```c
#ifndef AVRX_H
#define AVRX_H

// AVR linux defines
//
#define __SFR_OFFSET 0

// AVR processor-specific file
// containing the I/O port
// definitions  for the device
#include <avr/io.h>
#include <avr/interrupt.h>

// Directives
#define CSEG .text
#define DSEG .data
#define DB .byte
#define BYTE .space
#define ORG .org

// Operators
#define LOW(x) lo8(x)
#define HIGH(x) hi8(x)

#endif // AVRX_H
```

```

/********************************/
/*     Delay util functions     */
/********************************/

;---------------------------------------;
; Description:                          ;
;   - makes a ’delay’ of 1 mili second  ;
;                                       ;
; void mili_delay_1(void);              ;
;---------------------------------------;

.global mili_delay_1
mili_delay_1:

  push r16
  push r17
  push r18

        ldi r16, 2
startLoop3:
        ldi r17, 51
startLoop2:
        ldi r18, 51
startLoop1:
        dec r18
        brne startLoop1
outWhile1:
        dec r17
        brne startLoop2
outWhile2:
        dec r16
        brne startLoop3
outWhile3:

  pop r18
  pop r17
  pop r16

  ret
```

```
#ifndef GYRO_H
#define GYRO_H

// Acceleration Address register
// to start reading
#define ACCEL_ADDR 0x3B

// Gyroscope Address register
// to start reading
#define GYRO_ADDR 0x43

/////////////////////////////////////
//      Acceleration Registers     //
/////////////////////////////////////
//
#define ACCEL_X_H r18
#define ACCEL_X_L r19
#define ACCEL_Y_H r20
#define ACCEL_Y_L r21
#define ACCEL_Z_H r22
#define ACCEL_Z_L r23

//////////////////////////////////
//      Gyroscope Registers     //
//////////////////////////////////
//
#define GYRO_X_H r18
#define GYRO_X_L r19
#define GYRO_Y_H r20
#define GYRO_Y_L r21
#define GYRO_Z_H r22
#define GYRO_Z_L r23

#endif // GYRO_H
```

```
#include "avrx.h"
#include "i2c.h"
#include "gyro.h"

/****************************************/
/*   MPU-6050 Gyroscope/Accelerometer   */
/****************************************/

DSEG

.global ACCEL_X_H_VAL
ACCEL_X_H_VAL: BYTE 1
.global ACCEL_X_L_VAL
ACCEL_X_L_VAL: BYTE 1

CSEG

;-----------------------------------;
; Description:                      ;
;   - gets acceleration and stores  ;
;     them into global variables    ;
;                                   ;
; void get_acceleration(void);      ;
;-----------------------------------;

.global get_acceleration
get_acceleration:

  call i2c_start

  ldi r16, ACCEL_ADDR
  call i2c_connect

  call i2c_init_read

  ldi r17, MORE_BYTES
  call i2c_read
  sts ACCEL_X_H_VAL, r16

  ldi r17, STOP
  call i2c_read
  sts ACCEL_X_L_VAL, r16

  call i2c_end

  ret
```

```c
#ifndef I2C_H
#define I2C_H

// Enable sensor register
#define PWR_MGMT_1_RA 0x6B
// Who Am I? register
#define WHO_AM_I_RA 0x75

// More bytes
#define MORE_BYTES 0x1
// Stop: no more bytes needed
#define STOP 0x0

// Default MPU 6050 address
#define MPU_6050_DEF_ADDR 0x68

// Mask STATUS CODE
#define STAT_CODE_MASK 0xF8

///////////////////////////////////
//      General status code      //
///////////////////////////////////
//
// Start code
#define START 0x08
// Repeated Start
#define REP_START 0x10

/////////////////////////////////////////
//    Master Transmitter status code   //
/////////////////////////////////////////
//
// SLA+W transmitted and ACK received
#define SLA_W_ACK 0x18
// SLA+W transmitted and NACK received
#define SLA_W_NACK 0x20
// Data byte transmitted and ACK received
#define DATA_SEND_ACK 0x28
// Data byte transmitted and NACK received
#define DATA_SEND_NACK 0x30

/////////////////////////////////////////
//     Master Receiver status code     //
/////////////////////////////////////////
//
// SLA+R transmitted and ACK received
#define SLA_R_ACK 0x40
// SLA+R transmitted and NACK received
