Samuel Gandy

Log book 3 Digitial

Group 30

Contents

[Log Book 1 21/2/19 2](#_Toc5283026)

[Log Book 2 28/2/19 2](#_Toc5283027)

[Research 2](#_Toc5283028)

[Creating a sinewave with Kiel 4](#_Toc5283029)

[Digital Project 3 Pseudo Code 5](#_Toc5283030)

[Log book 3 Consolation Week 6](#_Toc5283031)

[Working out the values 6](#_Toc5283032)

[Log Book 4 14/3/19 8](#_Toc5283033)

[Log Book 21/3/19 8](#_Toc5283034)

[Log book 28/3/19 8](#_Toc5283035)

[Log book 1/4/19 9](#_Toc5283036)

[Testing and analysis 9](#_Toc5283037)

[Checking frequency’s 9](#_Toc5283038)

[Improved frequency’s 11](#_Toc5283039)

[Frequencies review 13](#_Toc5283040)

[RC Circuit 14](#_Toc5283041)

[Capacitor 14](#_Toc5283042)

[Resistor 14](#_Toc5283043)

[Block Diagram – Hardware 15](#_Toc5283044)

[Circuit - Schematic 15](#_Toc5283045)

[Final Program 16](#_Toc5283046)

[Evaluation 20](#_Toc5283047)

[Program 20](#_Toc5283048)

[Hardware 20](#_Toc5283049)

[Output frequencies 21](#_Toc5283050)

# Log Book 1 21/2/19

First week

To start the code, I have done some research and found this website.

<https://alselectro.wordpress.com/tag/how-to-write-c-code-in-keil/>

In the phycial log book is the notes about the lab book

Write up that in neat for this log book.

# Log Book 2 28/2/19

Starting the main project file from the template made last week. Then I can start the project as a basic structure then build on that for the final project.

I have followed the tutorial on how to make the C file in Kiel. I have made sure all the files are there. Now I need to write the program. I will start by naming all the pins that will be used in the code. To do this I need to choose the output pin, then I need to find a push button program to find out how inputs work in Kiel C.

## Research

In research this week I have found a link to how to create a sinewave from a micro controller in C. The link below is how to create a sinewave in C.

<http://www.keil.com/support/man/docs/uv4/uv4_sm_ai_sinewave.htm>

I have found some code that shows how to set up a push button in C. This will be very useful for the program as interrupts are included within this website.

<http://www.keil.com/support/man/docs/uv4/uv4_sm_di_push_button.htm>

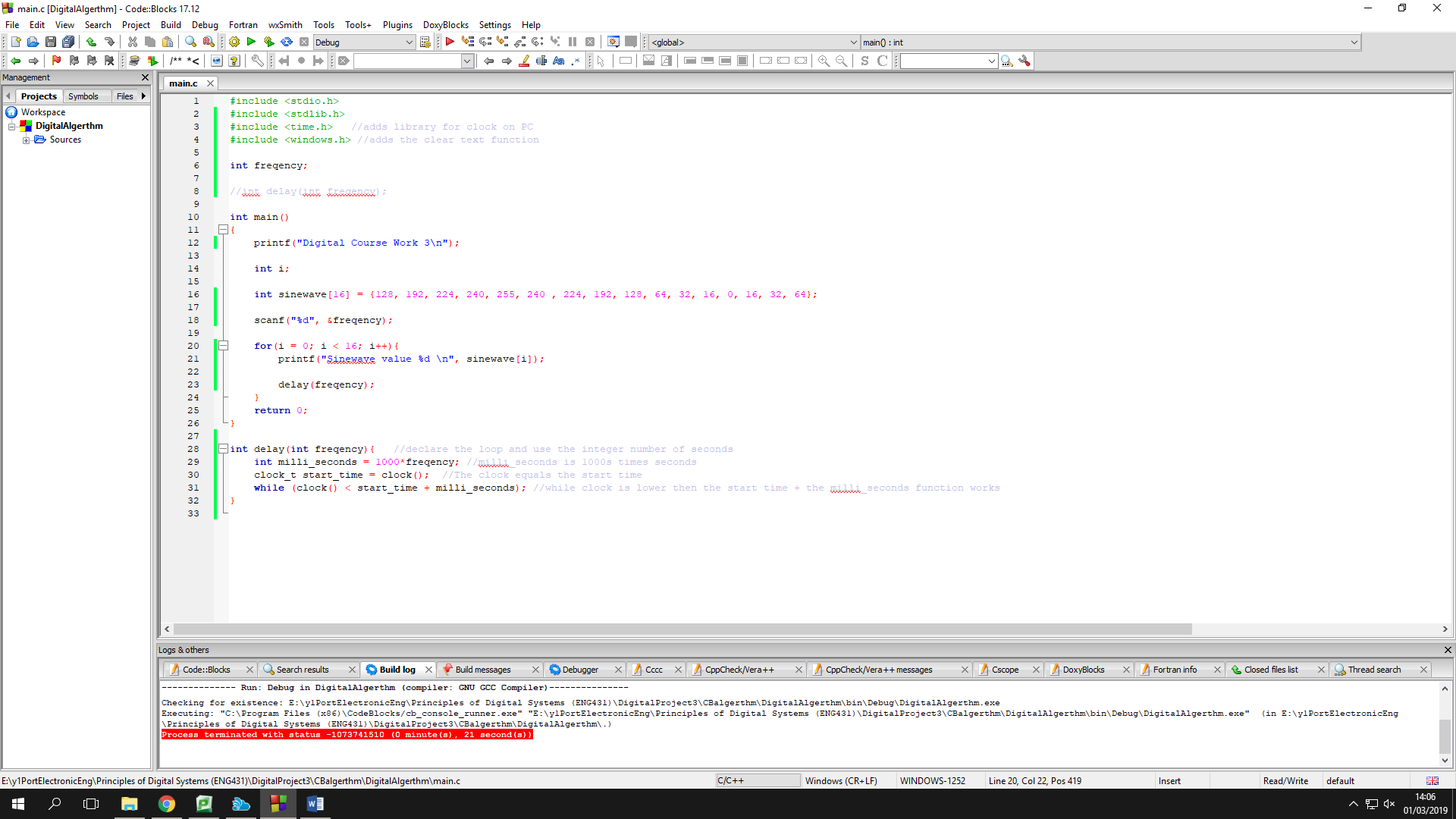
watched this video on how to program push buttons

<https://www.youtube.com/watch?v=IzPlCVgH-es>

I looked on Youtube to how to create a sine wave in kiel and found this video and gave me the values for a sinewave.

