Audie Escala

CPE301 – SPRING 2018

Final Project

**DO NOT REMOVE THIS PAGE DURING SUBMISSION:**

The student understands that all required components should be submitted in complete for grading of this assignment.

|  |  |  |  |
| --- | --- | --- | --- |
| **NO** | **SUBMISSION ITEM** | **COMPLETED (Y/N)** | **MARKS**  **(/MAX)** |
| 1 | COMPONENTS LIST AND CONNECTION BLOCK DIAGRAM w/ PINS |  |  |
| 2. | INFORMATION OF PROJECT |  |  |
| 3. | INITIAL CODE IN C OF MAIN CODE |  |  |
| 4. | INITIAL CODE IN MATLAB OF LORENZ EQUATION |  |  |
| 5. | FLOWCHART OF MAIN |  |  |
| 6. | SCREENSHOT OF OUTPUT FROM OSCILLOSCOPE AND MATLAB |  |  |
| 7. | SCREENSHOTS OF EACH MAIN ON BREADBOARD |  |  |
| 8. | CONCLUSION |  |  |
| 9. | YOUTUBE/GOOGLECODE LINK OF THE FINAL PROJECT |  |  |
|  |  |  |  |
|  |  |  |  |

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

ATmega328P Xplained Mini

MCP4725

1. **INFORMATION OF PROJECT**

The goal of this final project was to implement a model of a dynamic system on the AVR microcontroller. The system model I choose for this problem is the Chua Circuit Chaos System. To solve these equations, I used the Runge-Katta 4th order algorithm to solve the nonlinear ordinary differential equations of the Chua Circuit. Used the final calculated values of one point and then got the ATmega328p to send voltage values to the MCP4725 and see the output on an oscilloscope. The program on the ATmega328p communicates with the MCP4725 through i2c. I am only sending data to the MCP4725 and not receiving anything back. The MC4725 is a DAC (Digital to Analog Converter), which is a digital to analog converter.

1. **DEVELOPED CODE IN C OF MAIN**

#define *F\_CPU* 8000000UL

#define FOSC 16000000 // Clock Speed

#define BAUD 9600

#define MYUBRR FOSC/16/BAUD-1

#include <avr/io.h> //standard AVR header

#include <stdint.h> // need for uint8\_t

#include <util/delay.h> //delay header

#include <avr/interrupt.h>

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <util/twi.h>

// i2c Functions and Paramters

#define SLA\_W 0xC0 // write address of MCP4725

#define F\_SCL 100000UL // SCL frequency

#define Prescaler 1

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

#define TWCR\_CMD\_MASK 0x0F

#define MCP\_CTRL 0x00

void i2cSendStart(void); // start i2c

void i2cSendStop(void); // stop i2c

void i2cWaitForComplete(void); // wait of ACK

void i2cSendByte(unsigned char data); // change this based on your i2c lib

void i2cRecieveByte(unsigned char ackFlag); //

void i2c\_init (void); // initialize i2c

// Initialize arrays

float x[] = {0.7,0,0}; // hold the final values to be the output X Y Z

float temp[2]; // temporary holding array

float K1[3]; // holding array for Runge-Kutta Step 1

float K2[] ={0,0,0}; // holding array for Runge-Kutta Step 2

float K3[]={0,0,0}; // holding array for Runge-Kutta Step 3

float K4[]={0,0,0}; // holding array for Runge-Kutta Step 4

float final[3];

signed short finalX[100];

signed short finalY[100];

signed short finalZ[100];

// Constant Global Variables for Chua Circuit

float chua(int equation, float t,float y[]);

volatile float times = 0;

const float alpha = 15.6;

const float beta = 28;

const float m0 = -1.143;

const float m1 = -0.714;

const float dt = 0.01; // time step

static int uart\_putchar(char c, *FILE* \*stream)

{

if (c == '\n') uart\_putchar('\r', stream);

loop\_until\_bit\_is\_set(UCSR0A, UDRE0);

UDR0 = c;

return 0;

}

static *FILE* mystdout = *FDEV\_SETUP\_STREAM*(uart\_putchar, *NULL*, *\_FDEV\_SETUP\_WRITE*);

void USARTinit (void)

