CPE301 – SPRING 2022

Design Assignment 3

Student Name: Ernesto Ibarra-Ayala

Student #: 2001211571

Student Email: ibarre3@unlv.nevada.edu

Primary Github address: https://github.com/Ernesto-Ibarra/Work.git

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

Atmel Studio 7.0 Atmega328PB-Xmini Multi-Function Shield Logic Analyzer

- Assembler - Switches

- Simulator - LEDs

- Debugger

Diagram

Description automatically generated

1. **INITIAL/MODIFIED/DEVELOPED CODE OF TASK 1/2/3**

|  |
| --- |
| /\* |
|  | \* ADC2\_example.c |
|  | \* |
|  | \* Created: 10/10/2019 10:24:06 PM |
|  | \* Author : VenkatesanMuthukumar |
|  | \*/ |
|  |  |
|  | #include <avr/io.h> |
|  | #include <stdlib.h> |
|  | #define F\_CPU 16000000UL |
|  | #include <util/delay.h> |
|  | #define BAUDRATE 9600 |
|  | #define BAUD\_PRESCALLER (((F\_CPU / (BAUDRATE \* 16UL))) - 1) |
|  |  |
|  | uint16\_t adc\_value; //Variable used to store the value read from the ADC |
|  | char buffer[5]; //Output of the itoa function |
|  | uint8\_t i=0; //Variable for the for() loop |
|  |  |
|  | void adc\_init(void); //Function to initialize/configure the ADC |
|  | uint16\_t read\_adc(uint8\_t channel); //Function to read an arbitrary analogic channel/pin |
|  | void USART\_init(void); //Function to initialize and configure the USART/serial |
|  | void USART\_send( unsigned char data); //Function that sends a char over the serial port |
|  | void USART\_putstring(char\* StringPtr); //Function that sends a string over the serial port |
|  |  |
|  | int main(void){ |
|  | adc\_init(); //Setup the ADC |
|  | USART\_init(); //Setup the USART |
|  |  |
|  | for(;;){ //Our infinite loop |
|  | /\* for(i=0; i<3; i++){ |
|  | //USART\_putstring("Reading channel "); |
|  | //USART\_send('0'); //This is a nifty trick when we only want to send a number between 0 and 9 |
|  | //USART\_putstring(" : "); //Just to keep things pretty |
|  | adc\_value = read\_adc(i); //Read one ADC channel |
|  | itoa(adc\_value, buffer, 10); //Convert the read value to an ascii string |
|  | USART\_putstring(buffer); //Send the converted value to the terminal |
|  | USART\_putstring(","); //Some more formatting |
|  | \_delay\_ms(500); //You can tweak this value to have slower or faster readings or for max speed remove this line |
|  | \*/ |
|  | adc\_value = read\_adc(4); //Read one ADC channel |
|  | itoa(adc\_value, buffer, 10); //Convert the read value to an ascii string |
|  | USART\_putstring(buffer); //Send the converted value to the terminal |
|  | USART\_putstring(","); //Some more formatting |
|  | adc\_value = read\_adc(5); //Read one ADC channel |
|  | itoa(adc\_value, buffer, 10); //Convert the read value to an ascii string |
|  | USART\_putstring(buffer); //Send the converted value to the terminal |
|  | USART\_send('\n'); //Some more formatting |
|  | //} |
|  | //USART\_send('\r'); |
|  | //USART\_send('\n'); //This two lines are to tell to the terminal to change line |
|  | \*/ |
|  |  |
|  | } |
|  |  |
|  | return 0; |
|  | } |
|  |  |
|  | void adc\_init(void){ |
|  | ADCSRA |= ((1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)); //16Mhz/128 = 125Khz the ADC reference clock |
|  | ADMUX |= (1<<REFS0); //Voltage reference from Avcc (5v) |
|  | ADCSRA |= (1<<ADEN); //Turn on ADC |
|  | ADCSRA |= (1<<ADSC); //Do an initial conversion because this one is the slowest and to ensure that everything is up and running |
|  | } |
|  |  |
|  | uint16\_t read\_adc(uint8\_t channel){ |
|  | ADMUX &= 0xF0; //Clear the older channel that was read |
|  | ADMUX |= channel; //Defines the new ADC channel to be read |
|  | ADCSRA |= (1<<ADSC); //Starts a new conversion |
|  | while(ADCSRA & (1<<ADSC)); //Wait until the conversion is done |
|  | return ADCW; //Returns the ADC value of the chosen channel |
|  | } |
|  |  |
|  | void USART\_init(void){ |
|  |  |
|  | UBRR0H = (uint8\_t)(BAUD\_PRESCALLER>>8); |
|  | UBRR0L = (uint8\_t)(BAUD\_PRESCALLER); |
|  | UCSR0B = (1<<RXEN0)|(1<<TXEN0); |
|  | UCSR0C = (3<<UCSZ00); |
|  | } |
|  |  |
|  | void USART\_send( unsigned char data){ |
|  |  |
|  | while(!(UCSR0A & (1<<UDRE0))); |
|  | UDR0 = data; |
|  |  |
|  | } |
|  |  |
|  | void USART\_putstring(char\* StringPtr){ |
|  |  |
|  | while(\*StringPtr != 0x00){ |
|  | USART\_send(\*StringPtr); |
|  | StringPtr++;} |
|  |  |
|  | } |

