**B31DG Assignment 2 Report.**

**Contents:**

1. **Introduction.**
2. **Components.**
3. **Circuit design.**
4. **Tasks.**
   1. **Task 1.**
   2. **Task 2.**
   3. **Task 3.**
   4. **Task 4.**
   5. **Task 5.**
   6. **Task 6.**
   7. **Task 7.**
   8. **Task 8.**
   9. **Task 9.**
5. **Schedule.**
   1. **Initial Thoughts.**
   2. **Final Schedule.**
6. **Cyclic executive.**
   1. **General format.**
   2. **Exceptions.**
   3. **Issues/Changes.**
7. **Results.**
   1. **Task 1.**
   2. **Task 2.**
   3. **Task 3.**
   4. **Task 4/5/8.**
8. **Github.**
9. **Introduction.**

The aim of this project is to create a cyclic executive containing 9 different tasks with variable rates per second.

|  |  |  |
| --- | --- | --- |
| **Task Number** | **Rate (Hz)** | **Function** |
| 1 | 30 | Output a 50µs pulse |
| 2 | 5 | Monitor one digital input. |
| 3 | 1 | Measure the frequency of a 3.3v squarewave signal, 50% duty cycle and range of 500Hz-1000Hz. |
| 4 | 24 | Read 1 analogue input. |
| 5 | 24 | Compute an average of the past 4 analogue readings. |
| 6 | 10 | Execute “\_\_asm\_\_ \_\_volatile\_\_ ("nop")” 1000 times. |
| 7 | 3 | Performs the following check:  if (average\_analogue\_in > half of maximum range for  analogue input):  error\_code = 1 else:  error\_code = 0 |
| 8 | 3 | Turns LED on if error\_code = 1. |
| 9 | 0.2 | Prints to the serial monitor:  • State of the digital input (pushbutton / switch); • Frequency value (Hz, as an integer); • Averaged analogue input. |

1. **Components.**

* Breadboard.
* ESP-32.
* Push Button.
* 2 x LED.
* 2 x 220 Ohm resistors for LEDs.
* 1 x 10k Ohm resistor for the button.
* Potentiometer.

1. **Circuit Diagram.**

**A picture containing text, electronics, circuit

Description automatically generated**

Figure Photo and diagram of circuit.

1. **Tasks.**
   1. **Task 1.**

Output a 50µs pulse. Trivial to implement.

Graphical user interface, text, application

Description automatically generated

* 1. **Task 2.**

Monitor one digital input. Again, trivial to implement.

A picture containing logo

Description automatically generated

* 1. **Task 3.**

Measure the frequency of a 3.3v squarewave signal, 50% duty cycle and range of 500Hz-1000Hz.

This was the trickiest task to implement.

Chart, box and whisker chart

Description automatically generated

We start the task at *t0*. We save the state (low or high). We then check the state of the square wave until it changes, this is the point *t1*. We save the time of this change, then check the state again until the state changes again. This is the point *t2* and we save this time.

Now to calculate the frequency. The formula is: .

However we do not yet have the pulse length, the pulse length is actually *t3-t1*. We do know the duty cycle of 50% so we can calculate the pulse length as *2(t2-t1).*

Therefore, .

Text, letter

Description automatically generated

* 1. **Task 4.**

Read 1 analogue input. Back to a trivial implementation.

**Logo

Description automatically generated**

* 1. **Task 5.**

Compute an average of the past 4 analogue readings. This is quite obvious and easier to understand viewing the code than explaining in text.

Text, letter

Description automatically generated

* 1. **Task 6.**

Execute “\_\_asm\_\_ \_\_volatile\_\_ ("nop")” 1000 times. Simple *for* loop.

**Text

Description automatically generated with medium confidence**

* 1. **Task 7.**

Upon testing the potentiometer, the range is 0 to 4095. For half we round up to 2048.

Text

Description automatically generated

* 1. **Task 8.**

Turns LED on if error\_code = 1. Trivial.

Text

Description automatically generated

* 1. **Task 9.**

Prints to the serial monitor:

• State of the digital input (pushbutton / switch);

• Frequency value (Hz, as an integer);

• Averaged analogue input.

**Text

Description automatically generated**

1. **Schedule.**

Find the full schedule on the github/submitted excel spreadsheet named [Assignment 2 schedule.xlsx](https://github.com/DavidRhysGemmell/B31DG-Assignment-2/blob/main/Assignment%202%20schedule.xlsx) (Feel free to adjust the Slots/second in F16, the colours will automatically move to show clashes.)

* 1. **Initial Thoughts.**

So to start a cycle executive we need to find a suitable number of *slots* so that there are enough for each task to be computed the required number of times.

My Task 1 cycle length was initially 33750µS, which is 29.629Hz. This does not play well with the other frequencies of 3,5,10,24 etc. Therefore I rounded Task 1 up to 30 Hz. Then taking the LCM we get 120.

120 steps is technically schedulable, there are many clashes between tasks wanting to be computed at the same time (see fig). As we are using a cyclic executive rather than a ROS, we would have to consider every clash that occurs. Using 720 steps however naturally leaves us with no clashes.

Table

Description automatically generated with medium confidence

Figure Taken from Schedule excel spreadsheet with 120 steps. When colours are in the same column, there is a clash.

720 steps does come with one issue. The time taken for a single slot would be 1388µS. In Task 3, the time for a single pulse of a square wave of frequency of 500Hz is 2000µS. Under ideal conditions it is possible as our method calculates the frequency using half of the pulse, however, it would not be ideal conditions and if the first change is over 387µS into the task, there will not be enough time within the slot to measure the frequency. The solution for this was simple, just use 2 slots, one after another.

