B31DG-Assignment1-Report

Student Name: Shiyu Lu

Student ID: H00364918

Email: sl2031@hw.ac.uk

Major: Robotics

Grade: Year 4 Undergraduate

Date:03/03/2024

catalogue

Revision History

Revision	Date	Author	Note
V1.0	01/03/2024	Shiyu Lu (Andy)	Initial version ---- Complete the main content
V2.0	03/03/2024	Shiyu Lu (Andy)	Second version ---- Updated the interpretation analysis of experimental results and the code flow chart

1. Calculation of Output Timing Parameters

My surname is: Lu

So the five numbers we need in our program correspond to the five letters: luuuu

As shown in these tables, we can define our five numbers:

Letter	Letter	Numerical Mapping
a	z	1
b	y	2
c	x	3
d	w	4
e	v	5
f	u	6
g	t	7
h	s	8
i	r	9
j	q	10
k	p	11
l	o	12
m	n	13

Table 1: Alphanumeric Mapping

So L corresponds to 12; u corresponds to 6

Parameter	Definition
a	First Letter Numerical Mapping x 100us
b	Second Letter Numerical Mapping x 100us
c	Third Letter Numerical Mapping + 4
d	Fourth Letter Numerical Mapping x 500us

Table 2:Output Timing Parameter Definitions (for Normal Waveform)

a=l*100μS=12*100μS=1200μS

b=u*100μS=6*100μS=600μS

c=u+4=6+4=10

d=u*500μS=6*500μS=3000μS

The mode is determined by this formula:

$$\boldsymbol{Option\ Number\ =\ (Fifth\ Letter\ Numerical\ Mapping\ \%\ 4)\ +\ 1}$$

mode=remainder(u/4)+1=3

Option	Description
1	Remove the final 3 pulses from each data waveform cycle (i.e. c-3 pulses in a data waveform cycle) until the Output Select push button is pressed again.
2	Generate a reversed form of the data waveform (from the largest pulse to the shortest) until the Output Select push button is pressed again.
3	Insert an extra 3 pulses into each data waveform cycle (i.e. c+3 pulses in a data waveform cycle) until the Output Select push button is pressed again.
4	Half the **b** and **d** time intervals until the Output Select push button is pressed again.

