/\*

\* USART.h

\*

\*/

#ifndef USART\_RS232\_H\_FILE\_H\_ /\* Define library H file if not defined \*/

#define USART\_RS232\_H\_FILE\_H\_

#define F\_CPU 16000000UL /\* Define CPU clock Frequency e.g. here its 8MHz \*/

#include <avr/io.h> /\* Include AVR std. library file \*/

#define BAUD\_PRESCALE (((F\_CPU / (BAUDRATE \* 16UL))) - 1) /\* Define prescale value \*/

void USART\_Init(unsigned long); /\* USART initialize function \*/

char USART\_RxChar(); /\* Data receiving function \*/

void USART\_TxChar(char); /\* Data transmitting function \*/

void USART\_SendString(char\*); /\* Send string of USART data function \*/

#endif

/\*

\* USART.c

\*

\*/

#include "uart.h" /\* Include USART header file \*/

void USART\_Init(unsigned long BAUDRATE) /\* USART initialize function \*/

{

UCSR0B |= (1 << RXEN0) | (1 << TXEN0); /\* Enable USART transmitter and receiver \*/

UCSR0C |= (1 << UCSZ00) | (1 << UCSZ01); /\* Write USCRC for 8 bit data and 1 stop bit \*/

UBRR0L = BAUD\_PRESCALE; /\* Load UBRRL with lower 8 bit of prescale value \*/

UBRR0H = (BAUD\_PRESCALE >> 8); /\* Load UBRRH with upper 8 bit of prescale value \*/

}

char USART\_RxChar() /\* Data receiving function \*/

{

while (!(UCSR0A & (1 << RXC0))); /\* Wait until new data receive \*/

return(UDR0); /\* Get and return received data \*/

}

void USART\_TxChar(char data) /\* Data transmitting function \*/

{

UDR0 = data; /\* Write data to be transmitting in UDR \*/

while (!(UCSR0A & (1<<UDRE0))); /\* Wait until data transmit and buffer get empty \*/

}

void USART\_SendString(char \*str) /\* Send string of USART data function \*/

{

int i=0;

while (str[i]!=0)

{

USART\_TxChar(str[i]); /\* Send each char of string till the NULL \*/

i++;

}

}

//file is to prototype functions in .c file

#ifndef I2C\_MASTER\_H

#define I2C\_MASTER\_H

#define I2C\_READ 0x01

#define I2C\_WRITE 0x00

void i2c\_init(void);

*uint8\_t* i2c\_start(*uint8\_t* address);

*uint8\_t* i2c\_write(*uint8\_t* data);

*uint8\_t* i2c\_read\_ack(void);

*uint8\_t* i2c\_read\_nack(void);

*uint8\_t* i2c\_transmit(*uint8\_t* address, *uint8\_t*\* data, *uint16\_t* length);

*uint8\_t* i2c\_receive(*uint8\_t* address, *uint8\_t*\* data, *uint16\_t* length);

*uint8\_t* i2c\_writeReg(*uint8\_t* devaddr, *uint8\_t* regaddr, *uint8\_t*\* data, *uint16\_t* length);

*uint8\_t* i2c\_readReg(*uint8\_t* devaddr, *uint8\_t* regaddr, *uint8\_t*\* data, *uint16\_t* length);

void i2c\_stop(void);

#endif

#ifndef F\_CPU

#define F\_CPU 16000000UL

#endif

#include <avr/io.h>

#include <util/twi.h>

#include "i2c\_master.h" //include make file for i2c

#define F\_SCL 100000UL // set SCL frequency

#define Prescaler 1 //set prescaler

#define TWBR\_val ((((F\_CPU / F\_SCL) / Prescaler) - 16 ) / 2)

void i2c\_init(void) //initialize i2c

{

TWBR0 = (*uint8\_t*)TWBR\_val;

}

*uint8\_t* i2c\_start(*uint8\_t* address) //start i2c transmission

{

// reset TWI control register

TWCR0 = 0;

// transmit START condition

TWCR0 = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);

// wait for end of transmission

while( !(TWCR0 & (1<<TWINT)) );

// check if the start condition was successfully transmitted

if((TWSR0 & 0xF8) != *TW\_START*){ return 1; }

// load slave address into data register

TWDR0 = address;

// start transmission of address

TWCR0 = (1<<TWINT) | (1<<TWEN);

// wait for end of transmission

while( !(TWCR0 & (1<<TWINT)) );

