EPFL

Rapport Technique

Projet Microcontrôlers

Groupe 32

- Gestionnaire de salle -

Un système gérant l'accès a une salle limitée en occupation

1 Introduction

Ce projet consiste d'un système informatique implémenter sur microcontrôleur AVR sur une carte STK300, qui a comme but de réaliser un outille permettant de gérer l'entrée dune salle, comme salle d'exposition dans un musée pour limiter le nombre de personnes présente dans la salle en affichant le nombre de places disponible. Ce système mémorise le nombre de visiteurs total pendant une journée. Ce système gère une porte d'accès lorsque la limite est atteinte. Le système possède une sonnette qui permet d'appeler un responsable à partir de l'entrée lorsque la pote est fermée. L'affichage est implémenté en décompteur affin de savoir le nombre restant de places disponible. Ce système est pertinent spécialement pendant une période de pandémie comme le covid ou l'on a besoin de limiter le nombre de visiteurs du au conditions sanitaires, mais il peut être imaginé un autre besoin pour limiter le nombre de visiteur ou l'on peut effectue un monitoring de la situation pendant la journée.

Author :
Elmaleh Daniel Abraham

Submitted to :
Alexander Schmid

1er juin 2022

Table des matières

2 Technique

3 zones de mémoire dont lune est un registre de statuts, un deuxième pour mémoriser le nombre total de visiteurs pendant la journée, et un troisième pour stocker les donnes nécessaires pour l'affichage des LEDs. Le pointeur z sert à parcourir les adresses en mémoire programme et le pointeur x sert a parcourir les adresses dans la mémoire de données SRAM.

Une zone en mémoire programme pour des constante en tant que table de correspondance a été défini. Une zone en mémoire SRAM afin de mémoriser les donnes lors du déroulement du programme qui dépendaient de l'état et l'avancement de l'application. Un registre d'état a été défini affin d'avoir une indication claire a chaque moment du nombre de personnes présentes dans la salle ainsi que l'état de la porte.

Interface utilisateur L'interface est constituée du panneau LED de la sonnerie et du bouton 0 pour sonner à la porte. L'affichage sur le panneau LED est utilisée par un agent de sécurité par exemple afin de savoir combien de places restent disponible dans la salle et le nombre total de visiteurs pendant la journée.

Interruption Une interruption est utilisée dans ce projet, elle provient de la part des capteurs de distance. Lors d'un seuil de distance atteint, une interruption est effectuée pour indiquer le passage d'une personne.

2.1 Fonctions

Pour le panneau LED 3 fonctions principales ont été définis. Update_nb afin de mettre à jour le nombre de personnes dans la salle , store_(couleur) afin de stocker la bonne couleur du nombre à afficher en forme de 64*3 bits pour chaque pixel ces 3 registre de RGB, P_(COULEUR) afin de mémoriser la bonne couleur et store_affichage pour mettre dans la mémoire donne dans le bon ordre les pixels a afficher. Capteur_sortie et capteur_entree sont chargées de gérer la liaison entre le capteur, le seuil de distance et le pointeur x dans la mémoire données affin de pointée vers la bonne adresse pour stocker le nombre à afficher.

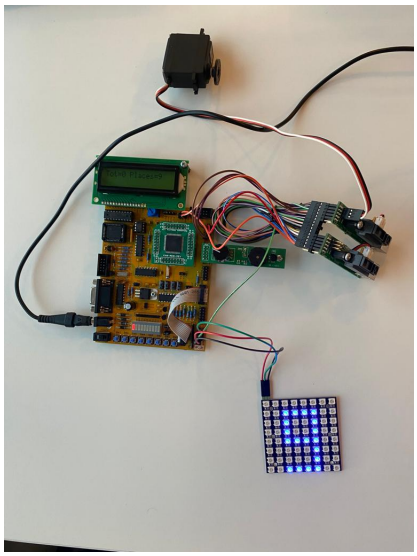

FIGURE 2.1 – Image du projet

3 Périphériques

Capteur de distance Le capteur de distance est implémenté en traduisant la valeur analogique qui mesure à partir de la distance en valeur numérique sur le port F, puis elle produit une interruption lors d'une valeur seuil prédéfinis.

