Electrical, Electronic and Computer Engineering
School of Engineering and Physical Sciences

Evrard EMONOT
H00385163

B31DG : Embedded Software

Assignment 2.

Summary

1 Introduction

In this assignment, we are going to execute task in the form of a cyclic executive. All of those tasks are going to be executed at a certain frequency on the ESP32

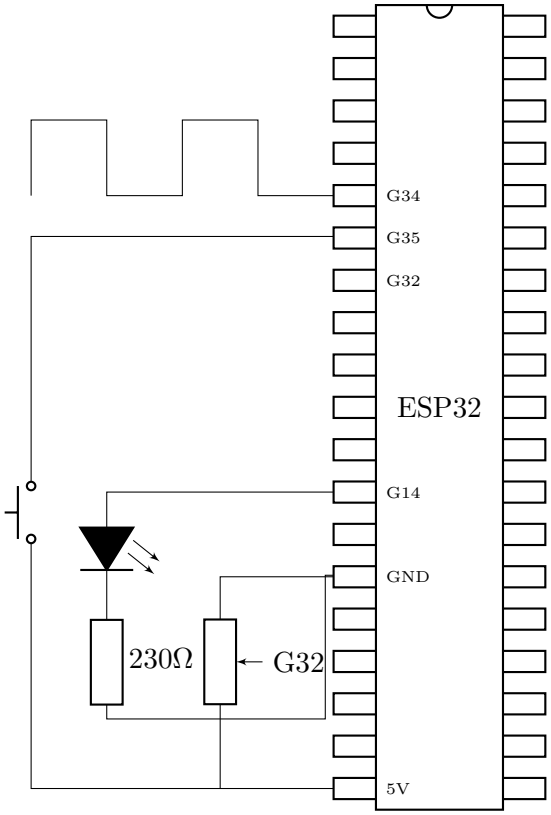
The commit and update of the project are available on that link : `https://github.com/Emonot/Embedded_Software_Assignement2`

2 Hardware

2.1 Schematics

Material used

- ESP 32
- 1 Resistor
- 1 Green LED
- 1 Switch
- 1 Potentiometer



2.2 Photos

Photos of the electronic assembly

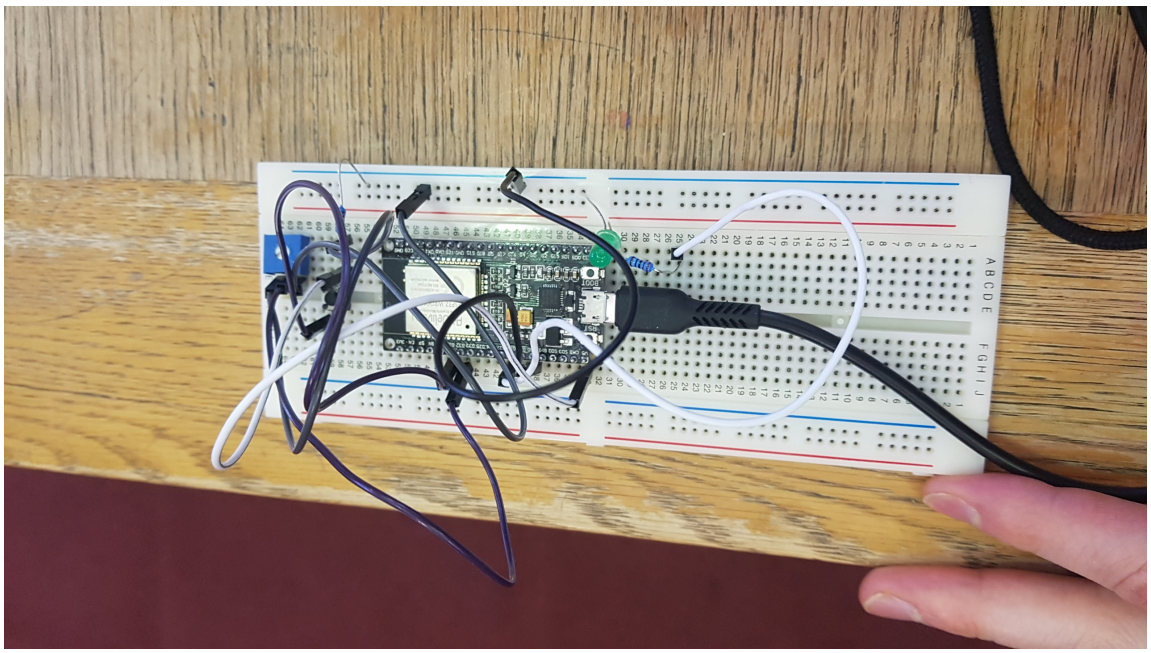


FIGURE 1 – Circuit

3 Method

3.1 Introduction and context diagram

This program contains 9 tasks to achieve.