// check if the device has acknowledged the READ / WRITE mode

*uint8\_t* twst = *TW\_STATUS* & 0xF8;

if ( (twst != *TW\_MT\_SLA\_ACK*) && (twst != *TW\_MR\_SLA\_ACK*) ) return 1;

return 0;

}

*uint8\_t* i2c\_write(*uint8\_t* data) //write i2c data

{

// load data into data register

TWDR0 = data;

// start transmission of data

TWCR0 = (1<<TWINT) | (1<<TWEN);

// wait for end of transmission

while( !(TWCR0 & (1<<TWINT)) );

if( (TWSR0 & 0xF8) != *TW\_MT\_DATA\_ACK* ){ return 1; }

return 0;

}

*uint8\_t* i2c\_read\_ack(void) //read acknowledge function

{

// start TWI module and acknowledge data after reception

TWCR0 = (1<<TWINT) | (1<<TWEN) | (1<<TWEA);

// wait for end of transmission

while( !(TWCR0 & (1<<TWINT)) );

// return received data from TWDR0

return TWDR0;

}

*uint8\_t* i2c\_read\_nack(void) //read non-acknowledge function

{

// start receiving without acknowledging reception

TWCR0 = (1<<TWINT) | (1<<TWEN);

// wait for end of transmission

while( !(TWCR0 & (1<<TWINT)) );

// return received data from TWDR0

return TWDR0;

}

*uint8\_t* i2c\_transmit(*uint8\_t* address, *uint8\_t*\* data, *uint16\_t* length) //transmit i2c values to uart

{

if (i2c\_start(address | I2C\_WRITE)) return 1;

for (*uint16\_t* i = 0; i < length; i++)

{

if (i2c\_write(data[i])) return 1;

}

i2c\_stop();

return 0;

}

*uint8\_t* i2c\_receive(*uint8\_t* address, *uint8\_t*\* data, *uint16\_t* length) //receive values to i2c

{

if (i2c\_start(address | I2C\_READ)) return 1;

for (*uint16\_t* i = 0; i < (length-1); i++)

{

data[i] = i2c\_read\_ack();

}

data[(length-1)] = i2c\_read\_nack();

i2c\_stop();

return 0;

}

*uint8\_t* i2c\_writeReg(*uint8\_t* devaddr, *uint8\_t* regaddr, *uint8\_t*\* data, *uint16\_t* length) //write to i2c register

{

if (i2c\_start(devaddr | 0x00)) return 1;

i2c\_write(regaddr);

for (*uint16\_t* i = 0; i < length; i++)

{

if (i2c\_write(data[i])) return 1;

}

i2c\_stop();

return 0;

}

*uint8\_t* i2c\_readReg(*uint8\_t* devaddr, *uint8\_t* regaddr, *uint8\_t*\* data, *uint16\_t* length) //read from i2c register

{

if (i2c\_start(devaddr)) return 1;

i2c\_write(regaddr);

if (i2c\_start(devaddr | 0x01)) return 1;

for (*uint16\_t* i = 0; i < (length-1); i++)

{

data[i] = i2c\_read\_ack();

}

data[(length-1)] = i2c\_read\_nack();

i2c\_stop();

return 0;

}

void i2c\_stop(void)

{

// transmit STOP condition

TWCR0 = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);

}

/\*

\* Harrin2\_Midterm\_2\_mpu.c

\*

\* Created: 12/13/2019 7:12:52 PM

\* Author : harrin2

\*/

#ifndef F\_CPU

#define F\_CPU 16000000UL

#endif

#include <avr/io.h>

#include <util/delay.h>

#include <math.h>

#include <stdlib.h> //Include standard library file

#include <stdio.h> //Include standard library file

#include "MPU6050\_def.h" //Include MPU6050 register define file

#include "i2c\_master.h" //Include I2C Master header file

#include "uart.h" //Include USART header file

#define MPU6050\_WRITE 0xD0

#define MPU6050\_READ 0xD1

float Acc\_x;

void init\_uart(*uint16\_t* baudrate){ //intialize uart and buad rate

*uint16\_t* UBRR\_val = (F\_CPU/16)/(baudrate-1);

UBRR0H = UBRR\_val >> 8;

UBRR0L = UBRR\_val;

