**ELEC 291 Section 20C**

**ELEC 291 Section 20C**

**LAB #: 6**

**LAB SECTION: 20D**

**TEAM #: D\_5C**

|  |  |  |
| --- | --- | --- |
| ***Student name*** | ***Student number*** | ***Contribution percentage*** |
| **LINDA MUNISI** | **22535158** | **33.3%** |
| **SIRINE TRIGUI** | **46443157** | **33.3%** |
| **YIXIN ZHAO** | **49427140** | **33.3%** |

*Contribution summary*:

We all worked together for this lab trying to figure out the AVR concept. We first took care of the simple Arduino then tried to modify it. In addition, everybody contributed in the processing part to make our oscilloscope as better as possible.

**A. Introduction and motivations**

The main aim of this lab was to use external interrupts and timers with Arduino. In this project we implement a simple oscilloscope. This oscilloscope is a simple one as it has limited allowed maximum input voltage and frequency, but it would serve well as the basis for any such application. As before, we will use Processing to graph the signal.

The motivation for this lab was to understand how timers and interrupts work in Arduino. Another motivation is to create an oscilloscope that looks as close as possible to the Tektronix Oscilloscope in the lab.

**B. Lab Description Summary**

**Components and circuitry:**

The components used in this lab are Arduino, USB cable, proto-shield + mini breadboard, push button, resistor, function generator and Oscilloscope.

To connect the circuit, we have one end of the switch connected to 5V while the other one is connected to the digital pin, and to ground through a resistor. Then we set the voltage on the probes to make sure that it does not exceed the maximum voltage allowed by Arduino that is 0-5V. To do this, we plugged in two wires in arbitrary places on the breadboard. Then we connected them to the function generator and the oscilloscope. We adjusted the frequency starting with a low frequency increasing it gradually. We then set the wave to a sine wave in the function generator and measured the peak to peak voltage, and adjusted it gradually using the offset and the amplitude button. After this, we set the peak to peak voltage of the channel we were using to approximately 4.0V which is smaller than the maximum voltage the Arduino can take (5v). Then we connected the positive probe of the function generator to analog pin 0, the negative one to the ground in order to read the data. We get our data displayed in the actual oscilloscope

**Implementation Procedures:**

**1) Using External Interrupts**

Having connected the circuit already, we started with testing a simple function that works with interrupts to see how it works. Here, we are using pin 3 which is an interrupt pin in Arduino. (We could have also used pin 2 since it is also an interrupt pin in Arduino). Then we added the following line of code for the external interrupt:

*attachInterrupt(digitalPinToInterrupt(3), isr, CHANGE);*

This states that the interrupt is triggered whenever the value of pin3 is changed. We are using the mode CHANGE since the switch that we have is a toggle switch and does not have a constant LOW or HIGH value. When our program begins to run, it reaches at the attachInterrupt in the setup function. Since we have set the interrupt detecting device to be the switch then the program checks if the switch is pressed and if it is, it will go to the function *isr()* :

*void isr() {*

*interFlag = true;*

*previousMillis = millis();*

*}*

We simply check the time that we are at the moment from the beginning of the interrupt and set a flag to call *isr()* whenever needed. Therefore, taking us have 20 seconds before the light goes off in our code, the code would work as follows:

When the code is uploaded, the led does not lightens up until we press the pushbutton (trigger the interrupt). After the button is pressed, the light stays on for 20 seconds and then goes off. If it occurs that the user interrupts the program within the 20 seconds that the LED was going to be on, it would save the time that it had reached at, and then add it at the end. For instance if the pushbutton is repressed again after 5 seconds , the LED will now be on for 25 seconds, meaning it compensates the 5 seconds that the user interrupted it at.

We tested this with the built in led (which is connected to pin 13 of the Arduino) and it worked as we expected.

After that, we tried to implement this using AVR. We wrote the code for this and tested it with the built in led to do the same function as above. However, we were not able to get it working with the requested logic. We were able to make the led lighten on and off every 5 seconds without using the pushbutton.