#define SLA_R_NACK 0x48
// Data byte received and ACK returned
#define DATA_RECV_ACK 0x50
// Data byte returned and NACK returned
#define DATA_RECV_NACK 0x58

////////////////////////
//    Slave Address   //
////////////////////////
//
// Slave Address in LOW mode (AD0=0)
#define SLA MPU_6050_DEF_ADDR
// Slave Address + Master mode: write
// SLA_W
#define SLA_W (SLA<<1 + 0)
// Slave Address + Master mode: Read
// SLA_R
#define SLA_R (SLA<<1 + 1)

/////////////////////
//  Register bits  //
/////////////////////
```

```
//
// TWCR bits
#define TWINT 7
#define TWEA 6
#define TWSTA 5
#define TWSTO 4
#define TWWC 3
#define TWEN 2
#define TWIE 0

// TWSR bits
#define TWPS0 0
#define TWPS1 1

#endif // I2C_H
```

```
#include "avrx.h"
#include "i2c.h"

/*****************************/
/*      TWI-I2C Protocol      */
/*****************************/

DSEG
WHO_AM_I_ADDR: BYTE 1

CSEG

;-----------------------------------------------;
; Description:                                  ;
;   - Starts the connection with                ;
;     the MPU-6050 device and ask it            ;
;     to receive the WHO_AM_I value,            ;
;     and checks if the value is correct.       ;
;     It also, enables the sensors, because     ;
;     by default, they come in sleep-mode.      ;
;                                               ;
; void i2c_init(void);                          ;
;-----------------------------------------------;

.global i2c_init
i2c_init:

  ; it uses the TWI:
  ; Two Wire Interface

  ; pre-scaler
  ldi r16, (0<<TWPS1)|(0<<TWPS0)
  sts TWSR, r16

  ; sets bit-rate
  ldi r16, 0x48
  sts TWBR, r16

  ; enables the TWI
  ; interface
  ldi r16, (1<<TWEN)|(0<<TWIE)
  sts TWCR, r16

  call i2c_start

  ; reads from WHO_AM_I
  ; register
  ldi r16, WHO_AM_I_RA
  call i2c_connect

  ; check WHO_AM_I
  ; received value
  call check_connection

  ; enables the sensor
  ; by writing the SLEEP-MODE
  ; bit (put a zero)
  ; into the PWR_MGMT_1 register
  call enable_sensor

  ret

;-----------------------------------------------;
; Description:                                  ;
;   - Starts the connection with                ;
;     the MPU-6050 device in WRITE mode         ;
;                                               ;
; void i2c_start(void);                         ;
;-----------------------------------------------;

```

```
.global i2c_start
i2c_start:

  ; sends the START request
  ; to receive data from slave
  ldi r16, (1<<TWINT)|(1<<TWSTA)|(1<<TWEN)
  sts TWCR, r16

  ; waits TWINT flag set
  call wait_int_i2c

  ; check TWI status register
  ; to verify START status
  ; goto error otherwise
  lds r16, TWSR
  andi r16, STAT_CODE_MASK
  cpi r16, START
  call check_error

  ; loads SLA_W into TWDR
  ; and clears TWINT in TWCR
  ; register to start transmission
  ; of address
  ldi r16, SLA_W
  sts TWDR, r16
  ldi r16, (1<<TWINT)|(1<<TWEN)
  sts TWCR, r16

  ; waits TWINT flag set
  call wait_int_i2c

  ; check TWI status register
  ; to verify SLA_W ACK received,
  ; goto error otherwise
  lds r16, TWSR
  andi r16, STAT_CODE_MASK
  cpi r16, SLA_W_ACK
  call check_error

  ret

;---------------------------------------------;
; Description:                                ;
;   - Connects with the MPU-6050 device, and  ;
;     tells it which register it’s going to   ;
;     read                                    ;
;                                             ;
; void i2c_connect(uint8_t regaddr);          ;
;   regaddr: r16                          ;
;---------------------------------------------;

.global i2c_connect
i2c_connect:

  ; register address comes
  ; in r16
  sts TWDR, r16
  ldi r16, (1<<TWINT)|(1<<TWEN)
  sts TWCR, r16

  ; waits TWINT flag set
  call wait_int_i2c

  ; check TWI status register
