Porte Automatique

Microcontrôleurs 2023

Manon Benarafa, Ismail Essaidi

EPFL
Lausanne
May 29, 2023

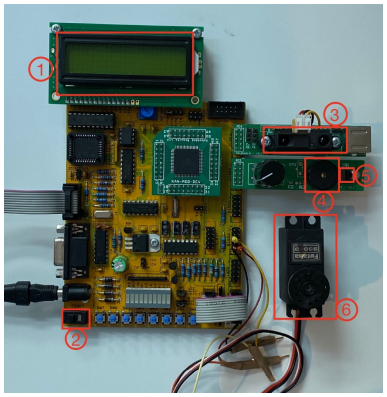
1 Introduction

Introduction Dans le contexte du cours de Microcontroleurs et systèmes numériques nous avons décidé de developper comme projet une porte à ouverture automatique.

Description de l'application La porte est modéelisée par un moteur servo qui effectue une rotation de 180° dans un sens ou dans l'autre pour l'ouverture et la fermeture respectivement. Ce moteur peut être contrôlé soit par une télécommande soit par un détecteur de distance, un LCD est utilisé afin d'afficher les états de la porte.
Deux modes d'utilisation sont possibles: manuel et automatique. Le mode manuel désactive l'ouverture de la porte à travers le capteur de distance et des pressions successives sur un des bouttons permettent tour à tour l'ouverture et la fermeture de la porte. Le mode automatique quant à lui exploite le capteur de distance qui va ouvrir la porte lorsqu'il détecte un objet dans sa proximité. Dès que l'objet ne se trouve plus dans sa portée, la porte se refermera.
De plus, la télécommande peut être utilisée afin de changer de mode ainsi que de réinitialiser l'appareil et ce, peu importe le mode d'utilisation.

2 Mode d'emploi

(a)



(b)

1	LCD: affichage de messages sur l'état du système
2	ON/OFF: allume et éteind la porte
3	Détecteur de distance: détecte si un objet est à moins de 20 cm de la porte
4	Buzzer: émet des sons indiquant le changement de mode et la réinitialisation
5	Récepteur IR: reçoit le signal envoyé par la télécommande
6	Moteur servo: ouvre et ferme la porte

(c)

Figure 1: Mode d'emploi

Initialisation: Un court message d'acceuil est affiché, puis la porte se ferme. Pour rétinitialiser l'appareil, appuyer sur le boutton I.

Mode auto: L'appareil est initialisé en mode automatique. Si un objet est à une distance de moins de 20 cm du capteur de distance la porte s'ouvre. Lorsqu'aucun objet n'est à proximité, la porte se ferme.

Mode manuel: Appuyer sur le bouton III pour activer le mode manuel. La porte s'ouvre après pression sur le bouton II et reste ouverte jusqu'à la prochaine pression du bouton.

3 Description Technique

3.1 Utilisation des ports

Le capteur de distance est connecté au PORT F et le moteur servo est branché au PORT B. Le PORT E est occupé par le capteur infrarouge (PE7) ainsi que le buzzer. Les ports A et C sont laissés libres, et le LCD est branché à son port respectif.

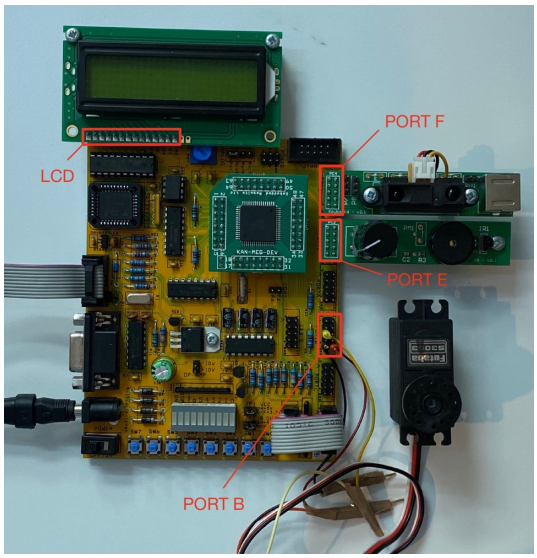


Figure 2: Utilisation des ports

3.2 Interfaces d'affichage

LCD (Hitachi 44780U 2x16) L'affichage se fait entièrement à travers le LCD, qui permet la visualisation du mode d'utilisation, de l'action effectuée sur la porte ainsi qu'un court message lors de l'initialisation.