Note for other tasks such as Task 6, I checked the computation time using a script in the github called Task6TimeTakenToCompute.ino.

* 1. **Final Schedule.**

We settle on the number of slots per second as 720. To know what task to complete, we divide the slot number by rate of each task and the remainder will tell us which task to compute.

|  |  |  |
| --- | --- | --- |
| **Task Number** | **Divisor** | **Remainder** |
| 1 | 24 | 0 |
| 2 | 144 | 1 |
| 3 | 720 | 2,3 |
| 4 | 30 | 4 |
| 5 | 30 | 5 |
| 6 | 72 | 6 |
| 7 | 240 | 8 |
| 8 | 240 | 8 |
| 9 | 3600 | 3599 |

There are no clashes between tasks so there is no need for any slot-shifts.

For all other slots, not tasks are computed.

1. **Cyclic Executive.**
   1. **General format.**

Now we know our schedule, we can implement it into our cyclic executive. The executive is within the Void Loop function. The following is the format for each task, with a couple of exceptions.

Text

Description automatically generated

The comments explain how this works suitably. The final delay makes sure that this slot completes the full slot length before moving on to the next slot.

* 1. **Exceptions.**

Task 3:

Text

Description automatically generated

Task 3 takes 2 slots. Therefore the differences are 2(slotlength) and added 2 to the slot count.

Task 4:

Graphical user interface, text

Description automatically generated

Another requirement was to set an output pin for the duration of Task 4 such that we can visualise execution times on an oscilloscope.

Task 9:

Graphical user interface, text

Description automatically generated

This is at the end of the cycle. I have an extra delay that makes up for any errors/remainders as my 1386µS slot length is a slight underestimate. Also we reset the Slot to 0 to start again.

* 1. **Issues/Changes.**

Although I did use *if, if else* statements to check each condition, I believe if I were to write it again, I may use the *case* statement instead.

Another issue is that if there is no square wave input, the cyclic executive gets stuck in Task 3 waiting for a change in state that isn’t coming. The solution for this would be to have timings within the task such that if the time taken > slot length then outputs an error and terminates the function.

1. **Results.**
   1. **Task 1.**



Figure Task 1 output on an oscilloscope.

