GEBZE TECHNICAL UNIVERSITY

ELECTRONIC ENGINEERING

ELEC 335

LAB 3

İÇİNDEKİLER

1 PROBLEM 1

1.1 DEFINITION (PROBLEM 1)

In this problem, a program has been written to flash an external LED at approximately 1 second intervals. The circuit is set up as follows. Oscilloscope images are given below.

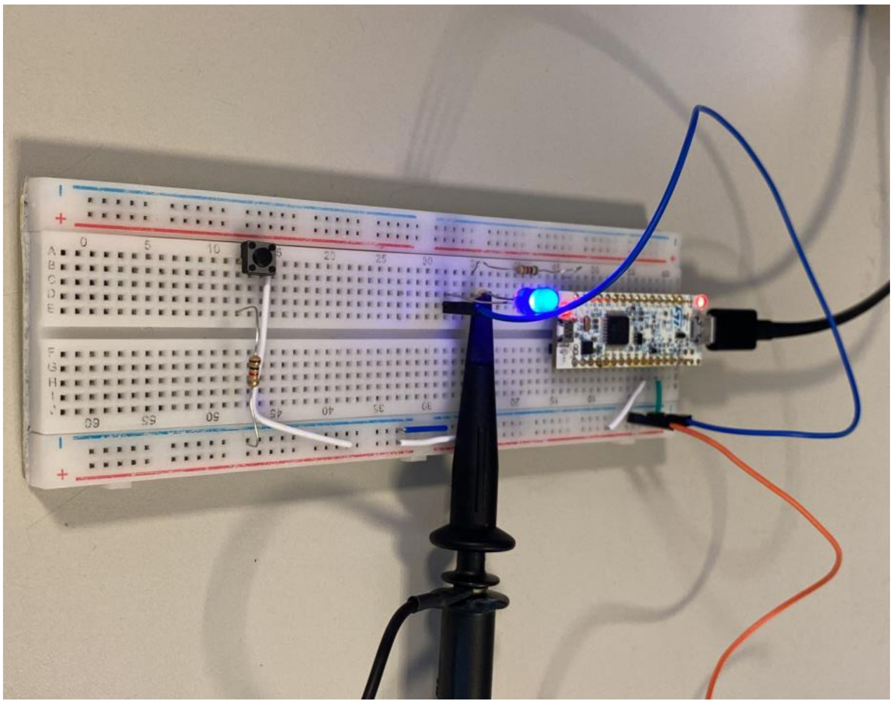


Figure 1: Circuit In Breadboard

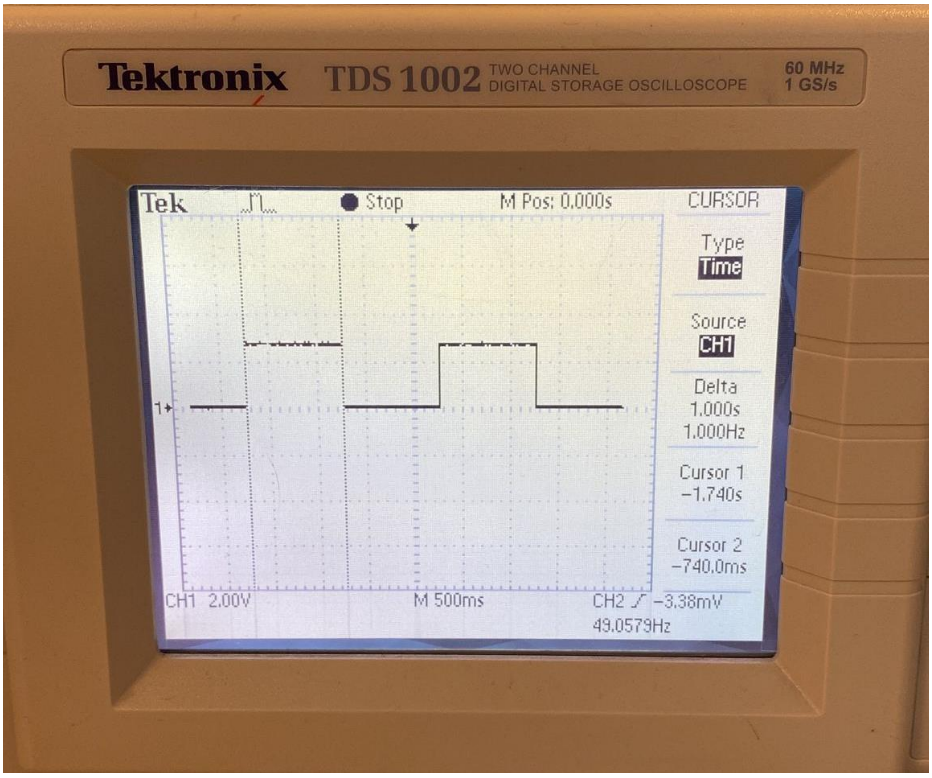
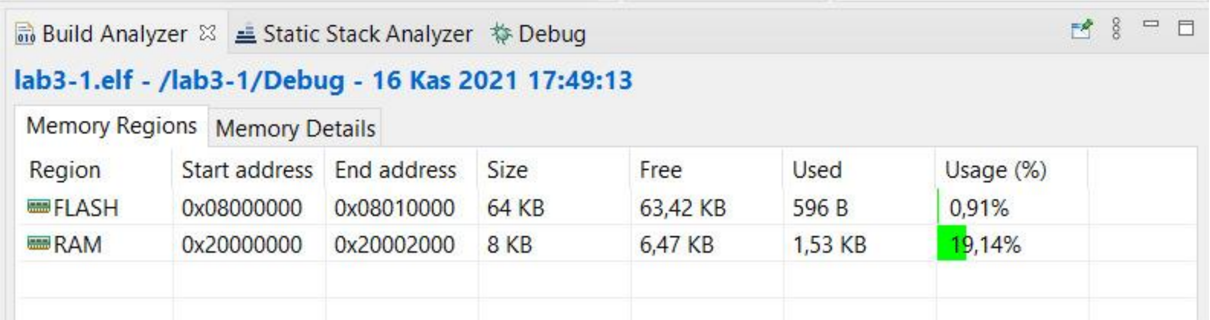