Buzzer Le buzzer sert à sonne à la porte que lorsqu'elle est fermée du a la limite de visiteurs. La sonnerie est une mélodie de « twinkle twinkle little star ». Une sonnerie joue lorsque le bouton dédier est pressé. La mélodie est mémoire en mémoire programme et fait appelle à une table de correspondance pour jouer la bonne note.

Affichage sur panneau LED L'affichage est effectué en deux parties. La première pour stocker la valeur en pixels et en couleur du nombre à afficher tirée de la mémoire SRAM ou est stocker le nombre. La deuxième pour afficher le nombre sur le panneau LED en envoyant des groupes de 3 bytes afin de définir la composante RG et B respectivement. Puis ce cycle est répété 64 fois afin de remplir tout le panneau ave l'information a afficher.

Moteur servo Le moteur est sensé ouvrir et fermer la porte lors de la limite de visiteurs atteinte. Le code fonctionne individuellement, mais un problème d'implémentation a fait en sorte qu'il ne fonctionne pas comme attendu sur le résultat final.

Ports utilisées Les port D,B,E,F sont utilisés pour le panneau LEDs et le moteur, les boutons, le buzzer, et les capteurs de distances avec le moteur servo respectivement.

4 Annexe

```
;
; projet.asm
;
; Created: 22/05/2022 13:44:12
; Author : Daniel Elmaleh
;


; Replace with your application code


    ;=== explanation ===

    ;=== general includes ===
.include    "libraries\macros.asm"
.include    "libraries\definitions.asm"
.include    "libraries\Macros_P.asm"

    ;=========memoire=========
.dseg
.org    0x0400
    ETAT:           .byte   1       ; bit7 = porte, bit0-5 = nb de visiteur actuel
    NB_TOTAL:       .byte   1       ; nb de visiteur total de la journee
    NB_A_AFFICH:    .byte   200

.cseg
    .org 0
        jmp reset
    .org 0x400
        ZERO_P:         .db     0b10000001, 0b01000010, 0b00100100, 0b00011000,
          0b00011000, 0b00100100, 0b01000010, 0b10000001
        UN_P:           .db     0b00000000, 0b00010000, 0b00011000, 0b00010100,
          0b00010000, 0b00010000, 0b00010000, 0b00010000
        DEUX_P:         .db     0b00000000, 0b00111100, 0b00100000, 0b00100000,
          0b00111100, 0b00000100, 0b00000100, 0b00111100
        TROIS_P:        .db     0b00000000, 0b00111100, 0b00100000, 0b00100000,
          0b00111100, 0b00100000, 0b00100000, 0b00111100
        QUATRE_P:       .db     0b00000000, 0b00100100, 0b00100100, 0b00100100,
          0b00111100, 0b00100000, 0b00100000, 0b00100000
        CINQ_P:         .db     0b00000000, 0b00111100, 0b00000100, 0b00000100,
          0b00111100, 0b00100000, 0b00100000, 0b00111100
        SIX_P:          .db     0b00000000, 0b00111100, 0b00000100, 0b00000100,
          0b00111100, 0b00100100, 0b00100100, 0b00111100
        SEPT_P:         .db     0b00000000, 0b00111100, 0b00100000, 0b00100000,
          0b00100000, 0b00100000, 0b00100000, 0b00100000
        HUIT_P:         .db     0b00000000, 0b00111100, 0b00100100, 0b00100100,
          0b00111100, 0b00100100, 0b00100100, 0b00111100
