CPE301 – SPRING 2020

Design Assignment 6

Student Name: Jason Villanueva

Student #: 2001255420

Student Email: villaj21@unlv.nevada.edu

Primary Github address: <https://github.com/jasonvillanuevagit/submission_designAssignments->

Directory: <https://github.com/jasonvillanuevagit/submission_designAssignments-/tree/master/DesignAssignment6>

1. **COMPONENTS LIST AND CONNECTION BLOCK DIAGRAM w/ PINS**

Atmel Studio 7.0 Atmega328PB-Xmini PC MPU6050 Sensor

- Assembler -Multifunction Shield

- Simulator

- Debugger

A close up of text on a white background

Description automatically generated

1. **INITIAL/MODIFIED/DEVELOPED CODE OF TASK 1/A**

#define *F\_CPU* 16000000UL

#include <avr/io.h>

#include <avr/interrupt.h>

#include <stdio.h>

#include <math.h>

//TW\_CONFIG

#define PRESCALER 1U

#define SCL\_FREQ\_200kH 200000UL

#define SCL\_FREQ\_400kH 400000UL

#define TW\_TWBR (((*F\_CPU*/SCL\_FREQ\_400kH) - 16)/(2\*PRESCALER))

#define TW\_config\_SCL\_200kH

#define TW\_TWCR\_TWINT (1U<<7)

#define TW\_TWCR\_TWEA (1U<<6)

#define TW\_TWCR\_TWSTA (1U<<5)

#define TW\_TWCR\_TWSTO (1U<<4)

#define TW\_TWCR\_TWWC (1U<<3)

#define TW\_TWCR\_TWEN (1U<<2)

#define TW\_TWCR\_TWIE (1U)

#define TW\_TWSR\_scaler64 (3U<<0)

#define TW\_TWSR\_scaler4 (1U<<0)

#define TW\_TWSR\_scaler1 (0U<<0);

//PWM

#define PWM\_8Scalar 0x02

#define PWM\_noScalar 0x01

#define PWM\_WGM\_11\_10 0x02

#define PWM\_WGM\_13\_12 0x02

#define PWM\_nonInvert 0x02

#define PWM\_timerOVR 0x01

//Bool

#define bool *uint8\_t*

#define true 0x01U

#define false 0x00U

#define MPU6050\_conversion\_accel\_2g 16384.0

#define MPU6050\_conversion\_gyro\_250dps 131.0

#define MPU6050\_conversion\_gyro\_2000dps 16.4

#define MPU6050\_conversion\_radians\_to\_degrees (180.0/*acos*(-1))

typedef enum{

MasterTX\_Data,

MasterTX\_Done,

}status\_t;

typedef struct{

volatile *int8\_t* rx[12];

volatile *uint8\_t* rxCounter;

volatile *uint8\_t* tx;

volatile *uint8\_t* slaveAddr;

volatile *uint8\_t* regAddr;

volatile bool busy;

volatile status\_t status;

}TW\_data\_t;

typedef struct{

volatile float X;

volatile float Y;

volatile float Z;

}accel\_t;

typedef struct{

volatile float X;

volatile float Y;

volatile float Z;

volatile float xErr;

volatile float yErr;

volatile float zErr;

}gyro\_t;

typedef struct{

volatile float curr\_Angle;

volatile float prev\_Angle;

}tilt\_angle\_t;

TW\_data\_t TW\_data;

accel\_t accel\_data;

gyro\_t gyro\_data;

tilt\_angle\_t roll\_angle;

tilt\_angle\_t pitch\_angle;

volatile bool sampleReady = false;

#define MPU6050\_FIFO\_RegEn TW\_send(0x68, 35U, ((1U<<6)|(1U<<5)|(1U<<4)|(1U<<3)))

#define MPU6050\_FIFO\_EN TW\_send(0x68, 106U, (1U<<6))

#define MPU6050\_CLEAR\_STATUS TW\_receive(0x68, 58U, 1)

#define MPU6050\_FIFO\_CLEAR TW\_send(0x68, 106U, (1U<<2));

