B31DG - Assignment 1 - H00484764 - Wenbin Lin

Contact Details

Name: Wenbin Lin
Student Number: H00484764
Email Address:wl4001@hw.ac.uk

Revision History

Date	Arthor Name	Version Number	Notes
2025-02-27	Wenbin Lin	1.0	initial draft
2025-03-01	Wenbin Lin	1.1	updating formating

1.Calculation of My Application Parameters

My name is Wenbin Lin. The surname is Lin. As the surname only has three letters, the fourth and fifth letters are duplicated from the third letter ‘n’ for the purposes of numerical mapping. The calculation detail is displayed in Table 1.

Parameter	Numerical mapping(for Wenbin Lin)	Calculation(for Wenbin Lin)
a	“L” maps to 12	12x100us=1200us
b	“i” maps to 9	9X100us=900us
c	“n” maps to 13	13+4=17
d	“n” maps to 13	13X500us=6500us=6.5ms
Alternative Behaviour Selection	“n” maps to 13	(13 % 4) + 1 = 2, therefore Option 2 is selected

Table 1. Calculation of My Application Parameters

2. Oscilloscope Screen Captures

2.1 Select Button On and Enable Button On

There are four situations for the oscilloscope. Firstly, the enable button is on and the select button is on. The oscilloscope screen capture is shown in Figure 1. The signal with blue color is the select signal. And the signal with brown color is the output signal. In this situation, for the output signal ,in one cycle, the waveform gradually shortens from a longer pulse. In debug mode, each reduction is 50ms. Each cycle consists of 17 pulses, with an interval of 0.9s between pulses. The longest pulse starts at 2.05s and gradually decreases to 1.2s. The interval between different cycles is approximately 6.5s. And for one cycle, the data signal starts just after the select signal. The select signal only lasts for 50ms in debug mode.