* 1. **Task 2.**

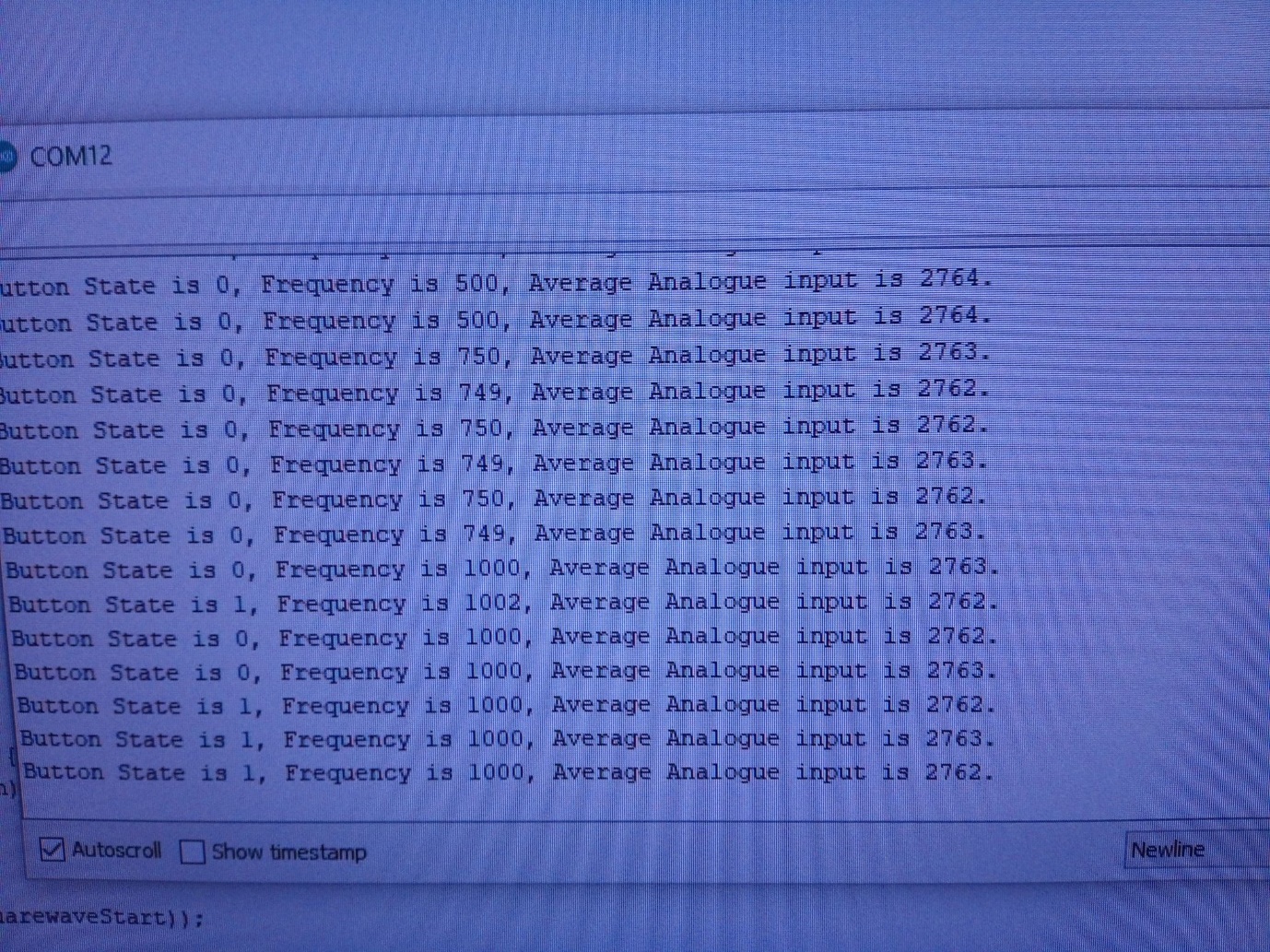


Figure Change in button state when the button is pushed. Shown on serial monitor.

* 1. **Task 3.**

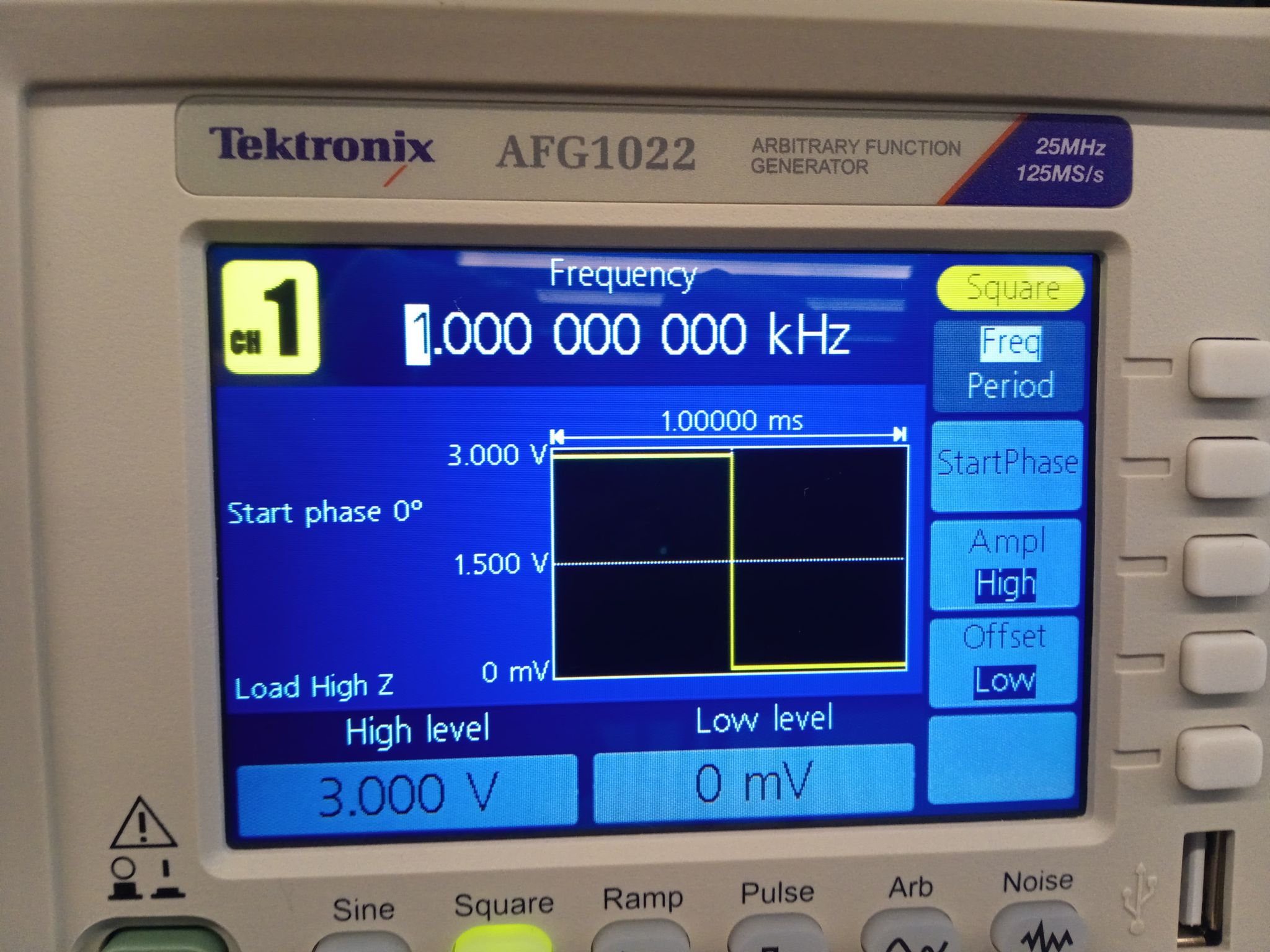


Figure 1000Hz square wave set.