Figure 2: Oscilloscope Images On/Off Time is 1sn

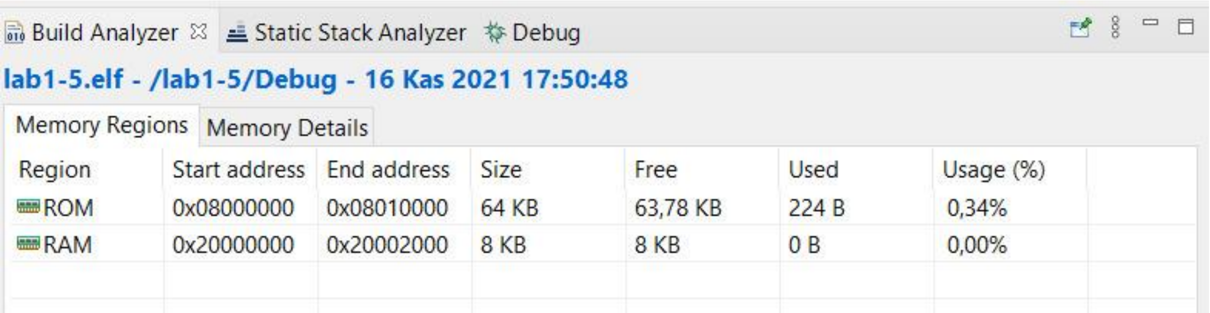
Many more instructions are used to write the same code in C. The code size is smaller in assembly, as can be seen from this situation. Since there are fewer instructions in assembly, the number of delays in C and assembly are different.



lab3-1.elf - /lab3-1/Debug - 16 Kas 2021 17:49:13

Region	Start address	End address	Size	Free	Used	Usage (%)
FLASH	0x08000000	0x08010000	64 KB	63,42 KB	596 B	0,91%
RAM	0x20000000	0x20002000	8 KB	6,47 KB	1,53 KB	19,14%

Figure 3: Code Size In C



lab1-5.elf - /lab1-5/Debug - 16 Kas 2021 17:50:48

Region	Start address	End address	Size	Free	Used	Usage (%)
ROM	0x08000000	0x08010000	64 KB	63,78 KB	224 B	0,34%
RAM	0x20000000	0x20002000	8 KB	8 KB	0 B	0,00%

Figure 4: Code Size In Assembly

As seen in figures 3 and 4, code size is larger in C.

When optimization is turned on, the on and off time of the led is reduced **400ms.** Oscilloscope image is given below.