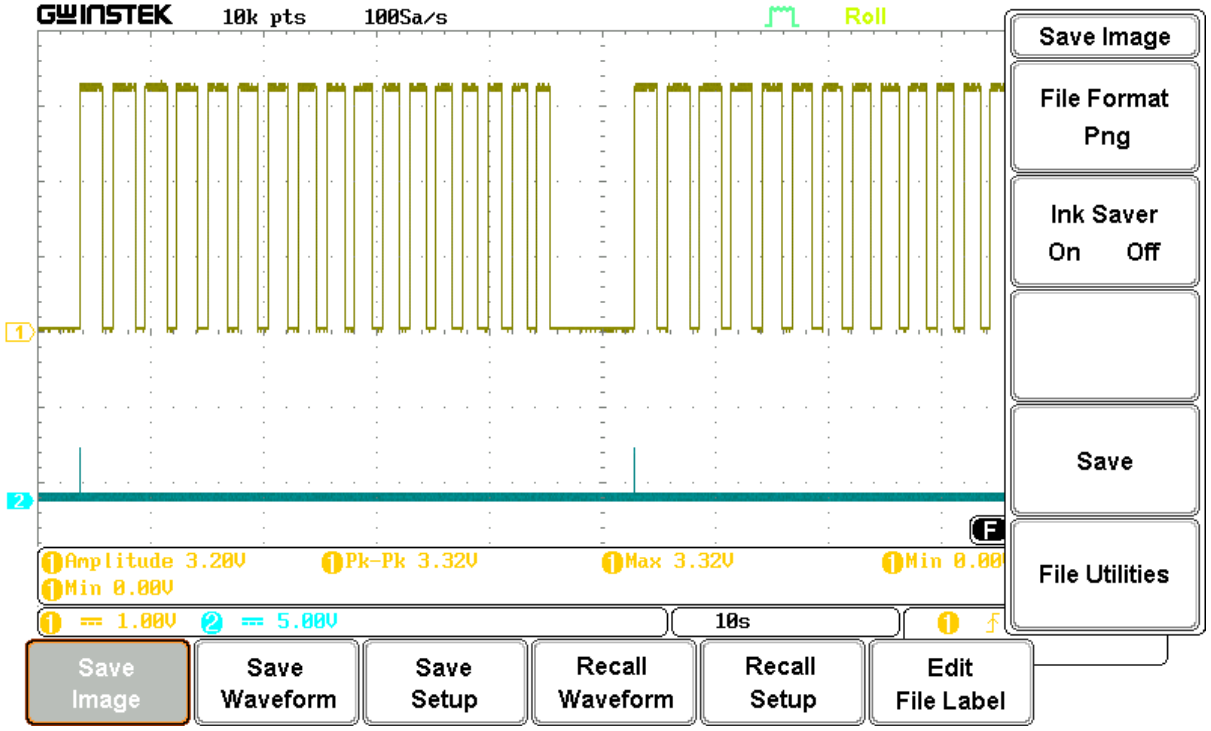

Figure 1. oscilloscope picture with select button on and enable button on

2.2 Select Button Off and Enable Button On

The second situation is that the enable button is on while the select button is off. The oscilloscope screen capture is shown in *Figure 2*. There is no select signal. In this situation, the output signal is just a reversed form of *Figure 1*. For this case ,in one cycle, the waveform gradually gets longer from a shorter pulse. In one cycle, there are also 17 pulses.

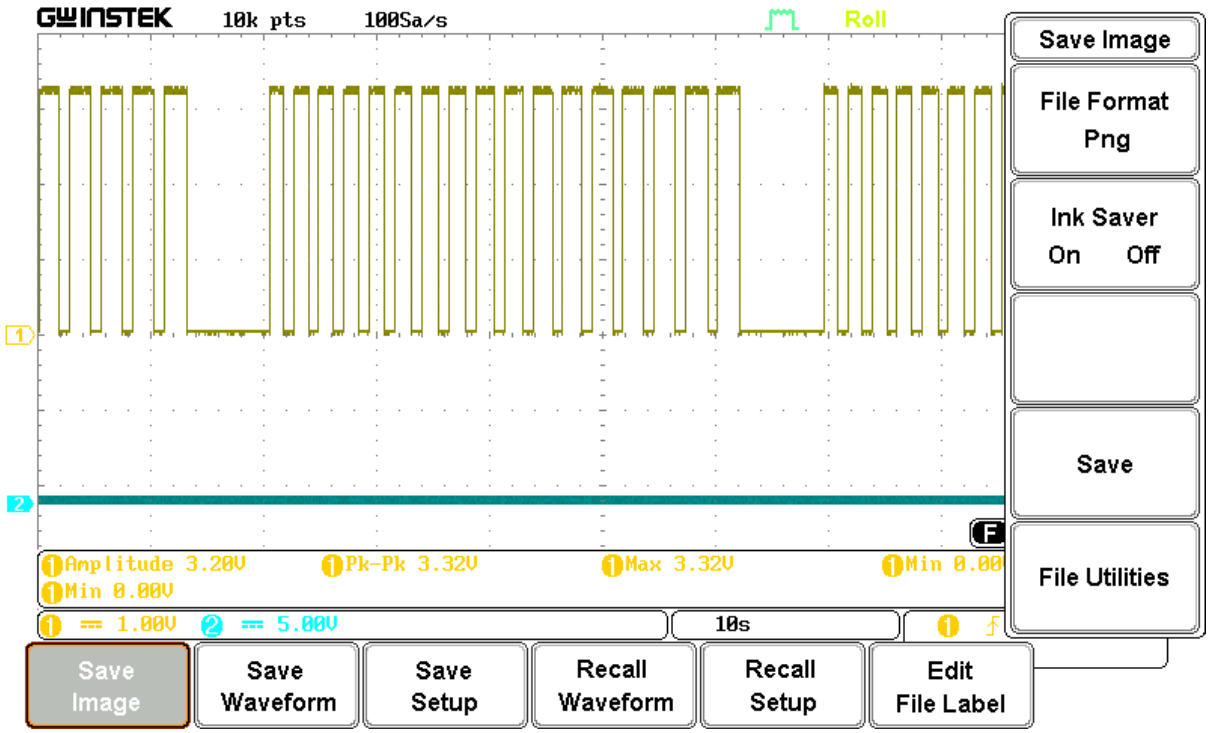

Figure 2. oscilloscope picture with select button off and enable button on

2.3 Select Button On and Enable Button Off

The third situation is that the enable button is off and the select button is on. In this situation, there are select signals but without output data. The oscilloscope picture is shown in *Figure 3*.