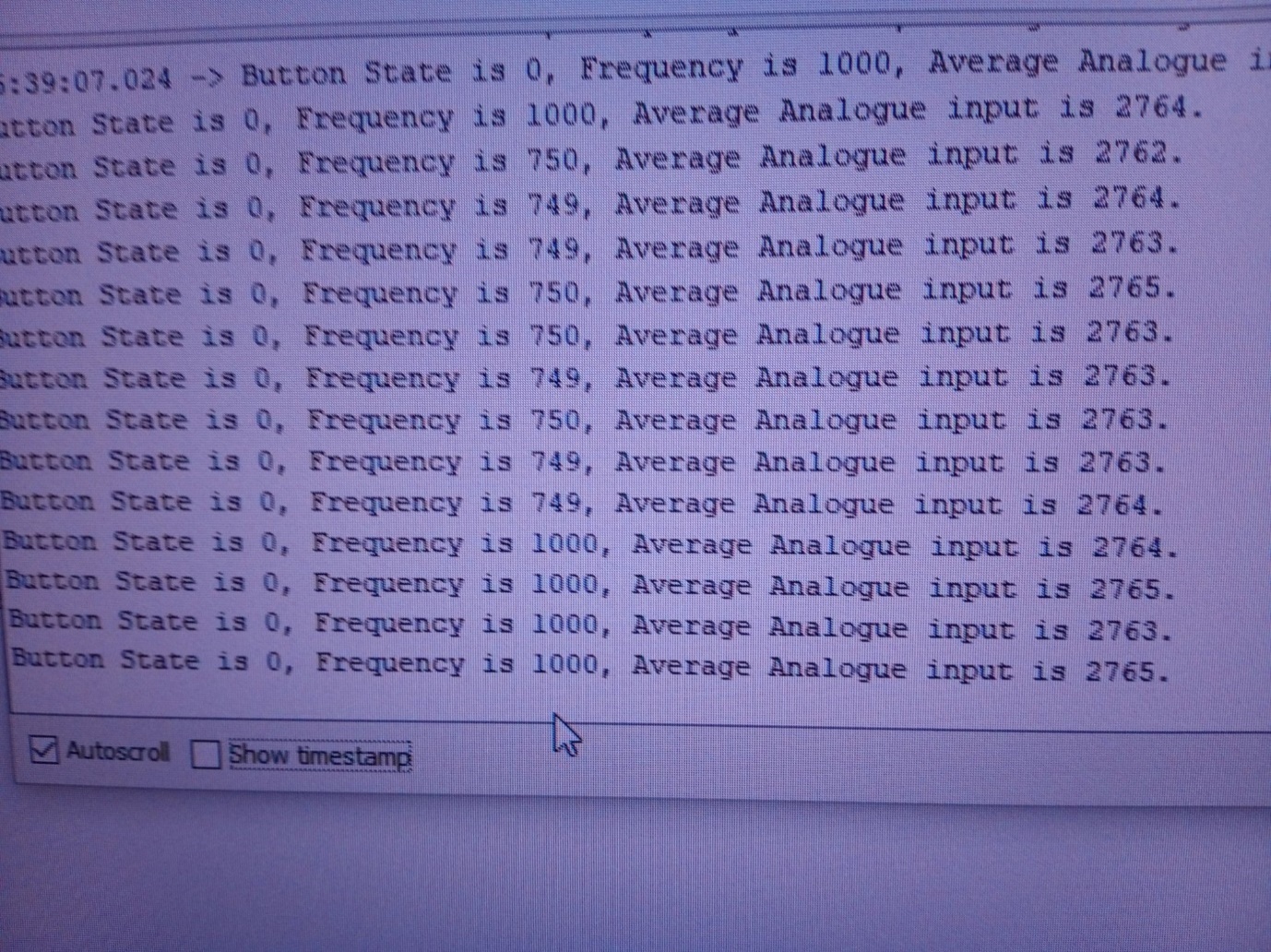


Figure Serial Monitor, 1000Hz measured.



Figure 750Hz square wave set.

