CPE301 – SPRING 2019

Design Assignment 6A

Student Name: Tanner Tindall

Student #: 8000733733

Student Email: tindat1@unlv.nevada.edu

Primary Github address: <https://github.com/TannerTindall51>

Directory:

<https://github.com/TannerTindall51/tindalltannerm_submission/upload/master/Design_Assignments/DA6A>

Design Assignment 6

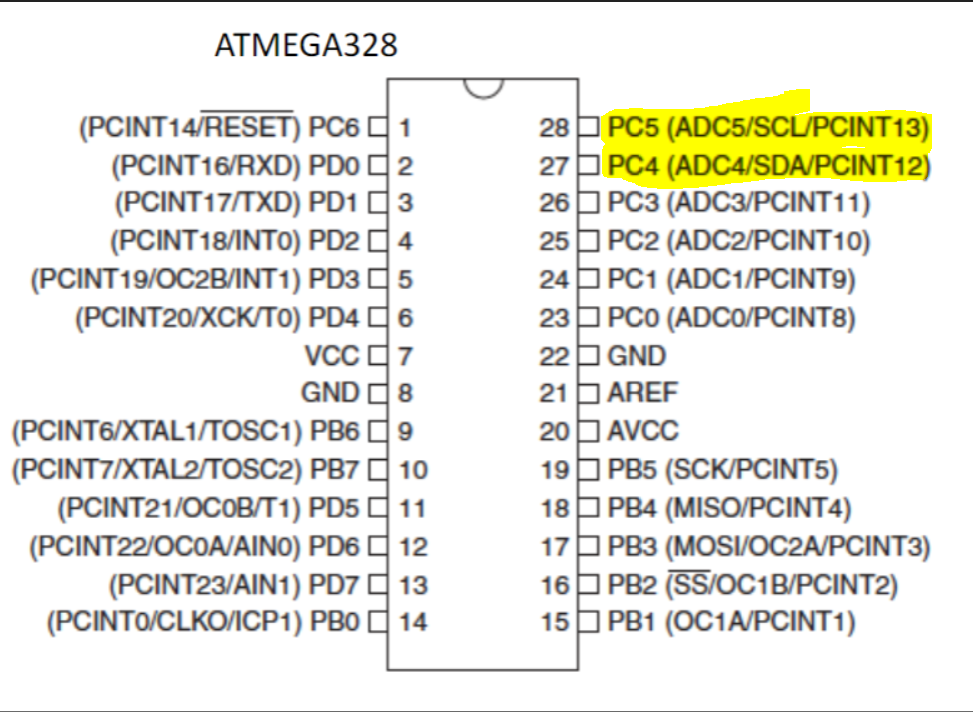
The goal of the assignment is to develop the above code to do the following:

1. Interface the provided MPU-6050 6-DOF IMU Sensor to the ATmega328p using the I2C interface. Using the earlier developed code for UART, display the accelerometer and gyro data to the UART Terminal. Visualizing the accelerometer and gyro values using the serial plotter.

2. Apply complementary filtering\* on the sensor data, calculate the smooth/filtered pitch and roll angles and display the filtered values in the graph.

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

* Atmel Studio 7.0 (Assembler, Simulator, & Debugger)
* Atmega328PB-Xmini
* Micro USB
* MP6050
* External Power Source



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

//

//CPE301 - DA6A Task 1

//Tanner Tindall

//

#define BAUD 9600

#define *F\_CPU* 16000000UL

#include <avr/io.h>

#include <util/setbaud.h>

#include <stdlib.h>

#include <stdio.h>

#include <util/delay.h>

#define SLAVE\_ADDRESS 0xD0

#define DIVIDER\_ADDRESS 0x19

#define POWER\_ADDRESS 0x6B

#define CONFIG\_ADDRESS 0x1A

#define GYRO\_CONFIG\_ADDRESS 0x1B

#define INTERRUPT\_ADDRESS 0x38

#define ACC\_START\_ADDRESS 0x3B

#define GYRO\_START\_ADDRESS 0x43

#define UBBR\_VALUE 103

#define BITRATE 3

volatile *uint8\_t* data;

volatile *uint8\_t* value;

volatile float Ax,Ay,Az;

volatile float Gx=0, Gy=0, Gz=0;

float Acc\_x,Acc\_y,Acc\_z,Temperature,Gyro\_x,Gyro\_y,Gyro\_z;

void i2cInit(void)