        NEUF_P:         .db     0b00000000, 0b00111100, 0b00100100, 0b00100100,
          0b00111100, 0b00100000, 0b00100000, 0b00111100
```

```
twinkle:
        .db     do2,0,do2,0,so2,0,so2,0,la2,0,la2,0
        .db     so2,so2,0,fa2,0,fa2,0,mi2,0,mi2,0,re2
        .db     0,re2,0,do2,do2,0,so2,0,so2,0,fa2,0
        .db     fa2,0,mi2,0,mi2,0,re2,re2,0,so2,0,so2
        .db     0,fa2,0,fa2,0,mi2,0,mi2,0,re2,re2,0
        .db     do2,0,do2,0,so2,0,so2,0,la2,0,la2,0
        .db     so2,so2,0,fa2,0,fa2,0,mi2,0,mi2,0,re2,0,re2,0,do2,do2,0

    ;=== interrupt table ===
.org    ADCCaddr
    jmp ADCCaddr_sra

    ;=== interrupt service routines ===
ADCCaddr_sra:
    ldi     r23, 0x01
    reti

    ;=== initialization (reset) ===
reset:

reset_led:
    LDSP    RAMEND          ; set up stack pointer (SP)
    rcall   ws2812b4_init   ; initialize LEDs
    STI     ETAT,0x09       ; porte ouverte , limite de visiteurs
    STI     NB_TOTAL,0

reset_moteur:
    OUTI    DDRD,0xff       ; configure portD to output

reset_son:
    OUTI    DDRE,0xff
    sbi     DDRE,SPEAKER
    OUTI    DDRB,0x00       ; configure les buttons en entrees

reset_capteur_de_distance:
    sei
    OUTI    ADCSR, (1<<ADEN)+(1<<ADIE)+1
    rcall   LCD_init
    jmp     main            ; jump ahead to the main program

    ;=== routines includes ===
.include    "libraries\lcd.asm"
.include    "libraries\printf.asm"
.include    "libraries\Routines_P.asm"
.include    "libraries\sound.asm"

    ;=== main program ===
```

```
main:

    ;=========LED=========
main_led:
    LDIX    ETAT
    ld      _w,x            ; liens avec nombre
    andi    _w,0b00011111   ; masking to get nb. of visitors
    clz
    ldi     b0,9
    rcall   update_nb       ; sets z (FLASH) acording to x (SRAM)

    LDIX    NB_A_AFFICH
    rcall   store_affich

    LDIX    NB_A_AFFICH
    _LDI    r0,64

loop:
    ld a0, x+
    ld a1, x+
    ld a2, x+

    cli
    rcall ws2812b4_byte3wr
    sei

    dec r0
    brne loop
    rcall ws2812b4_reset

end_led:

    ;=========mtest de zero place=========

    LDIX        ETAT
    ld          _w,x
    andi        _w,0b00011111
    clz
    tst         _w
    brne        main_moteur
    ori         _w,0b10000000
    st          x,_w

    ;=========moteur=========
main_moteur:
    P0  PORTD,SERVO1            ; pin=4
    ;WAIT_US    200
```

```
    LDIX            ETAT
    ld              _w,x
    andi            _w,0x80     ; mascage pour l'etat de la porte
    UPDATE_PORTE    _w

    P1  PORTD,SERVO1            ; pin=4

loop_moteur:
    SUBI2   a2,a3,0x01
    brne    loop_moteur

end_moteur:

    ;==========son==========
main_son:
    in      w,PINB
    andi    w,0b00000001    ; test de l'etat du button 0 (sonnerie)
    clz
    tst     w
    brne    end_son
    LDIX    ETAT
    ld      _w,x
    andi    _w,0b10000000   ; test de l'etat de la porte
    clz
    tst     _w
    breq    end_son
    LDIZ    twinkle*2
    LDIX    0x800
    ldi     w,90
    st      x,w

play:
    lpm
    adiw    zl,1
    mov     a0,r0
    ldi     b0,100