Each task has simple instructions, Some of them are independent but most of them communicates between each others. Several tasks gather external input or send output.

To simplify the structure of the program, we use 2 context diagram. The first one shows the global structure with the ticker object and the second one describes the link between tasks inside the main process.

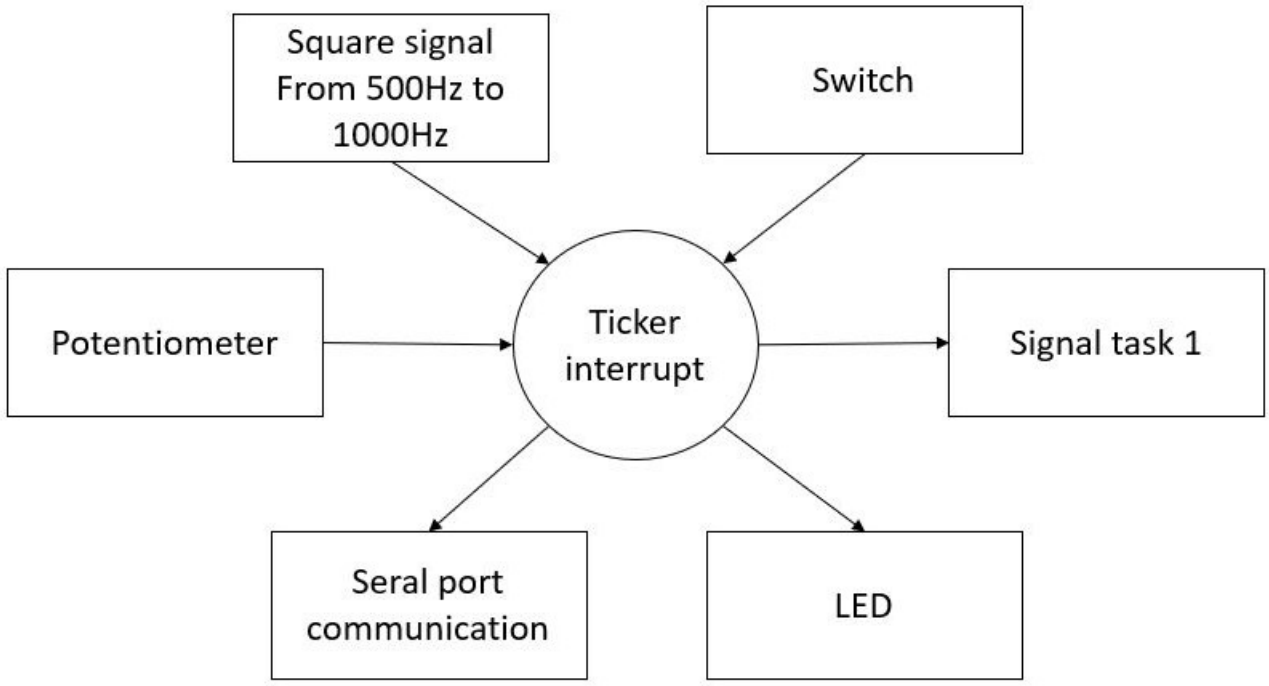


FIGURE 2 – Context diagram with the main process

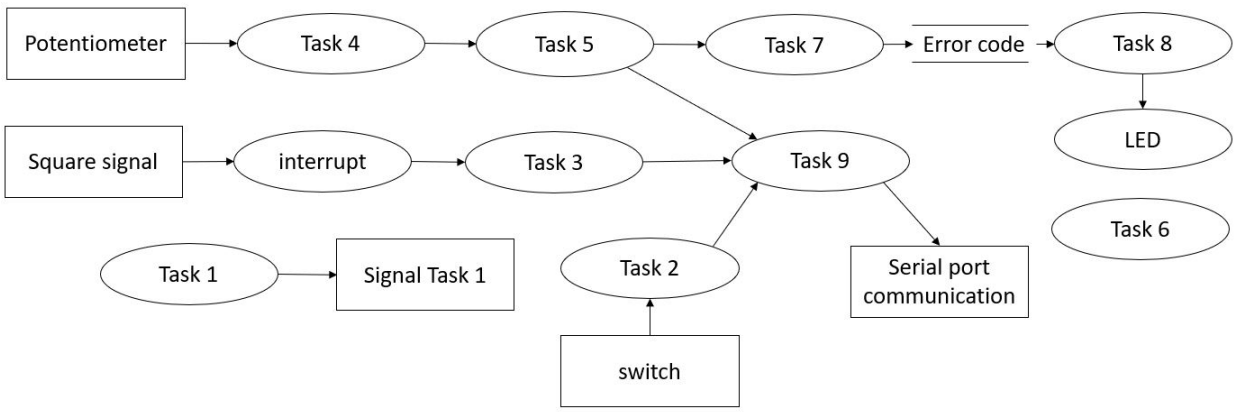


FIGURE 3 – Structures and link between tasks

3.2 Cycling executive parameters

Each task has a different frequency. To get a different frequency for each task, we create a ticker object and we attached it to a function that called the task at the right frequency using a counter. The ticker will be called every ms. Knowing that, we can use a counter that will be incremented every millisecond. Because every integer is a multiple of one, we can easily know when to call the function using the modulo operation.

Let's take the example of task 2. We need to execute it at 5Hz. This means that we need to call task2 every 200ms.

$$T = \frac{1}{5} = 200ms$$