UCSR0B |= (1<<TXEN0) | (1<<RXEN0) | (1<<RXCIE0); // UART TX

UCSR0C |= (1<<USBS0) | (3<<UCSZ00); //Modus Asynchronous 8N1

}

void uart\_putc(unsigned char c){ //function helps send chars through uart

while(!(UCSR0A & (1<<UDRE0))); // wait for sending

UDR0 = c; // output character saved

}

void uart\_puts(char \*s){ //function helps send strings through uart

while(\*s){

uart\_putc(\*s);

s++;

}

}

void init\_MPU6050(void){ //initialize MPU

*\_delay\_ms*(150); //Power up time greater than 100ms

i2c\_start(MPU6050\_WRITE); // Set Gyroscope Sample Rate = 1 KHz, Accelerometer Sample Rate = 1 KHz (default)

i2c\_write(SMPLRT\_DIV); // Sample Rate is generated by dividing the gyroscope output rate by SMPLRT\_DIV

i2c\_write(0x07); // Gyroscope Output Rate = 8kHz, Sample Rate = Gyroscope Output Rate / (1 + SMPLRT\_DIV)

i2c\_stop();

i2c\_start(MPU6050\_WRITE);

i2c\_write(PWR\_MGMT\_1);

i2c\_write(0x01); // PLL with X axis gyroscope reference

i2c\_stop();

i2c\_start(MPU6050\_WRITE);

i2c\_write(CONFIG); //Frame Synchronization & Digital Low Pass Filter setting

i2c\_write(0x00);

i2c\_stop();

i2c\_start(MPU6050\_WRITE);

i2c\_write(GYRO\_CONFIG); //gyroscopes’ scale range = FS\_SEL selects = 11 = +/- 2000 °/s

i2c\_write(0x18); // default accelerometer range = +/- 2g

i2c\_stop();

i2c\_start(MPU6050\_WRITE);

i2c\_write(INT\_ENABLE); // DATA\_RDY\_EN = 1

i2c\_write(0x01);

i2c\_stop();

}

void getreading(void){ //read values from i2c

i2c\_start(MPU6050\_WRITE);

i2c\_write(ACCEL\_XOUT\_H); // set pointer

i2c\_stop();

i2c\_start(MPU6050\_READ);

Acc\_x = (((int)i2c\_read\_ack()<<8) | (int)i2c\_read\_ack());

i2c\_stop();

}

int main(void){ //Main function

char buffer[20], float\_[10];

float Xa;

init\_uart(9600);

i2c\_init();

init\_MPU6050();

while(1){

getreading();

Xa = Acc\_x/16384.0; //Divide raw value by sensitivity scale factor to get real values

*dtostrf*( Xa, 3, 2, float\_ ); //Take values in buffer to send all parameters over USART

*sprintf*(buffer,"%s, ",float\_);

USART\_SendString(buffer);

*\_delay\_ms*(1000);

}

return 0;