1. **DEVELOPED MODIFIED CODE OF TASK 1/2/3**

#define *F\_CPU* 16000000UL //Set clock frequency

#define BAUDRATE 9600 //Set the baud rate

#define BAUD\_PRESCALLER (((*F\_CPU* / (BAUDRATE \* 16UL))) - 1) //Prescalar for the baud rate

#include <avr/io.h>

#include <stdlib.h>

#include <util/delay.h>

#include <avr/interrupt.h>

*uint16\_t* ADCX; //ADC X-axis

*uint16\_t* ADCY; //ADC Y-axis

char buffer[5]; //String for output of string function

float XF = 0.0; //x-axis value to frequency for timer

float YDC = 0.0; //y-axis value to duty cycle for timer

float percent = 0.0;// Stores value of duty cycle to convert to a percentage

float Hzz = 0.0;// variable used to display frequency

void adc\_init(void); //Function to initialize/configure the ADC

*uint16\_t* read\_adc(*uint8\_t* channel); //Function to read the analog input

void USART\_init(void); //Function to initialize and configure the USART/serial

void USART\_send( unsigned char data); //Function that sends a char over the serial port

void USART\_putstring(char\* StringPtr); //Function that sends a string over the serial port

void adc\_convert(void); //Main function used to read the ADC values and use our timbers with those values

void timer\_init(void); //Function to initialize the CTC timer1

void timer\_update(void); //Function to update the timer values when new ADC values come in

int main(void){

DDRC &= (0 << 4) | (0 << 5); //Set PC4 and PC5 as input

DDRB |= (1 << 5); //set PB5 as output

PORTB &= (0 << 5);

adc\_init(); //Start ADC

USART\_init(); //Start the USART

timer\_init(); //Start Timer1

while(1)

{

adc\_convert(); //Grab ADC values and process them

timer\_update(); //Update timer

}

return 0;

}

void adc\_convert(void)

{

// Here is part 2 of the HW here we will display the RAW values of the ADC

ADCX = read\_adc(4); //Read ADC channel 4

USART\_putstring("RAW FREQ: ");

*itoa*(ADCX, buffer, 10);

USART\_putstring(buffer);

USART\_putstring(",");

ADCY = read\_adc(5); //Read ADC channel 5

USART\_putstring("RAW DUTY: ");

*itoa*(ADCY, buffer, 10);

USART\_putstring(buffer);

USART\_send('\n');

//Here we take care of part 3 of the assignment by using this IF statements to create the correct Frequency and Duty Cycle

if (ADCX >= 512)

{

XF = 260 + (1023-ADCX)/2; //If valX > 512 it will use this formula to scale down

}

else

{

XF = 260+(512-ADCX)\*15; //If valX < 512 it will use this formula to scale up

}

YDC = XF \* (1.0 \* ADCY / 1000.0); //This formula will convert valY into a percentage to get the duty cycle

if(YDC >= XF)

{

YDC = XF;

}

percent = ADCY / 10; //This will give us the duty cycle from 0 - 100

//Hzz = XF

USART\_putstring("Freq: ");