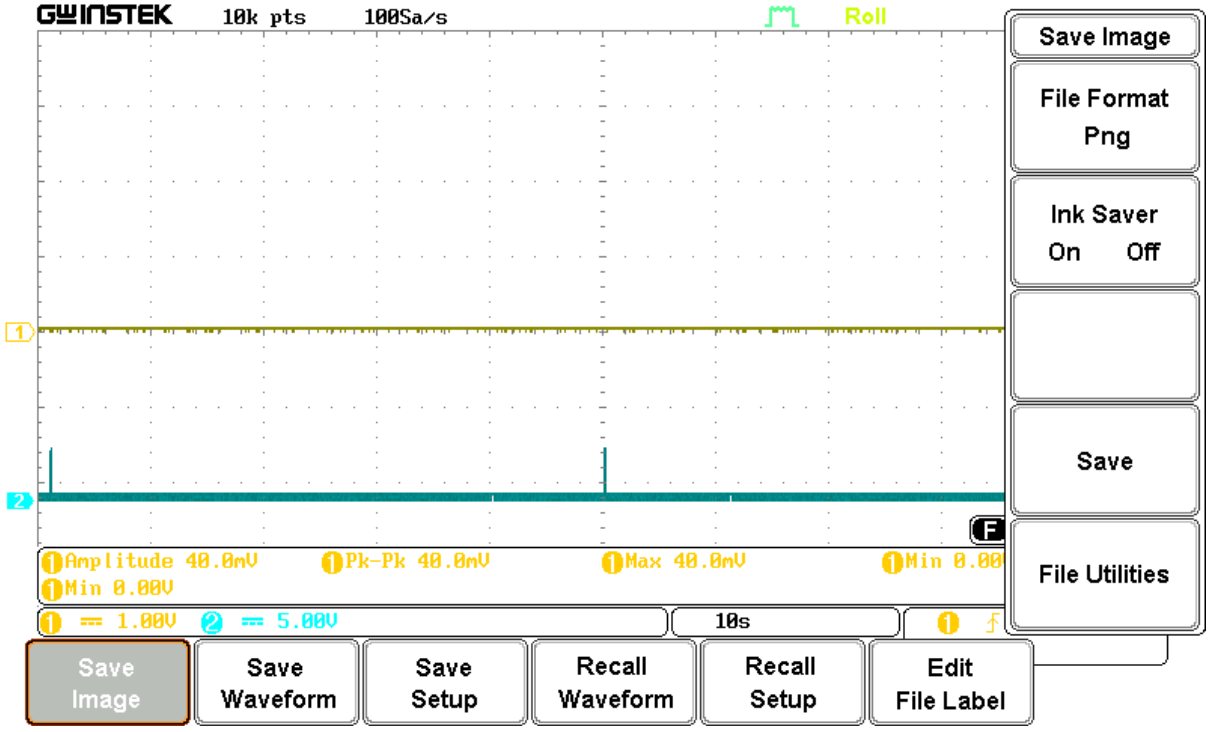

Figure 3. oscilloscope picture with select button on and enable button off

2.4 Select Button Off and Enable Button Off

The last situation is that the enable button is off and the select button is off. In this situation, there are no output and select signals. The oscilloscope picture is shown in *Figure 4*.