3.3 Interfaces d'aquisition

Telecommande IR (Vivanco UR Z2) La télécommande infrarouge permet l'aquisition de l'ensemble des signaux de contrôle de l'application. Ceci comprend notamment l'instruction de restart, l'ouverture et la fermeture de la porte ainsi que le changement de mode. Les détails techniques de cet interface sont décrits en *section 6.1* et les détails d'implémentation en *section 5.2.1*.

Detecteur de distance (Sharp GP2Y0A21) Le detecteur de distance a quant à lui un contrôle plus limité, il permet l'ouverture et la fermeture de la porte. Son implémentation est décrite en *section 5.2.2* et son fonctionnement en *section 6.3*

3.4 Interruptions

Nous avons fait usage de l'interruption externe INT7 afin de détecter la pression d'un bouton de la télécommande. Celle-ci est physiquement connectée au pin E7, qui est aussi celui sur lequel est

connecté le récepteur IR et c'est précisément cette dualité qui va permettre le fonctionnement de l'interruption.
Lorsqu'un signal est reçu par le récepteur IR, le pin E7 va passer au niveau bas (voir *section 6.1* pour les détails techniques) et déclencher une interruption. La routine de service d'interruption va ensuite procéder au décodage de la commande et tant que cette action n'est pas terminée aucune autre interruption ne pourra être déclenchée car le bit **I** de **SREG** sera à 0.
Cette méthode rend la routine relativement lente mais ceci ne constitue pas un problème en soi car le programme ne contient aucune autre interruption et que les instructions envoyées par la télécommande sont jugées prioritaires par rapport au détecteur de distance.

4 Présentation des modules

La figure ci-dessous illustre l'interaction entre les différents modules.

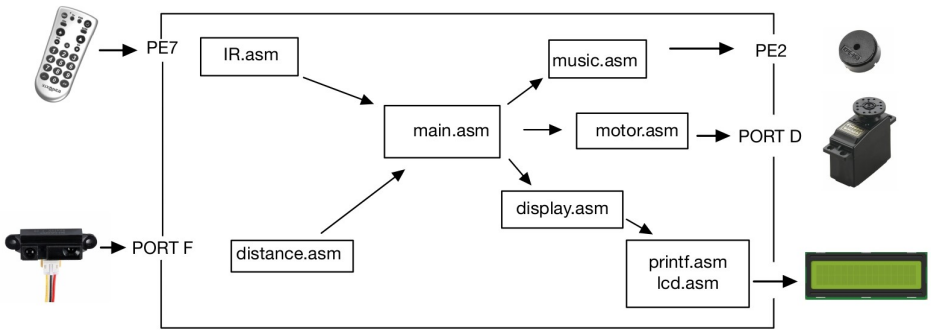


Figure 3: Modules

5 Fonctionnement du programme

5.1 Fonctionnement général

Le fonctionnement général du programme (*Figure 4*) est basé sur une boucle infinie dans le main, qui va tester le mode dans lequel l'appareil se trouve. En mode manuel aucune action n'est effectuée et le programme attend simplement une interruption. Celle-ci pourra à son tour décoder l'instruction de la télécommande et potentiellement agir sur le programme selon la touche pressée. En mode automatique l'interruption est toujours active et elle peut encore agir sur le programme avec un reset ou un changement de mode, mais elle n'a plus d'effet sur le moteur. Tant qu'il n'y a pas d'interruption, le detecteur de distance cherche constamment si un objet se trouve à proximité. Pour le contrôle de l'ensemble du programme un registre est utilisé: **motor_reg**. Celui-ci stocke les informations principales de l'état actuel de l'appareil et est décrit en détail dans la *Table 1*.