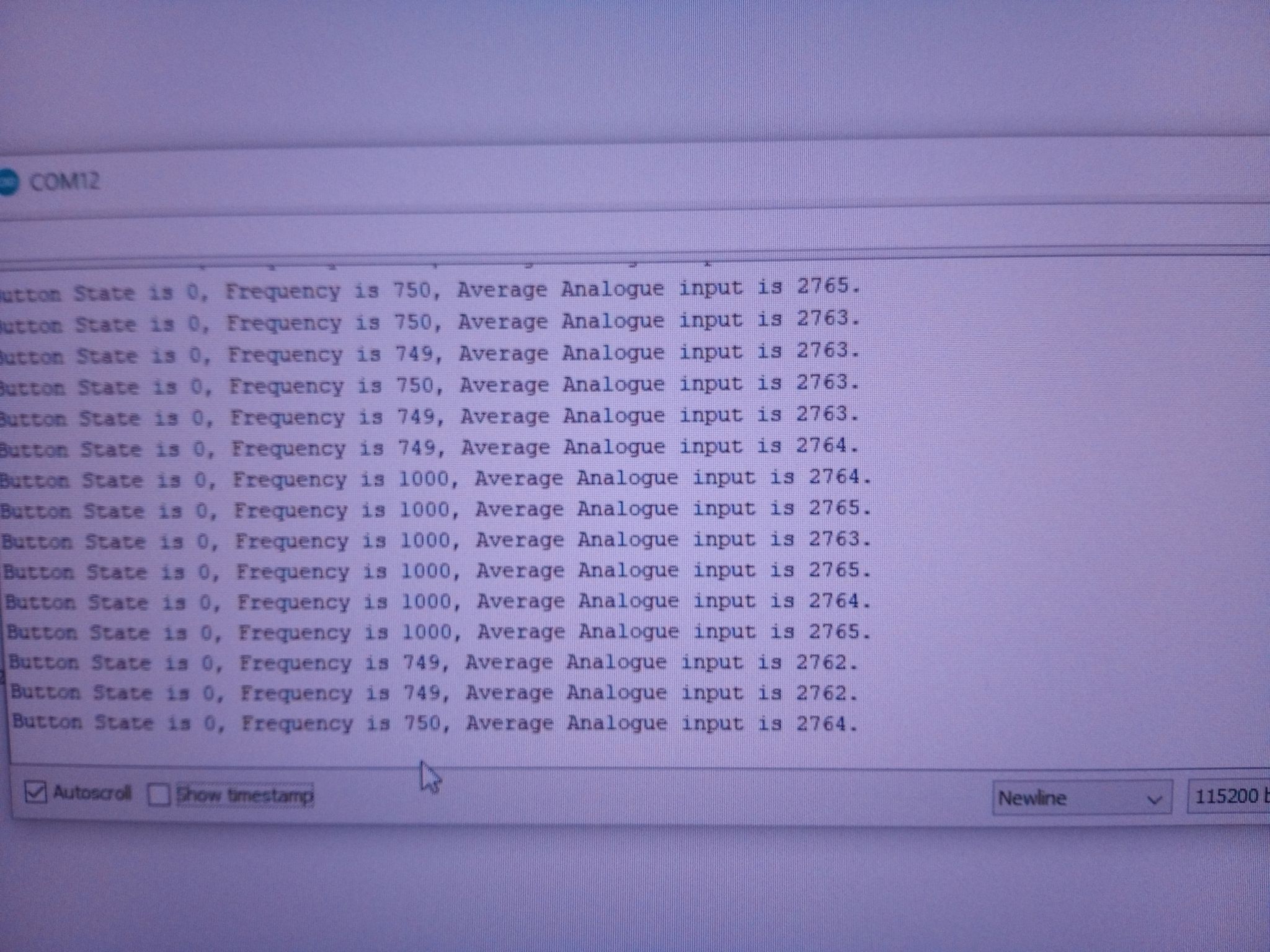


Figure 750Hz frequency measured on serial monitor.

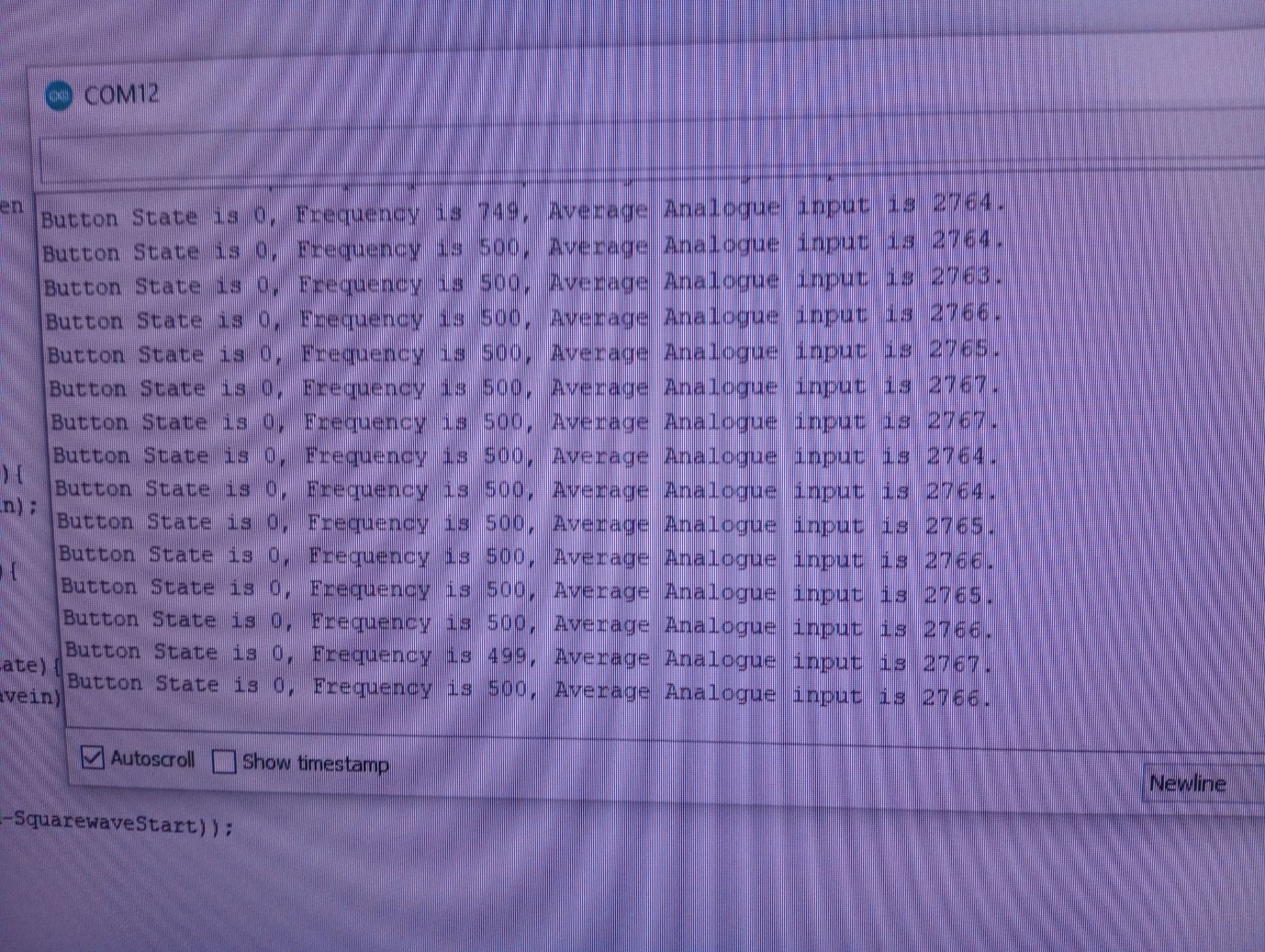


Figure 500Hz signal measured.