Table 3:Definition of Possible Alternative DATA Output Behaviours

In conclusion: a=1200μS ; b=600μS; c=10; d=3000μS; mode=3

2. The application controls the flow & Flowchart

2.1 Flowchart

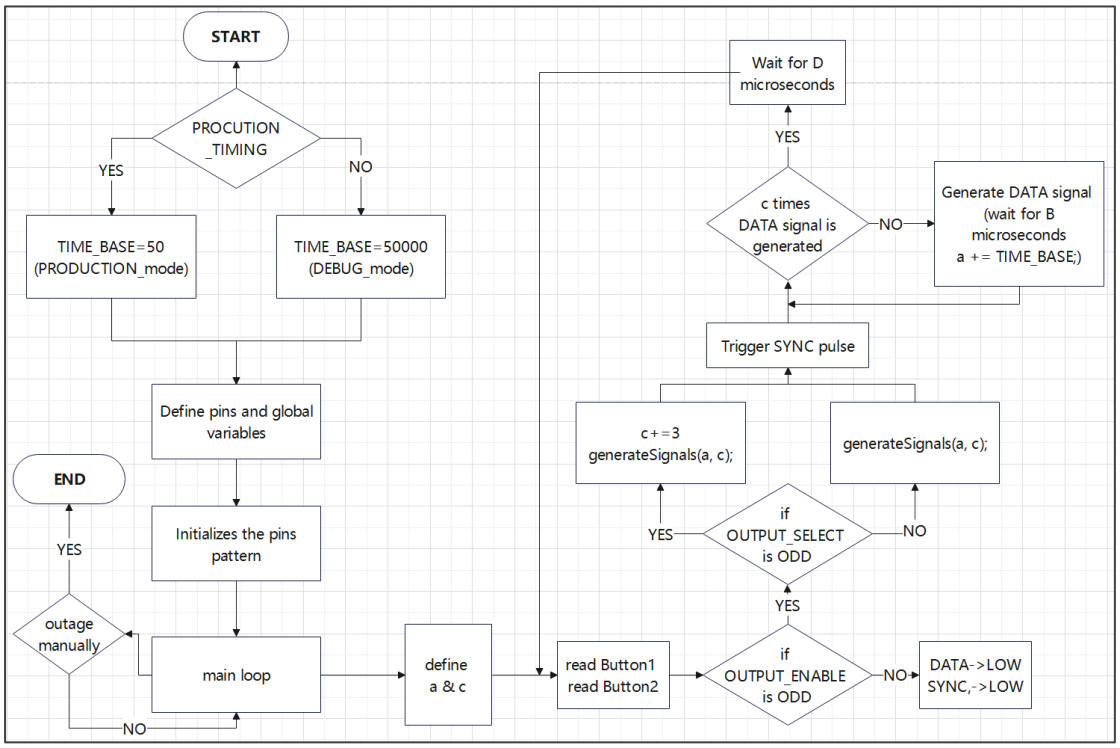

Table 4: Flowchart of the code

2.2 Code logical

Define part

First, the program can be run in two modes: **production mode and debug mode.**
The desired pattern can be selected with the preprocessing directive '#define PRODUCTION_TI MING'.
If **'PRODUCTION_TIMING'** is defined, **the 'TIME_BASE' macro is defined as 50** and enters production mode. **Otherwise, 'TIME_BASE' is defined as 50000**, and the program runs in debug mode.

```
#define A 24*TIME_BASE
#define B 12*TIME_BASE
```

```
#define C 10
#define D 60*TIME_BASE
```

Because several key time delays are defined by (xxμs*TIME_BASE), so TIME_BASE is changed from 50 to 50000 the time unit of the delay is also changed from μS to ms. So in this case, the blinking state of the LED light can be observed, we can not rely on the oscilloscope to observe the waveform, Instead, the basic function of the program is judged by the blinking of the LED light.

Setup part

In the setup phase of the program (the 'setup()' function), several pins are defined for different functions: two input pins for reading the state of the button (' Button1 'and' Button2 '), and two pins for output signals (' DATA 'and' SYNC '). These pins are initialized to the corresponding pattern for subsequent use.

```
  pinMode(Button1, INPUT_PULLDOWN); // Using internal pull-up resistors
  pinMode(Button2, INPUT_PULLDOWN); // Using internal pull-up resistors
  // In order to add a resistor instead of manually connecting two
resistors
```

Note here that I set the mode of Button1 and Button2 to INPUT_PULLDOWN. This allows you to use the built-in pull-up resistor instead of manually connecting two resistors to an external circuit. And using a built-in pull-down resistor ensures that when the button is not pressed, the pull-down resistor will ensure that the pin remains in a low state, providing a stable logical "0" signal. This is a better way to implement the required circuit diagram.

generateSignals() part

The signal generation function (' generateSignals() ') generates a signal based on the given pulse duration (' a ') and the number of pulses (' c '). It first generates a synchronization signal (' SYNC '), and then cycles to generate a DATA signal (' data '). There is a fixed delay between each data pulse (defined by 'B'), when the number of existing pulses is less than c, new pulses will be iterated, and the duration of each pulse updates the value of A after each iteration, each time incrementing 'TIME_BASE'. After each set of waveforms is completed, there is a longer delay (defined by D).

The benefit of defining this function is that the generateSignals() function encapsulates the logic of signal generation, making the code more modular. The variables a and c are updated after each iteration and passed into the function as input. In the call of the main function in different modes, the number of pulses c is different, you can only change the value of c, and then pass the generateSignals() as input, the code logic is clear and simple, easy to operate.

```
void generateSignals(int a, char c) {
  char i; // Declare the counter variable locally

  digitalWrite(SYNC, HIGH);
  delayMicroseconds(SYNCON);
```

```
  digitalWrite(SYNC, LOW);

  // Create iterative sequence of pulses for DATA(Signal A)
  for (i = 0; i < c; i++) {
    // Create rectangular pulse
    digitalWrite(DATA, HIGH);
    delayMicroseconds(a);
    digitalWrite(DATA, LOW);

    delayMicroseconds(B);
    a += TIME_BASE;
  }

  delayMicroseconds(D); // Wait for D microsecond for each full waveform
}
```

main_loop part

The main loop(the 'loop()' function) is the core of the program, which reads the button signal at the beginning, checks the state of the button, and changes the pattern of the output signal based on the number of times the button is pressed.