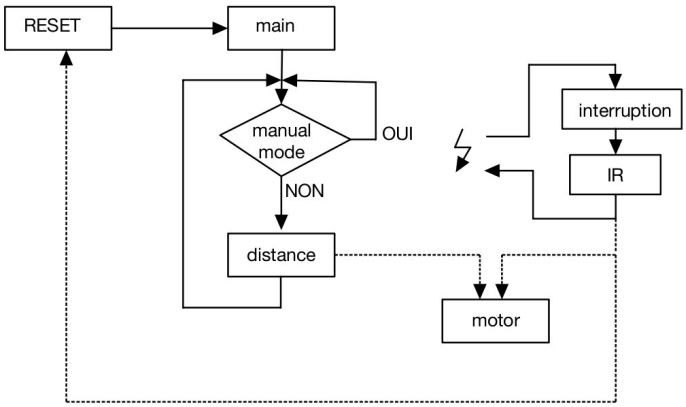
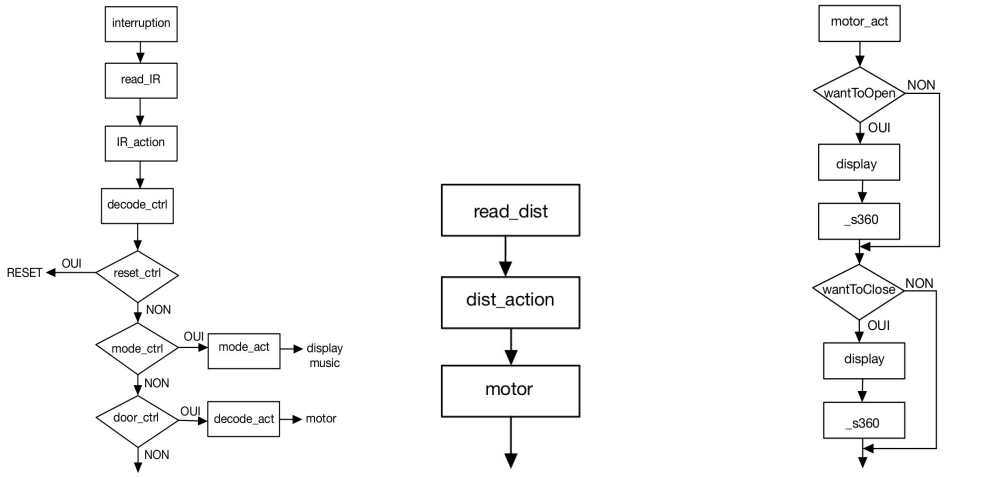


Figure 4: Fonctionnement général du programme, les traitillés sont des actions soumises à condition

5.2 Fonctionnement en détail



(a) Programme IR (b) Programme distance (c) Programme moteur

Figure 5: Fonctionnement du programme en détail

5.2.1 Module IR

Le module IR est constitué de deux sous-routines: **read_IR** et **IR_action**.
La sous-routine **read_IR** échantillone le signal reçu par la télécommande à des intervalles réguliers, elle est ponctuée par une boucle d'attente.
La sous-routine **IR_action** a pour rôle le décodage de la commande lue par **read_IR** suivi par l'appel éventuel du module approprié. On teste la valeur de la commande avec les trois constantes

correspondant aux reset, changement de mode et action sur la porte.
Dans le cas d'un reset, on effectue directement le branchement vers celui-ci. Pour un changement de mode on appelle la sous-routine **mode_act** qui va modifier le registre **motor_reg** (voir *Table 1*) puis appeler les modules display et music pour communiquer le changement à l'utilisateur. Pour une action sur la porte la sous-routine **decode_act** sera appelée, tout comme **mode_act** elle va modifier le registre **motor_reg**, puis elle appelera le module motor, qui effectuera la rotation.

5.2.2 Module distance

Le module distance est composé de deux sous-routines: **read_dist** et **dist_action**:
Le rôle de **read_dist** est de gérer l'interruption liée à la conversion analogique-numérique (détails techniques en *section 6.3*) et de stocker la valeur de la distance. Cette valeur, étant sur 10 bits, est stockée dans les deux registre b0 et b1, respectivement low et high byte.
Puis on appelle la sous-routine **dist_action** qui compare la valeur stockée dans b0 et b1 au *threshold* (défini dans le fichier *my_definitions.asm*). Cette comparaison est effectuée grâce à la macro **CPI2** qui compare un registre à 2 byte (b1:b0) à une constante. Puis le carry (résultat de **CPI2**) est affecté au bit0 du registre motor_reg. Enfin, on appelle le module moteur afin d'agir en fonction de **motor_reg**.

