car il ya a
      3 sonneries différentes)
    sts     son_,a0         ; met la valeur de a0 dans son_
    cp      a0,d0           ; vérifie si le son à changer par rapport au passage précedent
    breq    PC+3            ; si oui le branchement n'est pas pris et on joue le son ( afin de
      prévisualiser la sonnerie)
    rcall   quelle_sonnerie ;permet de placer le pointeur z sur la bonne partition
    rjmp    alarme
    rjmp    affichage


change_action:
    mov     _w,mode
    cpi     _w,mode_info
    breq    main_init       ; si le mode est le mode info, alors on rejoins le main

    mov     _w,action
    cpi     _w,SCROLL
    breq    change_to_edit  ; si l'action actuel est SCROLL alors on la change pour EDIT


change_to_scroll:           ; sinon on la change pour SCROLL
    ldi     _w,SCROLL
    mov     action,_w
    rjmp    affichage


change_to_edit:
    ldi     _w,EDIT
    mov     action,_w
    rjmp    affichage


;========== Entrée dans la partie de fonctionnement principale de notre programme ==========
main_init:
```

```
    sei                 ; réactive les interruptions pour pouvoir savoir quand le bouton de
      l'encodeur est pressé
    rcall   lcd_clear
    lds     a0,alr_     ; récupère la valeur de alr_ dans a0
    rcall   menui
.db "ALR OFF, |ALR ON,  ",0
    lds     a0,son_     ; récupère la valeur de son_ dans a0
    PRINTF  LCD
.db "SON : ",FDEC,a,LF,0


;========== main program ===========
main:
    P0          portMot,SERVO1  ; met le pin du servo à 0
    WAIT_US     20000           ; permets de créer un signal compréhensible par le servo

; === recuperation de la temp (a1,a0) ===
    rcall   wire1_reset         ; envoie une impulsion de reset
    CA      wire1_write, skipROM
    CA      wire1_write, convertT   ; commence la conversion de température
    WAIT_MS 750                 ; attend 750ms afin que le processus ait le temps de se faire

    rcall   wire1_reset
    CA      wire1_write, skipROM
    CA      wire1_write, readScratchpad
    rcall   wire1_read          ; lit le LSB de la température
    mov     d2,a0
    rcall   wire1_read          ; lit le MSB de la température
    mov     d3,a0               ; la température est stcoké dans d2,d3

    PRINTF  LCD                 ; affichage de la température actuelle
.db " temp =",FFRAC2+FSIGN,14,4,$32,"C",CR,0


verif_temp:     ; permet de vérifier que la température actuel est compris dans [min_;max_]
    clr     b1
    lds     b0,min_         ; récupère la valeur de min_
    MUL2B4  b1,b0           ; la multiplie par 16 (car le capteur nous donne T*16)
    CP2     d3,d2,b1,b0     ; la compare avec la température actuelle
    brlt    tooL            ; si la température est plus petite on va à tooL
    clr     b1
    lds     b0,max_         ; récupère la valeur de max_
    MUL2B4  b1,b0           ; la multiplie pas 16
    CP2     d3,d2,b1,b0     ; la compare avec la température actuelle
    brlt    PC+2            ; si la température est plus grande on va à tooH
    rjmp    tooH

    rjmp    calcul_servo    ; si la température est comprise dans [min_;max_] on va directement
       à calcul_servo


alarme: ; permet de faire biper le buzer avec la sonnerie définie auparavant
    lpm                     ; récupère la valeur à l'adresse de z et la stock dans r0
    tst     r0              ; vérifie si z0 est égale à 0
    breq    fin_alr         ; si oui, on a fini la partition, on jump donc a fin_alr
    adiw    zl,1            ; incrémente le pointeur z
    mov     a0,r0           ; stock la valeur de r0 dans a0 (car il faut pouvoir effectuer des
      op avec des imediate)
    ldi     b0,100
    rcall   sound           ; appelle la sous-routine sound qui permet de jouer le son d'une
```

