Rapport Microcontrôleur

1 Introduction

Ce rapport documente le développement de notre projet réalisé dans le cadre du cours *EE-208 Microcontrôleurs et système numérique*. Suite à l'acquis de formation de ce cours, nous avons entrepris la réalisation d'un dispositif employant un microcontrôleur ATmega-128. Celui-ci s'y retrouve dans de nombreux objets du quotidien, ainsi programmer un système embarqué en faisant l'utilisation est donc quelque chose de très raisonnable. La réalisation de ce projet nous a permis d'ajouter une partie pratique à notre apprentissage, nous entrainant à consolider fortement nos connaissances quant à la programmation des microcontrôleurs en langage assembleur.

1.1 Description de l'application

Le dispositif que nous avons réalisé est un système paramétrable, par l'utilisateur, qui permet d'activer une alarme lorsqu'une certaine température seuille est dépassée et qui s'arrête uniquement lorsqu'un certain code est entré. A sa désactivation, une pompe montée sur un moteur servo permet l'arrosage à 360 degrés afin de refroidir, avec de l'eau, le potentiel environnement dans lequel le système se trouve. Le temps de rotation est choisi à l'aide de l'équation d'Arrhenius qui est la loi, dans le monde des chimistes, qui lie la vitesse d'une réaction chimique à la température. Ainsi nous avons remplacé la vitesse de réaction chimique par celle du temps de rotation du moteur servo. Une interface visuelle est proposée avec un écran LCD. L'équation d'Arrhenius est la suivante :

$$\mathrm{k} = \mathrm{A}e^{\frac{-E_a}{RT}}$$

Où k est la vitesse de réaction chimique, dans notre cas le temps de rotation du servo moteur. A est une constante, pour ce projet nous avons choisi $A = 1$. E_a est l'énergie d'activation, dans nos calculs, nous avons choisis $E_a = -100$. R est la constante des gaz parfaits et T la température.

1.2 Objectifs

Notre objectif était de créer un dispositif alignant plusieurs périphériques ainsi qu'une interface utilisateur qui soit la plus instinctive possible. Nous voulions également que notre système soit utilisé afin de récolter une donnée d'un capteur, avec laquelle il fasse un calcul complexe et qu'en fonction de cette donnée et ce calcul, il génère autonomement un certain comportement afin d'avoir système automatisé. L'idée était de créer un system de control avec un « feedback loop » complet et rigoureux.

Un autre objectif fixé était celui d'avoir un code optimisé afin de bénéficier du plein avantage de programmer en assembleur et d'économiser de la mémoire et des coups d'horloge inutiles.

2 Mode d'emploi

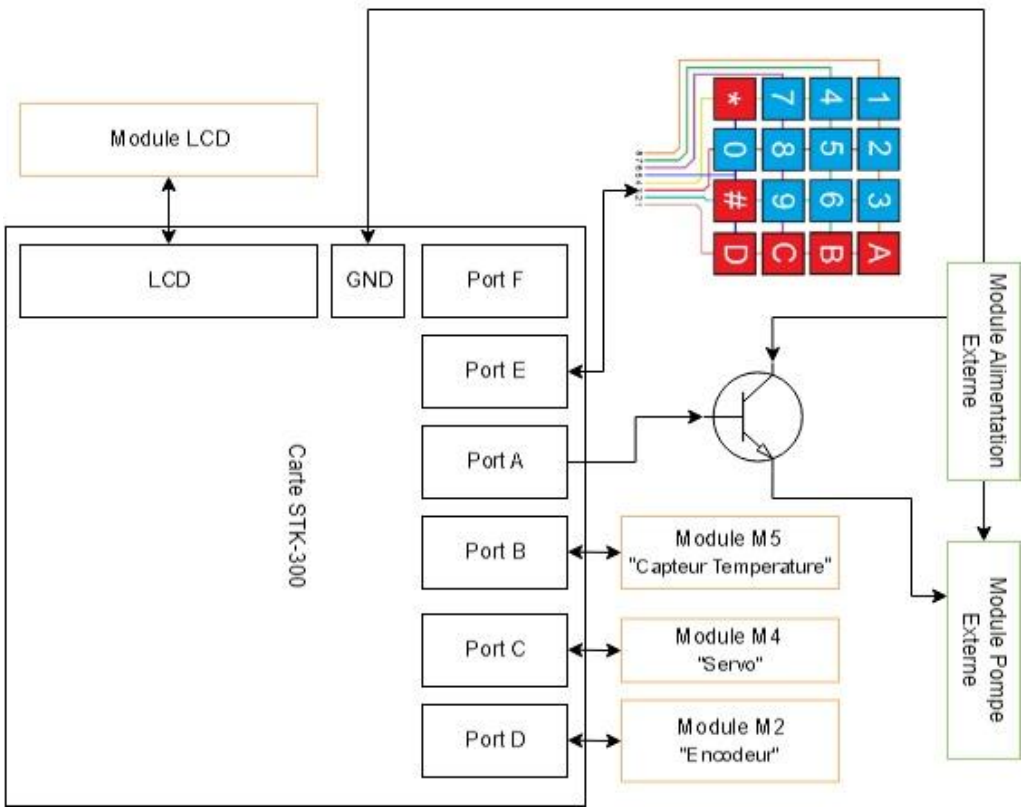
2.1 Montage du système

Avant de détailler l'utilisation du dispositif, il est important d'effectuer le bon montage afin que le dispositif fonctionne comme il se doit.

1. Alimenter la carte sur une prise secteur.

2. Branchement les périphériques :
   2.1. **Key-pad :** Brancher les lignes 1 à 4 du keypad sur les bits 4 à 7, respectivement du Port E.
   2.2. **Encodeur :** Brancher le module de l'encodeur sur le Port D.
   2.3. **Servo :** Brancher le module du moteur servo sur le Port C
   2.4. **Capteur de température** : Brancher le sur le Port B.
   2.5. **Pompe:** Brancher la base d'un transistor NPN au pin 1 du Port A et, en utilisant le module d'alimentation externe, alimenter la pompe avec la sortie 5 V. Connecter le GND de la carte avec celui du module d'alimentation ainsi qu'au collecteur du transistor. Finalement, connecter l'émetteur du transistor au GND de la pompe.
   2.6. **LCD:** Mettre l'écran LCD tout en haut de la carte, sur ces pins attitrés.

3. Allumer la carte et flasher le programme depuis un ordinateur.

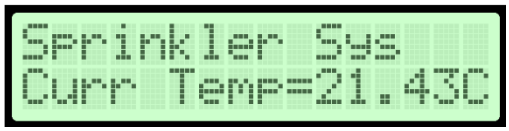


1.Schéma du montage

2.1 Utilisation du système

Mode d'accueil :

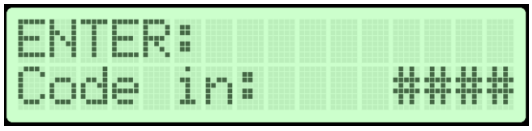
Le système dans son mode de repos affiche sur l'écran LCD la température actuelle qui se rafraichi environ toutes les 2 secondes. A chaque rafraichissement, le système vérifie si la température lue dépasse la température seuille, et si cela est le cas, on entre dans le mode alarme.