*itoa*(XF, buffer, 10);

USART\_putstring(buffer);

USART\_putstring(",");

USART\_putstring("DUTY: ");

*itoa*(percent, buffer, 10);

USART\_putstring(buffer);

USART\_putstring("%");

USART\_send('\n');

*\_delay\_ms*(500);

}

void adc\_init(void)

{

ADCSRA |= ((1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)); //16Mhz/128 = 125Khz the ADC reference clock

ADMUX |= (1<<REFS0); //Voltage reference from Avcc (5v)

ADCSRA |= (1<<ADEN); //Turn on ADC

ADCSRA |= (1<<ADSC); //Do an initial conversion because this one is the slowest and to ensure that everything is up and running

}

*uint16\_t* read\_adc(*uint8\_t* channel)

{

ADMUX &= 0xF0; //Clear the older channel that was read

ADMUX |= channel; //Defines the new ADC channel to be read

ADCSRA |= (1<<ADSC); //Starts a new conversion

while(ADCSRA & (1<<ADSC)); //Wait until the conversion is done

return ADCW; //Returns the ADC value of the chosen channel

}

void timer\_init(void)

{

TCCR1B |= (1 << WGM12) | (1 << CS12) | (1 << CS10); //Sets prescalar to 1024

TIMSK1 |= (1 << OCIE1A) | (1 << OCIE1B); //Enable OCR1A and OCR1B

TCNT1 = 0; // Start timer at 0

sei(); //Activate global interrupts

}

void timer\_update(void)

{

OCR1A = XF; //Store new frequency value

OCR1B = YDC; //Store new duty cycle value

}

ISR(TIMER1\_COMPB\_vect) //Interrupt for duty cycle

{

PORTB |= (1 << 5); //turn on the LED

}

ISR(TIMER1\_COMPA\_vect) //Interrupt for frequency

{

PORTB &= (0 << 5); // turn off LED

TCNT1 = 0; //Reset to zero

}

void USART\_init(void)//Function to initialize the USART

{

UBRR0H = (*uint8\_t*)(BAUD\_PRESCALLER>>8);

UBRR0L = (*uint8\_t*)(BAUD\_PRESCALLER);

UCSR0B = (1<<RXEN0)|(1<<TXEN0);

UCSR0C = (3<<UCSZ00);

}

void USART\_send( unsigned char data)//Function to send data using USART

{

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

UDR0 = data;

}

void USART\_putstring(char\* StringPtr)//Function to turn send a string on USART

{

while(\*StringPtr != 0x00){

USART\_send(\*StringPtr);

StringPtr++;}

}

1. **SCHEMATICS**

Diagram, schematic

Description automatically generated

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

Here is an output of my code which will take care of all three parts of the assignment. Below we can see the Raw ADC Values right above the frequency and duty cycle values. I added a small delay of 500ms to allow me to be able to screenshot the results.

Graphical user interface, text, application, email

Description automatically generated

Here is me using the data analyzer to get the correct values for the Joystick. This is joystick being in the center so we can see the Duty cycle is 50% and the Frequency is 30Hz which is what we wanted to display.

Text

Description automatically generated

Here we can see what happens when we move the Joystick all the way to one extreme, we get the duty cycle being 100% while the frequency remains unchanged at 30Hz.

A picture containing text, indoor, screen, screenshot

Description automatically generated

Here we can see what happens when we move the Joystick to the opposite end, we can basically turn off the LED.

Graphical user interface

Description automatically generated

Here is me changing the Frequency to one extreme causing it to go to zero or at least very close to zero which is basically 1Hz while the Duty cycle remains at 50% as you can see below.