    rcall   sound

    LDIX    0x800
    ld      w,x
    clz
    dec     w
    breq    end_son
    st      x,w

    rjmp    play

end_son:
```

```
    ;=========capteur distance=========
 main_dist:
    rcall   capteur_entree

    mov     c0,_w
    clr     c1
    PRINTF  LCD
.db         CR,"Tot=",FDEC,b," Places=",FDEC,c,0
    clz

    LDIX    NB_TOTAL
    st      x,b0            ; mise a jour du nb total de visiteurs
    LDIX    ETAT
    ld      w,x
    andi    w,0b10000000    ; ne pas modifier letat de la porte
    or      _w,w
    st      x,_w
    clz

    rcall   capteur_sortie

    mov     c0,_w
    clr     c1
    PRINTF  LCD
.db         CR,"Tot=",FDEC,b," Places=",FDEC,c,0
    clz

    LDIX    NB_TOTAL
    st      x,b0            ; mise a jour du nb total de visiteurs
    LDIX    ETAT
    ld      w,x
    andi    w,0b10000000    ; ne pas modifier letat de la porte
    or      _w,w
    st      x,_w
    clz

jmp main
```

```
/*
 * Macros_P.asm
 *
 *  Created: 28/05/2022 12:44:37
 *   Author: Daniel Elmaleh
 */

; WS2812b4_WR0  ; macro ; arg: void; used: void
; purpose: write an active-high zero-pulse to PD1
.macro  WS2812b4_WR0
    clr u
    sbi PORTD, 1
    out PORTD, u
    nop
    nop
    ;nop    ;deactivated on purpose of respecting timings
    ;nop
.endm

; WS2812b4_WR1  ; macro ; arg: void; used: void
; purpose: write an active-high one-pulse to PD1
.macro  WS2812b4_WR1
    sbi PORTD, 1
    nop
    nop
    cbi PORTD, 1
    ;nop    ;deactivated on purpose of respecting timings
    ;nop
.endm

.macro UPDATE_PORTE ; P_status_reg
    clz
    mov     w,@0
    andi    w,0b10000000
    tst     w
    breq    P_fermer
    LDI2    a2,a3,1600
    rjmp    END_UPDATE_PORTE
    P_fermer:
            LDI2    a2,a3,2500
    END_UPDATE_PORTE:
.endm
```

```
/*
 * Routines_P.asm
 *
 *  Created: 28/05/2022 12:45:00
 *   Author: Daniel Elmaleh
 */

;====================LED======================

; ws2812b4_init     ; arg: void; used: r16 (w)
; purpose: initialize AVR to support ws2812
ws2812b4_init:
    OUTI    DDRD,0x02
ret

; ws2812b4_byte3wr  ; arg: a0,a1,a2 ; used: r16 (w)
; purpose: write contents of a0,a1,a2 (24 bit) into ws2812, 1 LED configuring
;     GBR color coding, LSB first
ws2812b4_byte3wr:

    ldi w,8
ws2b3_starta0:
    sbrc a0,7
    rjmp    ws2b3w1
    WS2812b4_WR0            ; write a zero
    rjmp    ws2b3_nexta0
ws2b3w1:
    WS2812b4_WR1
ws2b3_nexta0:
    lsl a0
    dec w
    brne ws2b3_starta0

    ldi w,8
ws2b3_starta1:
    sbrc a1,7
    rjmp    ws2b3w1a1
    WS2812b4_WR0            ; write a zero
    rjmp    ws2b3_nexta1
ws2b3w1a1:
    WS2812b4_WR1
ws2b3_nexta1:
    lsl a1
    dec w
    brne ws2b3_starta1

    ldi w,8
ws2b3_starta2:
    sbrc a2,7
```

```
    rjmp    ws2b3w1a2
    WS2812b4_WR0            ; write a zero
    rjmp    ws2b3_nexta2
ws2b3w1a2:
    WS2812b4_WR1
ws2b3_nexta2:
    lsl a2
    dec w
    brne ws2b3_starta2
ret

; ws2812b4_reset    ; arg: void; used: r16 (w)
; purpose: reset pulse, configuration becomes effective
ws2812b4_reset:
    cbi PORTD, 1