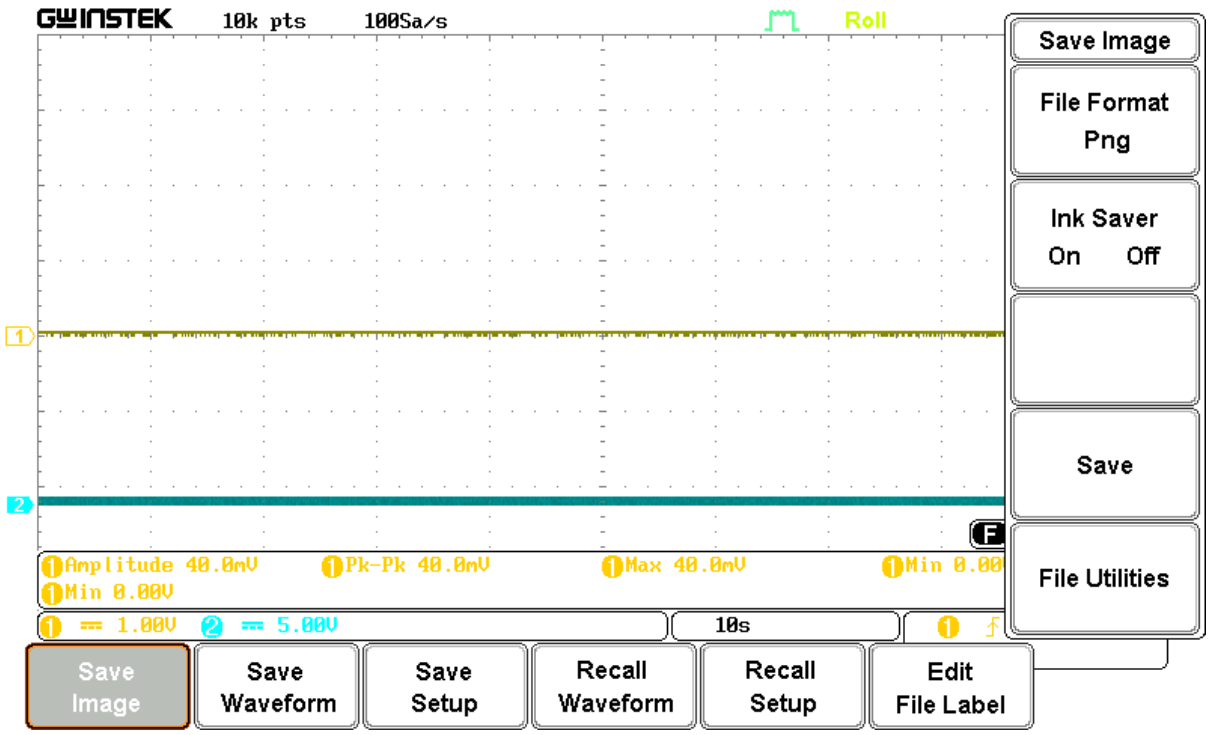

Figure 4. oscilloscope picture with select button off and Enable Button off

3. Hard Circuit Image of The Hardware

The circuit image is shown in *Figure 5*. The pins 16,17 of ESP32 PCB are connected to two buttons,which will be used to control the output signal and select signal. And at the other side the two buttons are connected to two 200Ω resistors and then to the VDD pin of the ESP32 PCB. The pins 26,27 are connected to two 200Ω resistors and then connected to two LEDs, which are used to check the select and output signal. Then the two LEDs are connected to the GND. The details are shown in the following Figure 5.