//Uart

#define rxEN 1

#define txIEN 1

#define drIEN 1

#define txEN 1

#define eightBit 0x03

#define async 0x00

#define sync 0x01

#define noParity 0x00

#define oneStop 0x00

#define twoStop 0x01

#define chSize8 0x03

#define BAUD 300000UL

#define BAUDPRESCALAR ((*F\_CPU*/(BAUD))/16 - 1) //Register Values

static volatile unsigned int usartBUSY; //ISR variables

static volatile char\* dataString;

static char charConversion[20];

void TW\_send(*uint8\_t* slaveAddr, *uint8\_t* regAddr, *uint8\_t* dataTx);

void TW\_receive(*uint8\_t* slaveAddr, *uint8\_t* regAddr, *uint8\_t* rxCounter);

void initiate\_TW();

void TW\_Status\_Code\_TX();

void TW\_Status\_Code\_RX();

float combineRegisters(*int16\_t* high, *int16\_t* low);

float calculateSensorData(*int16\_t* data, *int16\_t* conversion);

void convert\_float\_to\_string(float floatNum);

void sendString\_UART();

void initUART(*uint8\_t* umsel, *uint8\_t* upm, *uint8\_t* usbs, *uint8\_t* ucsz);

void UART\_Send\_Values();

void calculatePosition();

void inititate\_MPU6050();

void initiate\_INT0();

void storeSensorValues\_Accel();

void storeSensorValues\_Gyro();

float calculateAngle\_Pitch\_Accel();

float calculatetAngle\_Roll\_Accel();

float calculateAngle\_Pitch\_Gyro();

float calculatetAngle\_Roll\_Gyro();

float complementaryFilter(float gyro\_angle, float accel\_angle, tilt\_angle\_t\* angle);

void calculateGyroError();

void storeGyroError();

void initTimer1PWM(unsigned int top);

void startPWM1();

void sendMessage(char\* mess);

void send\_accel\_values(accel\_t\*);

void send\_gyro\_values(gyro\_t\*);

#define dt 0.0100

#define scale\_gyro .96

#define scale\_accel .02

volatile float temp;

int main(void){

TW\_data.rx[0] = 0xFF;

DDRB = (1U << 1); //DDRB.1 out OC1A

initiate\_TW();

initUART(async, noParity, oneStop, eightBit); //initiate UART 8-N-1

initiate\_INT0();

initTimer1PWM(20000);

sei();

startPWM1(); //start PWM

inititate\_MPU6050();

calculateGyroError();

while (1){

if(sampleReady){

sampleReady = false;

storeSensorValues\_Accel();

storeSensorValues\_Gyro();

send\_accel\_values(&accel\_data);

send\_gyro\_values(&gyro\_data);

pitch\_angle.curr\_Angle = complementaryFilter(calculateAngle\_Pitch\_Gyro(), calculateAngle\_Pitch\_Accel(), &pitch\_angle);

convert\_float\_to\_string(pitch\_angle.curr\_Angle);

sendString\_UART();

sendMessage(" ");

roll\_angle.curr\_Angle = complementaryFilter(calculatetAngle\_Roll\_Gyro(), calculatetAngle\_Roll\_Accel(), &roll\_angle);

convert\_float\_to\_string(roll\_angle.curr\_Angle);

sendString\_UART();

sendMessage(" ");

sendMessage("\r\n");

TW\_receive(0x68, 58U, 1);

}

}

}

void TW\_Status\_Code\_TX(){

switch(TWSR0 & 0xF8){

case 0x08:

(TWDR0 = (TW\_data.slaveAddr<<1));

(TWCR0 = TW\_TWCR\_TWEN|TW\_TWCR\_TWINT);

break;

case 0x18:

(TWDR0 = TW\_data.regAddr);

TW\_data.status = MasterTX\_Data;

(TWCR0 = TW\_TWCR\_TWEN|TW\_TWCR\_TWINT);

break;

case 0x28:

if(TW\_data.status == MasterTX\_Data){

(TWDR0 = TW\_data.tx);

TW\_data.status = MasterTX\_Done;

(TWCR0 = TW\_TWCR\_TWEN|TW\_TWCR\_TWINT);

}

else{

(TWCR0 = TW\_TWCR\_TWSTO|TW\_TWCR\_TWEN | TW\_TWCR\_TWINT);

TW\_data.busy = false;

}

break;

default:

TW\_data.busy = false;

}

}

void TW\_Status\_Code\_RX(){

switch((TWSR0 & 0xF8)){

case 0x08:

(TWDR0 = (TW\_data.slaveAddr<<1));

(TWCR0 = TW\_TWCR\_TWEN|TW\_TWCR\_TWINT);

break;

case 0x18:

(TWDR0 = TW\_data.regAddr);