5.2.3 Module moteur

Le module moteur est composé essentiellement des deux sous-routines principales **motor_act** et **_s360**:
La sous-routine **motor_act** a pour rôle le décodage du registre **motor_reg** qui peut prendre les valeurs indiquées dans la *Table 1*.
On commence par extraire les deux premiers bits de **motor_reg** qu'on stocke dans le registre intermédiaire c3. Puis, par exemple, si les conditions pour *open* sont remplies: c'est à dire que distance<threshold (bit0=0) et l'état de la porte est closed (bit1=0) on effectue les actions suivantes:
On appelle une sous-routine qui affiche l'état actuel de la porte sur LCD. Puis on utilise la macro **CA3** qui appelle la sous-routine **_s360**. Ensuite, on affecte à c3 la valeur *is_open*(0x02) qui, en fait, met le bit1 de c3 à 1. Enfin avant de quitter le module on affecte les 2 premiers bits de c3 (registre temporaire) aux 2 premiers bits de **motor_reg**. Un protocole complémentaire est effectué dans le cas *close* si les conditions nécessaires sont réunies.
La sous-routine **_s360** utilise le 2-byte register a1:a0 qui stocke la durée d'impulsion puis met le pin SERVO1 du PORTB à 1 pendant cette durée.
Le registre b0 indique le nombres d'impulsions de durée a1:a0. Ainsi, son effet est de faire tourner le moteur pendant un certain temps en générant un certain nombre d'impulsions. Nous avons défini la constante *nbr_impulse* (0x4B) empiriquement de façon à ce que le moteur tourne à 180°.

Table 1: Encodage du regsitre de contrôle *motor_reg*

motor_reg: 0bXXXXX $b_2b_1b_0$	
b_0	0: dist < threshold 1: dist > threshold
b_1	0: door closed 1: door open
b_2	0: auto 1: manual

6 Description de l'accès aux périphériques

6.1 Telecommande IR Vivanco UR Z2

La télécommande travaille de pair avec un module récepteur infrarouge connecté physiquement à un des ports du microcontrôleur. Celui-ci a pour charge de convertir le signal en un signal digital en éliminant la fréquence porteuse, ainsi que de changer la polarité du signal avec le logic-1 comme repos. Le format de données émis est le RC5 et il est décit ci-après:
Le format RC5 est composé de 14 bits biphasés qui suivent le schéma suivant: deux bits de start permettent la synchonisation, suivis de un bit de toggle qui change à chaque pression, 5 bits d'adresse pour indiquer l'appareil à controler et finalement 6 bits de commande permettent l'encodage des différents boutons.

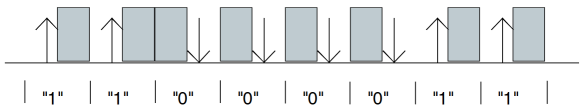


Figure 6: Encodage biphasé (flanc montant pour un logic-1, descendant pour un logic-0)

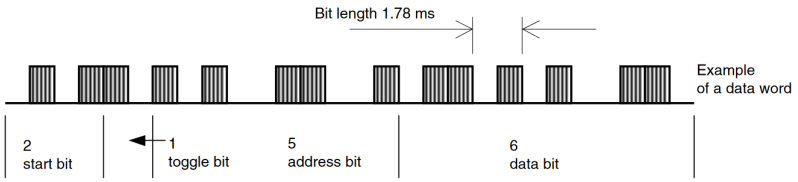


Figure 7: Format RC5

Le décodage des instructions de la télécommande est fait dans le module **IR**, elle est déclenchée après une interruption sur le pin E7 et se base sur une boucle d'attente qui échantillonne tour à tour les 14 bits. L'échantillonage se fait au millieu de la première demi-période de chaque bit et nécessite donc une complémentation (voir *Figure 6*).