**2) Implementing Oscilloscope (using Processing and Function Generator)**

To do this, first we had to observe the values that we get by printing them on the serial monitor. We printed the values and observed they had the ranges from 0 to 1024. Using these values we realized that the processing graph would be dense and flat and hence we had to scale the values. To do this, we used the formula;

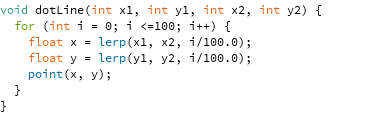
*float val = raw \* 5.0 / 1023.0;*

Where raw is the values that are in the range of 0-1023. Using this formula, we were able to get values in the rage of 0 to 5V.

Thereafter, we displayed the graph and it had a bit of spikes but not really the exact sine graph. Therefore, after researching, we found out that we had to change the serial monitor baud rate. We changed it from 9600 to 3200 and that changed the problem. Then we proceeded with other components of the processing graph to make it look like an oscilloscope.

First of all, we decided to draw a graph looks just like the one on oscillator. Since we did some lab about the communication of Processing and Arduino Serial Monitor before, we put this part at the last and designed our layout first.

There were a lot to write in the window so we put all the layout in a function called *drawStuff()* to make the *draw()* function looks more neat. The background was set as black and we drew several blue rectangles to create a basic layout on the Processing window. We chose to draw the screen by drawing several lines to form a rectangle instead of drawing a rectangle because when we tested it, the rectangle blocked our plotting. We used *lerp()* function to draw the dot lines on the screen and changed the parameter to *i / 100.0* to draw the dot lines with perfect density and length. Also, because we need to draw many dot lines in different position, we put it as a function and called it in a *for* loop. This made the task much easier and the code more readable.



We also wrote another function for the dash lines and we used it to make the edges on the frame of the graph. The edges are set to act as the units of the axes, *Volts and Time.* The other tasks were just putting texts in appropriate positions and separate them with bars. After finish those, we wanted to add some buttons to adjust the position of the plotting on the screen. We took the code from lab4 where we designed buttons and modified it to draw two triangle buttons that suggests putting the plot up or down. When user clicks on those buttons, we increase or decrease the *yPos* of the plotting function to change its position.



It was not hard to design a window like this.

However, we found some problem when doing the communication. From all the tests, we found that our Processing worked best with frequency from 52 to 54 Hz; our sketch reads the data from Serial Monitor and plots the graph on the screen as we wanted. But when we tried to use an array to take some of the *inByte*’s and compare them to find the maximum value and to calculate the frequency, we got errors and the Processing disabled the port. The following are the code we wrote, but it was deleted from the sketch since it was not working:

*float inputs[] = new float[30];*

*int arrayPos = 0;*

*float max = 0;*

*if (arrayPos < inputs.length) {*

*inputs[arrayPos] = inByte;*

*arrayPos++;*

*}*

*if (arrayPos == inputs.length) {*

*for (int i = 0; i < inputs.length; i ++)*

*if (inputs[i] <= inputs[i+1])*

*max = inputs[i+1];*

*arrayPos = 0;*

*}*

The additional functions that make the oscilloscope user friendly are the up and down arrows. These act like the offset button in the function generator. When they are pressed, the whole graph moves up or down depending on which one is pressed. That is the function of this portion of the code;

*void mousePressed() {*

*if (up) {*

*yPos += 5;*

*}*

*if (down) {*

*yPos -=10;*

*} }*

**C. Conclusions**

In general, we were able to implement an oscilloscope that looks similar to the one in the lab. We learnt how to use the timers to control the interrupts in this project. Also we have learned a bit about AVR while trying to implement the interrupts using the AVR (through inline assembly and port manipulation).

Despite that we were not able to use it for this lab, we have a fairly good foundation for the future projects to use the AVR and maybe inline assembly to control the interrupts.