(TWCR0 = TW\_TWCR\_TWEN|TW\_TWCR\_TWINT);

break;

case 0x28:

(TWCR0 = TW\_TWCR\_TWSTA|TW\_TWCR\_TWEN | TW\_TWCR\_TWINT);

break;

case 0x10:

(TWDR0 = ((TW\_data.slaveAddr<<1) | 0x01));

(TWCR0 = TW\_TWCR\_TWEN|TW\_TWCR\_TWINT);

break;

case 0x40:

if(TW\_data.rxCounter == 0){

(TWCR0 = TW\_TWCR\_TWEN|TW\_TWCR\_TWINT);

}

else{

(TWCR0 = TW\_TWCR\_TWEA|TW\_TWCR\_TWEN | TW\_TWCR\_TWINT);

}

break;

case 0x50:

if(TW\_data.rxCounter == 1){

TW\_data.rx[TW\_data.rxCounter] = (TWDR0);

(TWCR0 = TW\_TWCR\_TWEN|TW\_TWCR\_TWINT);

}

else{

TW\_data.rx[TW\_data.rxCounter] = (TWDR0);

TW\_data.rxCounter--;

(TWCR0 = TW\_TWCR\_TWEA|TW\_TWCR\_TWEN | TW\_TWCR\_TWINT);

}

break;

case 0x58:

TW\_data.rx[0] = (TWDR0);

(TWCR0 = TW\_TWCR\_TWSTO|TW\_TWCR\_TWEN | TW\_TWCR\_TWINT);

TW\_data.busy = false;

break;

default:

TW\_data.busy = false;

}

}

void sendMessage(char\* mess){

dataString = mess;

sendString\_UART();

}

void send\_gyro\_values(gyro\_t\* gyro){

convert\_float\_to\_string(gyro->X);

sendString\_UART();

sendMessage(" ");

convert\_float\_to\_string(gyro->Y);

sendString\_UART();

sendMessage(" ");

convert\_float\_to\_string(gyro->Z);

sendString\_UART();

sendMessage(" ");

}

void send\_accel\_values(accel\_t\* accel){

convert\_float\_to\_string(accel->X);

sendString\_UART();

sendMessage(" ");

convert\_float\_to\_string(accel->Y);

sendString\_UART();

sendMessage(" ");

convert\_float\_to\_string(accel->Z);

sendString\_UART();

sendMessage(" ");

}

void calculatePosition(){

if(pitch\_angle.curr\_Angle < 0){

OCR1A = (*uint16\_t*)(1500 + pitch\_angle.curr\_Angle\*7.5);

}

else{

OCR1A = (*uint16\_t*)(1500 + pitch\_angle.curr\_Angle\*7.5);

}

}

void initiate\_INT0(){

DDRD &= ~(1<<2);

PORTD &= ~(1<<2);

EICRA = 0x03;

EIMSK = 0x01;

}

void inititate\_MPU6050(){

sampleReady = false;

pitch\_angle.prev\_Angle = 0;

roll\_angle.prev\_Angle = 0;

gyro\_data.X = 0;

gyro\_data.Y = 0;

gyro\_data.Z = 0;

TW\_send(0x68, 27U, (3U<<3));

TW\_send(0x68, 25U, 47U);

TW\_send(0x68, 55U, 0x20);

TW\_send(0x68, 56U, 0x01);

TW\_send(0x68, 0x6B, 0x00);

TW\_receive(0x68, 58U, 1);

}

void storeSensorValues\_Gyro(){

volatile *uint8\_t* sensorValH;

volatile *uint8\_t* sensorValL;

TW\_receive(0x68, 0x43, 1);

sensorValH = TW\_data.rx[0];

TW\_receive(0x68, 0x44, 1);

sensorValL = TW\_data.rx[0];

gyro\_data.X = calculateSensorData(combineRegisters(sensorValH, sensorValL)-gyro\_data.xErr, MPU6050\_conversion\_gyro\_2000dps);

TW\_receive(0x68, 0x45, 1);

sensorValH = TW\_data.rx[0];

TW\_receive(0x68, 0x46, 1);

sensorValL = TW\_data.rx[0];

gyro\_data.Y = calculateSensorData(combineRegisters(sensorValH, sensorValL)-gyro\_data.yErr, MPU6050\_conversion\_gyro\_2000dps);

TW\_receive(0x68, 0x47, 1);

sensorValH = TW\_data.rx[0];