6.2 Moteur Servo Futaba S3003

Le moteur servo utilise trois lignes physiques: 2 lignes d'alimentation (GND et VCC) et une ligne de contrôle (SERVO1). La ligne de contrôle (SERVO1) est le pin B4 du PORT B. Le moteur servo est contrôlé par durée d'impulsion. C'est pourquoi nous avons défini deux durées d'impulsion *open_impulse* (500ms) et *close_impulse* (2400ms) qui correspondent respectivement aux angles 180° et 0°. Ces impulsion sont réalisées en mettant le pin SERVO1 à 1 pendant la durée de l'impulsions puis en le remettant à 0. Cette opération doit être effectuée un nombre de fois (*nbr_impulse*) qu'on a défini afin d'effectuer la rotation de 180° souhaitée.

6.3 Détecteur de distance Sharp GP2Y0A21

Le détecteur de distance employé est relié au **PORT F** qui est configuré en entrée. Le PORT F est ici utilisé comme entrée pour le convertisseur. Dans l'initialisation (reset) nous avons configuré le registre **ADMUX** en mettant les bits **MUX0** et **MUX1** à 1 pour permettre l'acquisition sur la ligne du détecteur de distance. Nous avons aussi régler la fréquence de convertion à 4Mhz/64= 62.5kHz en réglant les prescaler de registre **ADCSR**. Si nous sommes en mode automatique, nous appelons dans le main la sous-routine **read_dist** qui déclenche une interruption lorsqu'on met le bit **ADSC** du registre **ADCSR** à 1. Nous usons d'une sémaphore pour savoir que la conversion est achevée et que la valeur de la distance est affectée à ADCL et ADCH (Les deux registres de data du convertisseur) en mettant r23 à la valeur 0x01. Puis nous procédons au traitement de l'information comme expliqué en *section 5.2*.

6.4 LCD Hitachi44780U 2x16

La communication entre le microcontrôleur et le contrôleur LCD se fait à travers le bus d'adresse, de données et de contrôle de la carte STK300. Les librairies *printf.asm* et *lcd.asm* sont utilisées pour réaliser 3 affichages défilants et 2 affichages simples. Les affichages se trouvent dans le module *display.asm* qui a été crée pour assurer une interface utilisateur agréable et informative.

6.5 Buzzer piézoélectrique

Nous utilisons le buzzer afin de multiplier les moyens de communication avec l'utilisateur. En effet, un son est émis à chaque fois qu'on change de mode et une petite mélodie est jouée à l'initialisation. Nous utilisons le pointeur z pour se déplacer dans une zone mémoire allouée pour stocker les différentes fréquences des notes jouées. Tout cela est assuré par le module *music.asm*.

7 Annexes

- Code source

Main_prog.asm

Tuesday, 30 May 2023 00:06

```
...semblerApplication11\AssemblerApplication11\main_prog.asm                    1
/*
 * main_prog.asm
 *
 *  Created: 28/05/2023 13:50:24
 *   Author: ismai
 */
;=================MACROS & DEFINITIONS=================
 .include "macros.asm"
 .include "definitions.asm"
 .include "my_definitions.asm"

;================INTERRUPT VECTOR TABLE==================
.cseg
 .org 0
    jmp reset

.org INT7addr
    jmp ext_int7

.org ADCCaddr
    jmp ADCCaddr_sra

;=================INTERRUPT ROUTINES=========================
.org 0x45
ext_int7:
    in          _sreg, SREG
    CLR2        b1,b0       ;clear 2-byte register
    ldi         b2,14       ;load bit counter
    WAIT_US     (T1/4)      ;wait a quarter period
    rcall       read_IR
    rcall       IR_action
    out         SREG, _sreg
    reti

ADCCaddr_sra:
    in          _sreg, SREG
    ldi         r23,0x01            ;sémaphore
    out         SREG, _sreg
    reti

;=======================INITIALISATION====================

reset:
    LDSP        RAMEND
    OUTI        DDRB, 0xFF
    OUTI        DDRE, 0x7F

    ;init CAD
    OUTI        ADCSR, (1<<ADEN)+(1<<ADIE)+6
```

```
    OUTI        ADMUX, (1<<MUX1)+(1<<MUX0)

    ;init moteur
    P0          PORTB,SERVO1
    LDI2        a1,a0,close_impulse
    ldi         motor_reg, wantToClose              ;init moteur to wantToClose

    ;welcome/reset
    rcall       LCD_init                ;init LCD
    rcall       welcome_music
    rcall       welcome_display
    rcall       motor_act               ;close motor

    ;interrupt setup for IR
    OUTI        EIMSK,(1<<INT7)         ;enable int7
    OUTI        EICRB, 0x00             ;INT7 level 0
    sei

    rjmp        main
;=====================================================================
.include "lcd.asm"
.include "printf.asm"
.include "sound.asm"
.include "display.asm"
.include "motor_act.asm"
.include "music.asm"
.include "IR.asm"
.include "distance.asm"
;=========================MAIN CODE==============================
main:
    cpi         motor_reg, mode
    brsh        PC+2                    ;same or higher means manual mode
    rcall       read_dist
    rjmp        main
```