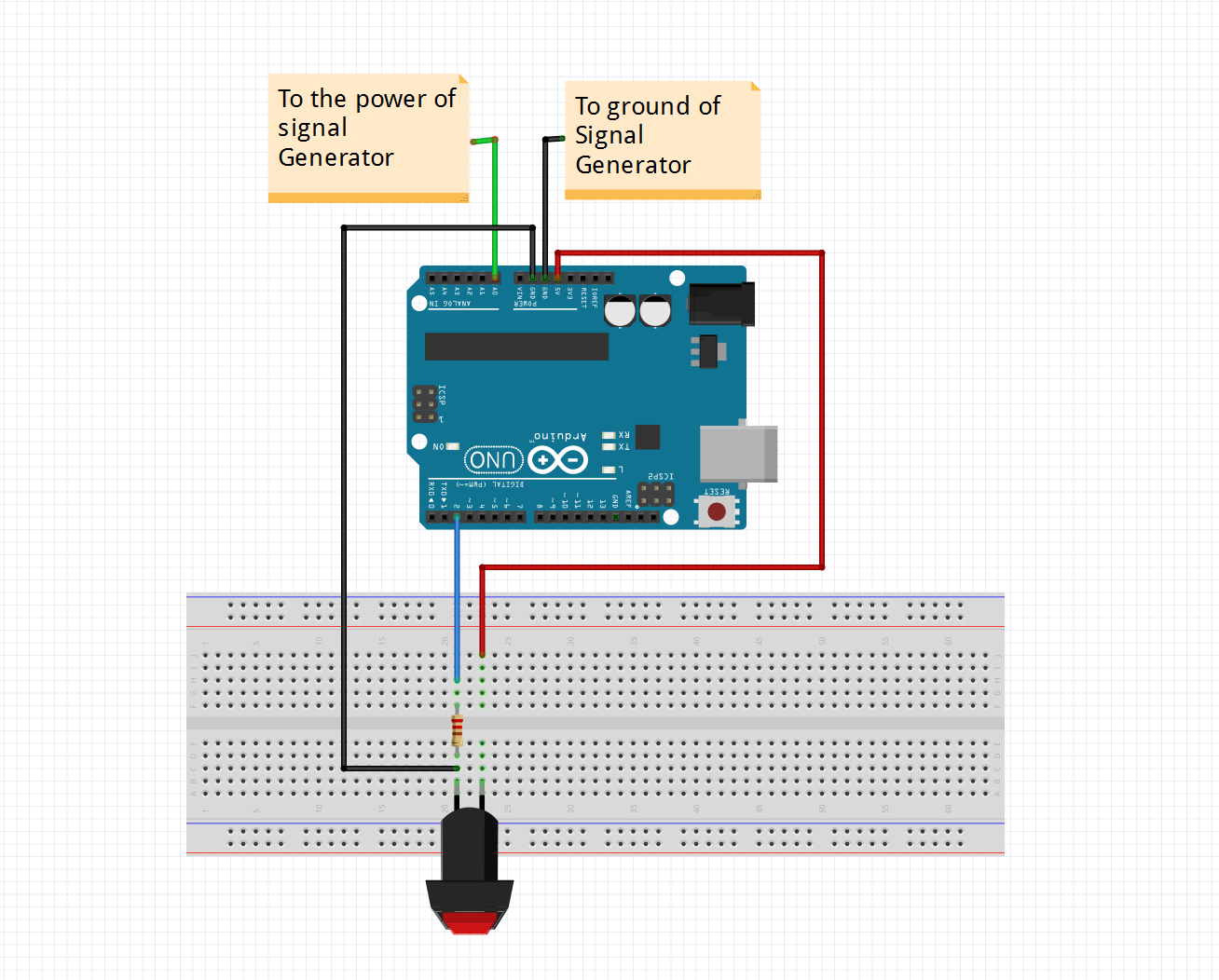
**D. References**

The references that were used in this report are;

* For the Arduino functions used <http://arduino.cc/en/Reference/HomePage>
* For the interrupt service routine: <http://arduino.cc/en/Reference/AttachInterrupt>
* For determining the pins used to configure the external interrupts: <http://playground.arduino.cc/Code/Interrupts>
* For the AVR documentation and features: <http://playground.arduino.cc/Main/AVR>
* For the function that draws dot line: <https://processing.org/reference/lerp_.html>

**Appendix I**

The fritzing snapshot:



**Appendix II**

**Arduino code including comment statements**

const int buttonPin = 3; // the number of the pushbutton pin

const int signalPin = A0; // the number of the signal pin

unsigned long previousMillis = 0; //the initial time

unsigned long currentMillis = 0; //the current time being observed

const long interval = 20000; // the to run after interrupt

// flag for interrupt to avoid run automatically after power on

boolean interFlag = false; //Flag used to control the interrupt to keep running

void setup() {

Serial.begin(3200);

pinMode(signalPin, OUTPUT);

// initialize the pushbutton pin as an input:

pinMode(buttonPin, INPUT);