TW\_receive(0x68, 0x48, 1);

sensorValL = TW\_data.rx[0];

gyro\_data.Z = calculateSensorData(combineRegisters(sensorValH, sensorValL)-gyro\_data.zErr, MPU6050\_conversion\_gyro\_2000dps);

}

void calculateGyroError(){

for(unsigned int i = 0; i < 200; i++){

while(!sampleReady);

sampleReady = false;

storeGyroError();

TW\_receive(0x68, 58U, 1);

}

gyro\_data.xErr = gyro\_data.X/200;

gyro\_data.yErr = gyro\_data.Y/200;

gyro\_data.zErr = gyro\_data.Z/200;

gyro\_data.X = 0;

gyro\_data.Y = 0;

gyro\_data.Z = 0;

TW\_receive(0x68, 58U, 1);

}

void storeGyroError(){

volatile *uint8\_t* sensorValH;

volatile *uint8\_t* sensorValL;

TW\_receive(0x68, 0x43, 1);

sensorValH = TW\_data.rx[0];

TW\_receive(0x68, 0x44, 1);

sensorValL = TW\_data.rx[0];

gyro\_data.X += combineRegisters(sensorValH, sensorValL);

TW\_receive(0x68, 0x45, 1);

sensorValH = TW\_data.rx[0];

TW\_receive(0x68, 0x46, 1);

sensorValL = TW\_data.rx[0];

gyro\_data.Y += combineRegisters(sensorValH, sensorValL);

TW\_receive(0x68, 0x47, 1);

sensorValH = TW\_data.rx[0];

TW\_receive(0x68, 0x48, 1);

sensorValL = TW\_data.rx[0];

gyro\_data.Z += combineRegisters(sensorValH, sensorValL);

}

void storeSensorValues\_Accel(){

volatile *uint8\_t* sensorValH;

volatile *uint8\_t* sensorValL;

TW\_receive(0x68, 0x3B, 1);

sensorValH = TW\_data.rx[0];

TW\_receive(0x68, 0x3C, 1);

sensorValL = TW\_data.rx[0];

accel\_data.X = calculateSensorData(combineRegisters(sensorValH, sensorValL), MPU6050\_conversion\_accel\_2g);

TW\_receive(0x68, 0x3D, 1);

sensorValH = TW\_data.rx[0];

TW\_receive(0x68, 0x3E, 1);

sensorValL = TW\_data.rx[0];

accel\_data.Y = calculateSensorData(combineRegisters(sensorValH, sensorValL), MPU6050\_conversion\_accel\_2g);

TW\_receive(0x68, 0x3F, 1);

sensorValH = TW\_data.rx[0];

TW\_receive(0x68, 0x40, 1);

sensorValL = TW\_data.rx[0];

accel\_data.Z = calculateSensorData(combineRegisters(sensorValH, sensorValL), MPU6050\_conversion\_accel\_2g);

}

float combineRegisters(*int16\_t* high, *int16\_t* low){

return ((high<<8) | low);

}

float calculateSensorData(*int16\_t* data, *int16\_t* conversion){

return ((float)data/(float)conversion);

}

float calculateAngle\_Pitch\_Accel(){

return (float)(*atan*(accel\_data.Y/*sqrt*(accel\_data.X\*accel\_data.X + accel\_data.Z\*accel\_data.Z))\*MPU6050\_conversion\_radians\_to\_degrees);

}

float calculatetAngle\_Roll\_Accel(){

return (float)(*atan*(accel\_data.X/(*sqrtf*(accel\_data.Y\*accel\_data.Y + accel\_data.Z\*accel\_data.Z)))\*MPU6050\_conversion\_radians\_to\_degrees);

}

float calculateAngle\_Pitch\_Gyro(){

return (gyro\_data.X \* dt);

}

float calculatetAngle\_Roll\_Gyro(){

return (gyro\_data.Y \* dt);

}

float complementaryFilter(float gyro\_angle, float accel\_angle, tilt\_angle\_t\* angle){

angle->prev\_Angle = (float)(((gyro\_angle + angle->prev\_Angle) \* scale\_gyro) + (accel\_angle \* scale\_accel));

return angle->prev\_Angle;

}

void TW\_send(*uint8\_t* slaveAddr, *uint8\_t* regAddr, *uint8\_t* dataTx){

TW\_data.busy = true;

TW\_data.slaveAddr = slaveAddr;

TW\_data.tx = dataTx;

TW\_data.regAddr = regAddr;