```
      note
    rjmp    alarme          ; revient à alarme pour jouer la suite de la partition
fin_alr:
    lds     b0,int_         ; récupère la valeur de int_ dans b0
    tst     b0              ; teste si c'est égal à zéro (si oui on est pas encore passé dans
      le main)
    breq    ps_main         ; si oui on jump à ps_main
    MOV2    a1,a0,b3,b2     ; récupère la valeur des impulsions à imprimmer au moteur
    rjmp    pos_servo       ; jmp au positionnement du servo
ps_main:
    rjmp    affichage       ; on revient à affichage


tooL:   ; fixe la valeur des pulsations à 1000 us et va à alarme si la température est trop
  froide de plus de 5 degrés
    LDI2    a1,a0,1000      ; permet de fixer la valeur des pulses à 1000us
    clr     d1
    lds     d0,tol_         ; place la valeur de la tolérance dans d0
    MUL2B4  d1,d0           ; permet de multiplier la tolérance par 16 (car le capteur nous
      donne 16*T)
    SUB2    b1,b0,d1,d0     ; soustrait tol_ degré à la temp min_
    CP2     d3,d2,b1,b0
    brge    ps_alrL         ; si la temp actuelle n'est pas plus froide que min_-tol on prend
      le branchement
    lds     b1,alr_         ; on charge la valeur de alr_ dans b1
    tst     b1              ; teste si b1 est égale à 0
    breq    ps_alrL         ; si oui l'alarme est désactiver et donc on prend le branchement
    rcall   quelle_sonnerie ; on place z à l'adresse de la bonne partition
    MOV2    b3,b2,a1,a0     ; permet de sauvegarder la valeur des impulsion a imprimer au
      moteur
    rjmp    alarme
ps_alrL:
    rjmp    pos_servo


tooH:   ; fixe la valeur des pulsations à 2000 us et v à alarme si la température est trop
  chaude de plus de 5 degrés
    LDI2    a1,a0,2000      ; permet de fixer la valeur des pulses à 2000us
    clr     d1
    lds     d0,tol_         ; place la valeur de la tolérance dans d0
    MUL2B4  d1,d0           ; permet de multiplier la tolérance par 16 (car le capteur nous
      donne 16*T)
    ADD2    b1,b0,d1,d0     ; ajoute tol_ degrés à la temp max_
    CP2     d3,d2,b1,b0
    brlt    ps_alrH         ; si la temp actuelle n'est pas plus chaude que max_+tol on prend
      le branchement
    lds     b1,alr_         ; on charge la valeur de alr_ dans b1
    tst     b1              ; teste si b1 est égale à 0
    breq    ps_alrH         ; si oui l'alarme est désactiver et donc on prend le branchement
    rcall   quelle_sonnerie ; on place z à l'adresse de la bonne partition
    MOV2    b3,b2,a1,a0     ; permet de sauvegarder la valeur des impulsion a imprimer au
      moteur
    rjmp    alarme
ps_alrH:
    rjmp    pos_servo

calcul_servo:
; === calcul de coef dir de la linéarisation de la température (b1,b0) ===
    CLR2    b3,b1
    lds     b2,min_         ; récupère la valeur de min_
```

```
    lds     b0,max_         ; récupère la valeur de max_
    SUB2    b1,b0,b3,b2     ; calcule deltaT
    LDI2    a1,a0,(2000-1000)
    rcall   div22           ; calcul du coef directeur
    MOV2    b1,b0,c1,c0     ; le stock dans b1,b0

; === calcul du nombre de pulse np = 1/16 * (cd*T + 16cd*min_ + 16000) ===
    MOV2    a1,a0,d3,d2     ; récupère la valeur de la température actuelle
    rcall   mul22           ; multiplie par le cd
    MOV4    d3,d2,d1,d0,c3,c2,c1,c0 ; stocke la valeur dans d3,d2,d1,d0

    MUL2B4  b1,b0           ; multiplie le cd par 16
    MOV2    a1,a0,b3,b2     ; récupère la valeur de min_
    rcall   mul22           ; multiplie les deux ensembles

    CLR4    a3,a2,a1,a0
    ADD4    a3,a2,a1,a0,d3,d2,d1,d0 ; les trois prochaines lignes effectuent le cd*T +
      16cd*min_ + 16000
    SUB4    a3,a2,a1,a0,c3,c2,c1,c0
    ADDI4   a3,a2,a1,a0,16000

    DIV4B4  a3,a2,a1,a0     ; division par 16 du tout


pos_servo:  ; permet de positionner le servo au bonne endroit
; === positionnement du moteur ===
    P1      portMot,SERVO1  ; met le pin du servo à 1
loop:   ;la loop permettant de faire des pulses de la bonne durée
    SUBI2   a1,a0,1
    brne    loop
    rjmp    main
```