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

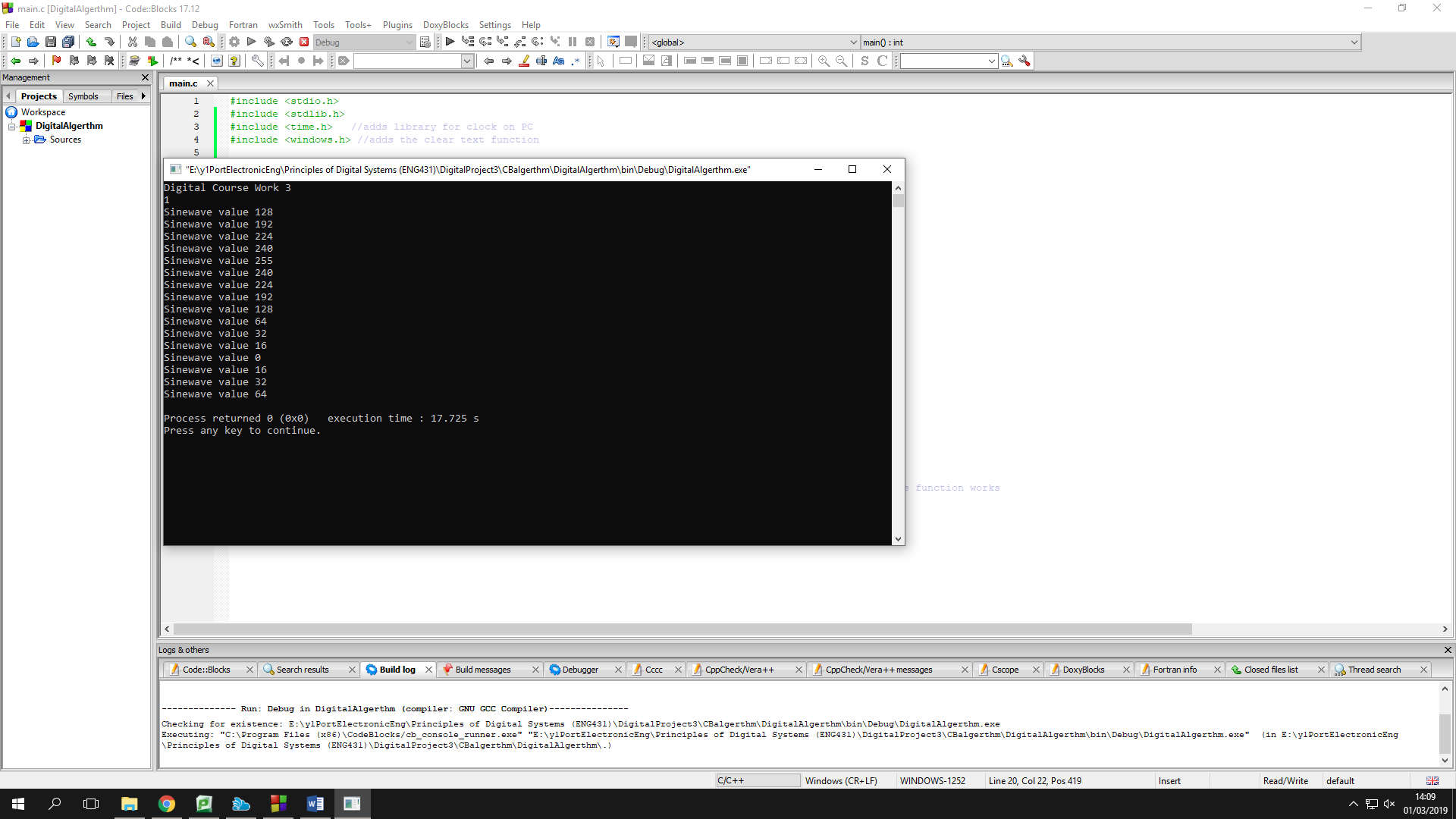
I had an idea that if the input of each button is equal to the difference between the values in the array the frequency of the sinewave can be changed. The value from this can be the button input into the algithrm. This means the code can very efficient. When there is an input, the wave waves are printed.

The values are in an array and run through. The speed of the inputs are controlled by what button is pressed. I have created this idea in code blocks IDE so I can test the system so when I can access Kiel I can adept the code and run the program.

The scan repeats the button inputs and the prints is the pin values to the speaker.

The console shows the output on the pin in one period of the wave.

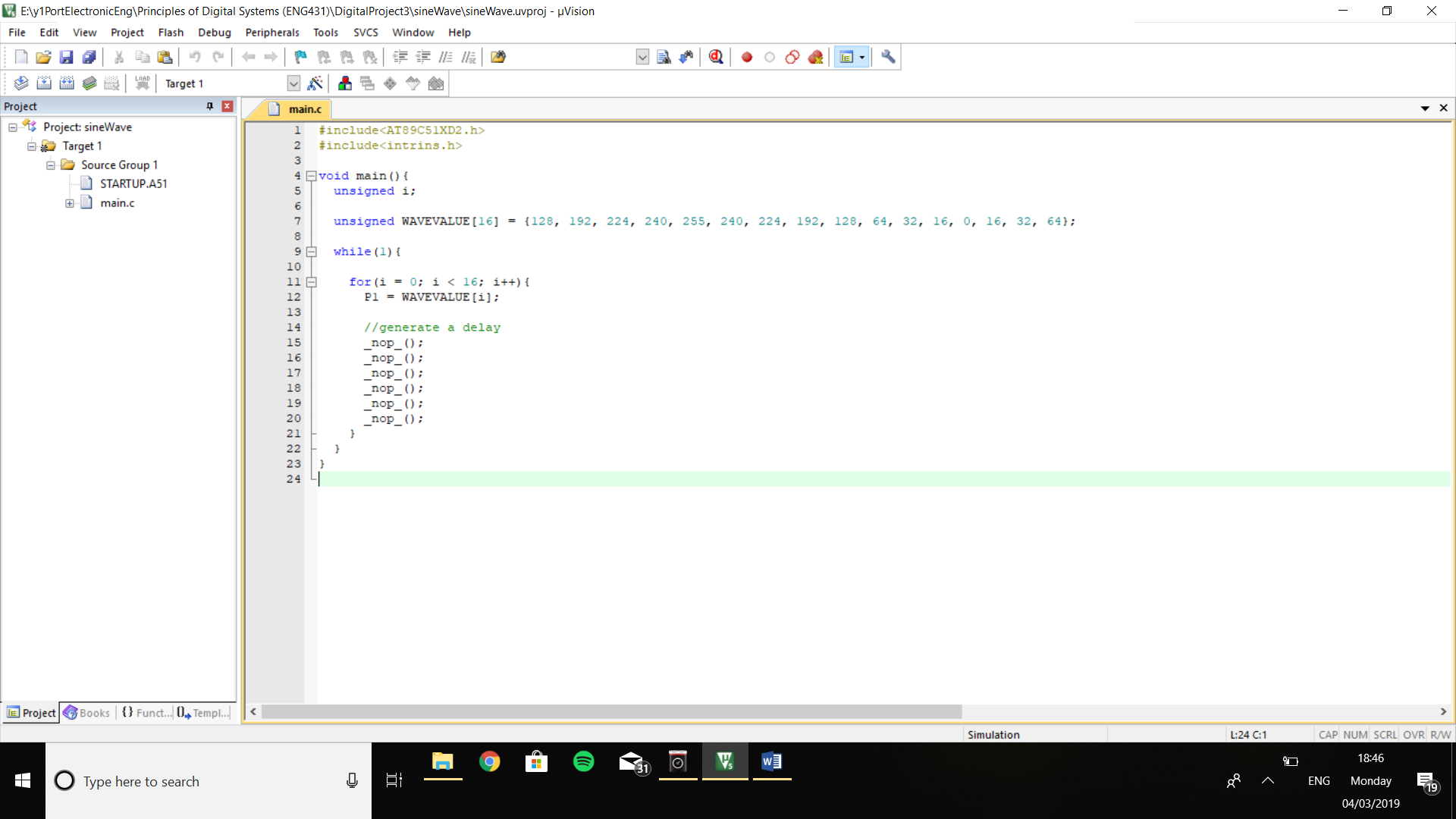
The delay function will be different on the micro-controller because the time is not set on a computers clock but the amount of loops that will run through. I will need to look through the video again to make sure i can have the sinewave working perfectly.



## Creating a sinewave with Kiel

From one of the links in the research part of this week on how to create a sinewave. The code works by using a for loop to run through all the variables in the sinewave array. The variables in the sinewave array are the instantons values on the wave every one sixth of the period. The for loop works by starting for a selected value until a set value and increments by the amount set. In this case start on the 0 value of the array (128) and count to the 16th value in stages of one.

The no-ops are the set time between the values. This is also in the for loop because this has to happen every time a value changes.



I have plotted the sinewave in excel to visually show the sinewave and how the sinewave will look coming from the microcontroller. Using this graph the value can be worked out from the degrees.