// initialize the build-in led as a flag for us to see when interrupt happpens

pinMode(13, OUTPUT);

// jump to the isr() when the button changes

attachInterrupt(digitalPinToInterrupt(3), isr, CHANGE);

}

void loop() {

// read currentMillis to calculate time has passed

currentMillis = millis();

// if the time passed is smaller than the interval and we are in interrupt

//then read data from analog pin and light up the led

if ( interFlag && currentMillis - previousMillis <= interval) {

float raw = analogRead(signalPin);

float val = raw \* 5.0 / 1023.0;

Serial.println(val);

digitalWrite(13, HIGH);

}

// otherwise, don't read and set the led to low

else

digitalWrite(13, LOW);

}

// set the interrupt flag as true and set previousMillis

void isr() {

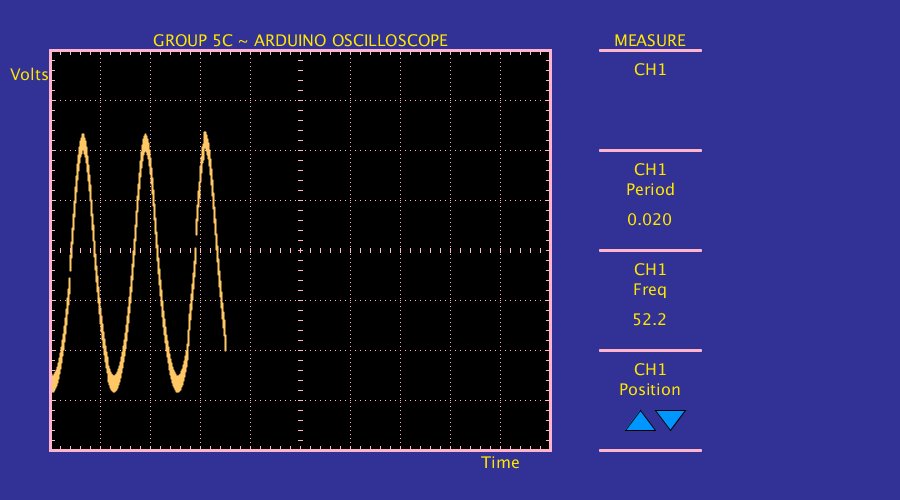
interFlag = true;

previousMillis = millis();

}

**Appendix III**

**Processing code including comment statements and clear snapshots of Processing user interface.**



// initialize the serial port to read data

import processing.serial.\*;

Serial myPort;

// variable that takes the input

float inByte = 0.0;

// set the flag for buttons

boolean up = false;

boolean down = false;

// colors for the buttons

color onUp = color(255, 150, 0);

color onDown = onUp;

color upColor = color(0, 150, 255);

color downColor = upColor;

// set the vertices for up and down buttons; w for width, h for hight

int upX = 640;

int downX = 670;

int w = 15;

int upYTop = 410;

int h = 20;

int downYTop = 430;

// set the position of the graph

float xPos = 50.0;

int yPos = 250;

// shift the result half of the 5V measured

float shift = 2.5;

// flag for new value that we take in

boolean newVal = false;

// measure range is from 52.0 to 54.0 Hz

void setup() {

size(900, 500); // set the size of the window

println((Object)Serial.list()); // enabling printing of the values on the console window

//initialize serial communication and the port being used

myPort = new Serial(this, Serial.list()[0], 3200 );

// generate a new serialEvent() when gets a newline character:

myPort.bufferUntil('\n');

background(0);

}

void draw() {

drawStuff(); // draw the screen of the oscillator

update(mouseX, mouseY); //check the position of the mouse

//fill different colors depending on whether the mouse is over the button

if (up) { //if the mouse is over the up button

fill(onUp);

} else {

fill(upColor);

}

stroke(0); //setting the background to black

triangle(upX, upYTop, upX - w, upYTop + h, upX + w, upYTop + h);//draw the button as a triangle pointing upwards

// do the same as the above one

if (down) { //if the mouse is over the down button

fill(onDown);

} else {

fill(downColor);

}

stroke(0);

triangle(downX, downYTop, downX - w, downYTop - h, downX + w, downYTop - h);

stroke(255, 200, 100);

// when the graph reaches the end of the screen, go back to the origin.