{

TWBR0 = BITRATE; //set bitrate

TWSR0 |= (0<<TWPS1)|(0<<TWPS0); //set prescaler to 1

}

unsigned char i2cRead(value)

{

if (value == 0) //ack bit & continue

{

TWCR0= (1<<TWINT) | (1<<TWEN) | (1<<TWEA); //enable TWI/ACK and clear flag

}

else

{

TWCR0= (1<<TWINT) | (1<<TWEN); //enable TWI & clear flag

}

while ((TWCR0& (1<<TWINT)) == 0); //wait until emptied

return TWDR0; //return value to TWDR

}

void i2cWrite(data)

{

TWDR0 = data; //pull data from TWI register

TWCR0 = (1<<TWINT) | (1<<TWEN); //enable TWI & clear flag

while ((TWCR0&(1<<TWINT)) == 0); //wait until done

}

void i2cStart(void)

{

TWCR0= (1<<TWINT) | (1<<TWSTA) | (1<<TWEN); //enable TWI, clear flag, start condition

while (TWCR0 & (1<<TWINT) == 0); //wait until emptied

}

void i2cStop(void)

{

TWCR0 = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO); //enable TWI, clear flag, stop condition

while ((TWCR0&(1<<TWINT)) == 0); //wait until done

}

void i2cIdle(char slave\_write\_address) //taken from Dr. Venkis I2C start wait function in Demo code

{

*uint8\_t* check; /\* Declare variable \*/

while (1)

{

TWCR0 = (1<<TWSTA)|(1<<TWEN)|(1<<TWINT); /\* Enable TWI, generate start condition and clear interrupt flag \*/

while (!(TWCR0 & (1<<TWINT))); /\* Wait until TWI finish its current job (start condition) \*/

check = TWSR0 & 0xF8; /\* Read TWI status register with masking lower three bits \*/

if (check != 0x08) /\* Check weather start condition transmitted successfully or not? \*/

continue; /\* If no then continue with start loop again \*/

TWDR0 = slave\_write\_address; /\* If yes then write SLA+W in TWI data register \*/

TWCR0 = (1<<TWEN)|(1<<TWINT); /\* Enable TWI and clear interrupt flag \*/

while (!(TWCR0 & (1<<TWINT))); /\* Wait until TWI finish its current job (Write operation) \*/

check = TWSR0 & 0xF8; /\* Read TWI status register with masking lower three bits \*/

if (check != 0x18 ) /\* Check weather SLA+W transmitted & ack received or not? \*/

{

i2cStop(); /\* If not then generate stop condition \*/

continue; /\* continue with start loop again \*/

}

break; /\* If yes then break loop \*/

}

}

void initializeUART()

{

DDRD |= 0x02;

UBRR0 = 103;

UCSR0C |= (0<<UPM01) | (0<<UPM00) | (1<<UCSZ01) | (1<<UCSZ00); //disable parity and set to asynchronous

}

void USART\_tx\_transmit()

{

UBRR0 = 103;

UCSR0B |= (1 << TXCIE0) | (1<<TXEN0); //enable data transmission

DDRD |= 0x02; //transmit through PD2

}

void USART\_tx(char\*data) //outputs data to terminal

{

while((\*data != '\0')) //loop until all data is emptied

{

while(!(UCSR0A & (1 << UDRE0))); //wait until data register in emptied

UDR0 = \*data; //once emptied, import into data register UDR0

data++; //increments pointer position for data

}

}

void gyroInit()

{

i2cIdle(SLAVE\_ADDRESS); //start condition at slave address

i2cWrite(DIVIDER\_ADDRESS); //write to sampling rate register

i2cWrite(0x07); //set sampling rate to 1kHz

i2cStop(); //stop condition

i2cIdle(SLAVE\_ADDRESS); //set power & start condition at slave address

i2cWrite(POWER\_ADDRESS); //write to that address

i2cWrite(0x01); //PLL with X axis gyroscope reference

i2cStop(); //stop condition

i2cIdle(SLAVE\_ADDRESS); //start condition at slave address

i2cWrite(CONFIG\_ADDRESS); //write to that address

i2cWrite(0x00); //set sampling rate to 8kHz

i2cStop(); //stop condition

i2cIdle(SLAVE\_ADDRESS); //start condition at slave address

i2cWrite(GYRO\_CONFIG\_ADDRESS); //write to that address

i2cWrite(0x18); //set range

i2cStop(); //stop condition

i2cIdle(SLAVE\_ADDRESS); //start condition at slave address

i2cWrite(INTERRUPT\_ADDRESS); //write to interrupt register address

i2cWrite(0x01); //toggle DATA\_RDY\_EN

i2cStop(); //stop condition

}

int main()