We can verify when the counter has reach a multiple of 200ms with the modulo operation.

$$counter \mod (200) = x$$

When x will be egal to 0, we will execute the task. It means that we are going to execute the task at 5Hz.

In terms of precision, the microcontroller has a frequency of 80MHz which represents one instruction every 12.5ns. We can consider the time of instruction negligible compared to the frequency of the task being called.

The longest instruction is the first one because it contains a delay of $50\mu S$.Because the ticker is at 1ms, it will only be 0.05% of instruction time before the ticker gets called again.

3.3 Cycling executive diagram

Now that we know what is the frequency of our task and at what time they should be executed. We can create a cycling executive diagram to represent the occurrence of our tasks in one cycle.

The first diagram shows the pattern for one cycle (until task9 that takes 5 second before occurring), the second diagram shows the pattern until task3 (every second).
A blue cell shows when the task is executed.

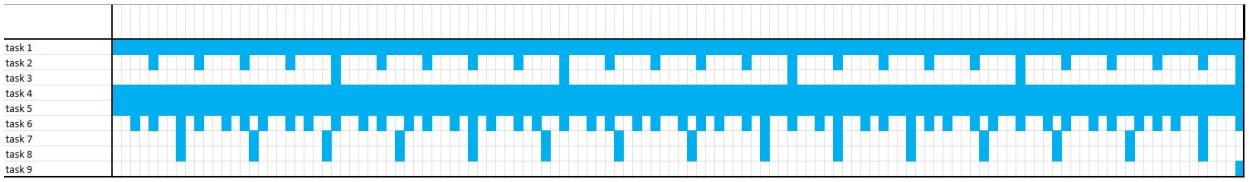


FIGURE 4 – Cycling executive diagram for a whole cycle (each cell's represents 41ms)