  ; to verify DATA ACK received,
  ; goto error otherwise
  lds r16, TWSR
  andi r16, STAT_CODE_MASK
  cpi r16, DATA_SEND_ACK
  call check_error

  ; repeats start
  ldi r16, (1<<TWINT)|(1<<TWSTA)|(1<<TWEN)
  sts TWCR, r16

  ; waits TWINT flag
  call wait_int_i2c

  ; check TWI status register
  ; to verify REPEATED START,
  ; goto error otherwise
  lds r16, TWSR
  andi r16, STAT_CODE_MASK
  cpi r16, REP_START
  call check_error

  ret

;------------------------------------------;
; Description:                             ;
;   - Initializes reading protocol, by     ;
;     sending the slave address in         ;
;     READ mode. It should only be called  ;
;     once before start reading bytes      ;
;     from the device.                     ;
;                                          ;
; void i2c_init_read(void);                ;
;------------------------------------------;

.global i2c_init_read
i2c_init_read:

  ; loads slave address in read mode
  ldi r16, SLA_R
  sts TWDR, r16
  ldi r16, (1<<TWINT)|(1<<TWEN)
  sts TWCR, r16

  ; waits TWINT flag
  call wait_int_i2c

  ; check TWI status register
  ; to verify SLA_R ACK received,
  ; goto error otherwise
  lds r16, TWSR
  andi r16, STAT_CODE_MASK
  cpi r16, SLA_R_ACK
  call check_error

  ret

;----------------------------------------------;
; Description:                                 ;
;   - Reads a byte from the device.            ;
;     Depending on the argument ’more’         ;
;     it will send an ACK if ’more’ is true    ;
;     or it will send a NACK, if ’more’ is     ;
;     false. The argument ’more’, stands for   ;
;     more bytes to be read after              ;
;     the current one.                         ;
;                                              ;
; void i2c_read(uint8_t more);                 ;
;   more: r17                              ;
;----------------------------------------------;

.global i2c_read
i2c_read:

  ; sends "signal" to slave
  ; to read data from it
  ser r16
  sts TWDR, r16

  ;*********************************
  ; Send ACK or NACK, and wait
  ; for it, depending on
  ; the amount of bytes to read.
  ; For example, if more bytes
  ; are going to be read,
  ; ACK must be sent, otherwise
  ; NACK must be sent.
  ;*********************************

  tst r17
  breq stop_read

  ldi r16, (1<<TWINT)|(1<<TWEN)|(1<<TWEA)
  sts TWCR, r16

  ; waits TWINT flag
  call wait_int_i2c

  ; check TWI status register
  ; to verify DATA received
  ; with ACK returned,
  ; goto error otherwise
  lds r16, TWSR
  andi r16, STAT_CODE_MASK
  cpi r16, DATA_RECV_ACK
  call check_error

  rjmp finish_read

stop_read:

  ldi r16, (1<<TWINT)|(1<<TWEN)
  sts TWCR, r16

  ; waits TWINT flag
  call wait_int_i2c

  ; check TWI status register
  ; to verify DATA received
  ; without ACK returned,
  ; goto error otherwise
  lds r16, TWSR
  andi r16, STAT_CODE_MASK
  cpi r16, DATA_RECV_NACK
  call check_error

finish_read:

  ; reads the data obtained
  ; with the previous transaction
  lds r16, TWDR

  ret

;--------------------------------;
; Description:                   ;
;   - Send the STOP signal.      ;
;                                ;
; void i2c_end(void);            ;
;--------------------------------;

.global i2c_end
i2c_end:

  ; sends STOP signal
  ldi r16, (1<<TWINT)|(1<<TWSTO)|(1<<TWEN)
  sts TWCR, r16

  ; waits for STOP condition
  ; to be executed
wait_stop:
  lds r16, TWCR
  andi r16, (1<<TWSTO)
  brne wait_stop

  ret

;************************
;** Auxiliar Functions **
;************************

;***********************************
;   Enables Sensors by clearing   **
;     the SLEEP-MODE default      **
;***********************************

enable_sensor:

  call i2c_start

  ; register address comes