Both extremes are shown to be working well. Confident it can measure the full range 500-1000Hz accurately.

* 1. **Task 4/5/9.**



Figure Showing the output of the computation time of task 4.

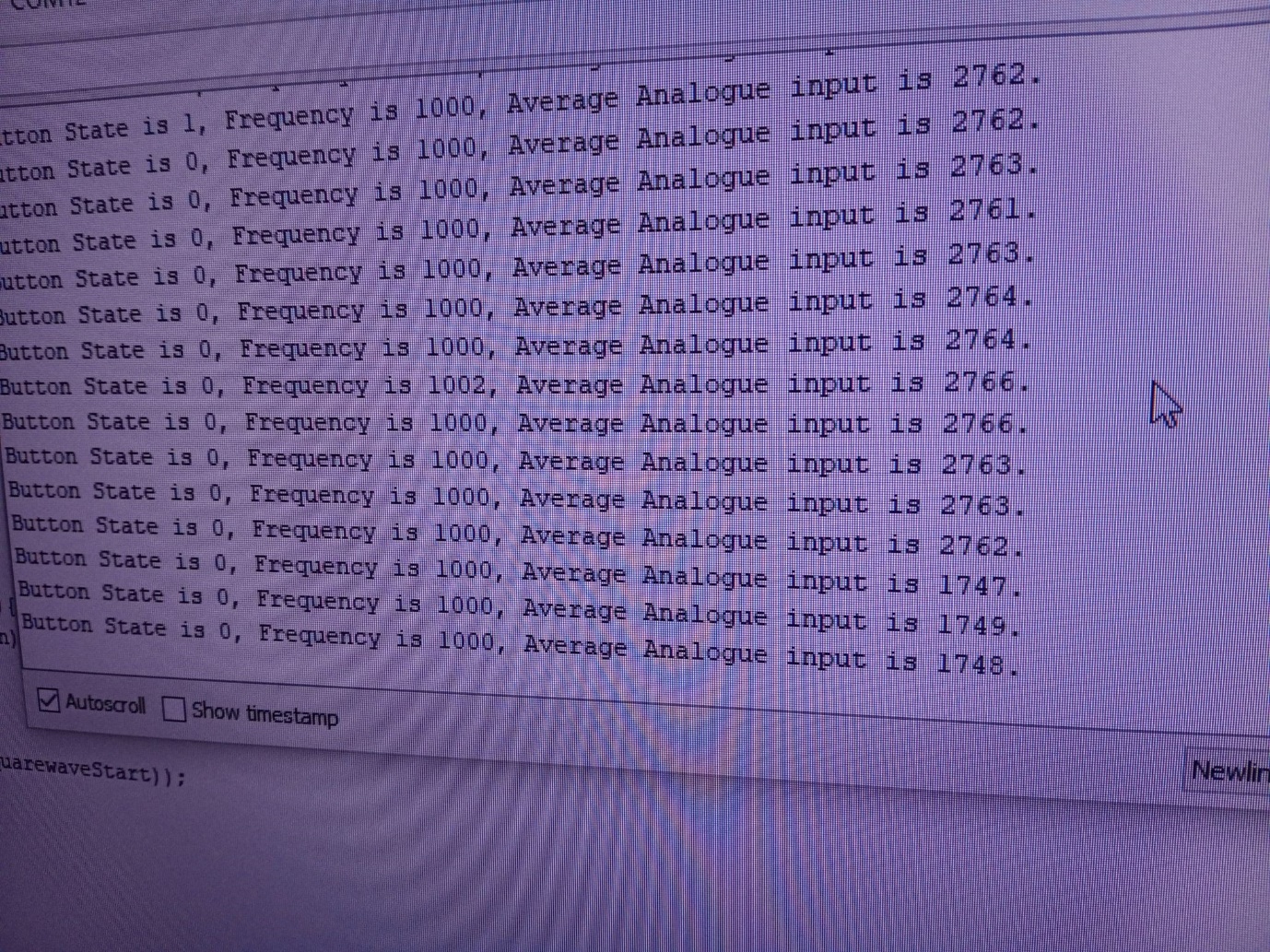


Figure Showing serial monitor, note analogue input < 2048

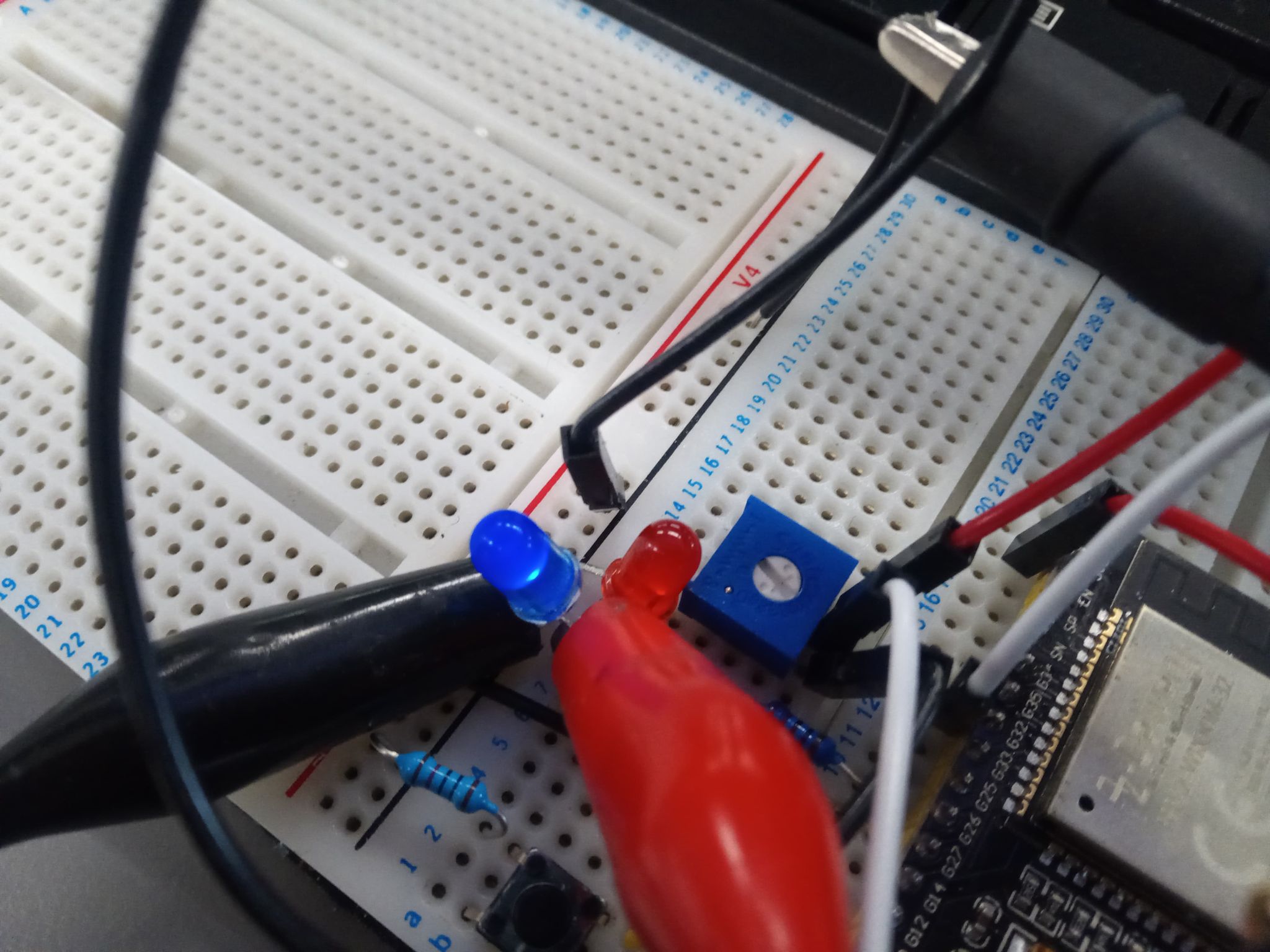


Figure Red LED is off for analogue input < 2048.