{

i2cInit(); //initializing functions

i2cStart();

USART\_tx\_transmit();

initializeUART();

gyroInit();

while(1)

{

Read\_RawValue(); //read in data

sendData(); //output data to UART

*\_delay\_ms*(250);

}

}

void Read\_RawValue()

{

i2cIdle(SLAVE\_ADDRESS); //start condition

i2cWrite(ACC\_START\_ADDRESS); //set acc address

i2cStart(); //read acc data

Acc\_x = (((int)i2cRead()<<8) | (int)i2cRead());

Acc\_y = (((int)i2cRead()<<8) | (int)i2cRead());

Acc\_z = (((int)i2cRead()<<8) | (int)i2cRead());

i2cStop(); //stop condition

i2cIdle(SLAVE\_ADDRESS); //start condition

i2cWrite(GYRO\_START\_ADDRESS); //set gyro address

i2cStart(); //read gyro data

i2cWrite(SLAVE\_ADDRESS);

Gyro\_x = (((int)i2cRead()<<8) | (int)i2cRead());

Gyro\_y = (((int)i2cRead()<<8) | (int)i2cRead());

Gyro\_z = (((int)i2cRead()<<8) | (int)i2cRead());

i2cStop(); //stop condition

}

void sendData()

{

char array[50]; //array data is input into

Ax = Acc\_x/16384.0; //scale collected value

*snprintf*(array,sizeof(array), "%f\r\n", Ax); //send to UART

USART\_tx(array);

Ay = Acc\_y/16384.0;

*snprintf*(array,sizeof(array), "%f\r\n", Ay);

USART\_tx(array);

Az = Acc\_z/16384.0;

*snprintf*(array,sizeof(array), "%f\r\n", Az);

USART\_tx(array);

Gx = Gyro\_x/16.4;

*snprintf*(array,sizeof(array), "%f\r\n", Gx);

USART\_tx(array);

Gy = Gyro\_y/16.4;

*snprintf*(array,sizeof(array), "%f\r\n", Gy);

USART\_tx(array);

Gz = Gyro\_z/16.4;

*snprintf*(array,sizeof(array), "%f\r\n", Gz);

USART\_tx(array);

}

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

//

//CPE301 - DA6A Task 1

//Tanner Tindall

//

#define BAUD 9600

#define *F\_CPU* 16000000UL

#include <avr/io.h>

#include <util/setbaud.h>

#include <stdlib.h>

#include <stdio.h>

#include <util/delay.h>

#define SLAVE\_ADDRESS 0xD0

#define DIVIDER\_ADDRESS 0x19

#define POWER\_ADDRESS 0x6B

#define CONFIG\_ADDRESS 0x1A

#define GYRO\_CONFIG\_ADDRESS 0x1B

#define INTERRUPT\_ADDRESS 0x38

#define ACC\_START\_ADDRESS 0x3B

#define GYRO\_START\_ADDRESS 0x43

#define UBBR\_VALUE 103

#define BITRATE 3

volatile *uint8\_t* data;

volatile *uint8\_t* value;

volatile float Ax,Ay,Az;

volatile float Gx=0, Gy=0, Gz=0;

float Acc\_x,Acc\_y,Acc\_z,Temperature,Gyro\_x,Gyro\_y,Gyro\_z;

volatile float pitch = 0;

volatile float roll = 0;

void i2cInit(void)

{

TWBR0 = BITRATE; //set bitrate

TWSR0 |= (0<<TWPS1)|(0<<TWPS0); //set prescaler to 1

}

unsigned char i2cRead(value)