}

/\*

\* MPU6050\_res\_define.h

\*

\* Created: 12/13/2019 20:45:23

\* Author: Harrin2

\*/

#ifndef MPU6050\_RES\_DEFINE\_H\_

#define MPU6050\_RES\_DEFINE\_H\_

#include <avr/io.h>

#define XG\_OFFS\_TC 0x00

#define YG\_OFFS\_TC 0x01

#define ZG\_OFFS\_TC 0x02

#define X\_FINE\_GAIN 0x03

#define Y\_FINE\_GAIN 0x04

#define Z\_FINE\_GAIN 0x05

#define XA\_OFFS\_H 0x06

#define XA\_OFFS\_L\_TC 0x07

#define YA\_OFFS\_H 0x08

#define YA\_OFFS\_L\_TC 0x09

#define ZA\_OFFS\_H 0x0A

#define ZA\_OFFS\_L\_TC 0x0B

#define XG\_OFFS\_USRH 0x13

#define XG\_OFFS\_USRL 0x14

#define YG\_OFFS\_USRH 0x15

#define YG\_OFFS\_USRL 0x16

#define ZG\_OFFS\_USRH 0x17

#define ZG\_OFFS\_USRL 0x18

#define SMPLRT\_DIV 0x19

#define CONFIG 0x1A

#define GYRO\_CONFIG 0x1B

#define ACCEL\_CONFIG 0x1C

#define FF\_THR 0x1D

#define FF\_DUR 0x1E

#define MOT\_THR 0x1F

#define MOT\_DUR 0x20

#define ZRMOT\_THR 0x21

#define ZRMOT\_DUR 0x22

#define FIFO\_EN 0x23

#define I2C\_MST\_CTRL 0x24

#define I2C\_SLV0\_ADDR 0x25

#define I2C\_SLV0\_REG 0x26

#define I2C\_SLV0\_CTRL 0x27

#define I2C\_SLV1\_ADDR 0x28

#define I2C\_SLV1\_REG 0x29

#define I2C\_SLV1\_CTRL 0x2A

#define I2C\_SLV2\_ADDR 0x2B

#define I2C\_SLV2\_REG 0x2C

#define I2C\_SLV2\_CTRL 0x2D

#define I2C\_SLV3\_ADDR 0x2E

#define I2C\_SLV3\_REG 0x2F

#define I2C\_SLV3\_CTRL 0x30

#define I2C\_SLV4\_ADDR 0x31

#define I2C\_SLV4\_REG 0x32

#define I2C\_SLV4\_DO 0x33

#define I2C\_SLV4\_CTRL 0x34

#define I2C\_SLV4\_DI 0x35

#define I2C\_MST\_STATUS 0x36

#define INT\_PIN\_CFG 0x37

#define INT\_ENABLE 0x38

#define DMP\_INT\_STATUS 0x39

#define INT\_STATUS 0x3A

#define ACCEL\_XOUT\_H 0x3B

#define ACCEL\_XOUT\_L 0x3C

#define ACCEL\_YOUT\_H 0x3D

#define ACCEL\_YOUT\_L 0x3E

#define ACCEL\_ZOUT\_H 0x3F

#define ACCEL\_ZOUT\_L 0x40

#define TEMP\_OUT\_H 0x41

#define TEMP\_OUT\_L 0x42

#define GYRO\_XOUT\_H 0x43

#define GYRO\_XOUT\_L 0x44

#define GYRO\_YOUT\_H 0x45

#define GYRO\_YOUT\_L 0x46

#define GYRO\_ZOUT\_H 0x47

#define GYRO\_ZOUT\_L 0x48

#define EXT\_SENS\_DATA\_00 0x49

#define EXT\_SENS\_DATA\_01 0x4A

#define EXT\_SENS\_DATA\_02 0x4B

#define EXT\_SENS\_DATA\_03 0x4C

#define EXT\_SENS\_DATA\_04 0x4D

#define EXT\_SENS\_DATA\_05 0x4E

#define EXT\_SENS\_DATA\_06 0x4F

#define EXT\_SENS\_DATA\_07 0x50

#define EXT\_SENS\_DATA\_08 0x51

#define EXT\_SENS\_DATA\_09 0x52

#define EXT\_SENS\_DATA\_10 0x53

#define EXT\_SENS\_DATA\_11 0x54

#define EXT\_SENS\_DATA\_12 0x55

#define EXT\_SENS\_DATA\_13 0x56

#define EXT\_SENS\_DATA\_14 0x57

#define EXT\_SENS\_DATA\_15 0x58

#define EXT\_SENS\_DATA\_16 0x59

#define EXT\_SENS\_DATA\_17 0x5A

#define EXT\_SENS\_DATA\_18 0x5B

#define EXT\_SENS\_DATA\_19 0x5C

#define EXT\_SENS\_DATA\_20 0x5D

#define EXT\_SENS\_DATA\_21 0x5E

#define EXT\_SENS\_DATA\_22 0x5F

#define EXT\_SENS\_DATA\_23 0x60

#define MOT\_DETECT\_STATUS 0x61

#define I2C\_SLV0\_DO 0x63

#define I2C\_SLV1\_DO 0x64

#define I2C\_SLV2\_DO 0x65

#define I2C\_SLV3\_DO 0x66

#define I2C\_MST\_DELAY\_CTRL 0x67

#define SIGNAL\_PATH\_RESET 0x68

#define MOT\_DETECT\_CTRL 0x69

#define USER\_CTRL 0x6A

#define PWR\_MGMT\_1 0x6B

#define PWR\_MGMT\_2 0x6C

#define BANK\_SEL 0x6D

#define MEM\_START\_ADDR 0x6E

#define MEM\_R\_W 0x6F

#define DMP\_CFG\_1 0x70

#define DMP\_CFG\_2 0x71

#define FIFO\_COUNTH 0x72

#define FIFO\_COUNTL 0x73

#define FIFO\_R\_W 0x74

#define WHO\_AM\_I 0x75

#endif /\* MPU6050\_RES\_DEFINE\_H\_ \*/