  ; in r16
  ldi r16, PWR_MGMT_1_RA
  sts TWDR, r16
  ldi r16, (1<<TWINT)|(1<<TWEN)
  sts TWCR, r16

  ; waits TWINT flag set
  call wait_int_i2c

  ; check TWI status register
  ; to verify DATA ACK received,
  ; goto error otherwise
  lds r16, TWSR
  andi r16, STAT_CODE_MASK
  cpi r16, DATA_SEND_ACK
  call check_error

  ; sends a zero to clear all
  ; the bits in that register
  ; to make sure that SLEEP-MODE
  ; bit is zero
  clr r16
  sts TWDR, r16
  ldi r16, (1<<TWINT)|(1<<TWEN)
  sts TWCR, r16

  ; waits TWINT flag set
  call wait_int_i2c

  ; check TWI status register
  ; to verify DATA ACK received,
  ; goto error otherwise
  lds r16, TWSR
  andi r16, STAT_CODE_MASK
  cpi r16, DATA_SEND_ACK
  call check_error

  call i2c_end

  ret

;***********************************
;   Verify Slave default Address  **
;***********************************

```

```
check_connection:

  call i2c_init_read

  ; reads the byte
  ldi r17, STOP
  call i2c_read

  sts WHO_AM_I_ADDR, r16

  call i2c_end

  ; compare the value with the
  ; default
  lds r16, WHO_AM_I_ADDR
  cpi r16, MPU_6050_DEF_ADDR
  call check_error

  ret

;************************************
;  Loops until TWINT flag is set   **
;************************************

wait_int_i2c:
  lds r16, TWCR
  sbrs r16, TWINT
  rjmp wait_int_i2c
  ret

check_error:
  brne ERROR
  ret

; error jump code
; loops for ever
ERROR: rjmp ERROR

```

```
#include "avrx.h"

#define STEPS_IN_BURST 8

#define THRESHOLD_ACCEL_POSITIVE 300
#define THRESHOLD_ACCEL_NEGATIVE −300

#define SLEEP_EN 0x1
#define IDLE_MODE 0x0

#define PRESCALE        0b00000011
#define ONLY_OVERFLOW   0b00000001

CSEG

;--------------------------------;
;              SETUP             ;
; Description:                   ;
;   Function that gets executed  ;
;   once the microcontroller is  ;
;   turned on                    ;
;                                ;
; void setup(void)               ;
;--------------------------------;

.global setup
setup:

  call i2c_init
  call stepper_init

  ; enables sleep mode and sets it
  ; in "idle−mode" for later
  ldi r16, (IDLE_MODE | SLEEP_EN)
  out _SFR_IO_ADDR(SMCR), r16

  ldi r16, PRESCALE
  sts TCCR1B, r16; start timer

  ldi r16, ONLY_OVERFLOW
  sts TIMSK1, r16

  sei ; enable interruptions

  ret

;--------------------------------;
;              LOOP              ;
; Description:                   ;
;   Function that gets executed  ;
;   constantly after the setup   ;
;                                ;
; void loop(void)                ;
;--------------------------------;

.global loop
loop:
  sleep
  ret

;-------------------------------;
;   INTER HANDLER TIMER1 OVF    ;
; Description:                  ;
;   Handler of timer1 overflow  ;
;   interruption. It reads      ;
;   from the accelerometer and  ;
;   turns the stepper to        ;
;   correct the deviation       ;
;   measured                    ;
;                               ;
; void timer1_ovf_vect(void)    ;
;-------------------------------;

.global TIMER1_OVF_vect
TIMER1_OVF_vect:

  push r16
  push r17
  push r18
  push r19
  push r20
  push r21
  push r22

  ldi r19, LOW(STEPS_IN_BURST)
  ldi r20, HIGH(STEPS_IN_BURST)

  call get_acceleration

  lds r16, ACCEL_X_L_VAL
  lds r17, ACCEL_X_H_VAL

  ; comparacion de mayor threshold mayor
  ldi r21, LOW(THRESHOLD_ACCEL_POSITIVE)
  ldi r22, HIGH(THRESHOLD_ACCEL_POSITIVE)
  sub r21, r16
  sbc r22, r17
  ; si es mayor que el threshold i.e. el threshold es menor que la acel
  brlt turn_cw ; ir a girar sentido horario

  ; sino comparacion con el threshold menor
  ldi r21, LOW(THRESHOLD_ACCEL_NEGATIVE)
  ldi r22, HIGH(THRESHOLD_ACCEL_NEGATIVE)
  sub r21, r16
  sbc r22, r17
  ; si es mayor que el threshold i.e. el threshold es menor que la acel