    WAIT_US 50  ; 50 us are required, NO smaller works
ret

store_affich:
        cpi     _w,4
        brsh    elseif1
    if:
        imgld_loop_R:
        lpm     _w, z+
        rcall   store_red
        dec     b0
        brne imgld_loop_R
        ret

    elseif1:
        cpi     _w,7
        brsh    elseif2
        imgld_loop_G:
        lpm     _w, z+
        rcall   store_green
        dec     b0
        brne imgld_loop_G
        ret

    elseif2:
        cpi     _w,10
        brsh    elseOFF
        imgld_loop_B:
        lpm     _w, z+
        rcall   store_blue
        dec b0
        brne imgld_loop_B
        ret
```

```
    elseOFF:                    ; pas correct
        imgld_loop_F:
        lpm     _w, z+
        rcall   P_OFF
        dec     b0
        brne imgld_loop_F
ret


P_OFF:
    ldi a0, 0x00    ; off
    st  x+,a0
    ldi a0,0x00
    st  x+,a0
    ldi a0, 0x00
    st  x+,a0
ret

P_BLUE:
    ldi a0, 0x00    ; light blue
    st  x+,a0
    ldi a0,0x00
    st  x+,a0
    ldi a0, 0x0f
    st  x+,a0
ret

P_RED:
    ldi a0, 0x00    ; light red
    st  x+,a0
    ldi a0,0x0f
    st  x+,a0
    ldi a0, 0x00
    st  x+,a0
ret

P_GREEN:
    ldi a0, 0x0f    ; light green
    st  x+,a0
    ldi a0,0x00
    st  x+,a0
    ldi a0, 0x00
    st  x+,a0
ret

update_nb:
    sw_case0:
        cpi     _w,0
        brne    sw_case1
```

```
        LDIZ    ZERO_P*2
        jmp next
    sw_case1:
        cpi     _w,1
        brne    sw_case2
        LDIZ    UN_P*2
        jmp next
    sw_case2:
        cpi     _w,2
        brne    sw_case3
        LDIZ    DEUX_P*2
        jmp next
    sw_case3:
        cpi     _w,3
        brne    sw_case4
        LDIZ    TROIS_P*2
        jmp next
    sw_case4:
        cpi     _w,4
        brne    sw_case5
        LDIZ    QUATRE_P*2
        jmp next
    sw_case5:
        cpi     _w,5
        brne    sw_case6
        LDIZ    CINQ_P*2
        jmp next
    sw_case6:
        cpi     _w,6
        brne    sw_case7
        LDIZ    SIX_P*2
        jmp next
    sw_case7:
        cpi     _w,7
        brne    sw_case8
        LDIZ    SEPT_P*2
        jmp next
    sw_case8:
        cpi     _w,8
        brne    sw_case9
        LDIZ    HUIT_P*2
        jmp next
    sw_case9:
        cpi     _w,9
        brne    next
        LDIZ    NEUF_P*2
        jmp next
    next:
ret
```

```
store_blue:
    clz

    sbrs    _w,0
    rcall   P_OFF
    sbrc    _w,0
    rcall   P_BLUE

    sbrs    _w,1
    rcall   P_OFF
    sbrc    _w,1
    rcall   P_BLUE

    sbrs    _w,2
    rcall   P_OFF
    sbrc    _w,2
    rcall   P_BLUE

    sbrs    _w,3
    rcall   P_OFF
    sbrc    _w,3
    rcall   P_BLUE

    sbrs    _w,4
    rcall   P_OFF
    sbrc    _w,4
    rcall   P_BLUE

    sbrs    _w,5
    rcall   P_OFF
    sbrc    _w,5
    rcall   P_BLUE

    sbrs    _w,6
    rcall   P_OFF
    sbrc    _w,6
    rcall   P_BLUE

    sbrs    _w,7
    rcall   P_OFF
    sbrc    _w,7
    rcall   P_BLUE
ret

store_red:
    clz

    sbrs    _w,0
```

```
    rcall   P_OFF
    sbrc    _w,0