{

/\* set baud rate \*/

UBRR0H = (MYUBRR>>8); //high value of baud rate

UBRR0L = MYUBRR; // low value of baud rate

UCSR0B |= (1 << RXEN0) | (1<<TXEN0); //enable receiver and transmitter

UCSR0B |= (1 << RXCIE0); // enable receiver input

UCSR0C = ((1<<UCSZ01)|(1<<UCSZ00)); //asynchronous

*stdout* = &mystdout;

}

void USARTsend(unsigned char Data) // function for sending data to the stream

{

while (!(UCSR0A & (1<<UDRE0)));

UDR0=Data;

}

volatile int count = 0;

int main(void)

{

i2c\_init();

USARTinit();

while(times < 1){ // can only take in a sample size of 100 since running out of memory

*printf*("Time = %f\n",times);

// Runge-Kutta Solving the ODE

// First Step

for(int i = 0; i < 3; i++)

{

K1[i] = chua(i,times,x);

temp[i] = x[i] + (K1[i]/2)\*dt;

}

// Second Step

for(int j = 0; j < 3; j++)

{

K2[j] = chua(j,times,temp);

temp[j] = x[j] + (K2[j]/2)\*dt;

}

for(int j = 0; j < 3; j++)

{

K3[j] = chua(j,times+dt/2,temp);

temp[j] = x[j] + (K3[j]/2)\*dt;

}

for(int j = 0; j < 3; j++)

{

K4[j] = chua(j,times+dt,temp);

}

// final calculated values of X Y Z

for(int j = 0; j < 3; j++){

x[j]= x[j]+dt\*((K1[j]/6)+(K2[j]/3)+(K3[j]/3)+(K4[j]/6));

}

// Set the offset of each graph to be visible on MCP7425

// Because the MCP4725 only accepts positive values, tried setting the highest negative value to 0

// it's not pretty since it's plotting over time

finalX[count]= (int)(x[0]+2.5\*100);

finalY[count]= (int)(x[1]+1\*100);

finalZ[count]= (int)(x[2]+2\*100);

times = times + dt; // next times value;

count = count + 1;

}

int p;

unsigned short control;

for (p = 0; p < 100; p++){

//Send start condition

i2cSendStart();

i2cWaitForComplete();

// send slave device address with write

i2cSendByte(SLA\_W);

i2cWaitForComplete();

// send first 4-bits of chua value

control = finalX[p];

control= (control >> 2);

// send the rest of the 8-bits of the chua value

i2cSendByte(control);

i2cWaitForComplete();

i2cSendByte(finalX[p]);

i2cWaitForComplete();

// send stop condition

i2cSendStop();

}

// Check if the values sent to MCP4725 was correct

int sCount = 0;

*printf*("Final values of X: \n");

for(int i = 0; i< 100; i++){

*printf*("%d ",finalX[i]);

if(sCount == 10){

*printf*("\n");

sCount = 0;

}

}

*printf*("\nFinal values of Y: \n");

for(int i = 0; i< 100; i++){

*printf*("%d ",finalY[i]);

if(sCount == 10){

*printf*("\n");

sCount = 0;

}

}

*printf*("\nFinal values of Z: \n");

for(int i = 0; i< 100; i++){

*printf*("%d ",finalZ[i]);

if(sCount == 10){

*printf*("\n");

sCount = 0;

}

}

return 0;

}

// The nonlinear ODE Chua

float chua(int equation, float t, float y[])

{

float h = m1\*y[0] + 0.5\*(m0-m1)\*(*abs*(y[0]+1))-(*abs*(y[0]-1));

// dx/dt = alpha\*(y - x - h)

if(equation == 0){

return (alpha\*(y[1]-y[0]-h));

}

// dy/dt = x - y + z

else if(equation == 1)

{

return (y[0]-y[1]+y[2]);

}

// dz/dt = - beta \* y

else

{

return (-1\*beta\*y[1]);

}

}

void i2cSendStart(void)

{

// send start condition

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

}

void i2cSendStop(void)

{

// transmit stop condition

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

}

void i2cWaitForComplete(void)

{

// wait for i2c interface to complete operation

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

}

void i2cSendByte(unsigned char data)

{

*printf*("This is the data: %d",data );

USARTsend('\n');