- When 'Button1' is pressed, the output enable flag (' OUTPUT_ENABLE_FLAG ') is increased. **If the flag is odd**, the program generates a signal; **If it is even,** the program will stop the signal output and set the DATA and SYNC pins to low. That is, pressing Button1 when the program is started will trigger the desired waveform, and pressing it again will stop. If you want to trigger the waveform again, press the Button1 button again and the process can be repeated. This switch has memory, do not need to hold the button, press the next time, the program will complete the switch between output and stop.

```
  if (OUTPUT_ENABLE_FLAG % 2 == 1) {
   ……
    Generate signal normal mode or mode3
   ……
    }
  } else if (OUTPUT_ENABLE_FLAG % 2 == 0) {
    digitalWrite(DATA, LOW);
    digitalWrite(SYNC, LOW);
  }
```

- When 'Button2' is pressed, the output selection flag (' OUTPUT_SELECT_FLAG ') is increased. If 'OUTPUT_ENABLE_FLAG' is odd, and 'OUTPUT_SELECT_FLAG' is also odd, the program enters mode 3, which means that the variable 'c' is increased by 3, affecting the number of signal sequences.
  If OUTPUT_SELECT_FLAG is an even number, the program runs in normal mode without changing the value of c. This switch is memorized and does not require holding down the button all the time. Similarly, when the program starts, it defaults to normal mode and switches

to mode3 when Button2 is pressed. If you want to return to normal mode again, press Button2 again and the process can be repeated.

```
if (OUTPUT_ENABLE_FLAG % 2 == 1) {
    // Generate signals when Button1 is pressed by odd times
    // Check if Button2 is pressed
    if (OUTPUT_SELECT_FLAG % 2 == 1) {
      c += 3; // Mode3: Increment c by 3
      // run mode3 when Button2 is pressed by odd times
      generateSignals(a, c);
    }
    else{
      generateSignals(a, c);
      // if the output_select is not odd, we run the normal mode
    }
```

3. The use of ESP-IDF

Firstly we download the ESP-IDF as the extension of vscode. Then we configure the extension with the EXPRESS(faster option). Then we create a new project of ESP-IDF, then we can change our program to fit the extension in the main.c