Display.asm

Tuesday, 30 May 2023 00:06

```
...AssemblerApplication11\AssemblerApplication11\display.asm                    1
/*
 * display_subroutines.asm
 *
 *  Created: 29/05/2023 10:44:43
 *   Author: manon
 */
.org    0x400

welcome_display:
    push        a1
    rcall       LCD_clear
    PRINTF      LCD
    .db         "WELCOME",0
    ldi         a1,0                    ;a1: counter
    loop1:
        WAIT_MS     175
        rcall       LCD_display_right   ;right-moving display
        subi        a1,-1
        cpi         a1,17               ;end of screen
        brne        loop1
    rcall       LCD_clear
    pop         a1
    ret

;============================================

manual_display:
    push        a1
    rcall       LCD_clear
    PRINTF      LCD
    .db         "MANUAL MODE",0
    ldi         a1,0                    ;a1: counter
    loop2:
        WAIT_MS     150
        rcall       LCD_display_right   ;right-moving display
        subi        a1,-1
        cpi         a1,17               ;end of screen
        brne        loop2
    rcall       LCD_clear
    pop         a1
    ret

;============================================

automatic_display:
    push        a1
    rcall       LCD_clear
    PRINTF      LCD
    .db         "AUTO MODE",0
```

```
    ldi         a1,0                    ;a1: counter
    loop3:
        WAIT_MS     150
        rcall       LCD_display_right   ;right-moving display
        subi        a1,-1
        cpi         a1,17               ;end of screen
        brne        loop3
    rcall       LCD_clear
    pop         a1
    ret

;=============================================

open_display:
    rcall       LCD_clear
    PRINTF      LCD
    .db         "OPEN ",0
    ret

;=============================================

close_display:
    rcall       LCD_clear
    PRINTF      LCD
    .db         "CLOSE",0
    ret
```

Distance.asm

Tuesday, 30 May 2023 00:07

```
...ssemblerApplication11\AssemblerApplication11\distance.asm                1
/*
 * distance.asm
 *
 *  Created: 29/05/2023 14:18:28
 *   Author: ismai
 */

 .macro CPI2    ; compare 2byte register with constant
        clc
        cpi     @1, low(@2)
        ldi     w, high(@2)
        cpc     @0, w
        .endmacro

 read_dist:
    clr         r23
    sbi         ADCSR, ADSC             ;start converting
    WB0         r23,0                   ;wait for sémaphore (r23=1)
    in          b0,ADCL                 ;read lower byte
    in          b1,ADCH                 ;read higher byte
    rcall       dist_action
    ret

dist_action:
    CPI2        b1,b0, threshold    ;carry=1 means far from door
    C2B         motor_reg, 0        ;carry to bit 0 of motor_reg
    rcall       motor_act
    ret
```