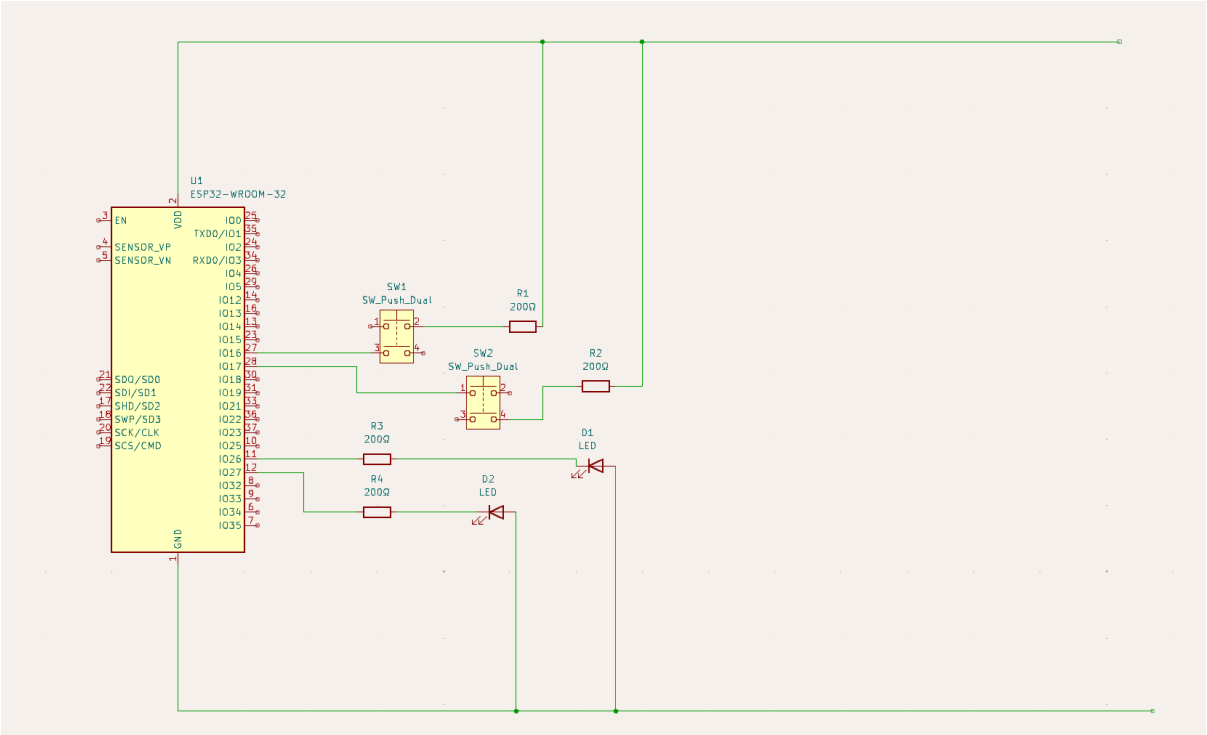

Figure 5. hardware circuit image

4. Flowchart showing application control flow

The flowchart of the application is shown in *Figure 6*.