{

if (value == 0) //ack bit & continue

{

TWCR0= (1<<TWINT) | (1<<TWEN) | (1<<TWEA); //enable TWI/ACK and clear flag

}

else

{

TWCR0= (1<<TWINT) | (1<<TWEN); //enable TWI & clear flag

}

while ((TWCR0& (1<<TWINT)) == 0); //wait until emptied

return TWDR0; //return value to TWDR

}

void i2cWrite(data)

{

TWDR0 = data; //pull data from TWI register

TWCR0 = (1<<TWINT) | (1<<TWEN); //enable TWI & clear flag

while ((TWCR0&(1<<TWINT)) == 0); //wait until done

}

void i2cStart(void)

{

TWCR0= (1<<TWINT) | (1<<TWSTA) | (1<<TWEN); //enable TWI, clear flag, start condition

while (TWCR0 & (1<<TWINT) == 0); //wait until emptied

}

void i2cStop(void)

{

TWCR0 = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO); //enable TWI, clear flag, stop condition

while ((TWCR0&(1<<TWINT)) == 0); //wait until done

}

void i2cIdle(char slave\_write\_address) //taken from Dr. Venkis I2C start wait function in Demo code

{

*uint8\_t* check; /\* Declare variable \*/

while (1)

{

TWCR0 = (1<<TWSTA)|(1<<TWEN)|(1<<TWINT); /\* Enable TWI, generate start condition and clear interrupt flag \*/

while (!(TWCR0 & (1<<TWINT))); /\* Wait until TWI finish its current job (start condition) \*/

check = TWSR0 & 0xF8; /\* Read TWI status register with masking lower three bits \*/

if (check != 0x08) /\* Check weather start condition transmitted successfully or not? \*/

continue; /\* If no then continue with start loop again \*/

TWDR0 = slave\_write\_address; /\* If yes then write SLA+W in TWI data register \*/

TWCR0 = (1<<TWEN)|(1<<TWINT); /\* Enable TWI and clear interrupt flag \*/

while (!(TWCR0 & (1<<TWINT))); /\* Wait until TWI finish its current job (Write operation) \*/

check = TWSR0 & 0xF8; /\* Read TWI status register with masking lower three bits \*/

if (check != 0x18 ) /\* Check weather SLA+W transmitted & ack received or not? \*/

{

i2cStop(); /\* If not then generate stop condition \*/

continue; /\* continue with start loop again \*/

}

break; /\* If yes then break loop \*/

}

}

void initializeUART()

{

DDRD |= 0x02;

UBRR0 = 103;

UCSR0C |= (0<<UPM01) | (0<<UPM00) | (1<<UCSZ01) | (1<<UCSZ00); //disable parity and set to asynchronous

}

void USART\_tx\_transmit()

{

UBRR0 = 103;

UCSR0B |= (1 << TXCIE0) | (1<<TXEN0); //enable data transmission

DDRD |= 0x02; //transmit through PD2

}

void USART\_tx(char\*data) //outputs data to terminal

{

while((\*data != '\0')) //loop until all data is emptied

{

while(!(UCSR0A & (1 << UDRE0))); //wait until data register in emptied

UDR0 = \*data; //once emptied, import into data register UDR0

data++; //increments pointer position for data

}

}

void gyroInit()

{

i2cIdle(SLAVE\_ADDRESS); //start condition at slave address

i2cWrite(DIVIDER\_ADDRESS); //write to sampling rate register

i2cWrite(0x07); //set sampling rate to 1kHz

i2cStop(); //stop condition

i2cIdle(SLAVE\_ADDRESS); //set power & start condition at slave address

i2cWrite(POWER\_ADDRESS); //write to that address

i2cWrite(0x01); //PLL with X axis gyroscope reference

i2cStop(); //stop condition

i2cIdle(SLAVE\_ADDRESS); //start condition at slave address

i2cWrite(CONFIG\_ADDRESS); //write to that address

i2cWrite(0x00); //set sampling rate to 8kHz

i2cStop(); //stop condition

i2cIdle(SLAVE\_ADDRESS); //start condition at slave address

i2cWrite(GYRO\_CONFIG\_ADDRESS); //write to that address

i2cWrite(0x18); //set range

i2cStop(); //stop condition

i2cIdle(SLAVE\_ADDRESS); //start condition at slave address

i2cWrite(INTERRUPT\_ADDRESS); //write to interrupt register address

i2cWrite(0x01); //toggle DATA\_RDY\_EN

i2cStop(); //stop condition

}

int main()

