**B31DG Assignment 1 Report.**

**Contents:**

1. **Introduction.**
2. **Components.**
3. **Circuit design.**
4. **Input Variables.**
5. **Switch Settings.**
6. **Timing.**
7. **Oscilloscope.**
8. **Improvements.**
9. **Github Link.**
10. **Introduction.**

The aim of this project is to use an ESP-32 to create pulses shown in Figure 1.

Diagram

Description automatically generated

Figure 1

Text

Description automatically generated

Figure 2

1. **Components.**

This project uses the following components.

* Breadboard.
* ESP-32.
* 2x Buttons
* 4x male-male wires
* 4x LEDs of different colours.
* 4x 220 Ohm resistors for LEDs
* 2x 10k Ohm resistors for Buttons.

1. **Circuit Diagram**

Diagram, schematic

Description automatically generated

Figure 3 Example Wiring Diagram

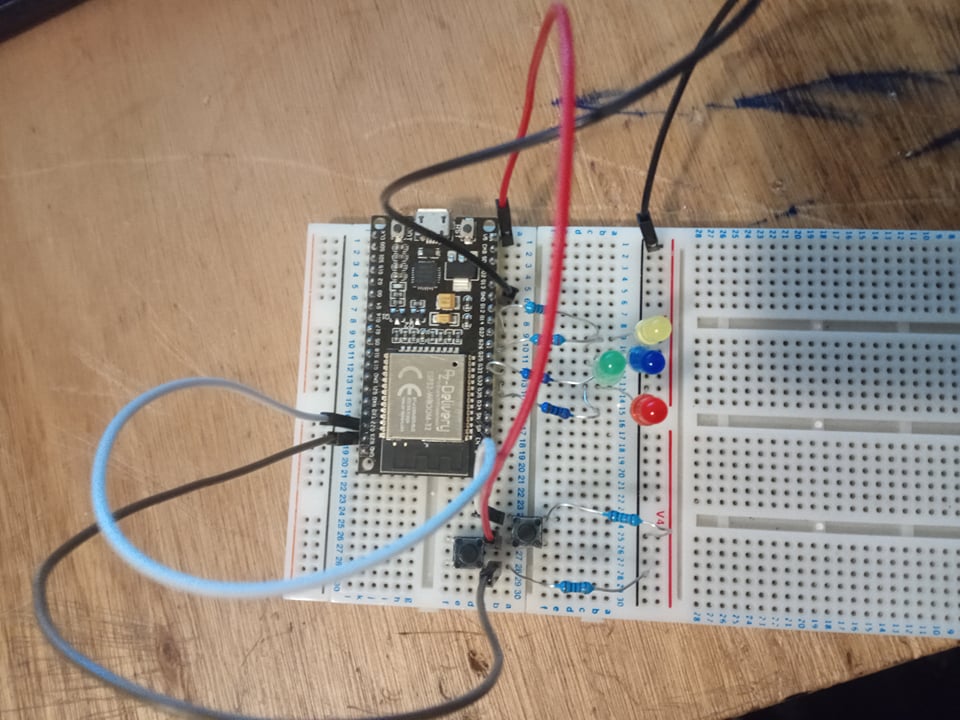


Figure 4 My circuit

1. **Input Variables.**

The input variables for *a, b, c*, *d,* and the modified SIG A waveform cycle are based on the first 5 letters of my surname; GEMME. Therefore I have;

* *a* = 700,
* *b* = 500,
* *c =* 17,
* *d* = 6500
* and I will be using mode 2 of the assignment which means my modified waveform will be an inverted form of SIG A where the pulses go from largest to shortest.

1. **Switch Settings.**

I have used 2 Switches, toggled from one state to the other by pressing the relevant button.

**Button Issues.**

I had some issues with the buttons, noticeably, when the button pins were connected it lead to the red LED flickering. I solved this by connecting the button pin to ground with a 10k resistor.

I have also added an extra LED for each switch, used to visualise the states and what should be expected.

|  |  |  |
| --- | --- | --- |
| **Switch Settings.** | **= 0** | **= 1** |
| Switch 1 | Enables Pulses in SIG A  Blue LED off | Disables Pulses in SIG A  Blue LED on |
| Switch 2 | Pulse length goes from smallest – largest.  Yellow LED off | Pulse length goes from largest – shortest.  Yellow LED on |

Text

Description automatically generated with medium confidence

Figure 5 Code showing how I used the buttons to change the state

However another condition is such that, when the switch setting changes, the effect only takes place after the current waveform cycle has completed. So a press of either button should only change the state on the next cycle. So I introduced a new variable that locks in the current switch setting at the start of each cycle, leaving the switch settings able to change without restarting the current cycle.

A screenshot of a computer

Description automatically generated

Figure 6 Code locking in the switch settings at the start of each waveform cycle.

**Alternate**

This gives 3 states for SIG A; off, normal waveform and modified waveform. Therefore, this could be altered and visualised using a single button and LED to visualise the states;

|  |  |  |  |
| --- | --- | --- | --- |
| **Switch Setting** | **0** | **1** | **2** |
| *a* Pulse | Normal Waveform Cycle | Modified Waveform Cycle | off |
| Blue LED | Off | Flashing | On |

1. **Timing.**

We need to use some inbuilt functions to measure the pulses lengths, the waveform cycle, the gaps between pulses and the gap at the end of the cycle. Initially I used the delay function so solve this.

Graphical user interface, text, application, email

Description automatically generated

Figure 7 Example where I used the delay function.

However this pauses the code whenever the delay function is “active”. Therefore I used the micros() and millis() functions to read the time since the loop started. From this I created variables, PreviousMicrosWaveform and PreviousMicrosa which are used to calculate how long the waveform has been going on for and how long each pulse has been going on for.



Figure 8 Calculates how long has elapsed in the current cycle, compares it to how long the waveform should be.

1. **Oscilloscope.**

I have used an oscilloscope to visualise the outputs as at the microsecond scale it is impossible to measure using our eyes and LEDs.



Figure . Modified waveform cycle visualised on an oscilloscope.

Note the blue line labelled “2” is Sig B and the yellow line showing Sig A. Notice the 17 pulses in the yellow line, starting large, separated by constant 500µS gaps, and gradually getting smaller until the last pulse which is 700µS long. Then a 6500µS gap before the cycle restarts. Times recorded were within 1-2% of the values intended. I consider this successful.



Figure . Switch 1 setting 0, no pulse for Sig A.

Again, showing what was intended with Sig A outputting no pulse when switch 1 = 0.

1. **Improvements.**

The waveform cycle length was calculated manually on matlab, this could however be implemented in the script;

int aw=a;

For (i=0; i=c-1; i++){

aw=aw+(i\*50);

}

With the way the button presses have been implemented, they can be slightly temperamental. I believe this to be when the button is pressed, it rapidly switches between switch settings until the button has been released. A solution to this would be to change the switch state only once no matter how long the button is pressed down, or potentially to use the average “tap” length so that the switch state can only change twice per second or so if the button is held down.

1. **Github link.**

<https://github.com/DavidRhysGemmell/B31DG-Assignment-1>