2. Affichage du LCD en mode Accueil

Mode alarme :

Dès lors qu'on entre dans ce mode, le buzzer retentit et l'affichage du LCD montre que le système attend un code de cette façon :

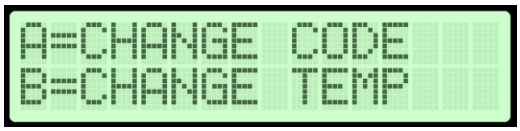


3. Affichage du LCD en mode Alarme

Une fois le bon code entré, il faut appuyer sur la touche « B » du key-pad pour le valider. Le servo moteur se met, ensuite, à tourner pendant x secondes, et la pompe fixée sur le servo permet de propulser un faible jet d'eau. x est calculé grâce à une série de Taylor sur 2-bytes qui approxime l'équation d'Arrhenius. Lorsque ces secondes sont écoulées, on revient au menu d'accueil et la température seuille est remise à celle originale (30 degrés Celsuis).

Select mode :

Si l'on désire modifier la température seuille ou le code, il faut, lorsque nous somme dans le mode d'accueil, appuyer sur n'importe quelle touche du key-pad, ensuite entrer le code actuel et le valider en appuyant sur « B ». Une fois cela fait, l'état du LCD devient le suivant :

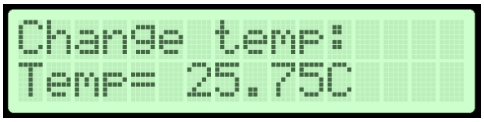


4. Affichage LCD mode : select mode

(Attention, une fois quitté le mode d'accueil l'alarme ne peut plus retentir). Comme indiqué, pour choisir quel paramètre modifier, il faut appuyer sur « A » ou « B ».

Mode changer température :

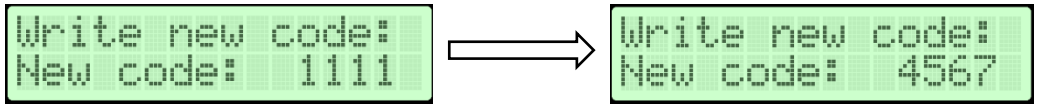
Pour changer la température la température seuille, il faut actionner l'encodeur. En le tournant dans le sens horaire, nous incrémentons la température et inversement pour la diminuer. Si nous actionnons l'encodeur sans appuyer dessus, cela inc/décrémentera la température à la valeur décimale le plus petit possible, cependant, si nous appuyons sur l'encodeur en même temps, la température est inc/décrémenté à l'unité. Pour sauvegarder cela, il faut appuyer sur « A » et nous retournons dans le mode d'accueil. Un exemple de l'affichage dans ce mode peut être vu ci-dessous :



5. Mode changer la température seuille

Mode changer le code

Pour changer le code, il faut réécrire son nouveau code (l'ancien code est lui disposé dès le début afin de pouvoir comparer). Une fois le code choisi, il faut appuyer sur « B » pour sauvegarder le nouveau code qui sera sauvegardé dans l'EEPROM interne du microcontrôleur. Une fois cela fait nous retournons dans le mode d'accueil. L'exemple ci-dessous montre une transition d'un code « 1111 » a « 4567 » :



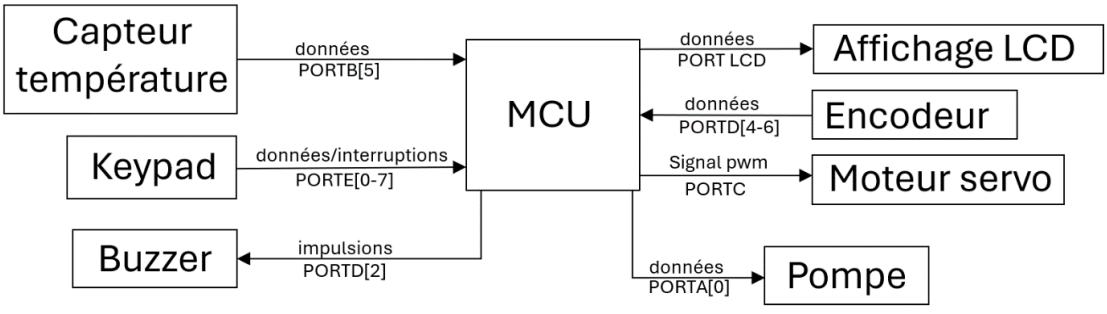
6.Mode changer le code

3 Aspect technique

Cette partie du rapport décrit les aspects techniques du dispositif d'un point de vue matériel et d'un point de vue du programme. Nous en profiterons pour présenter les différentes librairies utilisées.

3.1 Mise en place matérielle de l'application

Tous les périphériques utilisés ont été listé dans la section 2.1 et certains seront plus détaillés dans la section 4. La mise en place matérielle du dispositif peut être résumé de façon explicite à l'aide d'un schéma :



7.Logistique matérielle

Le microcontrôleur (MCU) est donc au milieu et gère tout ce qui est niveau code pour le transfert de données et exécution d'instructions.

3.2 Bibliothèques

Le lien entre les périphériques est la plupart du temps effectué à l'aide de bibliothèques qui sont expliquées ci-dessous :

Certains modules ont été emprunté du cours, parmi eux, certains ont été utilisés tel-quel, sans modifications. Font partis de cette catégorie : *wire1.asm, macro.asm, lcd.asm, definitons.asm, eeprom.asm* et *math.asm*.

Toujours parmi les modules du cours, nous avons fait la décision de modifier la bibliothèque *encoder.asm*, afin de pouvoir l'adapter à notre système afin que l'inc/décrémentation se fasse sur 2 bytes et non 1. Nous voulions également ajouter la fonctionnalité de pouvoir, lorsque l'encodeur est appuyé, inc/décrémenter un certain bit du premier octet. Cela devait être fait afin d'inc/décrémenter par unité (en degré Celsius) et donc gagner un temps considérable et une excellente précision pour régler la température seuille.