## Digital Project 3 Pseudo Code

declare loop variables (i)

declare all the buttons input

declare the buzzer

char sinewave [16] = {128, 192, 224, 240, 255, 240, 224, 192, 128, 64, 32, 16, 8, 0, 16, 32, 64, 128}

main function (){

while (loop equals 1){

while(button is pressed){

set selectedPeroid = the peroid of time between stages of array

call function sinePrint using selected peroid

}

}

}

sinePrint(selected peroid){

for(i = 0; i < selected period; increment by 1){

sinewave[i] = buzzerPin

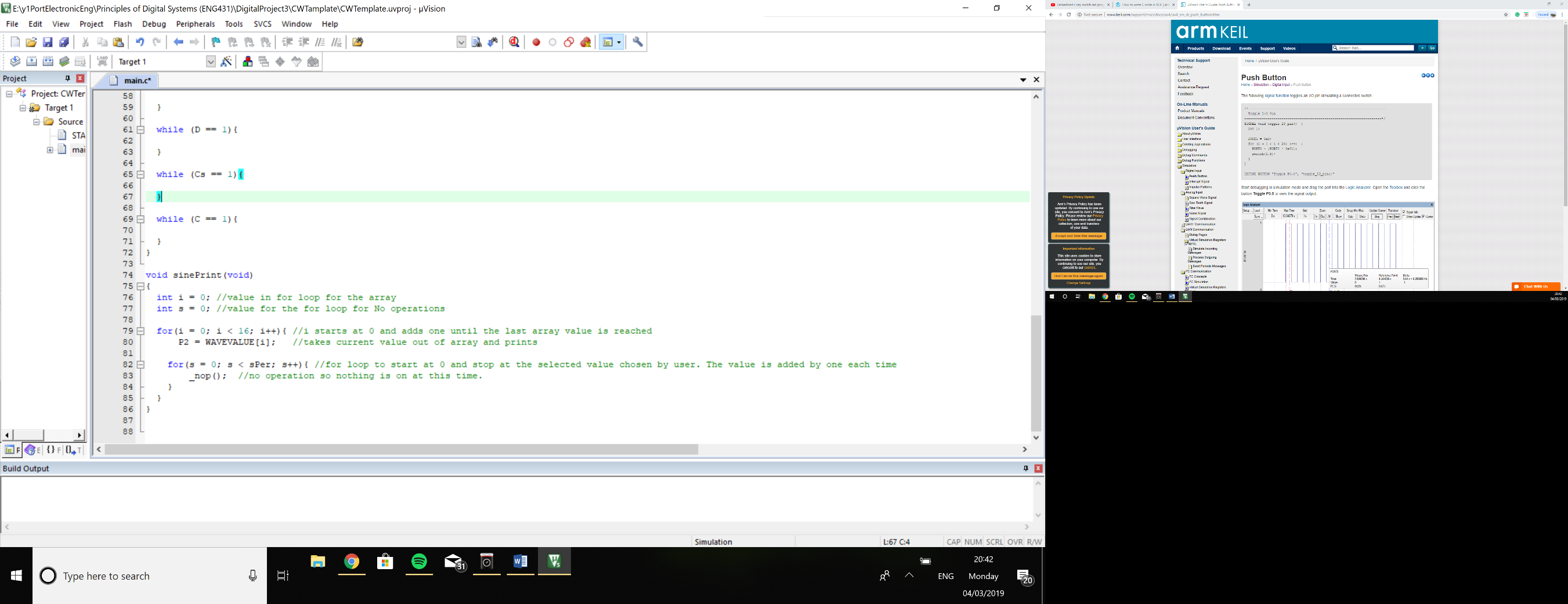
}

}

# Log book 3 Consolation Week

This week I need to focus on how to read an input signal to the micro-controller. This means I can read when a button is pressed. This is very useful for the project as there are 13 buttons that need to be read. I will look on YouTube for a tutorial to follow through to achieve this. Finding it hard to find a good source of information for the push button.

I am going to work out the structure of the main program for the project. This will help me find more problems to solve to complete the project. This is important to find the problems early.

I have written a function called sinePrint and the functions role is to print the sinewave value to the buzzer pin. The frequency is also controlled in this function using a for loop with a no operation.

The amount of no operations depends on the value that is inputted. Each button has a set value which is the period of the wave dived by the amount of sine wave values (16). The next step is to work out how long a No operation is. Then work out how many operations are required for each 1/16th of the period of each wave shown as variable sPer. The function is shown at the side of this text.

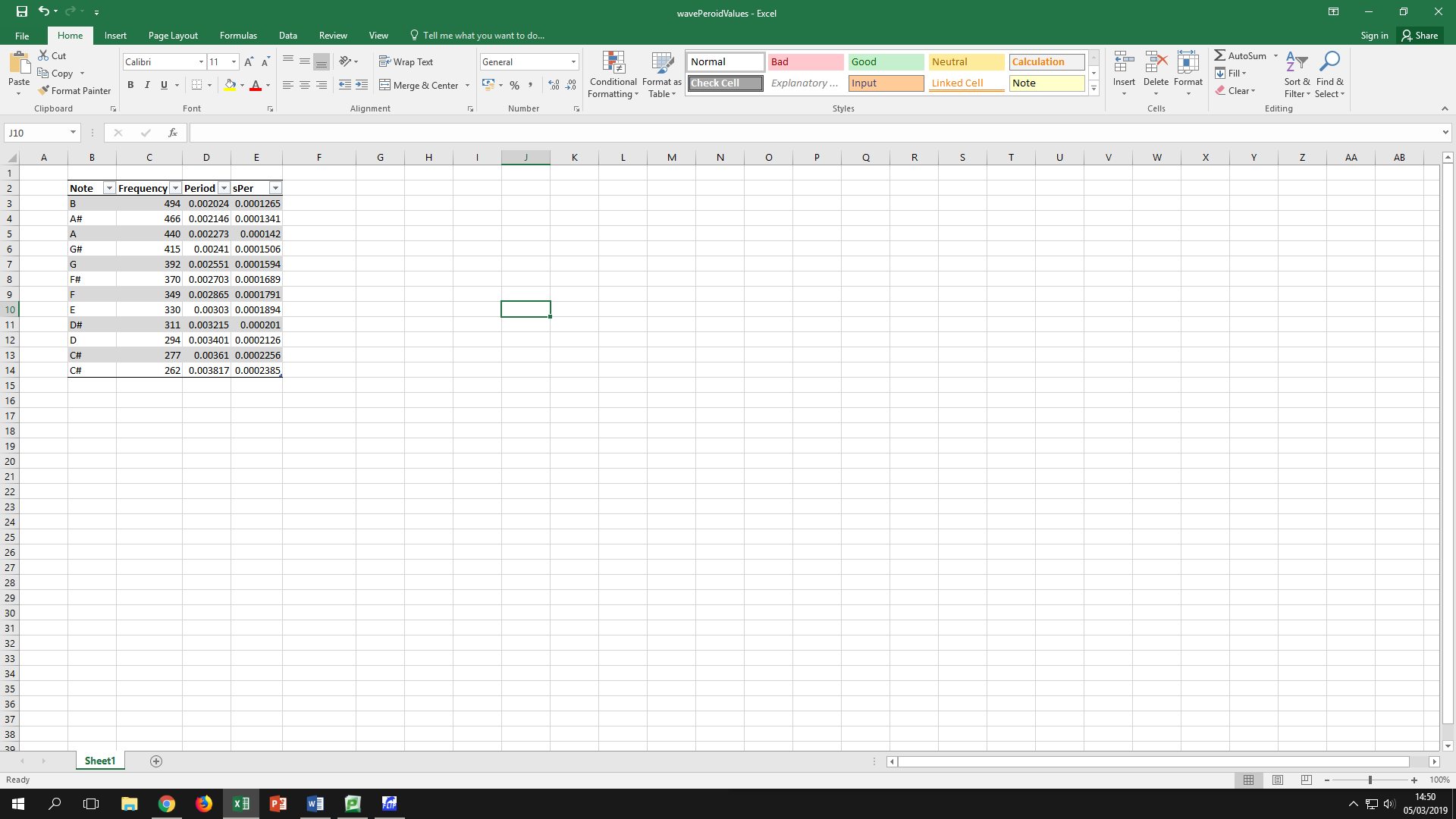
## Working out the values

These formulas is the algthrim to working out the amount of no operations each sound requires to get the correct frequency.









Found the for loop runs though the loop at the same rate as the processes cycle so the for loop has to run the same amount of times as needed for the delay. This saves us time and makes the code easier to understand.

# Log Book 4 14/3/19

We tested the button on the oscilloscope and found when the button is pressed the signal goes low when pressed so we were looking for when the button is 1 not a 0 so the button code was not working.

The buttons are being set to the pins in the code and these match wires that are going to the micro-controller. This is important otherwise when a button is pressed the wrong output will play.

The sinewave will not work because the port is 0 to 255 not the pins but I wonder if the whole port can be used for the buzzer. This means more wire but much more efficient code. This is a trade-off we will have to do.

We need to turn the port from digital to analogue so the signal for the buzzer. Looked into digital to analogue converter IC but we don’t have one available so, we can make a resistor leader to reduce the voltage using potential difference.