IR.asm

Tuesday, 30 May 2023 00:07

```
...\7.0\AssemblerApplication11\AssemblerApplication11\IR.asm                    1
/*
 * IR.asm
 *
 *  Created: 29/05/2023 14:11:13
 *   Author: ismai
 */
 read_IR:
    P2C         PINE, IR    ;move Pin to Carry
    ROL2        b1, b0      ;roll carry into 2-byte reg
    WAIT_US     (T1-4)      ;wait bit period - compensation
    DJNZ        b2, read_IR ;Decrement and jump if not zero
    ret


 IR_action:

    com         b0                      ;car on échantillonne à T1/4

    decode_crtl:
        cpi         b0, reset_ctrl
        _BREQ       reset
        cpi         b0, mode_crtl
        breq        mode_act
        cpi         b0, door_ctrl
        breq        decode_act
        rjmp        exitIR

    mode_act:
        _EORI       motor_reg, mode     ;toggle mode (manual/auto)
        rcall       mode_sound          ;sound of changing modes

        display_decode:
            cpi         motor_reg, mode
            brsh        PC+3                    ;same or higher means manual mode
            rcall       automatic_display
            rjmp        PC+2
            rcall       manual_display
        ret

    decode_act:
        cpi         motor_reg, is_open + mode           ;is_open + 0x04
        brne        PC+2
        ldi         motor_reg, wantToClose + mode       ;wantToClose + 0x04

        cpi         motor_reg, is_closed + mode         ;is_closed + 0x04
        brne        PC+2
        ldi         motor_reg, wantToOpen + mode        ;wantToOpen + 0x04

    rcall       motor_act
```

```
    ret

    exitIR:
        ret
```

Motor_act.asm

Tuesday, 30 May 2023 00:07

```
...semblerApplication11\AssemblerApplication11\motor_act.asm                1
/*
 * motor_act.asm
 *
 *  Created: 28/05/2023 13:50:46
 *   Author: ismai
 */

.org 0x250

.macro CA3          ;call a subroutine with three arguments in a1:a0 b0
    ldi         a0, low(@1)     ;low byte impulse duration
    ldi         a1, high(@1)    ;high byte impulse duration
    ldi         b0, @2          ;impulse number
    rcall       @0
.endmacro
;=====================================================================
motor_act:

    mov         c3, motor_reg
    _ANDI       c3, 0b00000011          ;select 2 first bits

    open:
        _CPI        c3, wantToOpen
        _BRNE       close
        rcall       open_display
        CA3         _s360, open_impulse, nbr_impulse    ; opening (CW 180, high-
          speed)

        _LDI        c3, is_open     ;we just opened
        rjmp        exit

    close:
        _CPI        c3, wantToClose
        _BRNE       exit
        rcall       close_display
        CA3         _s360, close_impulse, nbr_impulse   ;closing (CCW 180, high-
          speed)

        _LDI        c3, is_closed       ;we just closed

    exit:
        MOVMSK      motor_reg,c3,0b00000011     ;mov 2 first bits of c3 to
          motor_reg
        rcall       LCD_clear
        ret

; _s360, in a1:a0, a2 out void, mod a2,w=========================
_s360:
```

```
    ls3601:
        rcall       servoreg_pulse
        dec         b0
        brne        ls3601
        ret

; servoreg_pulse, in a1,a0, out servo port, mod a3,a2
; purpose generates pulse of length a1,a0
    servoreg_pulse:
        WAIT_US     20000
        MOV2        a3,a2, a1,a0
        P1          PORTB,SERVO1        ; pin=1

    lpssp01:
        SUBI2       a3,a2,0x1
        brne        lpssp01
        P0          PORTB,SERVO1        ; pin=0
        ret
```

Music.asm

Tuesday, 30 May 2023 00:08

```
...0\AssemblerApplication11\AssemblerApplication11\music.asm                    1
/*
 * music.asm
 *
 *  Created: 29/05/2023 14:11:13
 *   Author: ismai
 */



welcome_music:
    clr         c1
    ldi         zl, low(2*keys-1)   ; load table base into z
    ldi         zh, high(2*keys-1)
    _LDI        c2,0x03
    rjmp        music
mode_sound:
    clr         c1
    ldi         zl, low(2*keys+2)   ; load table base into z
    ldi         zh, high(2*keys+2)
    _LDI        c2,0x02
    rjmp        music
music:
    inc         zl      ; add offset to table base
    lpm         ; load program memory, r0 <- (z)
    mov         a0,r0       ; load oscillation period
    ldi         b0,40       ; load duration (40*2.5ms = 100ms)
    rcall       sound
    inc         c1
    cp          c1,c2
    brlo        music

    ret

keys:
    .db         do2,fa2,la2,re3,la3,0
```