  brlt finish_handler; ir al fin

  ; sino girar a la izquierda
  ldi r18, 1
  call stepper_move

  rjmp finish_handler

turn_cw:

  ldi r18, 0
  call stepper_move

finish_handler:

  pop r22
  pop r21
  pop r20
  pop r19
  pop r18
  pop r17
  pop r16

  reti

```

```
#include "avrx.h"

/************************************************/
/* AUTOLEVELING LASER                           */
/* CODE TO BE EXECUTED IN ATMEGA2056 AT 16MHZ   */
/************************************************/

CSEG
  rjmp main

ORG _VECTORS_SIZE

.global main
main:
  ; initialize the stack
  ; pointer to RAMEND
  ldi r16, HIGH(RAMEND)
  out _SFR_IO_ADDR(SPH), r16
  ldi r16, LOW(RAMEND)
  out _SFR_IO_ADDR(SPL), r16

  call setup

here:
  call loop
  rjmp here

```

```
#ifndef SERIAL_H
#define SERIAL_H

#define BAUD_RATE 103

// UCSR0A bits
#define U2X 1

// UCSR0B bits
#define UMSEL1 7
#define UMSEL0 6
#define UPM1 5
#define UPM0 4
#define USBS 3
#define UCSZ1 2
#define UCSZ0 1

// UCSR0C bits
#define RXCIE 7
#define UDRIE 5
#define RXEN 4
#define TXEN 3

#endif // SERIAL_H
```

```c
#ifndef STEPPER_H
#define STEPPER_H

#define MAX_SEQ 8

// Used pins
#define PIN_0 3
#define PIN_1 4
#define PIN_2 5
#define PIN_3 6

// Direction
#define CW_MODE 0x1
#define CCW_MODE 0x0

// Steps
#define STEPS_PER_REV 4096
#define STEPS_PER_REV_HALF 2048
#define STEPS_PER_REV_QUAR 1024
#define STEPS_PER_REV_OCTA 512
#define STEPS_PER_REV_SIXT 256

#endif // STEPPER_H
```

```
#include "avrx.h"
#include "stepper.h"

/**********************************/
/*   28BYJ-48 5V - Stepper motor    */
/**********************************/

DSEG
STEP_NUM: BYTE 1

CSEG
ROT_TABLE: DB 0x08, 0x18, 0x10, 0x30, 0x20, 0x60, 0x40, 0x48

;------------------------------------;
; Description:                       ;
;    - Initializes the stepper motor ;
;      by setting the corresponding  ;
;      pins in output mode.          ;
;                                    ;
; void stepper_init(void);           ;
;------------------------------------;

.global stepper_init
stepper_init:

  ; config digital pins
  ; mapping:
  ;  - IN1 → pin 9: PORTH[6]
  ;  - IN2 → pin 8: PORTH[5]
  ;  - IN3 → pin 7: PORTH[4]
  ;  - IN4 → pin 6: PORTH[3]
  ; in output mode

  ldi r16, (1<<PIN_0)|(1<<PIN_1)|(1<<PIN_2)|(1<<PIN_3)
  sts DDRH, r16

  ldi r16, 0x0
  sts STEP_NUM, r16

  ret
;--------------------------------------------------;
; Description:                                     ;
;    - Moves 'steps' steps in                      ;
;      'dir' direction                             ;
;                                                  ;
; void stepper_move(uint8_t dir, uint16_t steps);  ;
;   dir: r18                                       ;
;   steps: r20:r19                                 ;
;--------------------------------------------------;

.global stepper_move
stepper_move:

  push r26
  push r27

  ; max iteration
  mov r26, r19
  mov r27, r20

forloop:
  rcall one_step
  call mili_delay_1

  sbiw r26, 0x1
  brne forloop

  pop r27
  pop r26

  ret

;---------------------------------;
; Description:                    ;
;   Make one step of the motor    ;
;   going counter-clockwise       ;
;   if dir is diff than 0, anti   ;
;   counter-clockwise otherwise   ;
;                                 ;
;   void one_step(uint8_t dir)    ;
;   dir: r18                      ;
;---------------------------------;

one_step:

  push r16
  push r17

  lds r17, STEP_NUM

  tst r18
  breq ccw_rot

  mov r16, r17

  rjmp finish

ccw_rot:

  ldi r16, MAX_SEQ-1
  sub r16, r17

finish:

  ldi ZH, HIGH(ROT_TABLE)
  ldi ZL, LOW(ROT_TABLE)

  add ZL, r16
  clr r0
  adc ZH, r0

  lpm r16, Z

  sts PORTH, r16

  inc r17

  andi r17, MAX_SEQ-1

  sts STEP_NUM, r17

  pop r17
  pop r16

  ret

```

Table of Contents