<https://www.youtube.com/watch?v=bXUfDLF4MVc>

# Log Book 21/3/19

The lecture advised us to use this website that shows the different type of resistor ladders.

<http://mosaic.cnfolio.com/ParchizadehB222L>

<https://hackaday.com/2015/11/05/logic-noise-digital-to-analog-with-an-r-2r-dac/>

<https://www.best-microcontroller-projects.com/R-2R-ladder.html>

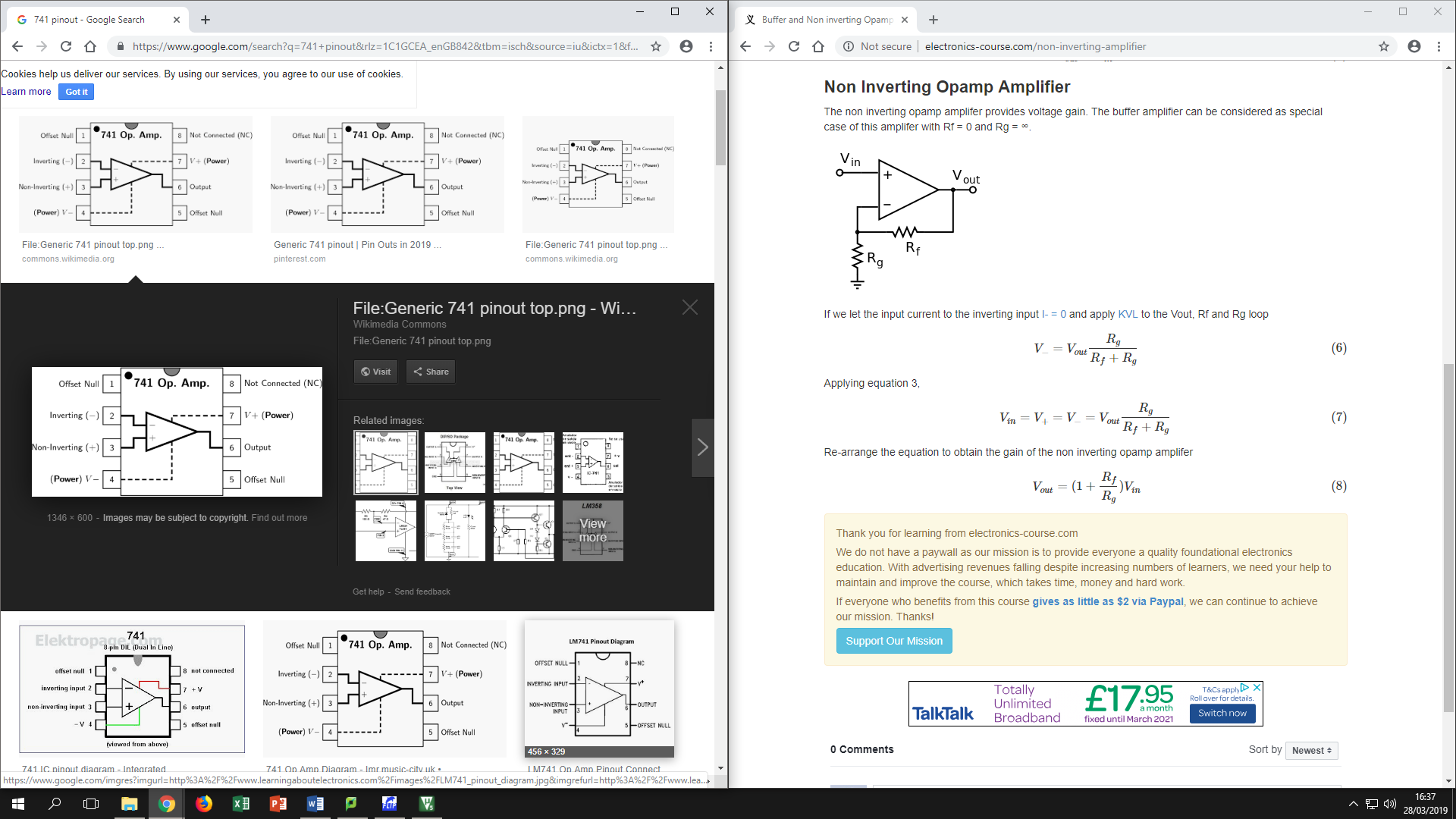
This week we a research week and we didn’t have much action on the project. The notes are in physical logbook under the date.

# Log book 28/3/19

This week the resistor ladder was made. We tested the signal and there was no output for all buttons. Then I plugged the oscilloscope into the output of resistor ladder. There was a signal for most of the buttons.

G button didn’t work and this is because the while loop for the button was not working. This small error was found on the oscilloscope. This was because when the button was pressed the signal on the output from resistor ladder was flat.

The problem of there being know sound from the buzzer is still a problem. I have an idea to solve this problem by fixing the op amp. This op amp will be a non-inverting buffer model.



<http://electronics-course.com/non-inverting-amplifier>

# Log book 1/4/19

The resistor ladder is working and we know this from the oscilloscope signal. The signal as shown below is the output from the resistor ladder. The resistor ladder breaks the signals from the outputs from the microcontroller.

The signal will not make a sound. This is because the signal is too weak. This means an op amp is required. The op amp can boost the signal. There are different ways to use the op amp to change the variables that are increased. The op amp I will be using is the LN741 as I have used it before.

The op amp has been connected up and it produced a much distorted signal and there was no sound as the amplification was not working. The op amp was in the non-inverting configuration meaning the signal will be increased and not flipped. This is prefect for the project as the signal is just increased.

The signal will still not be outputted correctly. This is very annoying. This problem has made me think about using a speaker instead of a buzzer. The speaker will be used in the final testing. I plugged the speaker into the end of the resistor ladder. Then pressed a button and the sound come out. I tested all the buttons and they worked.

The system is working but the only problem is the tones are off. This means the system needs to be tuned. I can do this by changing the numbers on the variable sPer. This is very important for the final test.

# Testing and analysis

## Checking frequency’s

These tables show the output signals and there frequency’s. Using the oscilloscope to find the frequency. These values are compared to the notes.

|  |  |  |  |
| --- | --- | --- | --- |
| **Note** | **Expected Frequency (Hz)** | **Frequency outcome (Hz)** | **Picture** |
| B | 494 | 519 |  |
| A# | 466 | 483 |  |
| A | 440 | 453 |  |
| G# | 415 | 426 |  |
| G | 392 | 426 |  |
| F# | 370 | 403 |  |
| F | 349 | 381 |  |
| E | 330 | 362 |  |
| D# | 311 | 329 |  |
| D | 294 | 315 |  |
| C# | 277 | 301 |  |
| C | 262 | 278 |  |

The frequencies are all wrong but close. I now need to tune all the sounds to the notes by changing the frequency. To make the values more accurate I need to change the selected period value and use trial and error to see if the frequency is correct. The graph below shows the final frequencies.

## Improved frequency’s

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Note** | **Selected period** | **Expected Frequency (Hz)** | **Frequency outcome (Hz)** | **Picture** |
| B | 12 | 494 | 518 |  |
| A# | 13 | 466 | 483 |  |
| A | 14 | 440 | 452 |  |
| G# | 15 | 415 | 425 |  |
| G | 16 | 392 | 403 |  |
| F# | 18 | 370 | 362 |  |
| F | 19 | 349 | 346 |  |
| E | 20 | 330 | 330 |  |
| D# | 21 | 311 | 315 |  |
| D | 22 | 294 | 301 |  |
| C# | 24 | 277 | 278 |  |
| C | 26 | 262 | 258 |  |