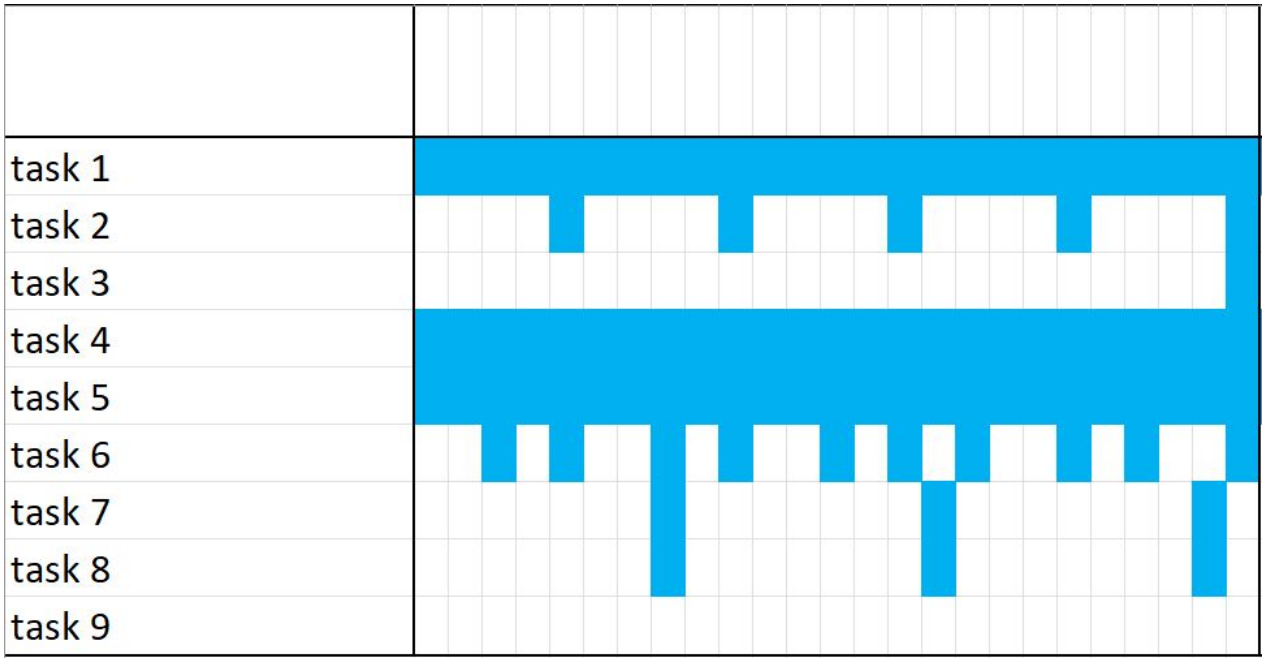


FIGURE 5 – Cycling executive diagram for 1 second (each cell's represents 41ms)

3.4 Code and task description

In the this section, We are going to describe the code and specify each task.

First we have our periodic ticker that will be called every milliseconds.

```
periodicTicker.attach_ms(1,do_all_task);
```

Each millisecond we check if the clock period matched with the period of a task. The clock is being updated by a counter variable that is being incremented every millisecond

```
void do_all_task(){

  if (counter % 42 == 0) task1();
  if (counter % 200 == 0) task2();
  if (counter % 1000 == 0) task3();
  if (counter % 41 == 0) task4();
  if (counter % 41 == 0) task5();
  if (counter % 100 == 0) task6();
  if (counter % 333 == 0) task7();
  if (counter % 333 == 0)task8();
  if (counter % 5000 == 0) task9();

  counter += 1;
}
```

The first task follows the SIGNAL B of the assignment 1. We write a small python script to get the period of the SIGNAL B.

We get the period of 42150ms, our ticker is called every millisecond. And, the first task contains a delay of 50 μS.We choose then to round the number to 42.

We can then calculate the error percentage :

$$e = \frac{42150}{42000} \approx 1\% < 5\%$$

The error is below 5%, the task 1 meets the specification.

```
void task1(){

  digitalWrite(TASK1, HIGH);
  delayMicroseconds(50);
  digitalWrite(TASK1, LOW);
}
```

For task 2 and 4, we read the analog input of our pin and the potentiometer value.

```
void task2(){

  input_Btn_t2 = digitalRead(INPUT_1);
}
```

```
void task4(){

  input_potentiometer_t4 = analogRead(POTENTIOMETER_PIN);
}
```

For task 3, we calculate the frequency of our square signal using an interrupt and the micro function.

```
void interrupt(){

  passedTime = actualTime;
  actualTime = micros();
}
```

Instead of using the millis's function, we have to use the micros's function. The signal can provide a frequency up to 1000Hz. The millis function will not be enough accurate.

```
void task3(){

  frequency_measured_t3 = (1/((actualTime - passedTime)*0.000001));
}
```