// save data to the TWDR

TWDR = data;

// begin send

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

}

void i2cRecieveByte(unsigned char ackFlag)

{

// begin receive over i2c

if(ackFlag)

{

// ackFlag = TRUE: ACK the received data

outb(TWCR, (inb(TWCR)&TWCR\_CMD\_MASK)|BV(TWINT)|BV(TWEA));

}

else

{

// ackFlag = FALSE: NACK the received data

outb(TWCR, (inb(TWCR)&TWCR\_CMD\_MASK)|BV(TWINT));

}

}

unsigned char i2cGetStatus(void)

{

// retrieve current i2c status from i2c TWSR

return (inb(TWSR));

}

void i2c\_init(void)

{

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

TWCR = (1<<TWEN); // ENABLE I2C

TWSR = 0x00; // Prescaler set to 1

}

1. **DEVELOPED CODE IN MATLAB OF CHUA EQUATION**

function out = chua(t,in)

x = in(1);

y = in(2);

z = in(3);

alpha = 15.6;

beta = 28;

m0 = -1.143;

m1 = -0.714;

h = m1\*x+0.5\*(m0-m1)\*(abs(x+1)-abs(x-1));

xdot = alpha\*(y-x-h);

ydot = x - y+ z;

zdot = -beta\*y;

out = [xdot ydot zdot]';

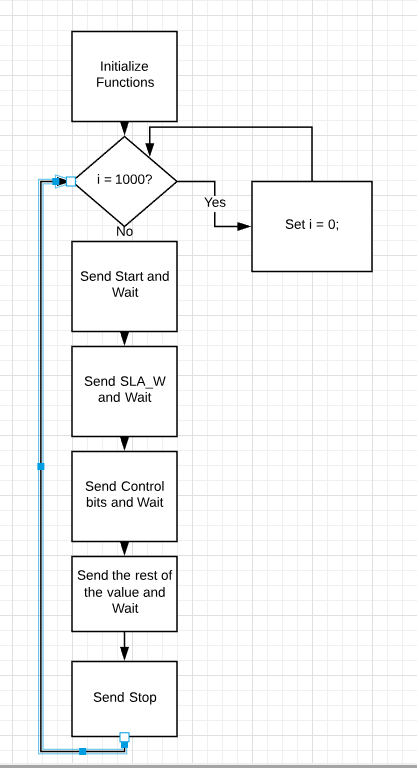
% run the program

[t,y] = ode45(@chua,[0 100],[0.7 0 0]);

plot3(y(:,1),y(:,2),y(:,3))