{

i2cInit(); //initializing functions

i2cStart();

USART\_tx\_transmit();

initializeUART();

gyroInit();

while(1)

{

Read\_RawValue(); //read in data

sendData(); //output data to UART

*\_delay\_ms*(250);

}

}

void Read\_RawValue()

{

i2cIdle(SLAVE\_ADDRESS); //start condition

i2cWrite(ACC\_START\_ADDRESS); //set acc address

i2cStart(); //read acc data

Acc\_x = (((int)i2cRead()<<8) | (int)i2cRead());

Acc\_y = (((int)i2cRead()<<8) | (int)i2cRead());

Acc\_z = (((int)i2cRead()<<8) | (int)i2cRead());

i2cStop(); //stop condition

i2cIdle(SLAVE\_ADDRESS); //start condition

i2cWrite(GYRO\_START\_ADDRESS); //set gyro address

i2cStart(); //read gyro data

i2cWrite(SLAVE\_ADDRESS);

Gyro\_x = (((int)i2cRead()<<8) | (int)i2cRead());

Gyro\_y = (((int)i2cRead()<<8) | (int)i2cRead());

Gyro\_z = (((int)i2cRead()<<8) | (int)i2cRead());

i2cStop(); //stop condition

}

void sendData()

{

char array[50]; //array data is input into

Ax = Acc\_x/16384.0; //scale collected value

*snprintf*(array,sizeof(array), "%f\r\n", Ax); //send to UART

USART\_tx(array);

Ay = Acc\_y/16384.0;

*snprintf*(array,sizeof(array), "%f\r\n", Ay);

USART\_tx(array);

Az = Acc\_z/16384.0;

*snprintf*(array,sizeof(array), "%f\r\n", Az);

USART\_tx(array);

Gx = Gyro\_x/16.4;

*snprintf*(array,sizeof(array), "%f\r\n", Gx);

USART\_tx(array);

Gy = Gyro\_y/16.4;

*snprintf*(array,sizeof(array), "%f\r\n", Gy);

USART\_tx(array);

Gz = Gyro\_z/16.4;

*snprintf*(array,sizeof(array), "%f\r\n", Gz);

USART\_tx(array);

}

void ComplementaryFilter()

{

float pitchAcc, rollAcc;

// Integrate the gyroscope data -> int(angularSpeed) = angle

pitch += ((float)Gy/16.4) \* 0.01;

// Angle around the X-axis

roll -= ((float)Gx/16.4) \* 0.01;

// Angle around the Y-axis

// Compensate for drift with accelerometer data if !bullshit

// Sensitivity = -2 to 2 G at 16Bit -> 2G = 32768 && 0.5G = 8192

int forceMagnitudeApprox = \

*abs*(Ax) + *abs*(Ay) + *abs*(Az);

if (forceMagnitudeApprox > 8192 && forceMagnitudeApprox < 32768)

{

// Turning around the X axis results in a vector on the Y-axis

pitchAcc = *atan2f*((float)Ax, (float)Ay) \* 180 / *M\_PI*;

pitch = pitch \* 0.98 + pitchAcc \* 0.02;

// Turning around the Y axis results in a vector on the X-axis

rollAcc = *atan2f*((float)Ax, (float)Ay) \* 180 / *M\_PI*;

roll = roll \* 0.98 + rollAcc \* 0.02;