The frequency we are measuring is in Hz we, so we multiply our actualTime - passedTime by a small factor to display the result in Hz.
For task 5, we gather the potentiometer value using the method of a FIFO list to calculate the average of the 4 last values of the potentiometer.

```
void task5(){

  all_potentiometer_t5[3] = all_potentiometer_t5[2];
  all_potentiometer_t5[2] = all_potentiometer_t5[1];
  all_potentiometer_t5[1] = all_potentiometer_t5[0];
  all_potentiometer_t5[0] = input_potentiometer_t4;

  average_potentiometer_t5 = 0;
  unsigned short i=0;
  for (i=0;i<=3;i++) average_potentiometer_t5 += all_potentiometer_t5[i];
  average_potentiometer_t5 /= 4;
}
```

For task 6, we execute a thousand times the nop instruction.

```
void task6(){

  unsigned short j=0;
  for(j=0;j<=999;j++) __asm__ __volatile__ ("nop");
}
```

We are using a local variable unsigned short j to save memory and optimize the program.
Task 7 is a condition for the average last 4 values of the potentiometer

```
void task7(){

  if (average_potentiometer_t5 > max_potentiometer_t7/2 ) error_code_t7 = 1;
  else error_code_t7 = 0;
}
```

The respond is a boolean that will be used inside task 8 to light an LED.

```
void task8(){

  if (error_code_t7 == 1) digitalWrite(LED, HIGH);
  else digitalWrite(LED, LOW);
}
```

In the last task we send our result in a csv file format.

```
void task9(){

  Serial.print(input_Btn_t2);
  Serial.print(",");
  Serial.print(frequency_measured_t3);
  Serial.print(",");
  Serial.println(average_potentiometer_t5);
}
```

Finally, we have here the full commented code also available on GitHub in the assignment 2 folder. Linked here : `https://github.com/Emonot/Embedded_Software_Assignement2`

```
/*
Second assignement embedded software

Name : EMONOT—DE CAROLIS Evrard
ID : H00385163

*/

#include <Ticker.h> //we import the Ticker librairie for using cycling executive method

//we define our pin on the ESP32 board

#define INPUT_1 35 //pin for the input button
#define LED 14 //pin for the LED output
#define TASK1 25 //pin to send the SIGNAL B's first assignement
#define INTERRUPT_PIN 34 //pin to read the frequency of a square wave
#define POTENTIOMETER_PIN 32 //pin to read the potentiometer


//we declare our variable and constant (t* correspond to the task where the variable has been used)

bool input_Btn_t2 = 0; //state of the button we read

float frequency_measured_t3;
unsigned long actualTime = micros();//we use micros to check the current time in microseconds
unsigned long passedTime = micros();

unsigned short input_potentiometer_t4 = 0; //value of the potentiometer

unsigned short all_potentiometer_t5[4] = {0,0,0,0}; //array that will received the last 4 values of our potentiometer
unsigned short average_potentiometer_t5 = 0; //value that will contains the average of the last 4 read of the potentiometer

const short max_potentiometer_t7 = 4095; //max value we can get from our potentiometer
bool error_code_t7 = 0; //error_code depending on the value we read on the potentiometer

Ticker periodicTicker; //our ticker object
int counter = 0;


//we declare our functions, each task is a function + 2 other functions (one for the square interrupt and the other for executing a task at a certain frequence)

//we check the period of the square wave, the interrupt is called for the rising edge of the square wave.
void interrupt(){

  passedTime = actualTime;
  actualTime = micros();
}

//we send the signal B from the first assignement
void task1(){