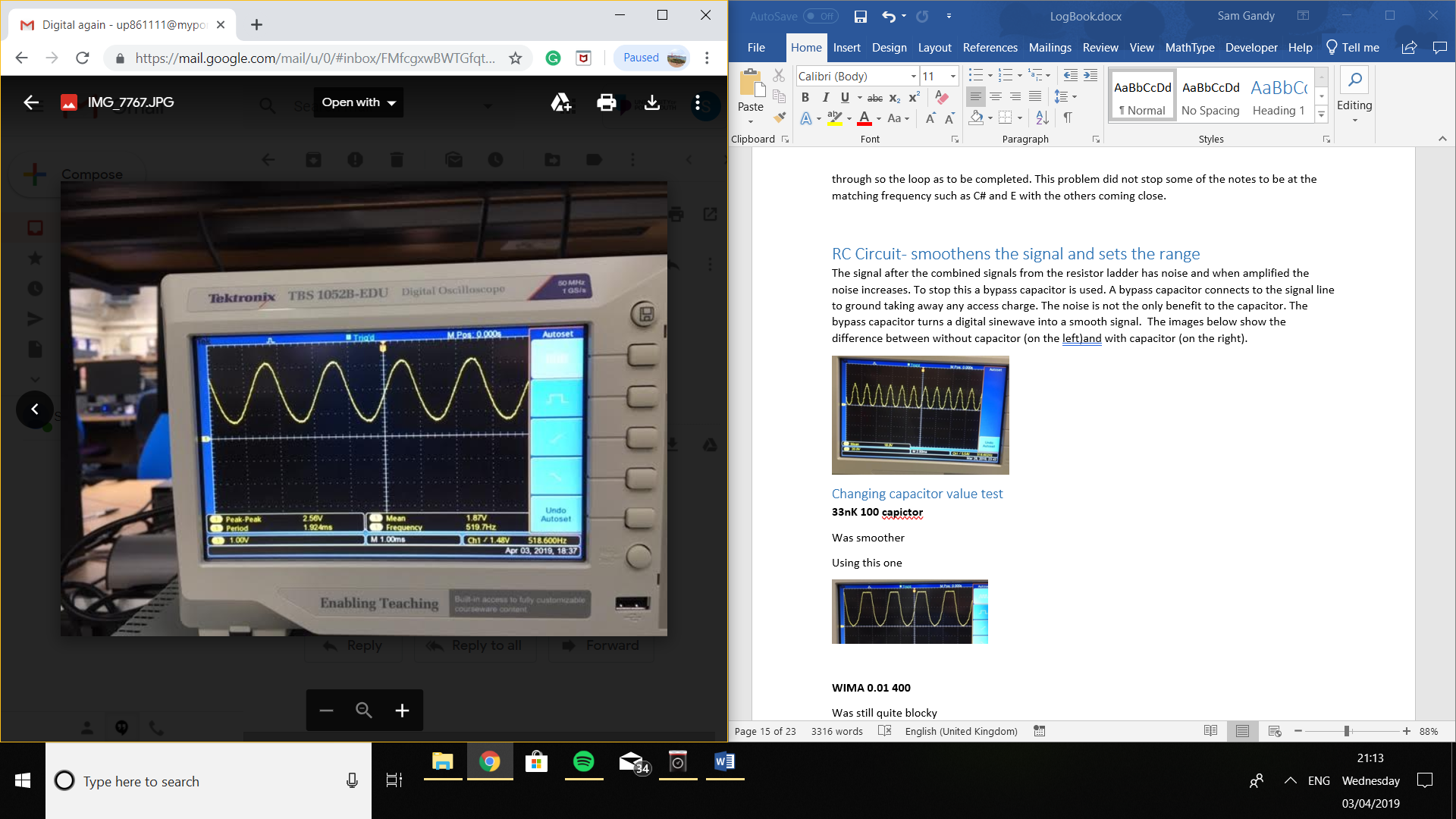
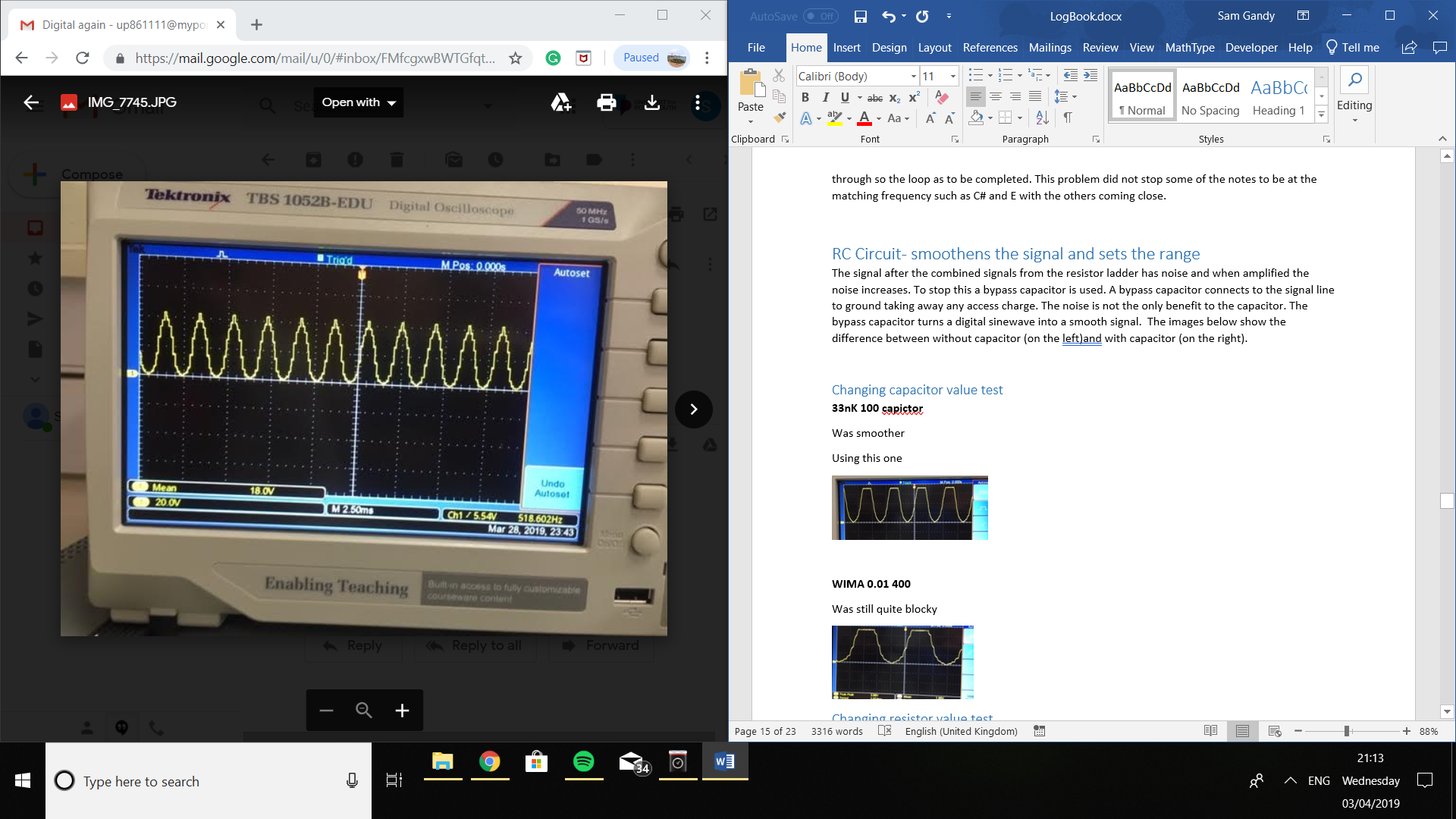
## Frequencies review

The frequencies where not correct to start with as the first tests shows. We found the results for both tests by using the oscilloscope and looking at the frequency reading. The readings after tuning are accurate as can be with the code. The code does restrict the accuracy because the code has to run through a for loop as the delay between the sinewaves change. The loop cannot stop half was through so the loop as to be completed. This problem did not stop some of the notes to be at the matching frequency such as C# and E with the others coming close.

# RC Circuit

## Capacitor

The signal after the combined signals from the resistor ladder has noise and when amplified the noise increases. To stop this a bypass capacitor is used. A bypass capacitor connects to the signal line to ground taking away any access charge. The noise is not the only benefit to the capacitor. The bypass capacitor turns a digital sinewave into a smooth signal. The images below show the difference between without capacitor (on the left) and with capacitor (on the right).