Graphical user interface

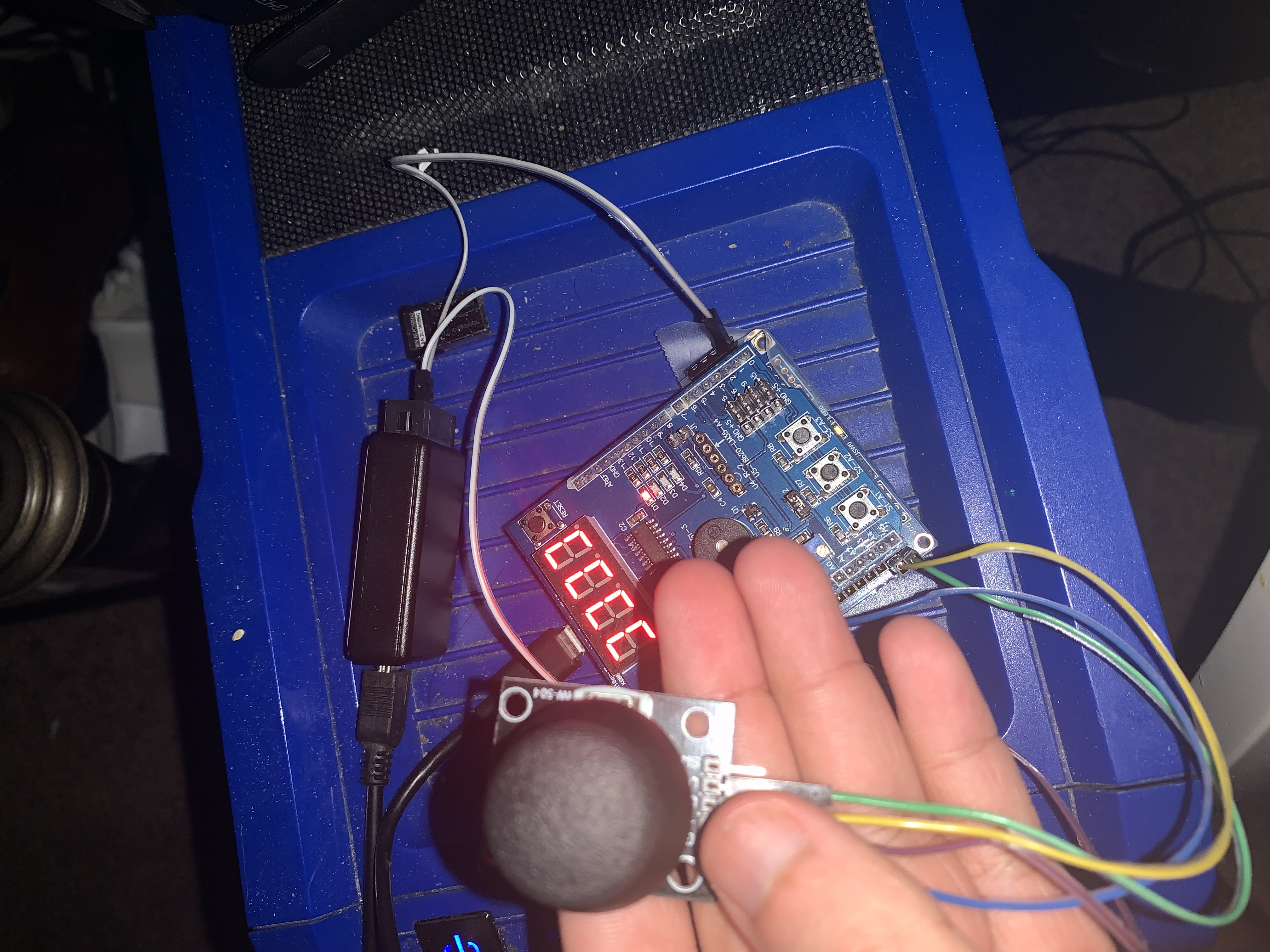
Description automatically generated with medium confidence

Here we can see changing the Frequency to the other extreme causing it to have a frequency of 60Hz as we wanted on the assignment, while the Duty cycle remains at 50% as stated. So we are basically able to see all 4 quadrants of the Joystick.

Graphical user interface

Description automatically generated

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



1. **VIDEO LINKS OF EACH DEMO**

**DA3 Part 2:** <https://youtube.com/shorts/j1wdf_MSlik?feature=share>

**DA3 Part 3:** <https://youtu.be/NsZ8_ZAMUuU>

1. **GITHUB LINK OF THIS DA**

https://github.com/Ernesto-Ibarra/Work/tree/main/DesignAssignments

**Student Academic Misconduct Policy**

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

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

Ernesto Ibarra