```

```
52   digitalWrite(TASK1, HIGH);
53   delayMicroseconds(50);
54   digitalWrite(TASK1, LOW);
55 }
56
57 //we read the input value of the button
58 void task2(){
59
60   input_Btn_t2 = digitalRead(INPUT_1);
61 }
62
63 //we calculate the frequency of the square signal using the values we get from the
       interrupt
64 void task3(){
65
66   frequency_measured_t3 = (1/((actualTime - passedTime)*0.000001));
67 }
68
69 //we read the value of the potentiometer
70 void task4(){
71
72   input_potentiometer_t4 = analogRead(POTENTIOMETER_PIN);
73 }
74
75 //we calculate the average of the last 4 reads of the potentiometer
76 void task5(){
77
78   //FIFO list to gather all last potentiometer values
79   all_potentiometer_t5[3] = all_potentiometer_t5[2];
80   all_potentiometer_t5[2] = all_potentiometer_t5[1];
81   all_potentiometer_t5[1] = all_potentiometer_t5[0];
82   all_potentiometer_t5[0] = input_potentiometer_t4;
83
84   average_potentiometer_t5 = 0;
85   unsigned short i=0;
86   for (i=0;i<=3;i++) average_potentiometer_t5 += all_potentiometer_t5[i];
87   average_potentiometer_t5 /= 4;
88 }
89
90 //execute 1000 times an empty instruction
91 void task6(){
92
93   unsigned short j=0;
94   for(j=0;j<=999;j++) __asm__ __volatile__ ("nop");
95 }
96
97 //if the average of the potentiometer is higher than half of the maximum of the
       potentiometer, we put an error code to one, otherwise it is 0
98 void task7(){
99
100   if (average_potentiometer_t5 > max_potentiometer_t7/2 ) error_code_t7 = 1;
101   else error_code_t7 = 0;
102 }
103
104 //we light the LED knowing the error code
105 void task8(){
106
107   if (error_code_t7 == 1) digitalWrite(LED, HIGH);
108   else digitalWrite(LED, LOW);
109 }
110
111 //we display in a csv file format the state of the input button, the frequency of
       the square wave and the average of the last 4 read of the potentiometer
```

```
112 void task9(){
113
114   Serial.print(input_Btn_t2);
115   Serial.print(",");
116   Serial.print(frequency_measured_t3);
117   Serial.print(",");
118   Serial.println(average_potentiometer_t5);
119 }
120
121 //we check the value of our counter and we execute each task at a certain
      frequency
122 //we use the modulo operation to execute the task at each new period (for exemple,
       if the number is 1000, we execute the task every second)
123 void do_all_task(){
124
125   if (counter % 42 == 0) task1(); //period of signal B in first assignement is
      42.15ms
126   if (counter % 200 == 0) task2();
127   if (counter % 1000 == 0) task3();
128   if (counter % 41 == 0) task4();
129   if (counter % 41 == 0) task5();
130   if (counter % 100 == 0) task6();
131   if (counter % 333 == 0) task7();
132   if (counter % 333 == 0)task8();
133   if (counter % 5000 == 0) task9();
134
135   counter += 1; //we increment our counter of 1ms each time the ticker is called
136   if (counter >= 500000) counter = 0; //to avoid an overflow of the counter we
      arbitrarily
137 }
138
139
140 //in the setup function, we initialise our pin, and add our ticker
141 void setup(){
142
143   //initialisation of the serial comunication to print the values in CSV format
144   Serial.begin(115200);
145   Serial.println("input_state,frequency_value,filtered_analog_input");
146
147   //we initialise our pin as output or input, but also the interruption
148   pinMode(INPUT_1, INPUT);
149   pinMode(LED, OUTPUT);
150   pinMode(TASK1, OUTPUT);
151   pinMode(INTERRUPT_PIN, INPUT_PULLUP);
152   attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN), interrupt, RISING);
153
154   periodicTicker.attach_ms(1,do_all_task); //the ticker will called the
      do_all_task function every ms
155 }
156
157 void loop(){
158   // the loop function stays empty as an interrupt will called do_all_task every
      millisecond
159 }
160 //END
```

4 Results

For this section, some tasks requires a minimum of accuracy.

For the first task, this picture shows the period accuracy of the signal.