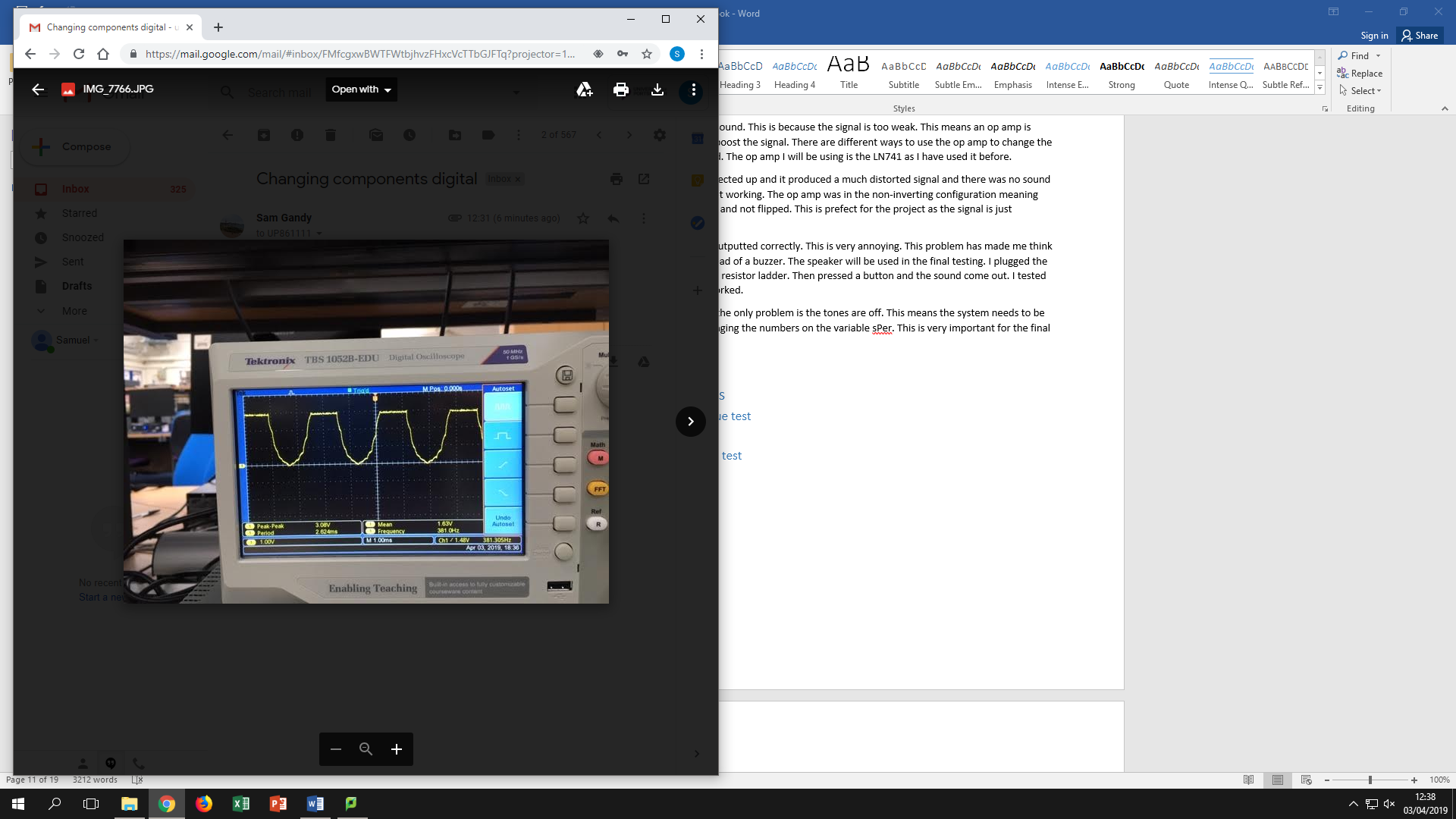
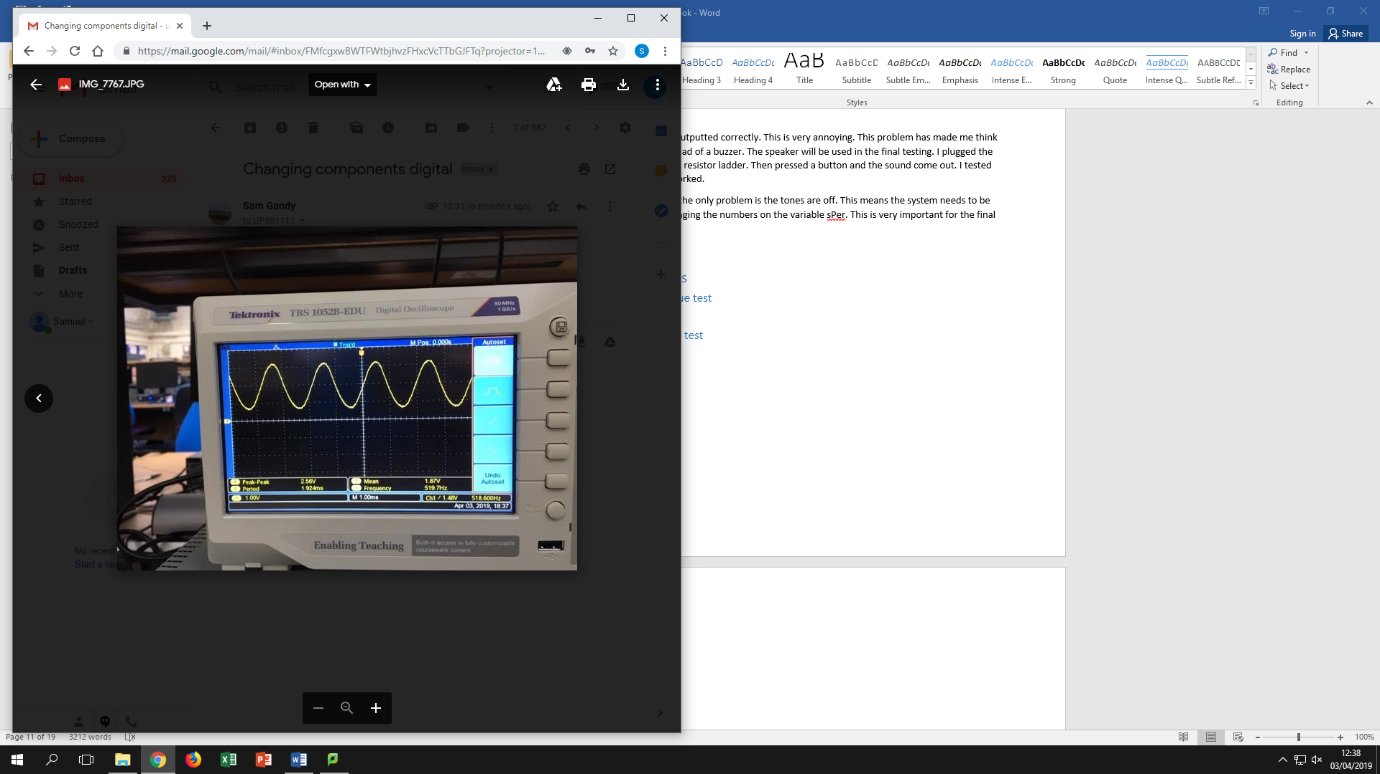


## Resistor

The resistor in the RC circuit is after the resistor ladder. The resistor sets the range of the signal before the op amp. The resistor reduces the voltage and the current before the op amp meaning there is less current and voltage before the op amp boasts the signal. This means a range can be set from 0 to a set voltage.

I experimented with the values of resistor to change the range and looked at the output at the oscilloscope. I tested 1k, 5K and 10K resistors. The 1K allowed too much voltage through and the waves where all cut off and all the sound could be heard. The 5k cut of half of the waves so half could be heard and the others could not. The 10K had no waves cut off and no sound. The waves looked the best. There was no sound because there was a limited voltage at the output. The waves below show the wave forms of the 1k and the 10k.

I had a choice to make sound or quality of the waves. I choose sound as the main role of the project is to make the selected sounds on a synthesiser. I chose the 1K resistor for the project.