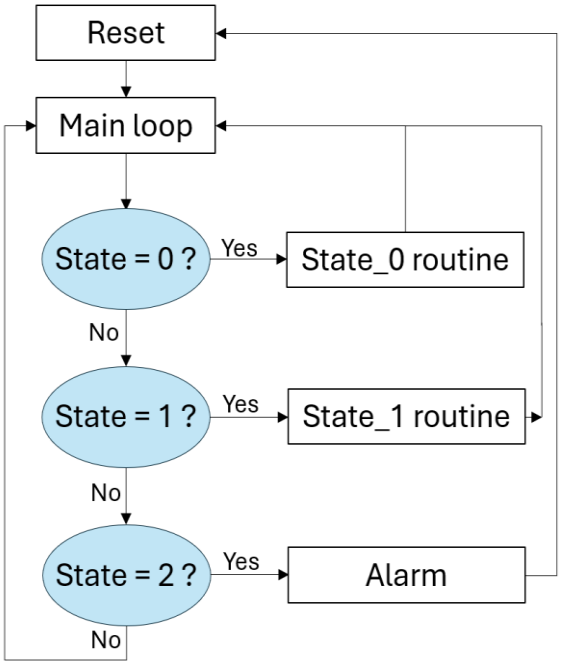
Finalement, nous avons créé la bibliothèques « taylor_2byte.asm ». Cette bibliothèque a pour objectif de calculer le coefficient d'Arrhenius en utilisant une série de Taylor pour e^x sur 2-bytes en utilisant une méthode itérative. Son fonctionnement principal commence par initialiser les variables nécessaires, stocker la valeur initiale de la température et push sur la pile les valeurs des registre r0 à r32. Le code procède ensuite à calculer le facteur exponentiel $k = e^{\frac{100}{8.31 \cdot T}}$, où la division et l'échelle sont gérées par des sous-routines spécifiques.

Parmi ce module, la sous-routine *exp_cal* calcule la série de Taylor pour e^x en utilisant la formule $1 + x + \frac{x^2}{2} + \frac{x^3}{6}$. La boucle itère à travers les calculs nécessaires pour chaque terme, mettant à jour le résultat en effectuant des multiplications pour les puissances de x et des divisions pour les termes factoriels. Les résultats intermédiaires sont stockés dans des registres et, après le calcul du terme final, le résultat est stocké en mémoire SRAM. L'opération entière implique plusieurs niveaux de sous-routines imbriquées pour la multiplication, la division et le calcul de puissance, assurant ainsi la précision dans le calcul de la série de Taylor. Enfin, le code restaure la pile et retourne, laissant le résultat de e^x dans des emplacements mémoire désignés.

3.3 Boucle principale

La boucle principale effectuée par le main est toute petite comparée à la taille du code. En effet, un registre *state* décrit dans quel état est le système et la boucle main compare ce registre avec ses valeurs possibles et effectue des branchements à d'autre sous-routines permettant la réalisation de toutes les fonctions du système. Voici le schéma illustrant sa manière d'opérer.

Sur le schéma on peut observer :



8. Main loop

State_0 routine qui correspond au dispositif dans son état d'accueil.

State_1 routine correspond à l'état lorsqu'on veut changer des paramètres du système. (Il est bon de savoir que d'autres branchements se font après cette routine.)

Alarm correspond au mode lorsque le dispositif détecte une température supérieure à celle qui est seuille. Ici aussi, des branchements se font après celui-ci. Ces branchements ne seront pas détaillés dans ce rapport car ils susciteraient trop de place mais vous pouvez les observer dans le code en annexe qui est très bien commenté.

Concernant les manières dont *state* est modifié, elles seront traitées dans la section 3.5, car nous devons d'abord développer les interruptions et timers utilisés dans le projet.

A l'issue de *Alarm*, nous retournons dans le reset car ceci nous permet de retourner le système à son état initial et de réallouer complétement la mémoire qui nous permet d'éviter de potentiel collision de données. Ceci n'impose pas de problème pour la suite de l'utilisation du système car le code, lorsqu'il est modifié, est stocké dans l'EEPROM garantissant que même si la carte est éteinte, le code choisit par l'utilisateur reste le même.

3.4 Timers et interruptions

Dans ce projet, nous utilisons les timers 0 et 2. Leurs paramétrisations est indiquées dans le tableau ci-dessous.

Timer	Nb de bits	Clock	Prescaler	Interrupt type	Overflow period
Timer 0	8	32'768 Hz	6	Overflow	2s
Timer 2	8	4MHz	2	Overflow	0.512ms

Le timer 0 est utilisé pour réactualiser la température toutes les 2 secondes. Tandis que le timer 2 est utilisé pour générer un son sur le buzzer lorsque le dispositif est en mode *alarm*.

Pour ce qui est des interruptions sur les ports utilisés, nous utilisons les 4 interruptions INT 4-7 qui sont configurées pour être activée si le pin détecte une tension au niveau bas. Celles-ci sont utilisées pour l'utilisation du key-pad.

3.5 Modification de *state*

Par défaut *state* est mis à 0 et le système est dans son mode d'accueil. Lorsqu'une interruption du key-pad est détectée, la sous routine lié vérifie l'état de *state* et s'il est à 0, elle le met à 1, afin d'entrer dans le mode *state 1 une fois de retour dans le main*. Une fois le branchement vers *state_1*, à l'issu des divers branchements effectués, *state* est remis à 0. Lorsqu'un excès de température est détecté, state est mis à 2 et une fois tous les branchements effectués, après le mode *alarm*, la boucle repasse par le reset est donc *state* est remis à 0.

4 Accès aux périphériques

Dans cette section figure quelques détails, concernant certains périphériques, qui n'ont pas encore était expliqué dans ce rapport.

4.1 Capteur de température

Le périphérique qui permet l'accès à la température courante communique, avec le microcontrôleur, grâce au protocole Dallas 1-wire. Étant donné le long temps de latence demandé par ce processus, lorsque *state* n'est pas égal à 0, la communication ne s'effectue pas, et l'interruption généré par le timer 0 n'a pas d'action particulière. Cela est nécessaire dans le cas où le microcontrôleur n'est pas au repos car le programme doit être réactif face aux interactions de l'utilisateur, que ça soit par le key-pad ou l'encoder.

4.2 Servo moteur

La rotation du servo moteur est générée, dans une boucle « loop », par un signal PWM. Le nombre de fois que cette boucle est parcourue dépend du résultat du calcul de l'équation d'Arrhenius, mais, est programmé avec un offset de 0xf0 afin de garantir dans tous les cas une rotation du moteur.

4.3 LCD

L'affichage sur l'écran LCD se fait grâce aux méthodes du fichier *printf.asm* fournit par le cours. Ce périphérique utilise également un module de transmission qui est UART. De plus, nous avons remarqué que dû au nombre de périphériques utilisé dans notre projet, des collisions sur les bus de la carte entre le LCD et certains ports pouvaient entrainer des comportements erronés. Ainsi notre solution pour remédier à ce problème, a été de réactualiser l'affichage du LCD le moins souvent possible.

4.4 Pompe

La pompe employée est un périphérique qui demande plus de puissance que ce que la carte permet de fournir. Ainsi l'utilisation d'une alimentation externe avec un transistor fonctionnant comme un switch était nécessaire pour la faire fonctionner comme il se doit.