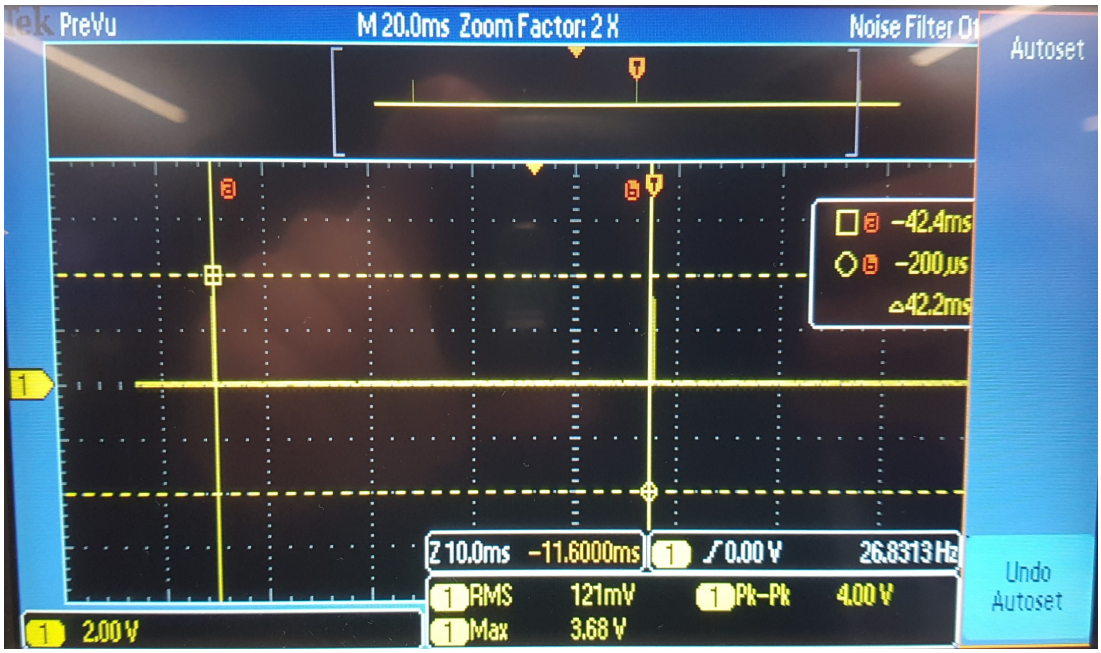


FIGURE 6 – Circuit

We find 42.2ms using cursors on the oscilloscope. This value is in the 5% accuracy of the 42150ms period .

We can then measured the frequency's accuracy of task3. For example, for a 900Hz square signal, we obtain this result.

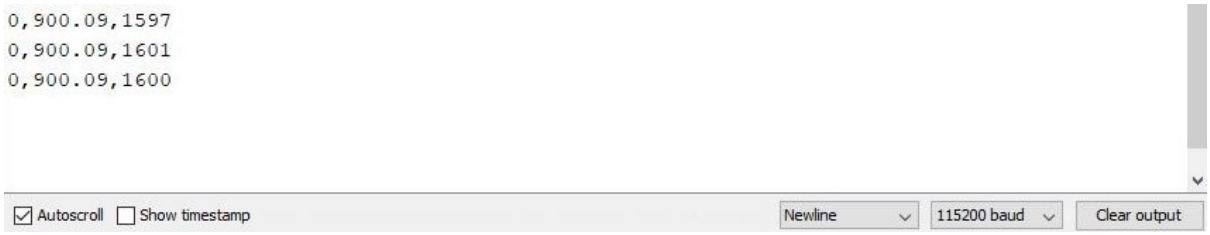


FIGURE 7 – switch input, frequency result for 900Hz, and potentiometer value (from left to right)

By calculating the error, The accuracy of task 3 is respected.

$$e = 900 \times 0.025 = 22.5$$
$$877.5Hz < 900.09Hz < 922.5Hz$$

On Figure 7, we can also see the average potentiometer value of task 5 printed in the 3rd column and the state of the input switch in the first column (task 2).

Finally we can verify the error code by turning the potentiometer and checking the LED. The results are shown in those 2 photos :