# Block Diagram – Hardware

Synthesizer

Analogue

Digital signal

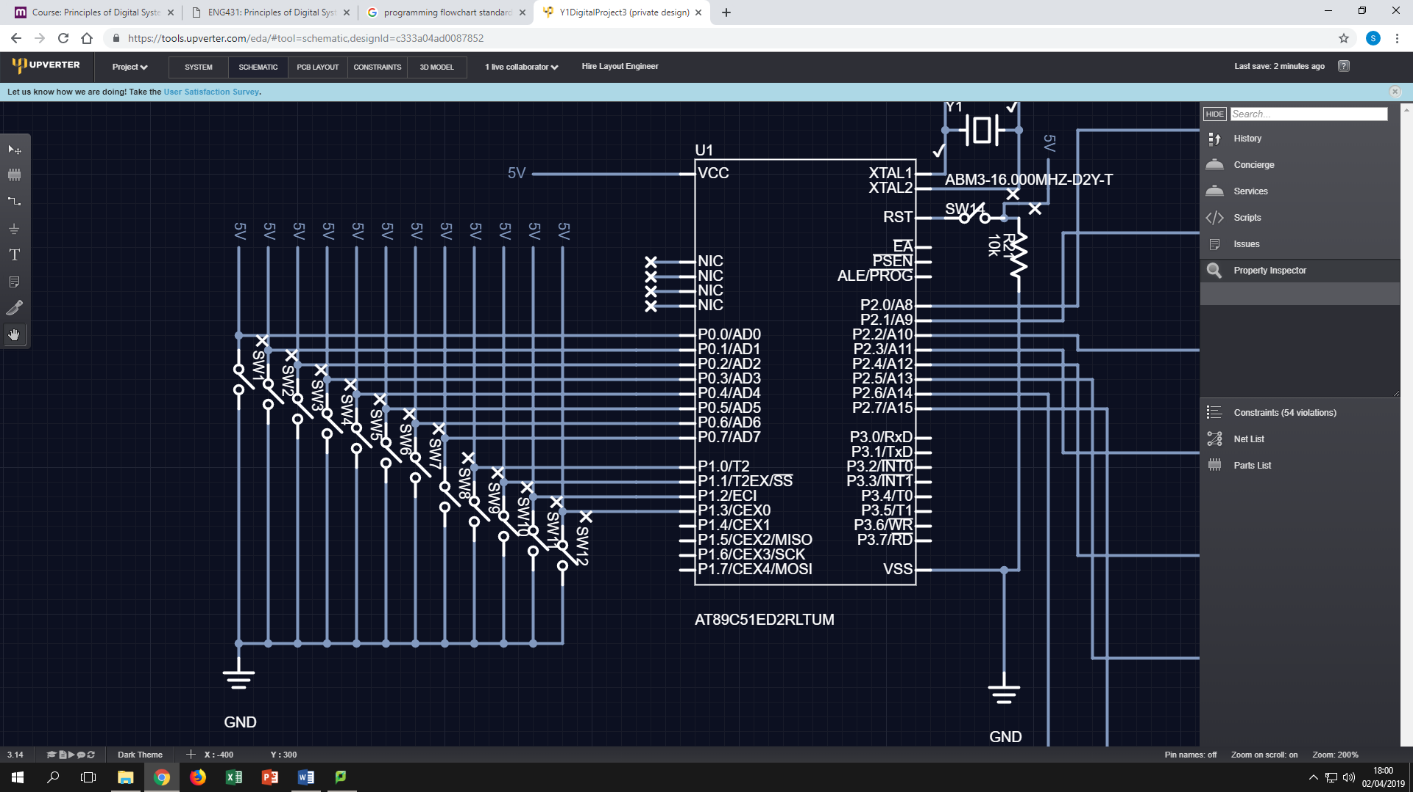
Speaker / buzzer sound

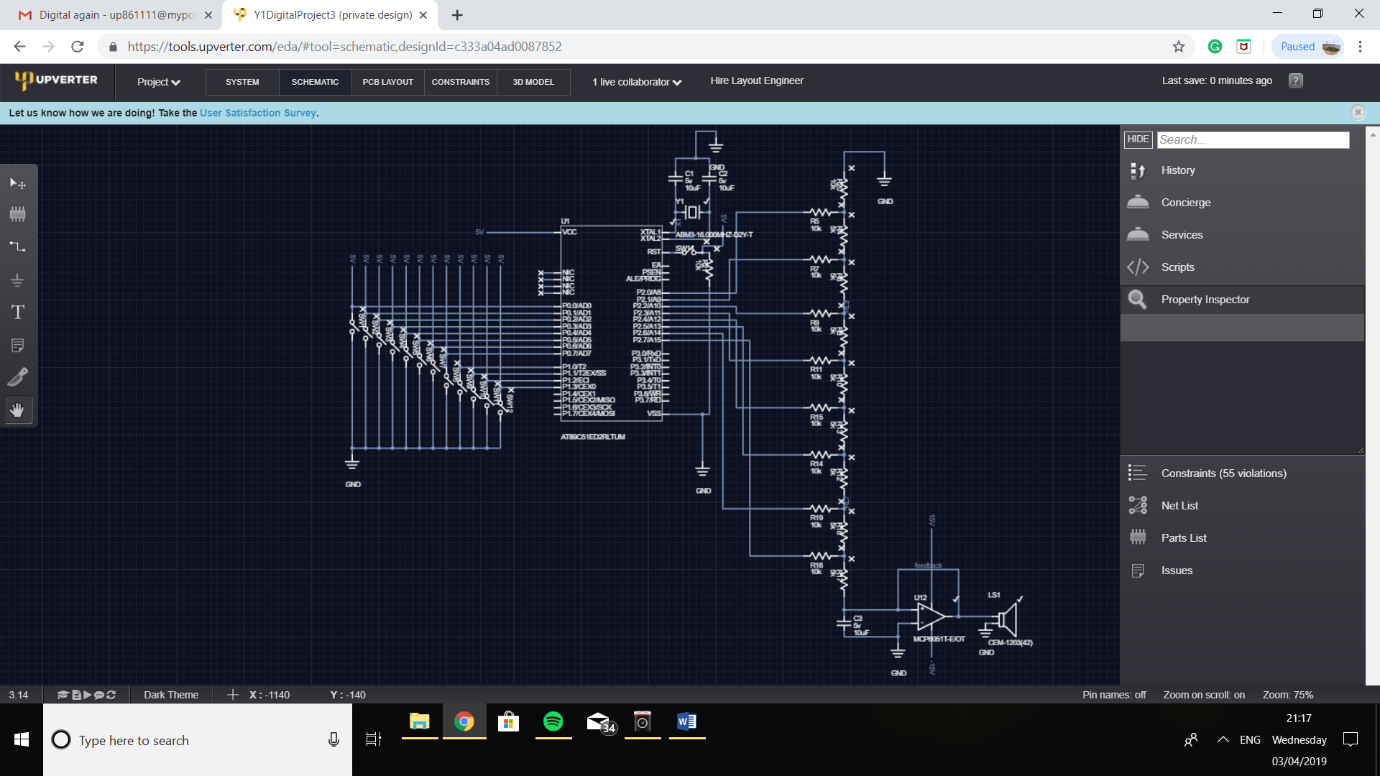
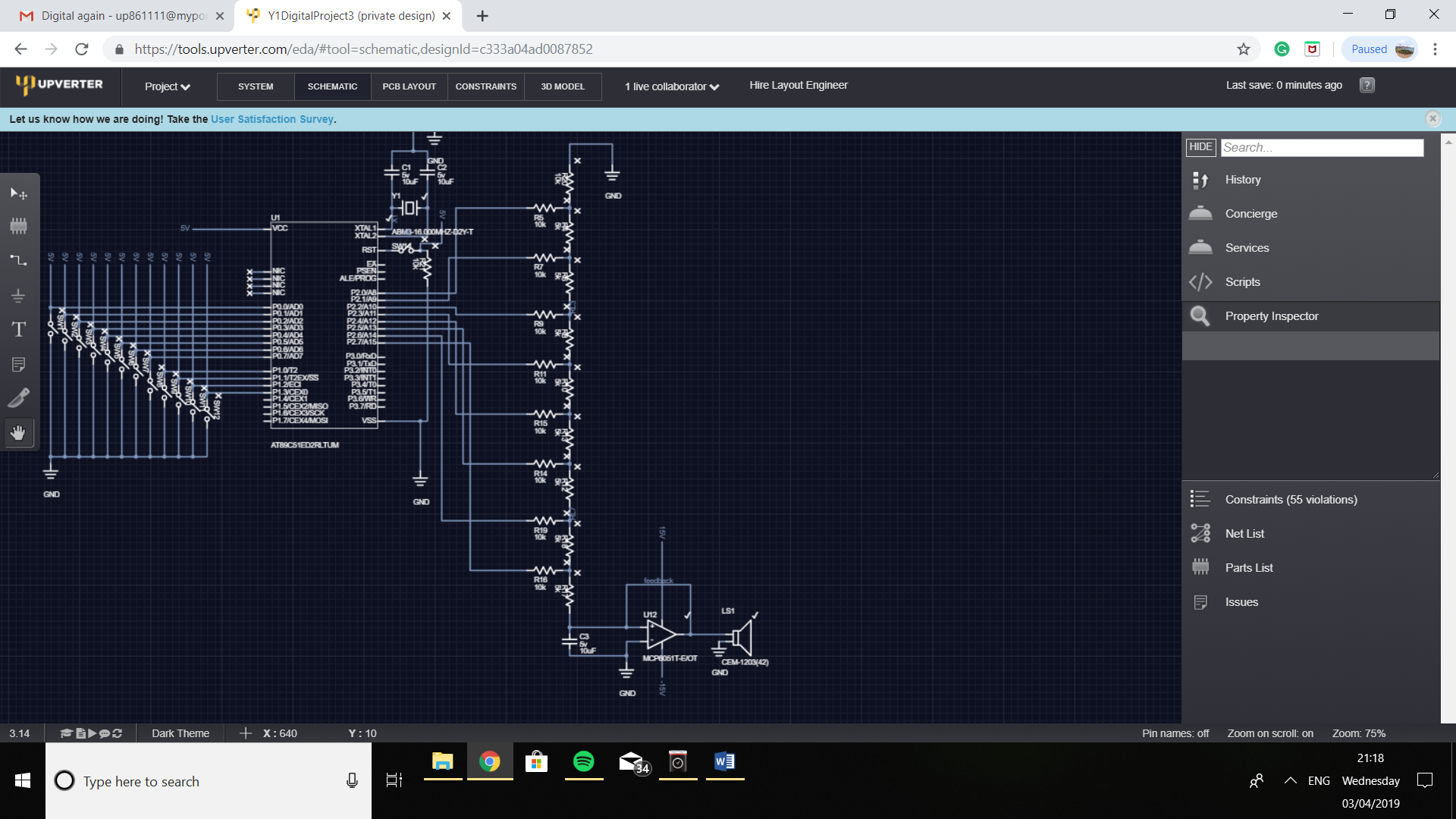
Micro-controller