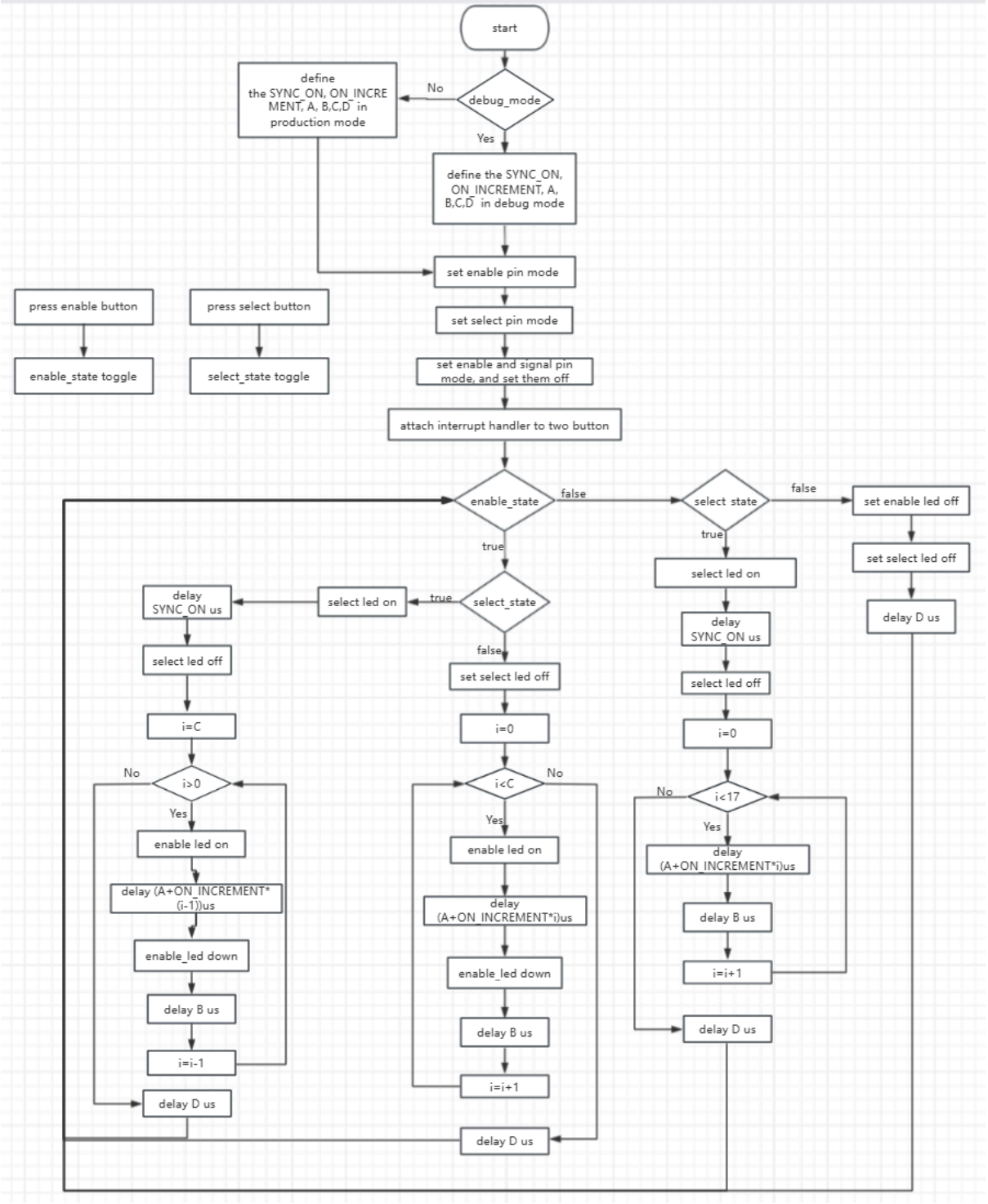

Figure 6. flowchart of the application control flow

4.1 Explain The Code in Arduino

I use the arduino code to explain the flowchart of the control flow. First part of the code is doing macro definition.

```
#define _DEBUG_MODE
```

```
#ifdef _DEBUG_MODE
  #define SYNC_ON (50*1000)
  #define A (12*100*1000)
  #define ON_INCREMENT (50*1000)
  #define B (9*100*1000)
  #define C (13+4)
  #define D (13*500*1000)
#else
  #define SYNC_ON 50
  #define A (12*100)
  #define ON_INCREMENT (50)
  #define B (9*100)
  #define C (13+4)
  #define D (13*500)
#endif
```

The above code defines two modes using #define _DEBUG_MODE. If in the production mode, the line can be commented out. In the debug mode, timing values are in milliseconds. While in the production mode, timing values are in microseconds. The variable SYNC_ON defines the synchronization pulse(select signal) duration. The variable A defines the base duration for the output signal. The variable ON_INCREMENT defines the reduction or increment per cycle. The variable B defines the time interval between pulses. The variable C defines the total number of pulses in a cycle. The variable D defines the interval between different cycles.

code:

```
#define ENABLE_PIN 16
#define SELECT_PIN 17
#define ENABLE_LED_PIN 27
#define SELECT_LED_PIN 26
#define DEBOUNCE_DELAY 100
volatile bool enable_state = true;
volatile bool select_state = true;
```

The above code defines the pins for output button(ENABLE_PIN), select button(SELECT_PIN), output led(ENABLE_LED_PIN) and select led(SELECT_LED_PIN). The code also defines the debounce delay for debouncing. Meanwhile, the code sets the initial state of the output data button and the select button to true(on).

code:

```
volatile unsigned long lastEnableInterruptTime = 0;
volatile unsigned long lastSelectInterruptTime = 0;
void IRAM_ATTR enable_pressed_handle() {
  unsigned long currentTime = millis();
```

```
  if (currentTime - lastEnableInterruptTime > DEBOUNCE_DELAY) {
      enable_state = !enable_state;
    lastEnableInterruptTime = currentTime;
  }
}
void IRAM_ATTR select_pressed_handle() {
  unsigned long currentTime = millis();
  if (currentTime - lastSelectInterruptTime > DEBOUNCE_DELAY) {
       select_state = !select_state;
    lastSelectInterruptTime = currentTime;
  }
}
```

The two functions handle the two button press interrupts. When a button is pressed, the corresponding state(enable_state or select_state) is toggled. Debouncing is implemented by checking if the time since the last interrupt is greater than DEBOUNCE_DELAY.

code:

```
void setup() {
  Serial.begin(115200);
  pinMode(ENABLE_PIN, INPUT_PULLDOWN);
  pinMode(SELECT_PIN, INPUT_PULLDOWN);
  attachInterrupt(digitalPinToInterrupt(ENABLE_PIN),
enable_pressed_handle, RISING);
  attachInterrupt(digitalPinToInterrupt(SELECT_PIN),
select_pressed_handle, RISING);
  pinMode(ENABLE_LED_PIN, OUTPUT);
  pinMode(SELECT_LED_PIN, OUTPUT);
  digitalWrite(SELECT_LED_PIN, LOW);
  digitalWrite(ENABLE_LED_PIN, LOW);
}
```

The above code just does setups for the application. It initializes serial communication for debugging, configures the button pins as inputs with pull-down resistors, attaches interrupt handlers to the button pins, triggering on the rising edge (button press), and configures the LED pins as outputs and sets them to LOW (off) initially.