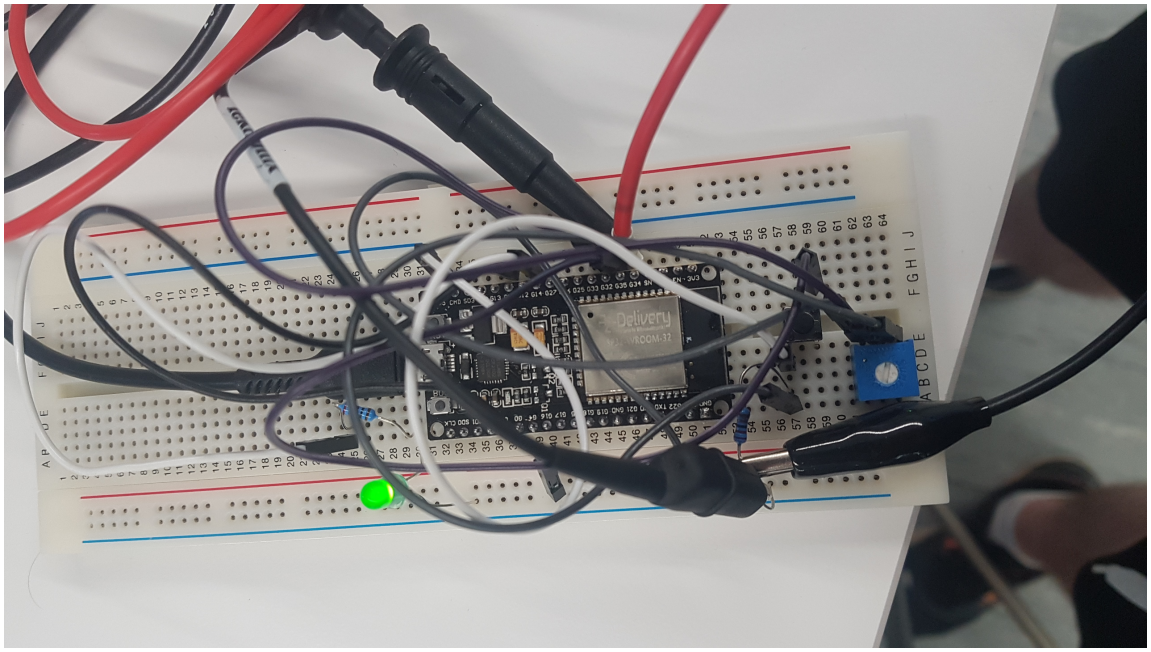


FIGURE 8 – LED state when $errorcode = 1$

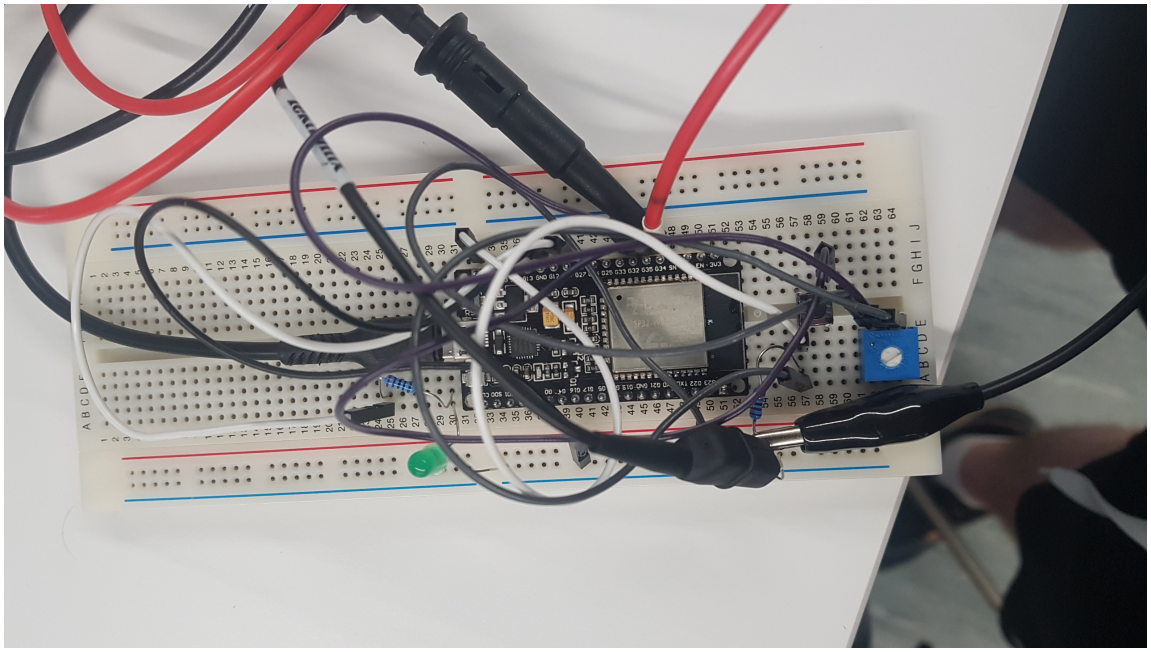


FIGURE 9 – LED state when $errorcode = 0$

The first image shows the state of the LED when the average of the last 4 reads of the potentiometer egal or over 2048.
The second image shows the state of the LED when the average of the last 4 reads of the potentiometer are are below 2047.