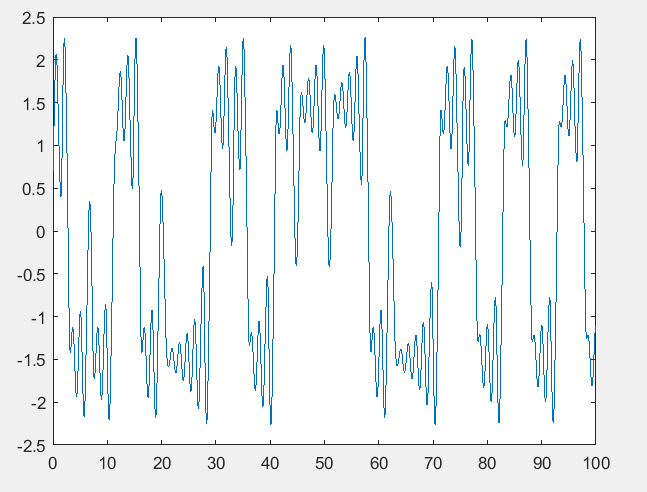
grid

1. **FLOWCHART OF MAIN**

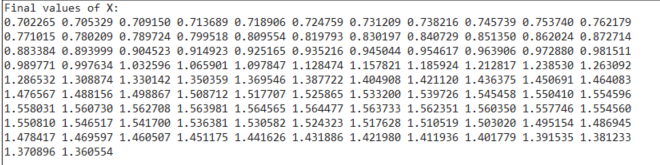


1. **SCREENSHOT OF OUTPUT FROM OSCILLOSCOPE AND MATLAB**

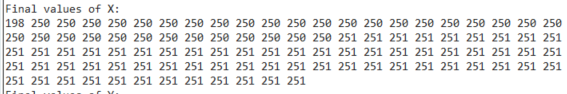
Plot of X axis values



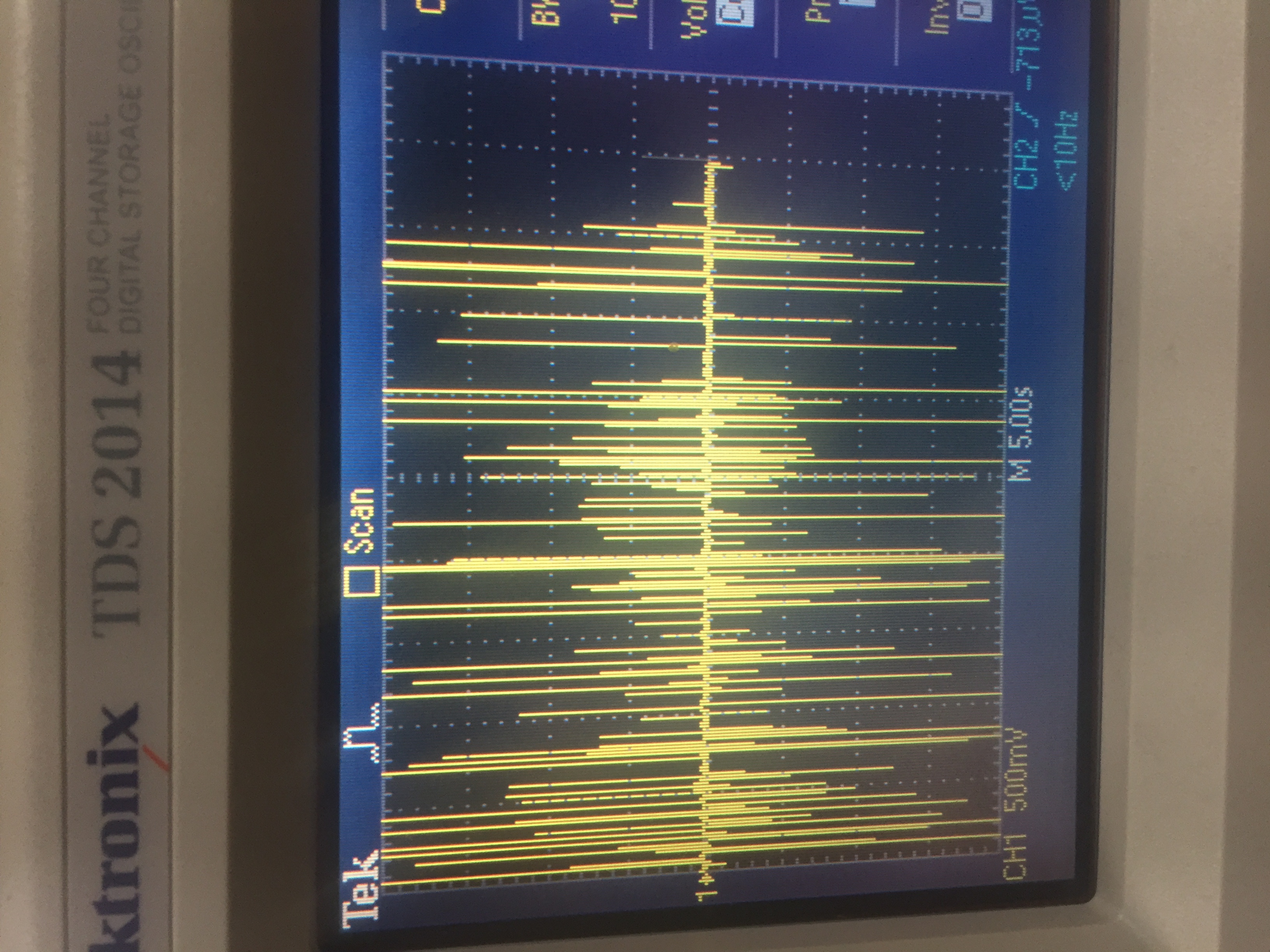
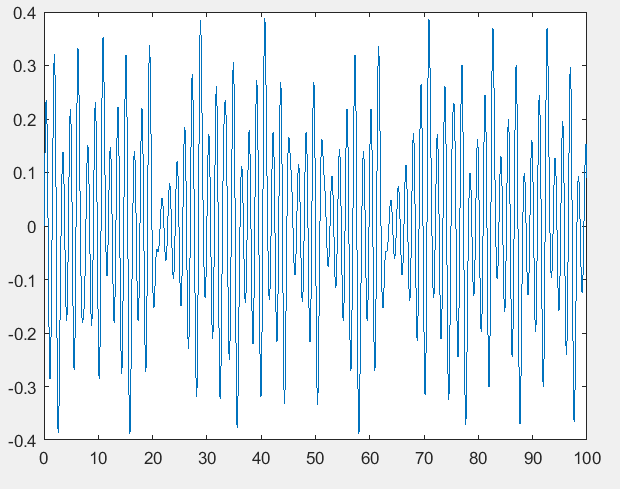
* Sample values in float