DAC using a resistor ladder

Button Press

# Circuit - Schematic





# Final Program

#include<AT89C51XD2.h> //uses the functions for the AT89C51XD2

#include<stdio.h> // prototype declarations for I/O functions

sbit buzz = P2; //sets buzzer as pin P2.0 on the chip

sbit b = P0^7; //button b is P0.7

sbit As = P0^6; //button A sharp is P0.6

sbit A = P0^5; //button A is P0.5

sbit Gs = P0^4; //button G sharp is P0.4

sbit G = P0^3; //button G is P0.3

sbit Fs = P0^2; //button F sharp is P0.2

sbit F = P0^1; //button F is P0.1

sbit E = P0^0; //button E is P0.0

sbit Ds = P1^0; //button D sharp is P1.0

sbit D = P1^1; //button D is P1.1

sbit Cs = P1^2; //button C sharp is P1.2

sbit C = P1^3; //button A is P1.3

int sPer = 0; //sPer is selected peroid and stores the amount of No operations are called

void sinePrint(int sPer) ; //sinePrint function prototype declaration

unsigned WAVEVALUE[16] = {128, 192, 224, 240, 255, 240, 224, 192, 128, 64, 32, 16, 0, 16, 32, 64}; //array that stores the 16 values for the sinewave

int loop = 1; //makes loop equal 1 this will never change so the program always runs

void main(void){ //the main function that the code runs through

//This is where the input code goes for each pin

while(loop == 1){ //while loop equals one run this loop. This means the loop always runs until turned off

while (b == 0){ //while button that represents b is pressed run loop

sPer = 12; //selected peroid is set to 13 when b is pressed

sinePrint(sPer); //call and run function sinePrint with the sPer value from the current loop

}

while (As == 0){

sPer = 13;

sinePrint(sPer);

}

while (A == 0){

sPer = 14;

sinePrint(sPer);

}

while (Gs == 0){

sPer = 15;

sinePrint(sPer);

}

while (G == 0){

sPer = 16;

sinePrint(sPer);

}

while (Fs == 0){

sPer = 18;

sinePrint(sPer);

}

while (F == 0){

sPer = 19;

sinePrint(sPer);

}

while (E == 0){

sPer = 20;

sinePrint(sPer);

}

while (Ds == 0){

sPer = 21;

sinePrint(sPer);

}

while (D == 0){

sPer = 22;

sinePrint(sPer);

}

while (Cs == 0){

sPer = 24;

sinePrint(sPer);

}

while (C == 0){

sPer = 26;

sinePrint(sPer);

}

}

}

void sinePrint(int sPer){

int i; //value in for loop for the array

int s; //value for the for loop for No operations

for(i = 0; i < 16; i++){ //i starts at 0 and adds one until the last array value is reached

P2 = WAVEVALUE[i]; //takes current value out of array and prints

for(s = 0; s < sPer; s++){ //for loop to start at 0 and stop at the selected value chosen by user. The value is added by one each time

}

}

}

# Evaluation

The overall project has been successful and very productive. The project to create an electronic synthesizer. The project has allow us to create a project that is mostly based around a program but also required hard ware to be built.

## Program

The first part of the program is to set the pins with sBit. sBit states the pins which are all the inputs and outputs. The variables are then set. The variables are the sPer, the function sinePrint, the sinewave array and the loop. The loop is a variable that is always 1 in a while loop so the program keeps looping so the program doesn’t finish unless the micro-controller turns off.

The main body of the code is made up of while loops for each button. Every while loop is the same because the sPer is set at a value to each button. Then the function sinePrint is called with the sPer value. The while loops are set so when the button is 0 run the loop will run this is because the buttons are always high only when pressed the pins read 0.

The function runs through the sineValue array which, has the fixed values that make one cycle. The cycle then repeats until a button is not pressed. The delay between the changes in value that are outputted on the port is the sPer value. The sPer value is set in a FOR loop as the maximum value reached. The for loop takes time with every loop which means the frequency is change by the amount of loops done in the for loop. This way changing the sPer means the frequency can be changed.

## Hardware

The main parts of the hard ware was given to us with the micro-controller and the keyboard itself. The buttons are in the shape of a key board and the buttons are on a PCB. The job we had was to connect the buttons to the micro controller. This needs to relate to the program so when C# is pressed the program knows C# signal is being registered as C#. The system used was to put them all in order to stop confusion.

Micro-controller is the AT89C51ED2. The micro-controller stores the program and runs the program. The program is the code that is written above. Port 0 and Port 1 read the inputs whereas port 2 is the outputs to the resistor ladder.

The resistor ladder is made up of 5k and 10k resistors. The resistor ladder is used as a DAC (Digital to Analogue converter). The DAC is made up of just resistors that weight the values from the micro-controller port 3. The port outputs in binary 0 to 255 (8-bit) but these signals is just them turning the pins on and off. This means there the digital signal needs to be converted to analogue. The resistor ladder works by using 16 resistors for 8 bit which are 10k and 5k. Each pin has a 10k to decrease the signal. Then there are 5k that connect each pin to the next one. This means the less significate bit has less of an impact then the most significate bit.

The resistor ladder decreases the signal but it’s an analogue signal. This signal is 0.6 mVs so this signal doesn’t create sound when connected to the buzzer but when connected to the speaker there is sound. This is because the signal is boasted in the speaker. This is very important to work on the buzzer. I have connected the output from the resistor ladder to an ln741 op amp with a limited resistive feedback to boost the signal to 5V.

## Output frequencies

The output frequencies and sound is nearly prefect. These sounds are not perfect because of the time delay in the program. This is caused from the speed of the microcontroller. There are also 16 values that are being run through.

Part of the specification sets when there are two buttons pressed only one of the sounds play instead of a mix of both. The while loops are set so when the first button is pressed the value is set and when the next button is pressed it is ignored. Even better when the first button is not pressed anymore the second one will play.