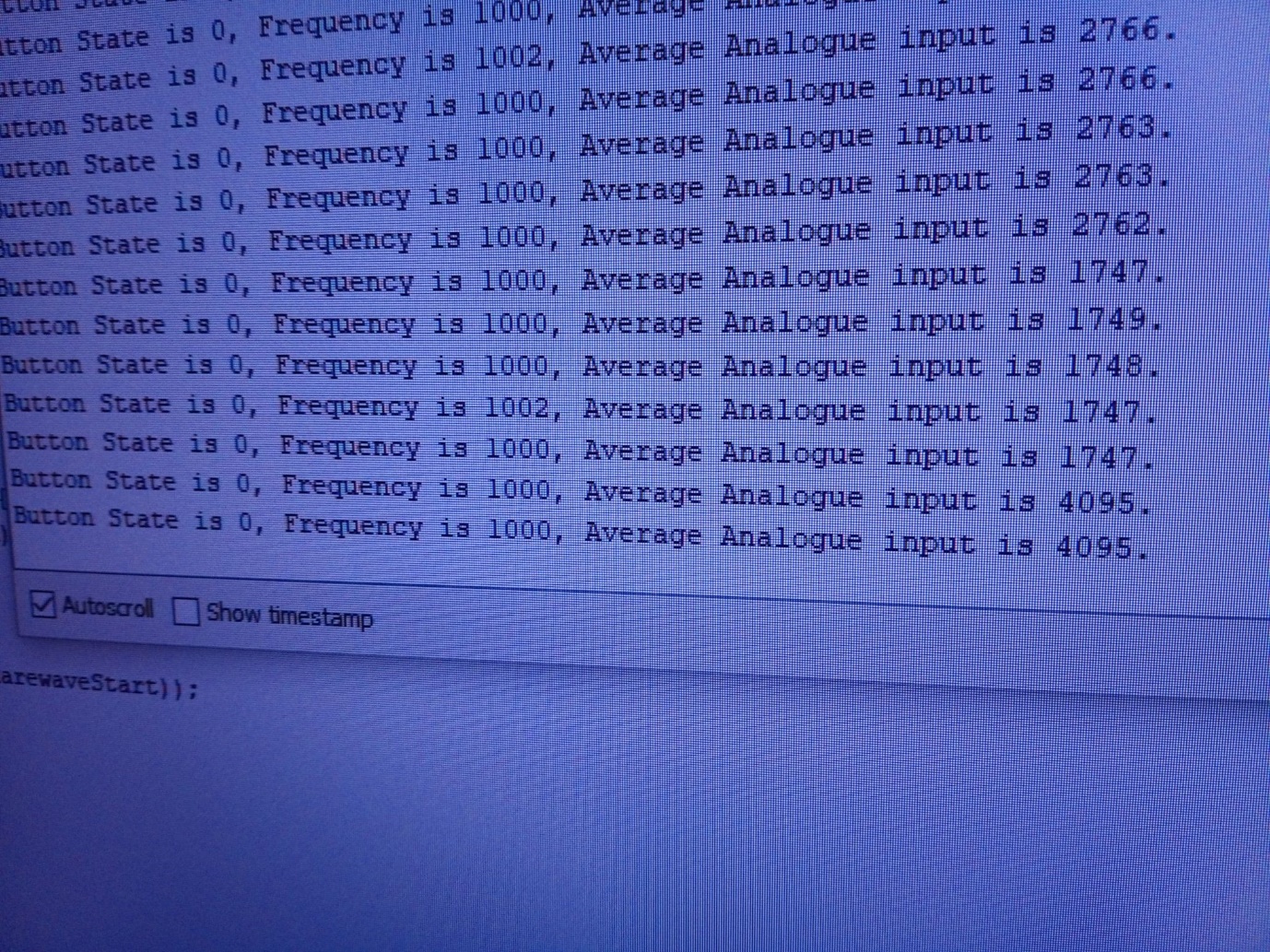


Figure Potentiometer turned up

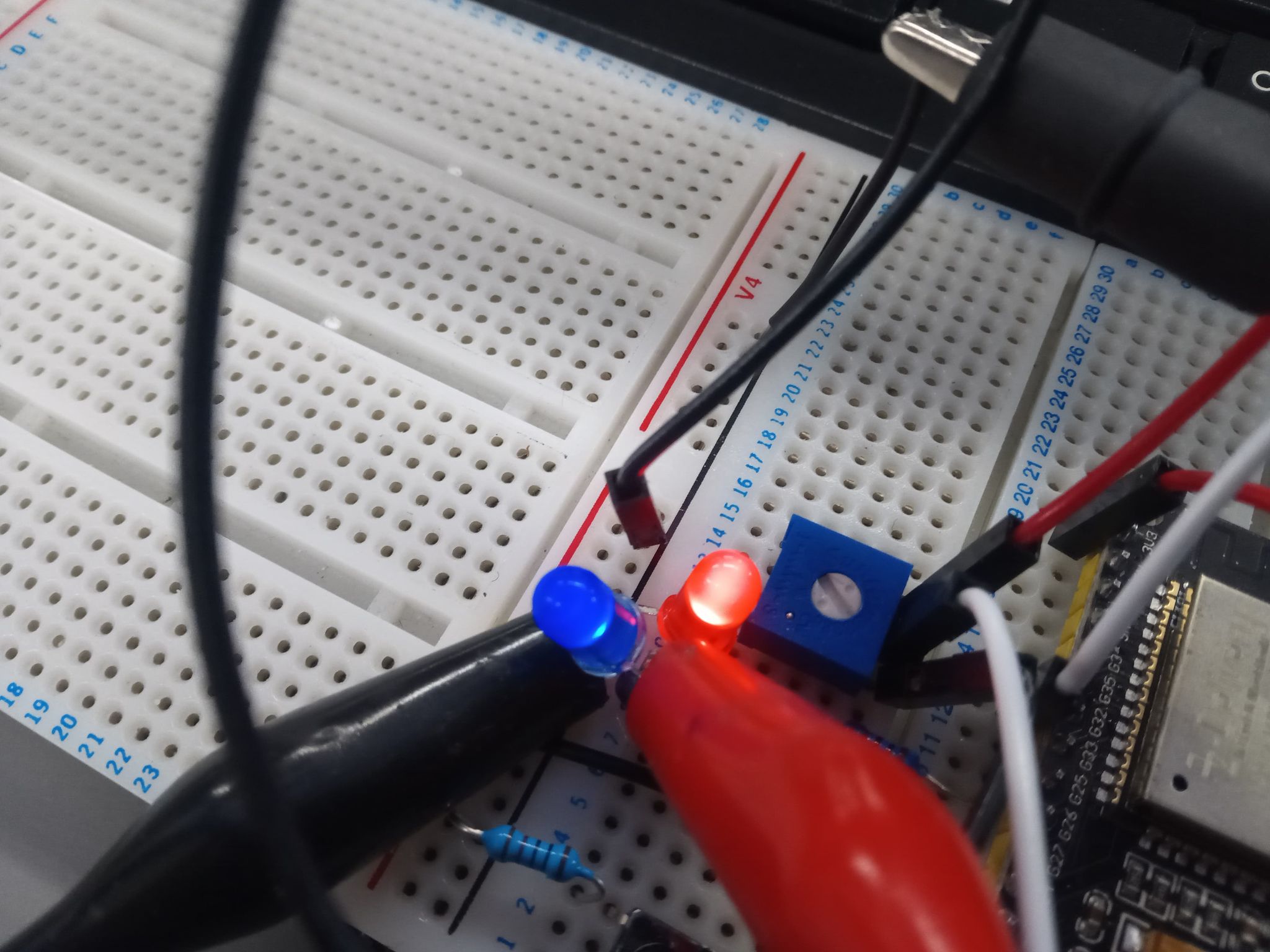


Figure Red LED on for analogue input > 2048.

1. **Github.**

<https://github.com/DavidRhysGemmell/B31DG-Assignment-2>

Graphical user interface, text, application, Teams

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Figure Note 1 more commit to be made, uploading this document.