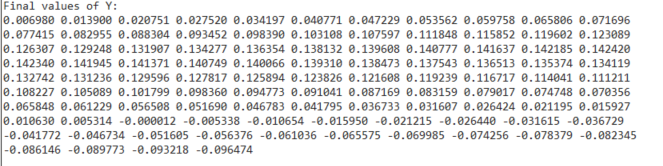
* Sample values with Conversion



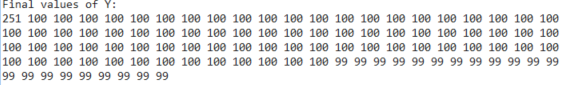
Plot of Y axis values



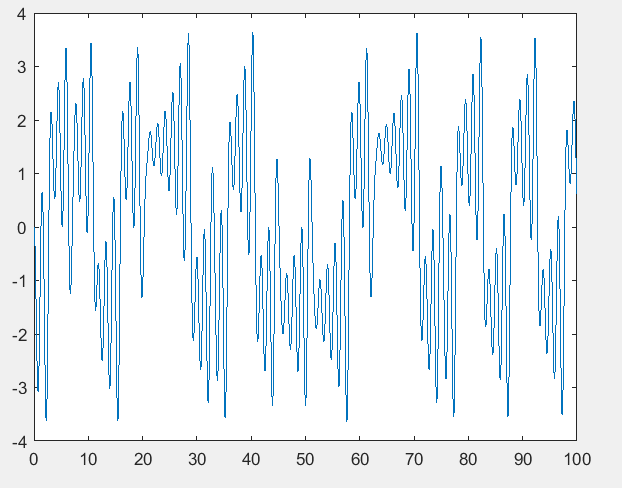
* Sample values in float



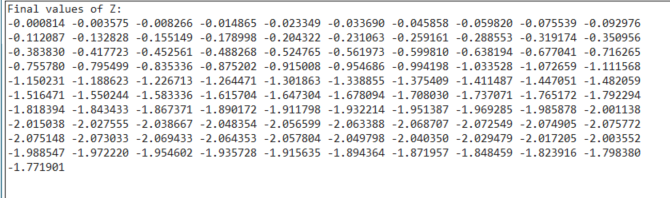
* Sample values with Conversion



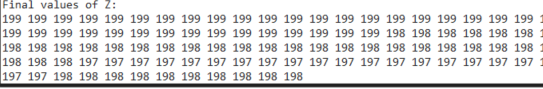
Plot of Z



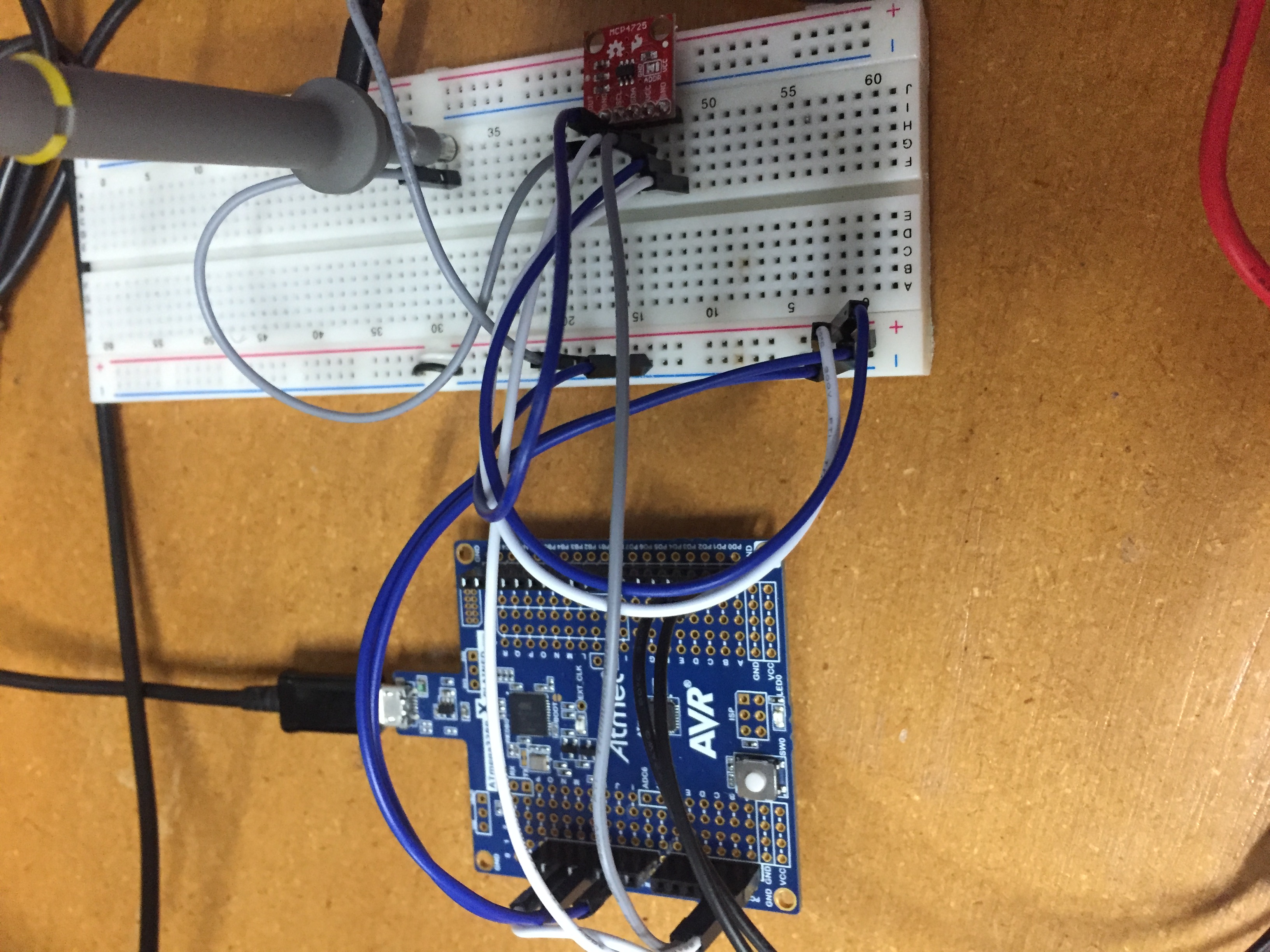
* Sample values in float



* Sample values with Conversion



1. **SCREENSHOT OF PROJECT ON BREADBOARD**



1. **Conclusion of Project**

Had a tough time implementing the project to the MC4725 because the values that can be sent to the MCP4725 ranged from 0-4095. This means that only positive numbers are accepted into the MC4725. Since the Chua equation deals with both positive and negative numbers, I had to set an offset of +2.5 for the X values and increase the scale to by 100 so it would reach clear value range from 1 to 5 V on the Oscilloscope. Other than that, the MCP4725 was difficult to debug through because sometimes the i2c commands wouldn’t send correctly and the MC4725 would never get the correct 12-bits.

1. **YOUTUBE/GITHUB LINK OF FINAL PROJECT AND RESOURCES**

<https://www.youtube.com/watch?v=o5EBwifCutI>

<https://github.com/escalaa/Final-Project>

<http://www.chuacircuits.com/matlabsim.php>

**Student Academic Misconduct Policy**

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

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

Audie Escala