5 Annexes

5.1 Code Principale – *355665_364154.asm*

```
/*
 * Created: 22/05/2024 23:41:19
 *  Author: eliot
 */
 /*
    EPFL - EE-208: Microcontroleurs et Systemes Numeriques
    Semester Project - Spring Semester 2024

    Groupe 014:
    Eliot Abramo - SCIPER 355665
    Mathias Rainaldi - SCIPER 364154

    Project description:
    Sprinkler System with user interface
*/

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

; ====================
; ====== macros ======
; ====================

.macro VERIFY_ENTER
    /* check entry of keypads, deactivates certain keys in certain modes. Check
        if 'B','C','D','*' or '#' and loads case into interm register. */
    ; @2 = interm = intermediate register used to temporary store values
    ; @1 = wr1 = column counter
    ; @0 = wr0 = row counter

    _CPI    @1,0x03
    brne    not_letter
    _CPI    @0,0x01
    breq    verif
    ldi     @2,0x02
    jmp     fin


verif:
    ldi     @2,0x01             ; update the interm to the wanted value
    jmp     fin                 ; jump to end

not_letter:
    _CPI    @0,0x03             ; check if on column 3
    brne    okay

    _CPI    @1,0x01             ; check if column 1
    breq    okay
    ldi     @2,0x02             ; update the interm to the wanted value
    jmp     fin

okay:
    ldi     @2,0x00             ; set default value

fin:
    nop
.endmacro

.macro CHECK_AND_SET
    ; @5 = count = bit of code you want to change
    ; @4 = interm = intermediate register used to temporary store values
    ; @3 = a3 = fourth bit of code
    ; @2 = a2 = third bit of code
    ; @1 = a1 = second bit of code
    ; @0 = a0 = first bit of code

    cpi     @5, 0x00            ; compare a3 with 0x23 ('#' ASCII code)
    breq    set_a0              ; if equal, branch to set_a0
    cpi     @5, 0x01            ; compare a2 with 0x23
    breq    set_a1              ; if equal, branch to set_a1
    cpi     @5, 0x02            ; compare a1 with 0x23
    breq    set_a2              ; if equal, branch to set_a2
    cpi     @5, 0x03            ; compare a0 with 0x23
    breq    set_a3              ; if equal, branch to set_a3
```

```
    set_a0:
        ldi     @5, 0x01
        mov     @0, @4              ; set a3 to interm
        rjmp    end                 ; jump to end
    set_a1:
        ldi     @5, 0x02
        mov     @1, @4              ; set a2 to interm
        rjmp    end                 ; jump to end
    set_a2:
        ldi     @5, 0x03
        mov     @2, @4          ; set a1 to interm
        rjmp    end                 ; jump to end
    set_a3:
        ldi     @5, 0x00
        mov     @3, @4          ; set a0 to interm
    end:
        nop
.endmacro

.macro DECODE_ASCII
    ; used to decode the input into an ASCII value
    ; @2 = interm = intermediate 'temporary' register used in calculations
    ; @1 = wr1 = r1 = column = high bit
    ; @0 = wr0 = r2 = row = low bit
    CLR2 ZL, ZH
    ;point Z to ASCII table
    ldi    ZL, low(2*(KeySet01))
    ldi    ZH, high(2*(KeySet01))
    ;move pointer to value pressed on keypad
    add    ZL, @0
    add    ZL, @0
    add    ZL, @0
    add    ZL, @0
    add    ZL, @1
    ;load ASCII value to temporary register to be used later
    lpm     @2,Z

.endmacro

.macro ROW_MACRO
    ; used to simplify the detection of the correct row of the keypad
    ; @4 = state = current state of system
    ; @3 = mask for the row
    ; @2 = wr0 = row counter
    ; @1 = mask to extract correct row
    ; @0 = row number on keypad (0->3)
    ; check if in state_0, if not update the position on keypad
    _CPI    @4, 0x00
    brne    not_state_0
    WAIT_MS 500
    _LDI    @4, 0x01
    reti

not_state_0:
    _LDI    @2, @0
    _LDI    @3, @1
.endmacro

.macro COLUMN_MACRO
    ; used to simplify the detection of the correct column of the keypad
    ; @2 = column number (0->3)
    ; @1 = next column subroutine address
    ; @0 = column identification

    WAIT_MS     KPD_DELAY
    OUTI        KPDO, @0
    WAIT_MS     KPD_DELAY
    in          w,KPDI
    and         w,mask
    tst         w
    brne        @1
    _LDI        wr1,@2
 .endmacro
```

```
.macro READ_EEPROM
    ; used to read the value at an EEPROM address and store it accordingly
    ; @1 = register to store value at address
    ; @0 = value address in EEPROM
    push xl
    push xh
    push a0

    clr a0
    ldi xl, low(@0)
    ldi xh, high(@0)
    rcall eeprom_load
    mov @1, a0

    pop a0
    pop xh
    pop xl
.endmacro

.macro WRITE_EEPROM
    ; used to write a value to an EEPROM address
    ;@1 = address in EEPROM to store value at
    ;@0 = value to store in EEPROM
    cli
    push    a0
    push    xl
    push    xh
    ;store selected value in the selected adress in the EEPROM
    mov     a0, @0
    ldi     xl, low(@1)
    ldi     xh, high(@1)
    rcall   eeprom_store
    pop     xh
    pop     xl
    pop     a0
    sei
.endmacro

.macro INITIALZE_CODE
    ; used to read the code in the EEPROM and check if it is a possible code
    ; (i.e. the values between 0-9) and store it in the system.
    ; @1 = address in EEPROM
    ; @0 = value to save at address
    push    b0
    clr     b0
    READ_EEPROM     @1, b0
    cpi     b0,0x31
    brlo    set_default
    cpi     b0, 0x44
    brsh    set_default

set_val:
    nop
    mov     @0, b0
    rjmp    end

set_default:
    _LDI    @0, 0x31

end:
    nop
    pop     b0
    nop
.endmacro

; === definitions ===
.equ  KPDD = DDRE
.equ  KPDO = PORTE
.equ  KPDI = PINE

.equ  KPD_DELAY = 30                ; msec, debouncing keys of keypad

.def    wr0 = r2                    ; detected row in hex
.def    wr1 = r1                    ; detected column in hex
.def    mask = r14                  ; row mask indicating which row has been detected in bin
```

```
.def    wr2 = r15                   ; semaphore: must enter LCD display routine, unary: 0 or other
.def    interm = r16                ; intermediate register used in calculations
.def    state = r6                  ; store state of system, three states total
.def    temp0 = r8                  ; temperature0 (LSB)
.def    temp1 = r9                  ; temperature1 (MSB)
.def    chg = r26                   ; reload temperature
.def    count = r27                 ; to know at which character of code to change

.equ    temp_MSB = 0xff0a           ; temperature MSB address
.equ    temp_LSB = 0xff0b           ; temperature LSB address
.equ    code1_address = 0xff0c      ; first address for passcode
.equ    code2_address = 0xff0d      ; second address for passcode
.equ    code3_address = 0xff0e      ; third address for passcode
.equ    code4_address = 0xff0f      ; fourth address for passcode

.dseg
.org 0x0100
    temp_seuil: .byte 2             ; stores the threshold value for the temperature
    code:       .byte 4             ; stores the code of the system
    exp_result: .byte 2             ; stores the result of the taylor series

; === interrupt vector table ===
.cseg
.org 0
    jmp reset

.org 10
    jmp isr_ext_int0                ; external interrupt INT4
    jmp isr_ext_int1                ; external interrupt INT5
    jmp isr_ext_int2                ; external interrupt INT6
    jmp isr_ext_int3                ; external interrupt INT7

.org OVF0addr                       ; external interrupt for temperature sensor
    rjmp read_temp

.org OVF2addr                       ; timer overflow 2 interrupt vector
    rjmp overflow2

; === interrupt service routines ===
.org    0x30

isr_ext_int0: ; detect row 0
    ROW_MACRO   0x00, 0b00010000, wr0, mask, state
    rjmp        column_detect

isr_ext_int1: ; detect row 1
    ROW_MACRO   0x01, 0b00100000, wr0, mask, state
    rjmp        column_detect

isr_ext_int2: ; detect row 2
    ROW_MACRO   0x02, 0b01000000, wr0, mask, state
    rjmp        column_detect

isr_ext_int3: ; detect row 3
    ROW_MACRO   0x03, 0b10000000, wr0, mask, state
    rjmp        column_detect

column_detect:
    OUTI        KPDO,0xff   ; bit4-7 driven high

col7: ; check column 7
    COLUMN_MACRO    0xf7, col6, 0x03
    rjmp            isr_return

col6: ; check column 6
    COLUMN_MACRO    0xfb, col5, 0x02
    rjmp            isr_return