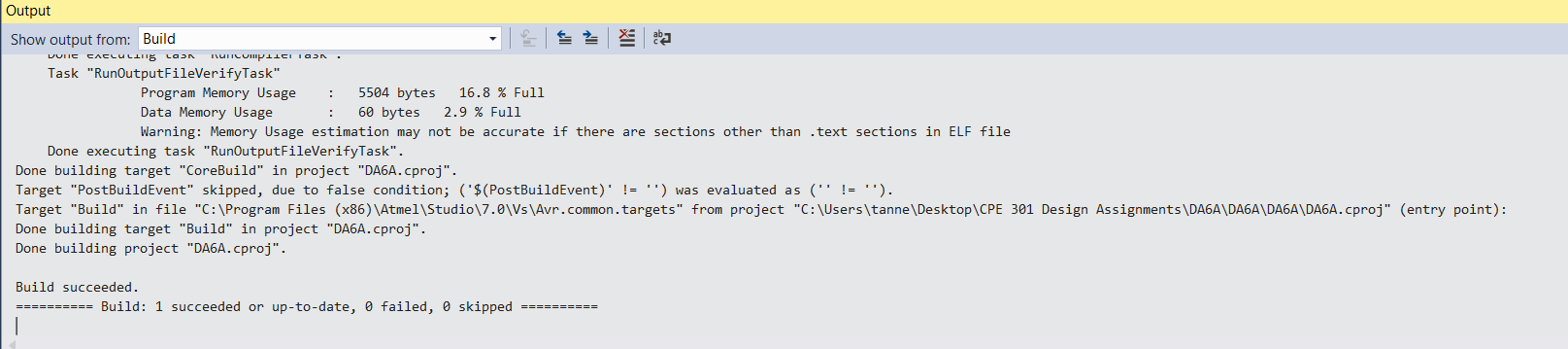
}

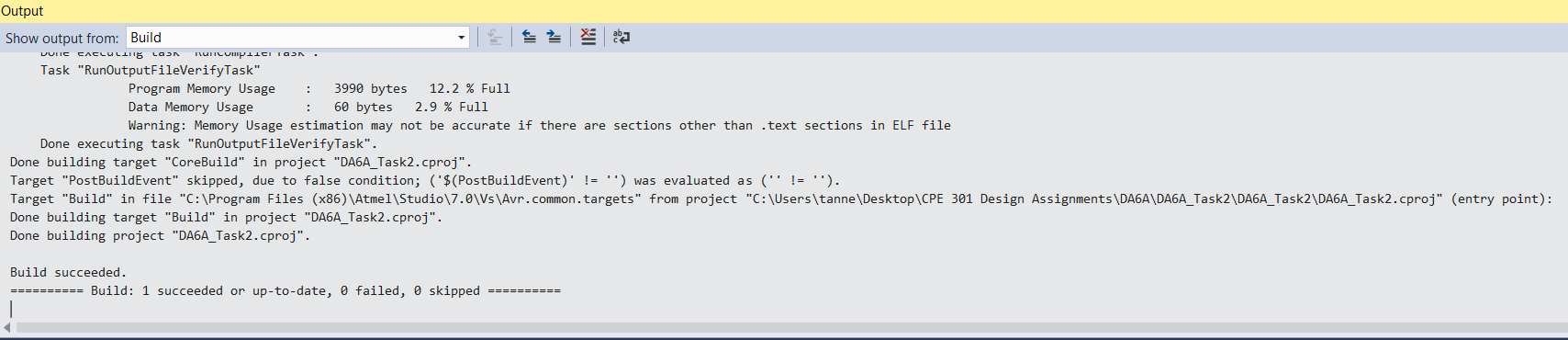
}

1. **SCHEMATICS**

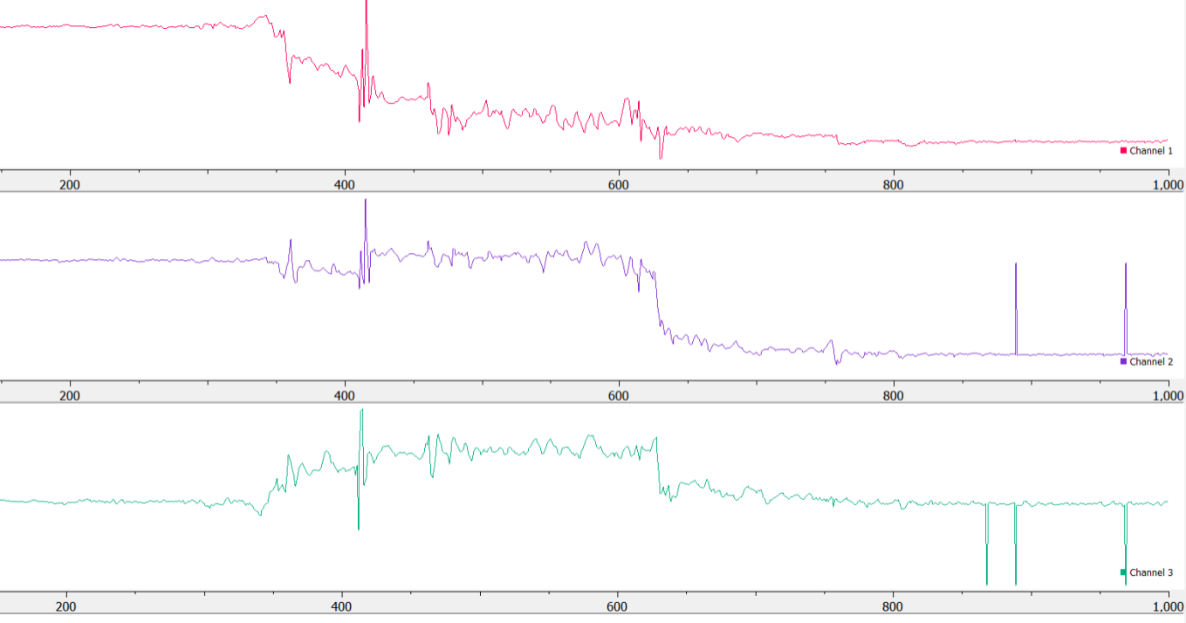