code:

```
void loop() {
    if (select_state==true && enable_state == true){
    Serial.println("into condition 1");
    digitalWrite(SELECT_LED_PIN, HIGH);
    delayMicroseconds(SYNC_ON);
    digitalWrite(SELECT_LED_PIN, LOW);
    for (int i=C;i>0;i--){
      digitalWrite(ENABLE_LED_PIN, HIGH);
      delayMicroseconds(A + ((i-1)*ON_INCREMENT));
      digitalWrite(ENABLE_LED_PIN,LOW);
      delayMicroseconds(B);
    }
    delayMicroseconds(D);
  }
```

```
  else if (select_state == false and enable_state == true){
    Serial.println("into condition 2");
    digitalWrite(SELECT_LED_PIN, LOW);
    for (int i=0;i<C;i++){
      digitalWrite(ENABLE_LED_PIN, HIGH);
      delayMicroseconds(A + (i*ON_INCREMENT));
      digitalWrite(ENABLE_LED_PIN,LOW);
      delayMicroseconds(B);
    }
    delayMicroseconds(D);
  }
  else if (select_state == true and enable_state == false){
    Serial.println("into condition 3");
    digitalWrite(SELECT_LED_PIN, HIGH);
    delayMicroseconds(SYNC_ON);
    digitalWrite(SELECT_LED_PIN, LOW);
    for (int i=0;i<C;i++){
      delayMicroseconds(A + (i*ON_INCREMENT));
      delayMicroseconds(B);
    }
    delayMicroseconds(D);
  }
  else if (select_state == false and enable_state == false){
    Serial.println("into condition 4");
    digitalWrite(SELECT_LED_PIN, LOW);
    digitalWrite(ENABLE_LED_PIN, LOW);
    delayMicroseconds(D);
  }
}
```

The above code is a loop function, which contains four conditions based on the states of select_state and enable_state. It will loop indefinitely until interrupts or other outside events.When the two buttons are pressed, the corresponding interrupt function will execute, and then return to the last state of the loop function. For the loop function, there are four situations.

For the first situation, when both select and enable are on, the application turns on the select LED for SYNC_ON microseconds, then blinks the enable LED in a loop with decreasing delays (A + (i-1) * ON_INCREMENT)(i from 17 to 1)us and the interval is B us for 17 times, after that, adds a final delay of D microseconds. And then continue the loop.

For the second situation,when select is off and enable is on, the select LED does not blink, while the enable LED blinks in a loop with increasing delays (A + i * ON_INCREMENT)(i from 0 to 16)) us and the interval is B us for 17 times, after that, adds a final delay of D microseconds. And then continue the loop.

For the third situation, when the select is on and the enable is off, the select LED does the same thing as situation 1 but the enable LED is off all the time.

For the fourth situation, when the select and enable are all off, the select LED and enable LED are all off all the time.

4.2 Expain The Code in VS Code with ESP-IDF Extension

```
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "driver/gpio.h"
#include "esp_timer.h"
#include "esp_log.h"
#include "esp32/rom/ets_sys.h"
```

The file is to import the header file in C which is used in the following code.

```
#define _DEBUG_MODE
#ifdef _DEBUG_MODE
  #define SYNC_ON (50*1000)
  #define A (12*100*1000)
  #define ON_INCREMENT (50*1000)
  #define B (9*100*1000)
  #define C (13+4)
  #define D (13*500*1000)
#else
  #define SYNC_ON 50
  #define A (12*100)
  #define ON_INCREMENT (50)
  #define B (9*100)
  #define C (13+4)
  #define D (13*500)
#endif
#define ENABLE_PIN 16
#define SELECT_PIN 17
#define ENABLE_LED_PIN 27
#define SELECT_LED_PIN 26
#define DEBOUNCE_DELAY 100
```

This block of code is setting the variables values. The first line #define DEBUG_MODE will make sure that the variables SYNC_ON,A, ON_INCREMENT,B,C,D can have different values depending on whether the line is commented out. Then the code sets the value of the LED and button GPIO number. The code also defines the debounce delay time for 100 ms.

```
void IRAM_ATTR enable_pressed_handle(void* arg) {
    uint64_t currentTime = esp_timer_get_time() / 1000;
    if (currentTime - lastEnableInterruptTime > DEBOUNCE_DELAY) {
        enable_state = !enable_state;
        lastEnableInterruptTime = currentTime;
    }
}
void IRAM_ATTR select_pressed_handle(void* arg) {
    uint64_t currentTime = esp_timer_get_time() / 1000;
       if (currentTime - lastSelectInterruptTime > DEBOUNCE_DELAY) {
```

```
        select_state = !select_state;
        lastSelectInterruptTime = currentTime;
    }
}
```