(TWCR0 = TW\_TWCR\_TWSTA|TW\_TWCR\_TWEN);

while(TW\_data.busy){

while(!(TWCR0 & (1U<<TWINT))){};

TW\_Status\_Code\_TX();

}

(TWCR0 = 0x00);

}

void TW\_receive(*uint8\_t* slaveAddr, *uint8\_t* regAddr, *uint8\_t* rxCounter){

TW\_data.busy = true;

TW\_data.slaveAddr = slaveAddr;

TW\_data.regAddr = regAddr;

TW\_data.rxCounter = rxCounter - 1;

(TWCR0 = TW\_TWCR\_TWSTA|TW\_TWCR\_TWEN);

while(TW\_data.busy){

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

TW\_Status\_Code\_RX();

}

(TWCR0 = 0x00);

}

void initiate\_TW(){

TWBR0 = TW\_TWBR;

TWSR0 |= TW\_TWSR\_scaler1;

}

void convert\_float\_to\_string(float floatNum){

*snprintf*(charConversion, sizeof(charConversion), "%f ", floatNum);

dataString = charConversion;

}

void sendString\_UART(){

//USART busy, enable TX and interrupts

usartBUSY = 0x01;

UCSR0B |= ((txIEN << TXCIE0) |

(drIEN << UDRIE0) |

(txEN << TXEN0));

while(usartBUSY);

}

void initUART(*uint8\_t* umsel, *uint8\_t* upm, *uint8\_t* usbs, *uint8\_t* ucsz){

//initiates UART Baud and sets it to a 8-N-1 setting

UBRR0H = (*uint8\_t*)(0x00);

UBRR0L = (*uint8\_t*)(BAUDPRESCALAR);

UCSR0C = (umsel << UMSEL00) |

(upm << UPM00) |

(usbs << USBS0) |

(ucsz << UCSZ00);

}

void initTimer1PWM(unsigned int top){

//inititate PWM

TCCR1A |= (PWM\_nonInvert << COM1A0);

ICR1 = (top);

}

void startPWM1(){

//start PWM

TCNT1L = 0x00;

TCNT1H = 0x00;

TCCR1A |= (PWM\_WGM\_11\_10 << WGM10);

TCCR1B |= (PWM\_WGM\_13\_12 << WGM12);

TIMSK1 |= (PWM\_timerOVR << TOIE1);

TCCR1B |= (PWM\_8Scalar << CS10);

}

ISR(USART0\_UDRE\_vect){

//if datastring is not empty send character

//else send disable interrupt and signal when usart is done

if(\*dataString != '\0'){

UDR0 = \*dataString;

dataString++;

}

else{

UCSR0B &= ~(drIEN << UDRIE0);

usartBUSY = 0x00;

}

}

ISR(TIMER1\_OVF\_vect){

calculatePosition();

}

ISR(USART0\_TX\_vect){

//disables TX and

UCSR0B &= ~((txIEN << TXCIE0) |

(txEN << TXEN0));

}

ISR(INT0\_vect){

sampleReady = true;

}

1. **DEVELOPED MODIFIED CODE OF TASK 2/A from TASK 1/A**

N/A

1. **SCHEMATICS**

A circuit board

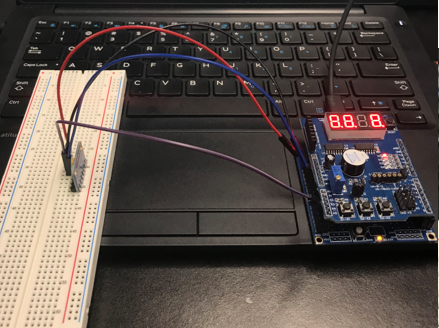
Description automatically generated

1. **SCREENSHOTS OF EACH TASK OUTPUT (ATMEL STUDIO OUTPUT)**

A screenshot of a computer screen

Description automatically generated

1. **SCREENSHOT OF EACH DEMO (BOARD SETUP)**

A picture containing computer, indoor, electronics, laptop

Description automatically generated

1. **VIDEO LINKS OF EACH DEMO**
2. **GITHUB LINK OF THIS DA**

<https://github.com/jasonvillanuevagit/submission_designAssignments-/tree/master/DesignAssignment6>

**Student Academic Misconduct Policy**

<http://studentconduct.unlv.edu/misconduct/policy.html>

“This assignment submission is my own, original work”.

Jason Villanueva