col5: ; check column 5
    COLUMN_MACRO    0xfd, col4, 0x01
    rjmp            isr_return

col4: ; check column 4
    COLUMN_MACRO    0xfe, isr_return, 0x00
```

```
    rjmp            isr_return

isr_return:
    ldi             _w,10  ; sound feedback of key pressed acknowledge

beep01:
    OUTI KPDO,      0xf0
    _LDI            wr2,0xff
    reti

read_temp: ; temperature sensor interrupt routine
    _CPI    state,0x00                  ; check if in first state
    breq    PC+2
    reti

    push    a0                          ; a0 might change, save on stack in case
    push    a1

    rcall   wire1_reset                 ; send a reset pulse
    CA      wire1_write, skipROM        ; skip ROM identification
    CA      wire1_write, convertT       ; initiate temp conversion
    WAIT_MS 750                         ; wait 750 msec

    rcall   lcd_home                    ; place cursor to home position
    rcall   wire1_reset                 ; send a reset pulse
    CA      wire1_write, skipROM
    CA      wire1_write, readScratchpad ; send address to read value from sensor

    rcall   wire1_read                  ; read temperature LSB
    mov     c0, a0                      ; save temperature LSB
    rcall   wire1_read                  ; read temperature MSB
    mov     temp1,a0                    ; save temperature MSB
    mov     temp0,c0
    ldi     chg,0xff                    ; indicate temperature has changed


    ; now we compare with the temperature seuil
    ldi     xl, low(temp_seuil)
    ldi     xh, high(temp_seuil)
    ld      a0, x+
    ld      a1, x
    CP2     temp1,temp0,a1,a0

    pop     a1                          ; restore a1
    pop     a0                          ; restore a0
    brlo    PC+2
    _LDI    state, 0x02
    reti

overflow2 :
    INVP DDRD,SPEAKER
    reti

; === include necessary libraries ===
.include "lcd.asm"      ; include UART routines
.include "printf.asm"   ; include formatted printing routines
.include "eeprom.asm"   ; include internal EEPROM routines
.include "wire1.asm"    ; include one-wire protocol routines
.include "encoder.asm"  ; include encoder routines

; === initialization and configuration ===
.org 0x400
reset:
    LDSP    RAMEND              ; Load Stack Pointer (SP)
    ;=== save state of MCU control register ===
    in      _w, MCUCR
    sts     0xDDDD, _w
    clr     _w

    ;=== initialize the protocols ===
    rcall   LCD_init            ; initialize UART
    rcall   wire1_init          ; initialize 1-wire(R) interface
    rcall   encoder_init        ; initialize encoder interface
```

```
    ;=== configure output pins ===
    OUTI    DDRA, 0x01          ; configure PORTA to output
    OUTI    PORTA, 0x00         ; ensure pin not yet activated
    OUTI    DDRC,0xff           ; configure portC to output
    OUTI    DDRD,0xff           ; configure portD to output
    sbi     DDRD,SPEAKER        ; set bit speaker is connected to 1

    ;=== configure keypad pins ===
    OUTI    KPDD,0x0f           ; bit0-3 pull-up and bits4-7 driven low
    OUTI    KPDO,0xf0           ; >(needs the two lines)

    ;=== configure interrupts ===
    OUTI    EIMSK,0xf0          ; enable INT4-INT7
    OUTI    EICRB,0b00          ; >at low level

    ;=== configure timer ===
    OUTI    TIMSK,(1<<TOIE0)    ; timer0 overflow interrupt enable
    OUTI    ASSR, (1<<AS0)      ; clock from TOSC1 (external)
    OUTI    TCCR0,6             ; CS0=1 CK
    OUTI    TCCR2,2             ; prescaler for the buzzer

    ;=== set temperature limit ===
    _LDI    a0,0xe0             ; corresponds to 30 degree celcius
    _LDI    a1,0x01

    ldi     xl,low(temp_seuil)
    ldi     xh,high(temp_seuil)
    st      x,a0
    inc     xl
    st      x,a1

    ;=== set initial code ===
    ; initializes all 4 bits of the code using macro
    INITIALZE_CODE      a0, code1_address
    INITIALZE_CODE      a1, code2_address
    INITIALZE_CODE      a2, code3_address
    INITIALZE_CODE      a3, code4_address

    ; add code loaded in to pointer x, to be used later in verification
    ldi     xl,low(code)
    ldi     xh,high(code)
    st      x+,a0
    st      x+,a1
    st      x+,a2
    st      x,a3

    ;=== set initial values ===
    PRINTF LCD                  ; display initial LCD message
.db FF,CR,"Sprinkler Sys",0

    _LDI    a0, 0x23            ; sets the a registers to #
    _LDI    a1, 0x23            ; for display purposes
    _LDI    a2, 0x23
    _LDI    a3, 0x23
    _LDI    state, 0x00         ; set initial state to 0

    ;=== clear registers ===
    CLR8 count, wr0, wr1, wr2, chg, b1, b2, b3
    sei                         ; enable interrupt

; === main program ===
main:
    _CPI    state,0x00  ; check if in state 0
    breq    state_0

    _CPI    state,0x01  ; check if in state 1
    brne    PC+2
    rjmp    state_1

    _CPI    state,0x02  ; check if in state 2
    brne    PC+2
    rjmp    alarm
```

```
    nop
    rjmp    main

;======================
; ==== sub-routines ====
;======================
/****** Ordered as followed ******
    -> State 0 - Home Page
    -> State 1 - Enter Code
    -> State 2 - Alarm and Servo

    -> Code Verifcation
    -> Store and Load from EEPROM
    -> Menu System
        -> Temperature sub-menu
        -> Code sub-menu

    -> Math calculations sub-routines
/*******************************/

;==== State 0 - Home Page ====
state_0:
    tst     chg         ; check flag/semaphore
    breq    main        ; if no change, back to main
    clr     chg
    mov     a0, temp0   ; update temperature values
    mov     a1, temp1
    PRINTF  LCD         ; print temperature
.db LF,"Curr Temp=",FFRAC2+FSIGN,a,4,$22,"C ",0
    rjmp    main

;==== State 1 - Enter code ====
state_1:
    rcall   LCD_clear       ; clear LCD
    clr     count
    PRINTF  LCD             ; update LCD display
.db FF,CR, "ENTER: ",0

    _LDI    a0, 0x23        ; reset a values to # for display
    _LDI    a1, 0x23        ; purposes
    _LDI    a2, 0x23
    _LDI    a3, 0x23

display_code:
    tst     wr2             ; check flag/semaphore
    breq    display_code    ; loop back till not 0
    clr     wr2

    VERIFY_ENTER    wr0,wr1,interm  ; check if BCD*# dont count,if A,verify code,otherwise ok
                                    ; interm=0 ok, interm=1 verify code, interm=2 dont count
    cpi     interm, 0x01
    brne    PC+2
    jmp     verify_code

    cpi     interm,0x02
    breq    display_code

    DECODE_ASCII    wr0, wr1, interm                ; decode the input from keypad into the ascii value
    CHECK_AND_SET   a0, a1, a2, a3, interm, count   ; Update value accordingly (change incremental
                                                    ; bit of the code)
    PRINTF LCD                                      ; update LCD
.db LF, "Code in:    ",FSTR, a,0
    rjmp    display_code

;==== State 2 - Alarm and Servo system ====
alarm :
    OUTI    TIMSK,(1<<TOIE2)            ; enable the timer
    rjmp    state_1

stop_alarm :                            ; sub-routine to turn off alarm
    WAIT_MS 1000
    rcall LCD_clear                     ; clear display
    PRINTF LCD                          ; update display
    .db FF,CR,"Sprinkler Sys",0
```

```
    OUTI  TIMSK,(1<<TOIE0)              ; Disable the timer
    _LDI state,0x00                     ; update the state of the system

servo_routine:
    cli                                 ; disable interrupts to ensure execution of sub-routine
    ldi _w, 0xf0                        ; add initial offset to duration of movement of servo
    call calculate_math                 ; call the sub-routine to calculate the coefficient of Arrhenius
    add _w, a0                          ; add result of exponential to offset
    add _w, a1
    rcall LCD_clear                     ; clear LCD
    PRINTF LCD                          ; update LCD
    .db FF,CR,"Servo activated",0

    OUTI DDRA, 0x01                     ; Activate the pump by activating correct pin
    OUTI PORTA, 0x01

    lds interm, 0xDDDD                  ; reset the MCUCR to initial state to ensure
    out MCUCR, interm                   ; correct working of servo motor

loop:
    tst _w                              ; test to see if end of loop
    breq end
    dec _w
    P0 PORTC,SERVO1                     ; send PWM impulse in order to move the servo
    WAIT_US 1900000
    P1 PORTC,SERVO1
    WAIT_US 100000
    rjmp loop

end:
    sei                                 ; reactivate the interrupts
    jmp reset                           ; reset the system to initial state

;==== Code verification ====
verify_code:
    rcall       LCD_clear               ; clear LCD
    PRINTF      LCD                     ; Update LCD
```

```
  .db CR, CR, "verification...",0
      WAIT_MS     1000                  ; Add wait to ensure message can be seen by user

      push    c0                        ; Push registers on SRAM in order to be able to
      push    c1                        ; recover them after
      push    c2
      push    c3

      ldi     xl,low(code)              ; load code from SRAM address into x pointer
      ldi     xh,high(code)
      ld      c0,x+
      ld      c1,x+
      ld      c2,x+
      ld      c3,x

      cp      a0,c0                     ; check if each bit of the code is correct
      breq    PC+2
      rjmp    wrong_code
      cp      a1,c1
      breq    PC+2
      rjmp    wrong_code
      cp      a2,c2
      breq    PC+2
      rjmp    wrong_code
      cp      a3,c3
      breq    PC+2
      rjmp    wrong_code
      nop

      ; restore values
      pop c3
      pop c2
      pop c1
      pop c0

  correct_code:
      nop
```

```
    PRINTF  LCD                     ; update LCD display
.db CR, LF, "Correct Code"
.db  0
    _CPI    state,0x02              ; check if in alarm state
    brne    PC+2
    rjmp    stop_alarm
    rjmp menu                       ; loop back to menu

wrong_code:
    pop     c3                      ; restore initial values of registers
    pop     c2
    pop     c1
    pop     c0

    PRINTF  LCD                     ; update LCD
.db LF, "Wrong code ",0
    WAIT_MS 1000

    _CPI    state,0x02              ; check if in Alarm mode
    brne    PC+2
    rjmp    state_1
    _LDI    state,0x00              ; check if in normal mode
    rcall   LCD_clear
    PRINTF  LCD                     ; update LCD
.db FF,CR,"Sprinkler Sys",0
    rjmp    main

;==== Menu System ====
menu:
    WAIT_MS 1000                    ; add wait offset in order to ensure correct functioning of system
    rcall  LCD_clear                ; clear LCD display
menu1:
    WAIT_MS 100                     ; add wait to ensure correct functioning

    PRINTF  LCD                     ; Update LCD display
.db FF,CR,"A=CHANGE CODE",0
    nop
```

```
    PRINTF  LCD                     ; Update LCD display
.db LF,"B=CHANGE TEMP   ",0

    tst     wr2                     ; check flag/semaphore
    breq    menu1
    clr     wr2

    DECODE_ASCII    wr0, wr1, interm    ; Decode input into ASCII value
    cpi     interm,0x41             ; branching operations based on the result of interm
    brne    PC+2
    rjmp    change_code             ; change code option
    cpi     interm,0x42
    breq    change_temp             ; change temp option
    rjmp    menu1

;==== Temperature sub-menu ====
change_temp:
    ldi     xl,low(temp_seuil)      ; load the threshold temperature into pointer X
    ldi     xh,high(temp_seuil)
    ld      a0,x+                   ; load values into a registers
    ld      a1,x
    rcall   LCD_clear               ; clear LCD display
    PRINTF  LCD                     ; update LCD display
.db FF,CR,"Change temp:",0

change_temp1:                       ; change temp sub-routine part 2, needed for looping
    WAIT_MS 10
    rcall   encoder                 ; poll encoder
    PRINTF LCD                      ; update LCD
.db LF,"Temp=",FFRAC2+FSIGN,a,4,$32,"C ",0

    tst    wr2                      ; check flag/semaphore
    breq   change_temp1             ; loop back
    clr    wr2
    DECODE_ASCII wr0, wr1, interm    ; decoode input into ASCII
    cpi     interm,0x41             ; branching on state of interm
    breq    set new temp            ; save new temp
```

```
    rjmp    change_temp1                ; loop back

set_new_temp :
    ldi     xl,low(temp_seuil)          ; load threshold temp into x pointer
    ldi     xh,high(temp_seuil)
    st      x+,a0
    st      x,a1

    rcall  LCD_clear                    ; clear LCD
    PRINTF LCD                          ; update LCD
.db LF, "NEW TEMP SET",0
    _LDI state,0x00                     ; reset state of system to default

    WAIT_MS 1000                        ; wait to ensure system has time to react
    rcall  LCD_clear                    ; clear LCD display
    PRINTF LCD                          ; update LCD display
.db FF,CR,"Sprinkler Sys",0

    rjmp main                           ; loop back to main

;==== Change code sub-menu ====
change_code :
    rcall   LCD_clear                   ; clear LCD display
    PRINTF  LCD                         ; update LCD display
.db FF,CR,"WRITE NEW CODE:",0

    ldi     xl,low(code)                ; load existing code
    ldi     xh,high(code)
    ld      a0,x+
    ld      a1,x+
    ld      a2,x+
    ld      a3,x

    PRINTF  LCD                         ; update LCD display
.db LF, "NEW CODE:   ",FSTR, a,0
    WAIT_MS 500                         ; wait to ensure system has time to react
```

```
    ldi     count,0x00                  ; reset count to 0

change_code_1:                          ; second change code sub-routine, used for looping
    WAIT_MS 1                           ; wait to ensure time has time to react

    tst    wr2                          ; check flag/semaphore
    breq   change_code_1
    clr    wr2
    VERIFY_ENTER wr0,wr1,interm         ; loads case of key pressed into interm
    cpi     interm,0x01                 ; performs branching to redirect to correct case
    brne    PC+2
    jmp     set_new_code
    cpi     interm,0x02
    brne    PC+2
    rjmp    change_code_1

    DECODE_ASCII    wr0, wr1, interm    ; decode input to ASCII value
    CHECK_AND_SET   a0, a1, a2, a3, interm, count
    PRINTF  LCD                         ; update LCD display
.db LF, "NEW CODE:   ",FSTR, a,0
    rjmp            change_code_1       ; loop back

set_new_code:
    ldi     xl,low(code)                ; load existing code
    ldi     xh,high(code)
    st      x+,a0
    st      x+,a1
    st      x+,a2
    st      x,a3

    push    a0                          ; push registers to SRAM in order to restore their
    push    a1                          ; initial values later
    push    a2
    push    a3

    rcall   LCD_clear                   ; clear LCD display
    PRINTF  LCD                         ; update LCD display
```

```
.db LF, "NEW CODE SET",0
    _LDI    state,0x00                  ; reset state of system

    WAIT_MS 1000                        ; wait to ensure the user can see message
    rcall   LCD_clear                   ; clear LCD display
    PRINTF  LCD                         ; update LCD display
.db FF,CR,"Sprinkler Sys",0

    pop a3                              ; restore registers
    pop a2
    pop a1
    pop a0

    WRITE_EEPROM a0, code1_address      ; write new code to EEPROM
    WRITE_EEPROM a1, code2_address
    WRITE_EEPROM a2, code3_address
    WRITE_EEPROM a3, code4_address

    rjmp main                           ; loop back

;==== Math calculations sub-routines ====
; these libraries were imported here as otherwise they occupy too much SRAM and it can cause
; conflicts with the rest of the program.
.include "math.asm"                     ; include math libraries
.include "taylor_2byte.asm"             ; include Taylor series library
calculate_math:
    call    calculation_speed           ; call function to calculate  the coefficient of Arrhenius
    lds     a0, exp_result              ; load results into register
    lds     a1, exp_result
    ret                                 ; return to where function was cllaed

; === look up table for ASCII values===
KeySet01:
    .db 0x31, 0x32, 0x33, 0x41 ; 1, 2, 3, A
    .db 0x34, 0x35, 0x36, 0x42 ; 4, 5, 6, B
    .db 0x37, 0x38, 0x39, 0x43 ; 7, 8, 9, C
    .db 0x2A, 0x30, 0x23, 0x44 ; *, 0, #, D
```