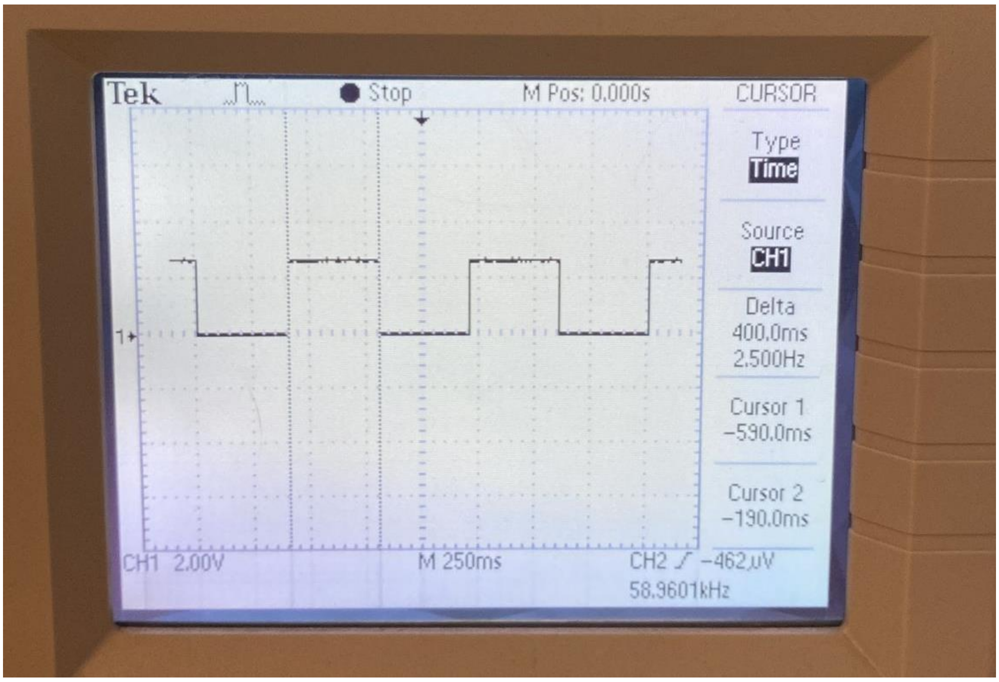
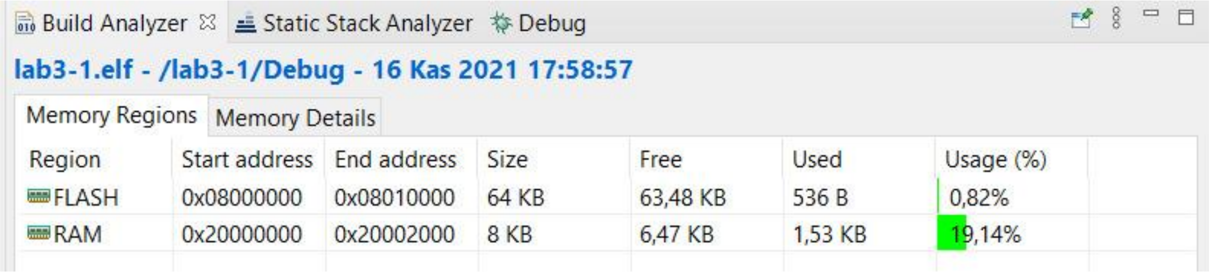


Figure 5: On/Off time 400ms



lab3-1.elf - /lab3-1/Debug - 16 Kas 2021 17:58:57

Region	Start address	End address	Size	Free	Used	Usage (%)
FLASH	0x08000000	0x08010000	64 KB	63,48 KB	536 B	0,82%
RAM	0x20000000	0x20002000	8 KB	6,47 KB	1,53 KB	19,14%

Figure 6: Code size when -O2 optimization open

As seen in figure 6, the code size has **decreased in -O2** because instruction number has reduced by assembly.