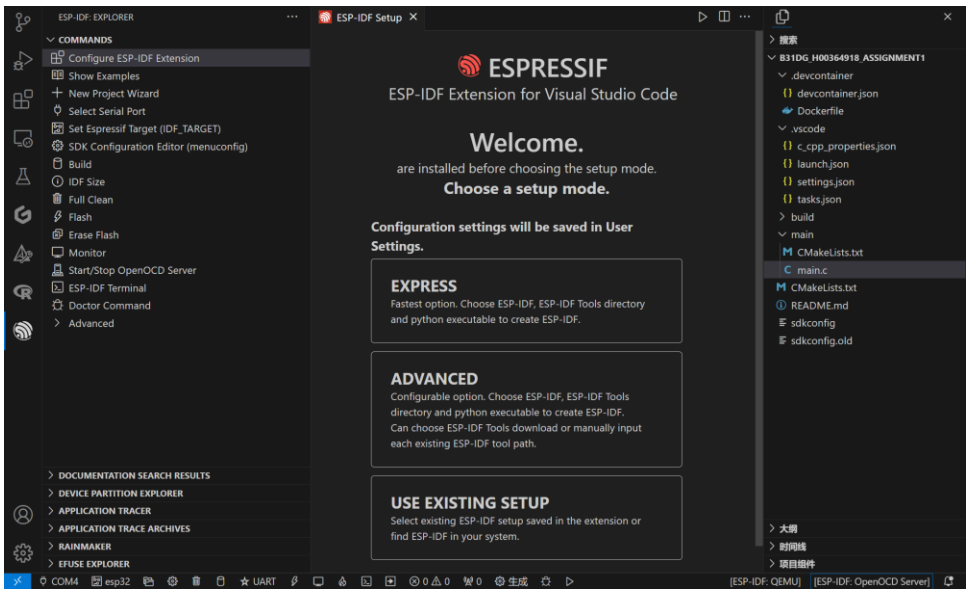

Figure 1:The IDF setup and the new project structure

The main things we should change:

1. Import libraries necessary for C code, which is not required for writing in Arduino

We can create a **new ".h" header file** where these references can be included in，and use `#include "project_includes.h"`to import these libraries in main.c

```
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "driver/gpio.h"
#include "esp_log.h"
#include "sdkconfig.h"
#include "rom/ets_sys.h"
```

2. Instead of using the pinMode function to set the pin mode, use the gpio_set_direction function to set the input or output mode of the pin, and use the gpio_set_pull_mode function to configure the pull-up/pull-down resistance of the pin.

```
gpio_set_direction(Button1, GPIO_MODE_INPUT);
gpio_set_pull_mode(Button1, GPIO_PULLDOWN_ONLY);
```

3. Change the digitalRead and digitalWrite functions to use gpio_get_level to read the pin level and gpio_set_level to set the pin level.

```
gpio_set_level(SYNC, 1);
```

4. Instead of using delayMicroseconds for delay, use ets_delay_us for microsecond delay

```
ets_delay_us(a);
```

5. Use app_main as the program entry point, similar to Arduino's setup, to manually create a while loop for the loop part of the code

```
void app_main() {
    setup();
    while (1) {
        loop();
    }
}
```

After we adjust the code, we choose the COM and Board to let ESP-IDF configure in the right way. Then we use the Build function to build the project:

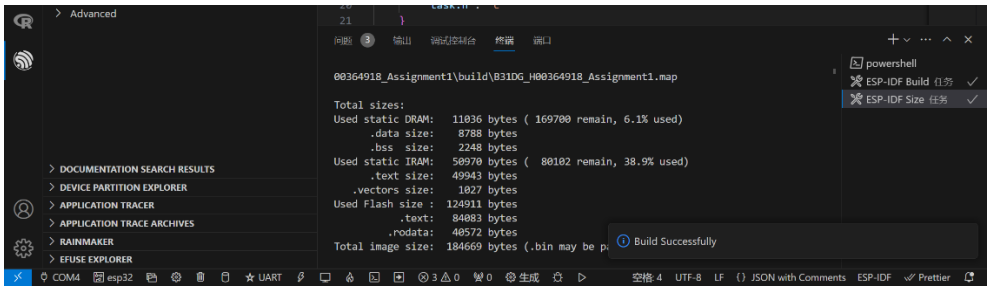

Figure 2:Build Successfully

And we use the Flash function to burn the program to the board:

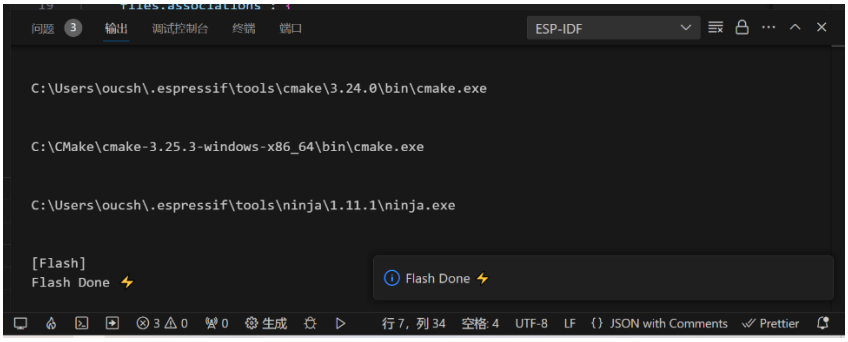

Figure 3:Flash successfully

Then our ESP32 board can work as well as before. So now we can use Arduino or ESP32-IDF to implement the ESP32 board function.