5.2 Bibliothèque pour calculer coefficient d'Arrhenius – *taylor_2byte.asm*

```
/*
    EPFL - EE-208: Microcontroleurs et Systemes Numeriques
    Semester Project - Spring Semester 2024

    Groupe 014:
    Eliot Abramo - SCIPER 355665
    Mathias Rainaldi - SCIPER 364154

    Custom written math library to calculate the coefficient of Arrhenius
    on a 2-byte floating decimal system.

    k = exp(E/(R*temp)) = exp(x)
*/

start_math:
    ; save current state of system in order to restore it after
    in      _sreg, SREG
    push    _sreg
    push    zh
    push    zl
    push    yh
    push    yl
    push    xh
    push    xl
    push    _w
    push    w
    push    d3
    push    d2
    push    d1
    push    d0
    push    c3
    push    c2
    push    c1
    push    c0
    push    b3
    push    b2
    push    b1
```

```
    push    b0
    push    a3
    push    a2
    push    a1
    push    a0

calculation_speed:
    lds     xl, high(exp_result)            ; load value of exponential from memory into x pointer
    lds     xh, low(exp_result)
    lds     zl, temp_LSB                    ; load threshold temperature from memory into z pointer
    lds     zh, temp_MSB
    ; Calculate k = exp(E/(R*temp)) = exp(x)

    ;x = x/temperature
    ;initial division
    ldi     a0, 0x10
    ldi     a1, 0x27
    mov     b0, zl
    mov     b1, zh
    call    div22
    mov     r17, c0

    ;scale remainder to adapt floating decimal
    mov     a0, d0
    mov     a1, d1
    ldi     b0, 0x64
    call    mul21

    ;second division to calculate decimal part
    mov     a0, c0
    mov     a1, c1
    mov     a2, c2
    mov     b0, zl
    mov     b1, zh
    call    div32
    mov     r16, c0
```

```
    ;update pointer
    mov     zl, r16
    mov     zh, r17
    clr     r16
    clr     r17

;result = exp(x)
exp_cal:
    ; initialize all of the counters and placeholder values needed
    ldi     r17, 0x01
    ldi     r26, 0x02
    ldi     r27, 0x02
    ADD2    r17, r16,   zh,zl

    ; Calculate e^x using Taylor series
    ; 1 + x + x^2/2 + x^3/6
exp_loop:
    ; Calculate x^i
    clr     r11
    inc     r11
    push    r17
    push    r16
    rcall   pow_loop

    ; Calculate x/i! = r28/r27
    pop     r16
    pop     r17

    ;initial division
    mov     a0, r29
    mov     b0, r27
    call    div11
    add     r17, c0

    ;second devision to adapt data in order to find decimal part
    add     r28, d0
    mov     a0, r28
```

```
    mov     b0, r27
    call    div11
    add     r16, c0

    ;Update i and factorial
    inc     r26

    mov     a0, r27
    mov     b0, r26
    call    mul44
    mov     r27, c0

    ; Repeat for 3 terms (3 because max value is ff = 256 and worst case 5^3 < 256 but not 5^4)
    cpi     r26, 4
    brne    exp_loop
    sts     high(exp_result), r17       ; write results to memory to ensure accessible later
    sts     low(exp_result), r16

end_math:
    ; restore system state
    pop     a0
    pop     a1
    pop     a2
    pop     a3
    pop     b0
    pop     b1
    pop     b2
    pop     b3
    pop     c0
    pop     c1
    pop     c2
    pop     c3
    pop     d0
    pop     d1
    pop     d2
    pop     d3
    pop     w

     pop     _w
     pop     xl
     pop     xh
     pop     yl
     pop     yh
     pop     zl
     pop     zh
     pop     _sreg
     out     SREG, _sreg
     ret     ; return to where math functions called in main program

;=======================
; ==== sub-routines ====
;=======================

pow_loop:
    ; calculate x^i
    mov     a0, zl
    mov     b0, zl
    call    mul11

    mov     a0, c0
    mov     a1, c1
    ldi     b0, 0x64
    call    div21
    add     r28, c0

    mov     a0, zh
    mov     b0, zh
    call    mul11
    add     r29, c0

    inc     r11
    cp      r11, r26 ; compare i and r11
    brlt    pow_loop ; if i != interm2, repeat loop
    ret
```