N/A

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

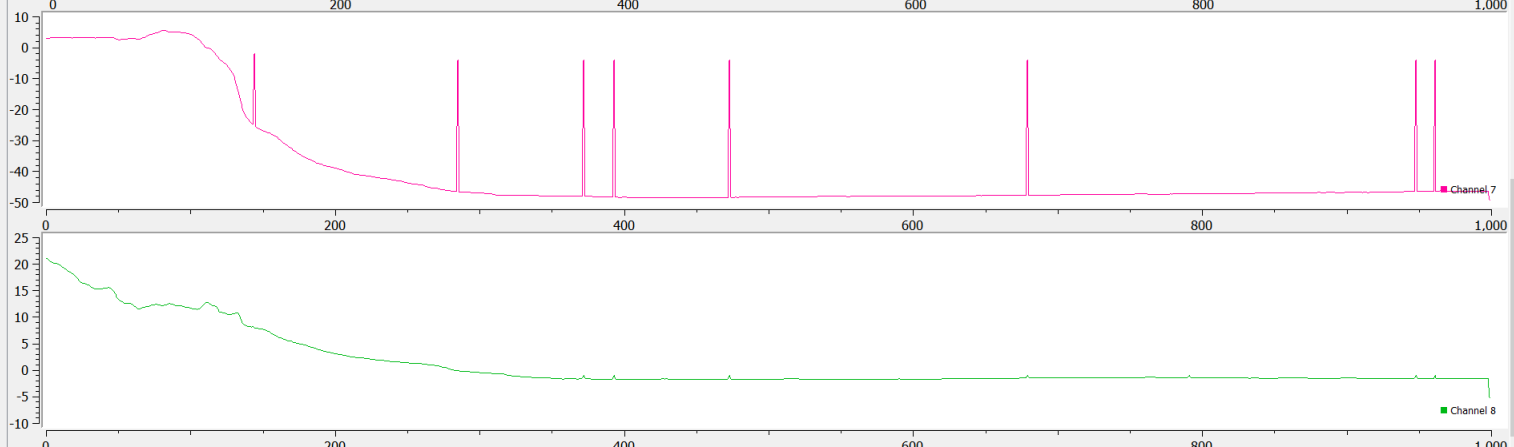
Task 1:

Task 2:

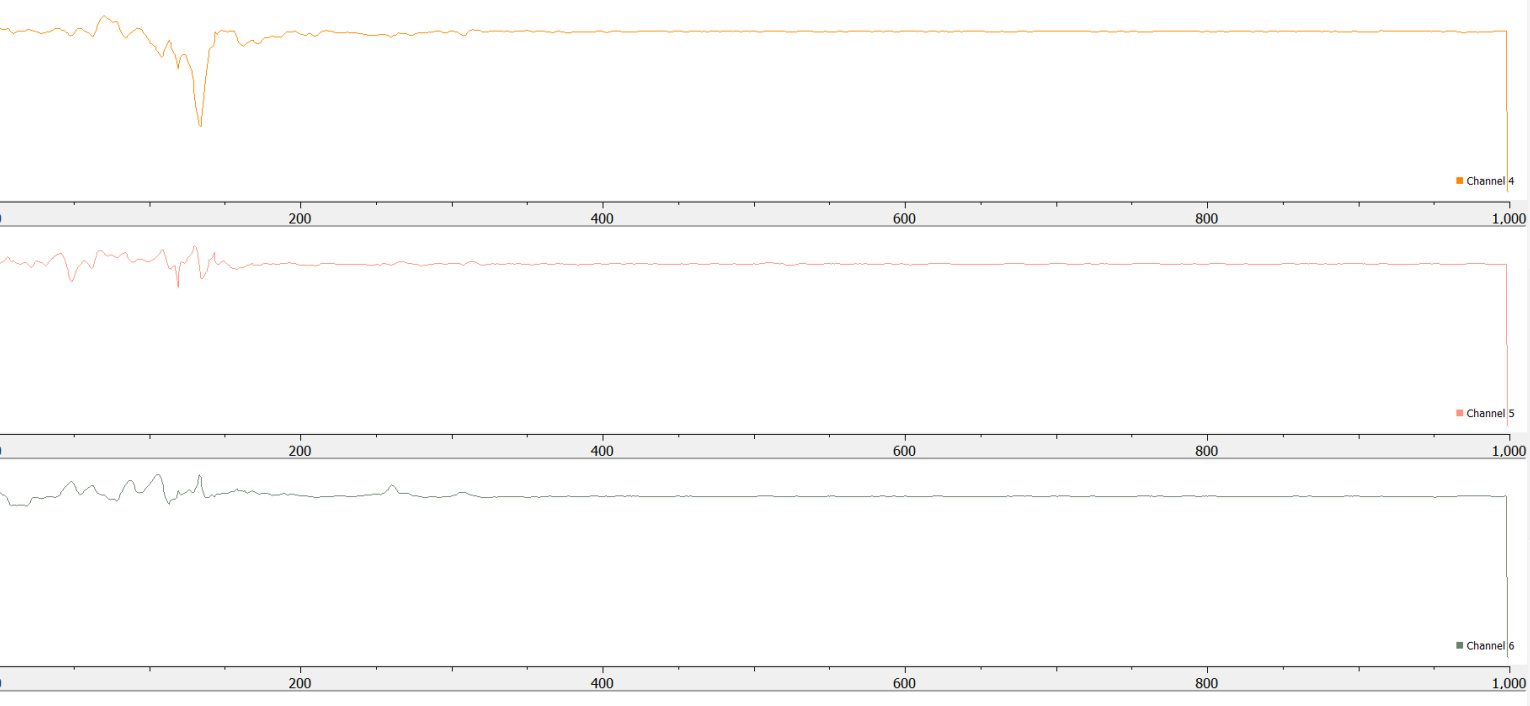
Accelerometer XYZ



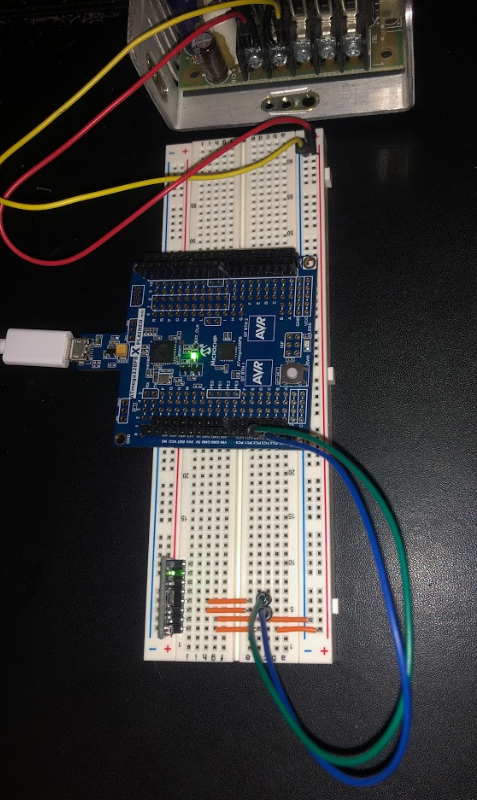
Pitch/Roll:



Degrees/sec:



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



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

<https://github.com/TannerTindall51/tindalltannerm_submission/upload/master/Design_Assignments/DA6A>

**Student Academic Misconduct Policy**

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

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

Tanner Tindall