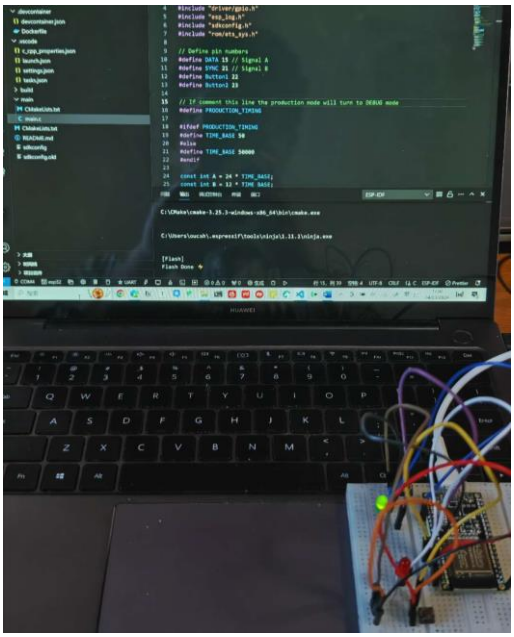

Figure 4:Successfully use ESP-IDF to implement the function of ESP32 Board

4. Use of oscilloscope and analysis of experimental results

4.1 Settings for using an oscilloscope

For the use of oscilloscopes, a very critical point is the setting of the trigger. By setting the trigger conditions, such as the voltage level, the oscilloscope can start to capture when the signal reaches a specific state, which is very helpful for debugging and analyzing the instantaneous events in the signal so that we can obtain a clear and stable periodic signal.

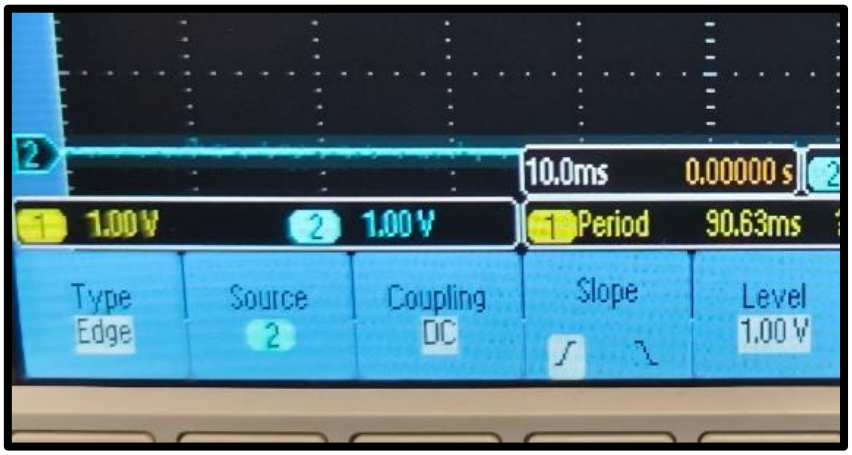

Figure 5：trigger values of the oscilloscope

For the trigger of DATA(SignalA), we can set a value of 2V， it works well on the trigger for the SignalA; for the trigger of SYNC(SingalB), we can set a value of 1V.
But in the end, we set the trigger source of the oscilloscope on the second channel (SYNC). **Select the SYNC channel as the trigger source, and set the trigger level to 1V.** This meets the task requirements: “ii. Oscilloscope to be triggered on Sync signal.”
The advantage of this is that no matter how the DATA signal changes, the oscilloscope always starts to synchronously capture the waveform when the SYNC signal reaches 1V. **This allows the display of the waveform to stabilize on the screen, rather than constantly moving, which is conducive to our observation and analysis of periodic signals.**

After determining the appropriate trigger value, we adjust the display range of the oscilloscope, adjust the scale of the two waveforms, and adjust the display position of the two waveforms, you can clearly and completely observe the waveform.