    rcall   P_RED

    sbrs    _w,1
    rcall   P_OFF
    sbrc    _w,1
    rcall   P_RED

    sbrs    _w,2
    rcall   P_OFF
    sbrc    _w,2
    rcall   P_RED

    sbrs    _w,3
    rcall   P_OFF
    sbrc    _w,3
    rcall   P_RED

    sbrs    _w,4
    rcall   P_OFF
    sbrc    _w,4
    rcall   P_RED

    sbrs    _w,5
    rcall   P_OFF
    sbrc    _w,5
    rcall   P_RED

    sbrs    _w,6
    rcall   P_OFF
    sbrc    _w,6
    rcall   P_RED

    sbrs    _w,7
    rcall   P_OFF
    sbrc    _w,7
    rcall   P_RED
ret

store_green:
    clz

    sbrs    _w,0
    rcall   P_OFF
    sbrc    _w,0
    rcall   P_GREEN

    sbrs    _w,1
```

```
    rcall   P_OFF
    sbrc    _w,1
    rcall   P_GREEN

    sbrs    _w,2
    rcall   P_OFF
    sbrc    _w,2
    rcall   P_GREEN

    sbrs    _w,3
    rcall   P_OFF
    sbrc    _w,3
    rcall   P_GREEN

    sbrs    _w,4
    rcall   P_OFF
    sbrc    _w,4
    rcall   P_GREEN

    sbrs    _w,5
    rcall   P_OFF
    sbrc    _w,5
    rcall   P_GREEN

    sbrs    _w,6
    rcall   P_OFF
    sbrc    _w,6
    rcall   P_GREEN

    sbrs    _w,7
    rcall   P_OFF
    sbrc    _w,7
    rcall   P_GREEN
ret

;====================moteur=======================

;====================capteur distance=======================

capteur_sortie:
    OUTI    ADMUX, 1        ;lecture sortie
    WAIT_MS 100

    clr     r23
    sbi     ADCSR, ADSC
    WB0     r23, 0

    in      a0, ADCL
    in      a1, ADCH
```

```
    LDIX    NB_TOTAL
    ld      b0,x            ; storage pour affichage du nb total de visiteurs
    LDIX    ETAT
    ld      _w,x
    andi    _w,0b00011111   ; masking to get nb. of visitors
    clz

    cpi     a1, 3
    brne    end_dist_s
    inc     _w
    WAIT_MS 1500

end_dist_s:

ret

 capteur_entree:
    OUTI    ADMUX, 3 ;lecture entree
    WAIT_MS 100

    clr     r23
    sbi     ADCSR, ADSC
    WB0     r23, 0

    in      a0, ADCL
    in      a1, ADCH

    LDIX    NB_TOTAL
    ld      b0,x            ; storage pour affichage du nb total de visiteurs
    LDIX    ETAT
    ld      _w,x
    andi    _w,0b00011111   ; masking to get nb. of visitors
    clz

    cpi     a1, 3
    brne    end_dist_e
    inc     b0
    dec     _w
    WAIT_MS 1500

end_dist_e:

ret
```

```
/*
 * Twinkle_Twinkle.asm
 *
 *  Created: 31/05/2022 12:19:08
 *   Author: Daniel Elmaleh
 */

twinkle:
.db     do2,0,do2,0,so2,0,so2,0,la2,0,la2,0
.db     so2,so2,0,fa2,0,fa2,0,mi2,0,mi2,0,re2
.db     0,re2,0,do2,do2,0,so2,0,so2,0,fa2,0
.db     fa2,0,mi2,0,mi2,0,re2,re2,0,so2,0,so2
.db     0,fa2,0,fa2,0,mi2,0,mi2,0,re2,re2,0
.db     do2,0,do2,0,so2,0,so2,0,la2,0,la2,0
.db     so2,so2,0,fa2,0,fa2,0,mi2,0,mi2,0,re2,0,re2,0,do2,do2,0


main:

        LDIZ    twinkle*2
        LDIX    0x800
        ldi     w,90
        st      x,w

play:

        lpm
        adiw    zl,1
        mov     a0,r0
        ldi     b0,100

        rcall   sound

        LDIX    0x800
        ld      w,x
        clz
        dec     w
        breq    end
        st      x,w

        rjmp    play

end:

        rjmp    end
```