if (newVal) {

strokeWeight(2.0);

line(xPos, height - yPos - inByte + 15, xPos, height - yPos - inByte);

if ((xPos +=1) >= 550) {

xPos =50;

//draw the background again to clear the screen

background(0);

}

newVal = false;

}

}

void serialEvent (Serial myPort) {

// get the ASCII string:

String inString = myPort.readStringUntil('\n'); //serial port reads until end of new line

if (inString != null) {

// trim off any whitespace:

inString = trim(inString);

// convert to an float and map to the screen height:

inByte = float(inString) - shift ;//separate the String with “,” to read different data

//the temperature button is pressed so that temperature values can keep looping until stopped

inByte = map(inByte, 0, 5, 0, height - 180);

// set a flag for the value received

newVal = true;

}

}

//function that draws all the elements of the window to create a screen of oscillator

void drawStuff() {

//cover the screen with black rectangles and leave one part as blue

noStroke();

fill(50, 50, 150);

rect(550, 0, 900, 500 ); //drawing the rectangles that act as the frame of the oscillator

rect(50, 0, 550, 50);

rect(0, 0, 50, 550);

rect(50, 450, 550, 50);

// set the color of the rectangle frame of the oscilloscope;

stroke(255, 180, 205);

strokeWeight(3.0);

strokeCap(ROUND);

line(50, 50, 550, 50);

line(50, 50, 50, 450);

line(550, 50, 550, 450);

line(50, 450, 550, 450);

// draw the lines on the left of the screen

drawBar(50);

drawBar(150);

drawBar(250);

drawBar(350);

drawBar(450);

// set the stroke weight to normal

strokeWeight(1.0);

// draw the dot lines on the screen

for (int m = 100; m < 550; m +=50) {

if ( m == 300);

else

dotLine(m, 50, m, 450); //for the vertical gridlines

if ( m == 250);

else

dotLine(50, m, 550, m); //for the horizontal grid lines

}

// draw the dash lines on the screen (the background lines)

drawDashVer(248);

drawDashVer(50);

drawDashVer(446);

drawDashHor(298);

drawDashHor(50);

drawDashHor(546);

// put all the text

textSize(16);

fill(255, 230, 0);

textAlign(CENTER);

text("MEASURE", 650, 46);

text("CH1", 650, 75);

text("CH1", 650, 175);

text("Period", 650, 195);

text("0.020", 650, 225);

text("CH1", 650, 275);

text("Freq", 650, 295);

text("52.2", 650, 325);

text("CH1", 650, 375);

text("Position", 650, 395);

text("GROUP 5C ~ ARDUINO OSCILLOSCOPE", 300, 46);

text("Volts", 30, 80);

text("Time", 500, 468);

}

void drawBar(int y) {

line(600, y, 700, y);

}

// draw vertical dash line

void drawDashVer(int y) {

for ( int x = 50; x < 550; x += 10)

line(x, y, x, y + 4);

}

// draw horizontal dash line

void drawDashHor(int x) {

for (int y = 50; y < 450; y += 10)

line(x, y, x + 4, y);

}

// draw dot line

void dotLine(int x1, int y1, int x2, int y2) {

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

float x = lerp(x1, x2, i/100.0);

float y = lerp(y1, y2, i/100.0);

point(x, y);

}

}

//if the mouse is on one of the button, set the “over” flag of this button as true, the others’ as false

void update(int x, int y) {

if ( overUp(upX, upYTop, w, h) ) {

up = true; // the mouse is over temperature button

down = false; //set “over” flag of humidity and light as false

} else if ( overDown(downX, downYTop, w, h) ) {

down = true; // the mouse is over humidity button

up = false; //set “over” flag of the other two as false

}

}

//check if the mouse is over the up button

boolean overUp(int x, int y, int width, int height) {

if (mouseX >= x - width&& mouseX <= x + width &&

mouseY >= y && mouseY <= y + height) {

return true;

} else {

return false;

}

}

//check if the mouse is over the down button

boolean overDown(int x, int y, int width, int height) {

if (mouseX >= x-width && mouseX <= x + width &&

mouseY >= y-height && mouseY <= y) {

return true;

} else {

return false;

}

}

// if click on the up buttons, increase the shift on y axis; same for the down button

void mousePressed() {

if (up) {

yPos += 5;

}

if (down) {

yPos -=10;

}

}