This block of code sets the two buttons interrupt handler function. When the button is pressed, the button state is toggled.

```
    gpio_config_t io_conf = {
        .pin_bit_mask = (1ULL << ENABLE_PIN) | (1ULL << SELECT_PIN),
        .mode = GPIO_MODE_INPUT,
        .pull_up_en = GPIO_PULLUP_DISABLE,
        .pull_down_en = GPIO_PULLDOWN_ENABLE,
        .intr_type = GPIO_INTR_POSEDGE
    };
    gpio_config(&io_conf);
    gpio_config_t led_conf = {
        .pin_bit_mask = (1ULL << ENABLE_LED_PIN) | (1ULL <<
SELECT_LED_PIN),
        .mode = GPIO_MODE_OUTPUT
    };
    gpio_config(&led_conf);
    gpio_install_isr_service(0);
    gpio_isr_handler_add(ENABLE_PIN, enable_pressed_handle, NULL);
    gpio_isr_handler_add(SELECT_PIN, select_pressed_handle, NULL);
    gpio_set_level(ENABLE_LED_PIN, 0);
    gpio_set_level(SELECT_LED_PIN, 0);
```

The above block of code sets the initial state and mode of the LEDs and buttons. The io_conf sets the two button GPIO state. The led_conf sets the state of two LED GIPO. The gpio_isr_handler_add sets the two button pin interrupting handler to enable_pressed_handle and select_pressed_handle. And the gpio_set_level function sets the two LED gpio's initial volts 0.

```
while (1) {
        if (select_state && enable_state) {
            ESP_LOGI(TAG, "into condition 1");
            gpio_set_level(SELECT_LED_PIN, 1);
            ets_delay_us(SYNC_ON);
            gpio_set_level(SELECT_LED_PIN, 0);
            for (int i = C; i > 0; i--) {
                gpio_set_level(ENABLE_LED_PIN, 1);
                ets_delay_us(A + ((i - 1) * ON_INCREMENT));
                gpio_set_level(ENABLE_LED_PIN, 0);
                ets_delay_us(B);
            }
```

```
            ets_delay_us(D);
        }
        else if (select_state && !enable_state) {
            ESP_LOGI(TAG, "into condition 2");
            gpio_set_level(SELECT_LED_PIN, 1);
            ets_delay_us(SYNC_ON);
            gpio_set_level(SELECT_LED_PIN, 0);
            for (int i = 0; i < C; i++) {
                ets_delay_us(A + (i * ON_INCREMENT));
                ets_delay_us(B);
            }
            ets_delay_us(D);
        }
        else if (!select_state && !enable_state) {
            ESP_LOGI(TAG, "into condition 3");
            gpio_set_level(SELECT_LED_PIN, 0);
            gpio_set_level(ENABLE_LED_PIN, 0);
            ets_delay_us(D);
        }
        else if (!select_state && enable_state) {
            ESP_LOGI(TAG, "into condition 4");
            gpio_set_level(SELECT_LED_PIN, 0);
            for (int i = 0; i < C; i++) {
                gpio_set_level(ENABLE_LED_PIN, 1);
                ets_delay_us(A + (i * ON_INCREMENT));
                gpio_set_level(ENABLE_LED_PIN, 0);
                ets_delay_us(B);
            }
            ets_delay_us(D);
        }
    }
```

The above code is a loop function, which contains four conditions based on the states of select_state and enable_state. It will loop indefinitely until interrupts or other outside events.When the two buttons are pressed, the corresponding interrupt function will execute, and then return to the last state of the loop function. For the loop function, there are four situations.

For the first situation, when both select and enable are on, the application turns on the select LED for SYNC_ON microseconds, then blinks the enable LED in a loop with decreasing delays (A + (i-1) * ON_INCREMENT)(i from 17 to 1)us and the interval is B us for 17 times, after that, adds a final delay of D microseconds. And then continue the loop.

For the second situation,when select is off and enable is on, the select LED does not blink, while the enable LED blinks in a loop with increasing delays (A + i *

ON_INCREMENT)(i from 0 to 16)) us and the interval is B us for 17 times, after that, adds a final delay of D microseconds. And then continue the loop.

For the third situation, when the select is on and the enable is off, the select LED does the same thing as situation 1 but the enable LED is off all the time.

For the fourth situation, when the select and enable are all off, the select LED and enable LED are all off all the time.