5.3 Bibliothèque pour l'encodeur – *encoder.asm*

Pour l'encodeur, puisque nos fonctions non seulement s'ajoute à la bibliothèque du cours mais la complémente aussi et donc il y a une grande différence entre la nôtre et celle du livre, nous avons fait le choix de la présenter en complet dans l'annexe. C'est pour cette raison nous avons inclus la bibliothèque en entier ci-dessous.

```
/*
    EPFL - EE-208: Microcontroleurs et Systemes Numeriques
    Semester Project - Spring Semester 2024

    Groupe 014:
    Eliot Abramo - SCIPER 355665
    Mathias Rainaldi - SCIPER 364154

    Modified encoder.asm file in order to function on a 2 byte system
    instead of 1 byte.
*/

; === definitions ===
.equ    ENCOD   = PORTD

.dseg
enc_old:.byte   1
.cseg

; === macro ===
.macro add_volker2          ;if button down, macro to increment the
    add @0,a2               ;fourth bit of the first byte while
    brcc end2               ;checking and avoid errors du to
    inc @1                  ;overflow/carry
    ldi a3,0x01
end2 :
    nop
.endmacro

.macro add_volker           ;if button up, macro to incrment on
    inc @0                  ;two bytes while checking and avoid
    cpi @0,0x00             ;errors due to overflow/carry
    brne end1
    inc @1
    ldi a3,0x01
end1 :
    nop

.endmacro

.macro sub_volker2          ;if button down, macro to decrement the
    push @0                 ;fourth bit of the first byte while
    push a2                 ;checking and avoid errors du to
    ldi a2,0xf0             ;overflow/carry
    and @0,a2
    pop a2
    cpi @0,0x00
    pop @0
    breq mala
    subi @0,0x10
    jmp end3
mala:
    subi @1,0x01
    subi @0,0x10
end3 :
    nop
.endmacro


.macro sub_volker           ;if button up, macro to decrement on
    cpi @0,0x00             ;two bytes while checking and avoid
    breq mala               ;errors due to overflow/carry
    subi @0,1
    jmp end
mala:
    subi @1,1
    subi @0,1
end :
    nop
.endmacro
; === routines ===

encoder_init:
    in  w,ENCOD-1       ; make 3 lines input
    andi    w,0b10001111
```

```
    out ENCOD-1,w
    in  w,ENCOD         ; enable 3 internal pull-ups
    ori w,0b01110000
    out ENCOD,w
    ret

encoder:
; a0,b0 if button=up   then increment/decrement a0
; a0,b0 if button=down then incremnt/decrement b0
; T     T=1 button press (transition up-down)
; Z Z=1 button down change
    clr a3
    ldi a2,0x10
    clt                     ; preclear T
    in  _w,ENCOD-2          ; read encoder port (_w=new)

    andi    _w,0b01110000   ; mask encoder lines (A,B,I)
    lds _u,enc_old          ; load prevous value (_u=old)
    cp  _w,_u               ; compare new<>old ?
    brne    PC+3
    clz
    ret                     ; if new=old then return (Z=0)
    sts enc_old,_w          ; store encoder value for next time

    eor _u,_w               ; exclusive or detects transitions
    clz                     ; clear Z flag
    sbrc    _u,ENCOD_I
    rjmp    encoder_button  ; transition on I (button)
    sbrs    _u,ENCOD_A
    ret                     ; return (no transition on I or A)

    sbrs    _w,ENCOD_I      ; is the button up or down ?
    rjmp    i_down
i_up:
    sbrc    _w,ENCOD_A
    rjmp    a_rise
a_fall:
```

```
    add_volker a0,a1                    ; if B=1 then increment
    sbrc    _w,ENCOD_B
    rjmp i_up_done
    subi    a0,1
    sub_volker a0,a1            ; if B=0 then decrement
    cpi a3,0x00
    breq PC+2
    subi a1,1
    rjmp    i_up_done
a_rise:
    add_volker a0,a1                    ; if B=0 then increment
    sbrs    _w,ENCOD_B
    rjmp i_up_done
    subi    a0,1
    sub_volker a0,a1
    cpi a3,0x00
    breq PC+2
    subi a1,1       ; if B=1 then decrement
i_up_done:
    clz                     ; clear Z
    ret

i_down:
    sbrc    _w,ENCOD_A
    rjmp    a_rise2
a_fall2:
    add_volker2 a0,a1                   ; if B=1 then increment
    sbrc    _w,ENCOD_B
    rjmp i_up_done
    subi    a0,0x10
    sub_volker2 a0,a1           ; if B=0 then decrement
    cpi a3,0x00
    breq PC+2
    subi a1,0x01
    rjmp    i_up_done
a_rise2:
    add_volker2 a0,a1                   ; if B=0 then increment
```

```
    sbrs    _w,ENCOD_B
    rjmp i_up_done
    subi    a0,0x10
    sub_volker2 a0,a1
    cpi a3,0x00
    breq PC+2
    subi a1,0x01
i_down_done:
    sez                     ; set Z
    ret

encoder_button:
    sbrc    _w,ENCOD_I
    rjmp    i_rise
i_fall:
    set                     ; set T=1 to indicate button press
    ret
i_rise:
    ret

.macro  CYCLIC  ;reg,lo,hi
    cpi @0,@1-1
    brne    PC+2
    ldi @0,@2
    cpi @0,@2+1
    brne    PC+2
    ldi @0,@1
.endmacro
```