My_definition.asm

Tuesday, 30 May 2023 00:08

```
...erApplication11\AssemblerApplication11\my_definitions.asm                    1
/*
 * my_definitions.asm
 *
 *  Created: 21/05/2023 19:10:48
 *   Author: ismai
 */

 .equ   door_ctrl = 0xfd            ;open/close door command
 .equ   mode_crtl = 0xfa            ;switch mode command
 .equ   reset_ctrl = 0xf3           ;reset command
 .equ   threshold = 300             ;distance treshold
 .equ   T1 = 1778                   ;bit period RC5
 .equ   close_impulse = 2400        ;angle = 0°
 .equ   open_impulse = 500          ;angle = 180°
 .equ   nbr_impulse = 0x4B          ;number of impulses in motor_act
 .def   motor_reg = r29
 .equ   mode = 0x04                 ;manual node in motor_reg
 .equ   wantToOpen = 0x00           ;open code for motor_reg
 .equ   is_open = 0x02              ;is_open code for motor_reg
 .equ   is_closed = 0x01            ;is_closed code for motor_reg
 .equ   wantToClose = 0x03          ;close code for motor_reg
```

Sound.asm

Tuesday, 30 May 2023 00:09

```
...0\AssemblerApplication11\AssemblerApplication11\sound.asm                    1
; file: sound.asm   target ATmega128L-4MHz-STK300
; purpose library, sound generation

sound:
; in    a0  period of oscillation (in 10us)
;   b0  duration of sound (in 2.5ms)

    mov b1,b0       ; duration high byte = b
    clr b0      ; duration  low byte = 0
    clr a1      ; period high byte = a
    tst a0
    breq    sound_off   ; if a0=0 then no sound
sound1:
    mov w,a0
    rcall   wait9us     ; 9us
    nop         ; 0.25us
    dec w       ; 0.25us
    brne    PC-3        ; 0.50us    total = 10us
    INVP    PORTE,SPEAKER   ; invert piezo output
    sub b0,a0       ; decrement duration low  byte
    sbc b1,a1       ; decrement duration high byte
    brcc    sound1      ; continue if duration>0
    ret

sound_off:
    ldi a0,1
    rcall   wait9us
    sub b0,a0       ; decrement duration low  byte
    sbc b1,a1       ; decrement duration high byte
    brcc    PC-3        ; continue if duration>0
    ret

; === wait routines ===

wait9us:rjmp    PC+1        ; waiting 2 cycles
    rjmp    PC+1        ; waiting 2 cylces
wait8us:rcall   wait4us     ; recursive call with "falling through"
wait4us:rcall   wait2us
wait2us:nop
    ret     ; rcall(4), nop(1), ret(3) = 8cycl. (=2us)

; === calculation of the musical scale ===

; period (10us) = 100'000/freq(Hz)
.equ    do  = 100000/517    ; (517 Hz)
.equ    dom = do*944/1000   ; do major
.equ    re  = do*891/1000
.equ    rem = do*841/1000   ; re major
.equ    mi  = do*794/1000
```

```
.equ    fa  = do*749/1000
.equ    fam = do*707/1000   ; fa major
.equ    so  = do*667/1000
.equ    som = do*630/1000   ; so major
.equ    la  = do*595/1000
.equ    lam = do*561/1000   ; la major
.equ    si  = do*530/1000

.equ    do2 = do/2
.equ    dom2    = dom/2
.equ    re2 = re/2
.equ    rem2    = rem/2
.equ    mi2 = mi/2
.equ    fa2 = fa/2
.equ    fam2    = fam/2
.equ    so2 = so/2
.equ    som2    = som/2
.equ    la2 = la/2
.equ    lam2    = lam/2
.equ    si2 = si/2

.equ    do3 = do/4
.equ    dom3    = dom/4
.equ    re3 = re/4
.equ    rem3    = rem/4
.equ    mi3 = mi/4
.equ    fa3 = fa/4
.equ    fam3    = fam/4
.equ    so3 = so/4
.equ    som3    = som/4
.equ    la3 = la/4
.equ    lam3    = lam/4
.equ    si3 = si/4
```