4.2 Oscilloscope display results

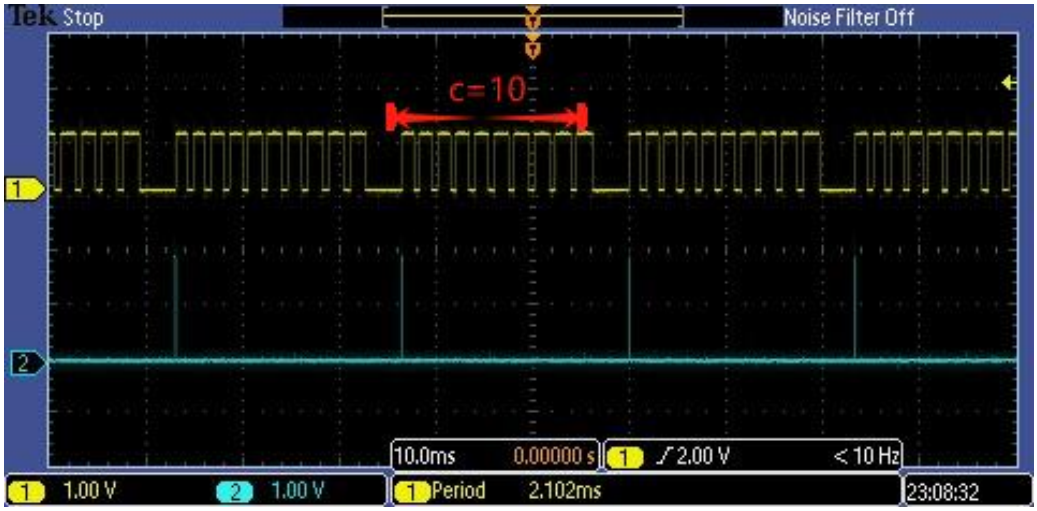

Figure 6：The waveform of the normal mode

We can observe that before the beginning of the DATA waveform, our SYNC signal appears first, signifying the start of a cycle. Since the number of waves contained in one cycle is relatively large, the display is correspondingly small. Also, the time interval between each signal in the DATA waveform is the same, at bμS. Furthermore, with each iteration, the duration of the next signal increases by 50μS. So we can observe that the duration of the last iteration's signal is much longer than that of the first signal. All these observations are in line with the experimental requirements.

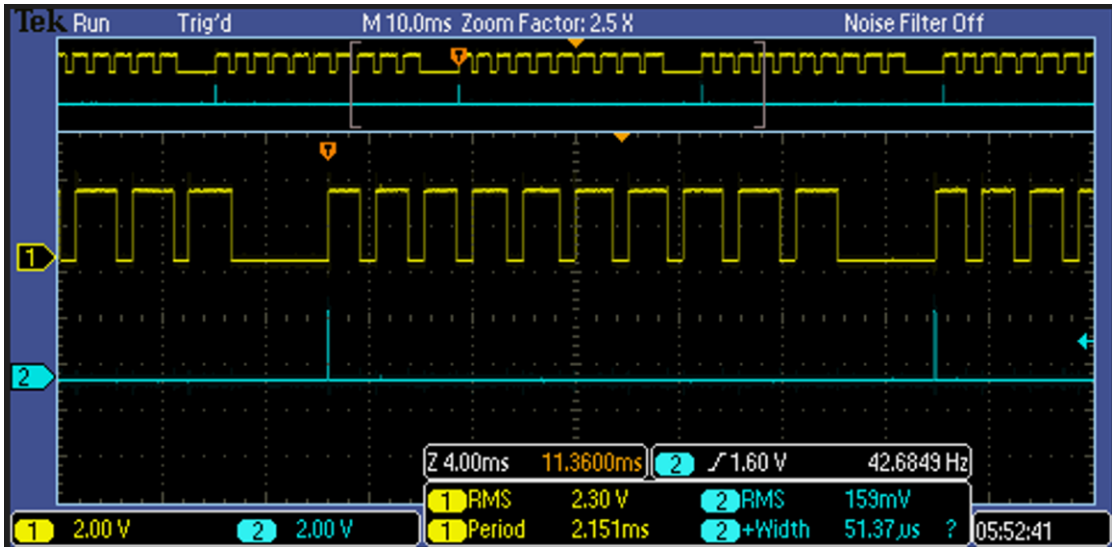

Figure 7: Larger diagram, information can be more intuitively observed

When we push the Button2 the signal mode will change to mode3:

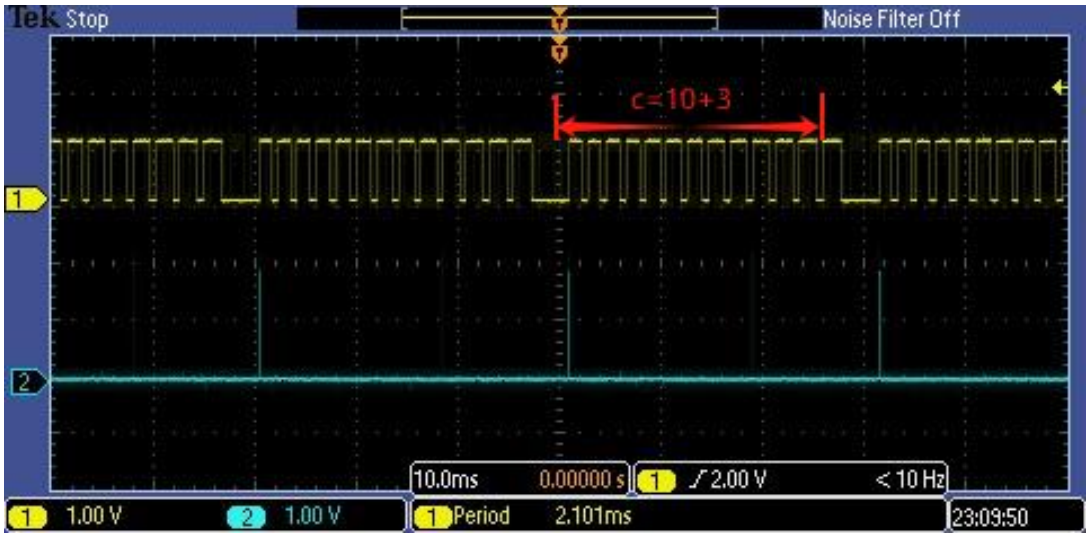

Figure 8: The waveform of mode3

We can see that the number of pulses in a DATA waveform cycle plus 3. And the other essential elements remain the same.

5. Image of the hardware circuit

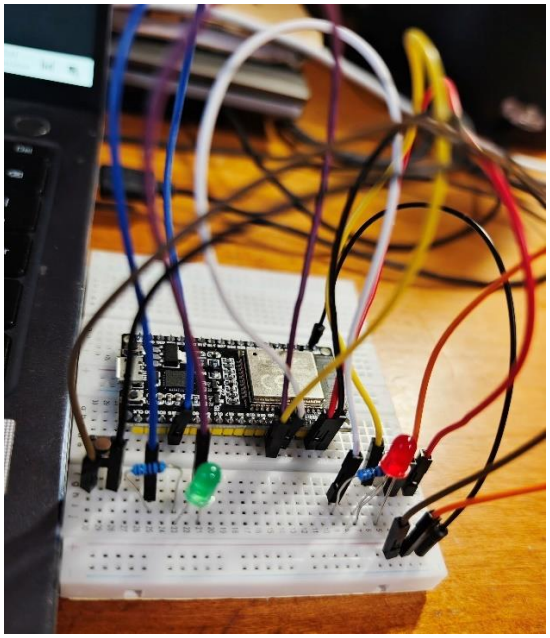

Figure 9: The image of the hardware circuit

Just as I mentioned above we use internal pull-up resistors with the button, so there are only two resistors connected to the LED in the circuit.
At the same time, the length of the wire may have some influence on the signal sensitivity, but in my experiment, they are not influenced a lot.