1.2 FLOWCHART (PROBLEM 1)

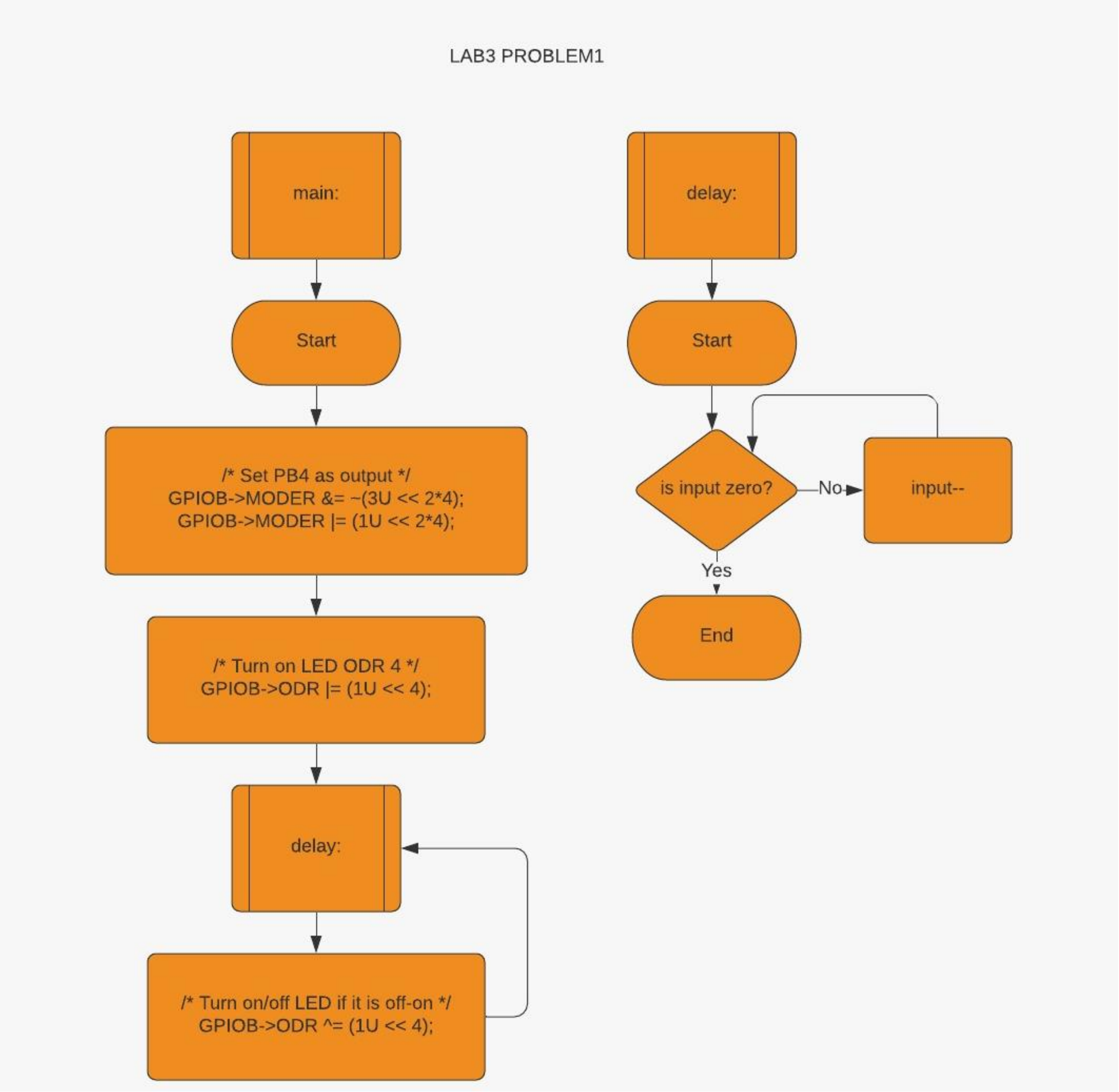


Figure 7: Flowchart Problem 1

1.3 BLOCK DIAGRAM (PROBLEM 1)

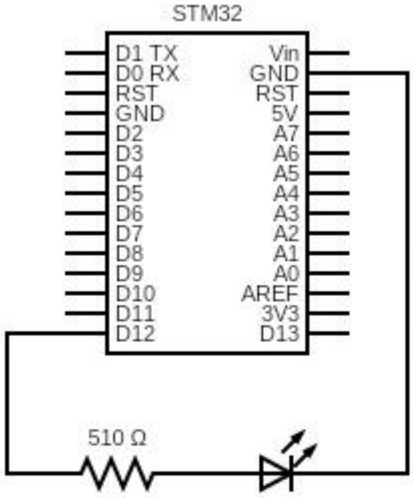


Figure 8: Block Diagram Problem 1

1.4 CODES (PROBLEM 1)

```
#include "stm32g0xx.h"


#define LEDDELAY    1600000


void delay(volatile uint32_t);


int main(void) {


   /* Enable GPIOB clock */


   RCC->IOPENR |= (1U << 1);


   /* Setup PB4 as output */
   GPIOB->MODER &= ~(3U << 2*4);
   GPIOB->MODER |= (1U << 2*4);


   /* Turn on LED ODR 4 */
   GPIOB->ODR |= (1U << 4);
```

```
    while(1) {
        delay(LEDDELAY);
        /* Toggle LED */
        GPIOB->ODR ^= (1U << 4);
    }

    return 0;
}

void delay(volatile uint32_t s) {
    for(; s>0; s--);
}
```

2 PROBLEM 2

2.1 DEFINITON (PROBLEM 2)

In this problem, using a state machine, the LED is turned on and off at different intervals. The setup of the circuit on the breadboard is as follows.

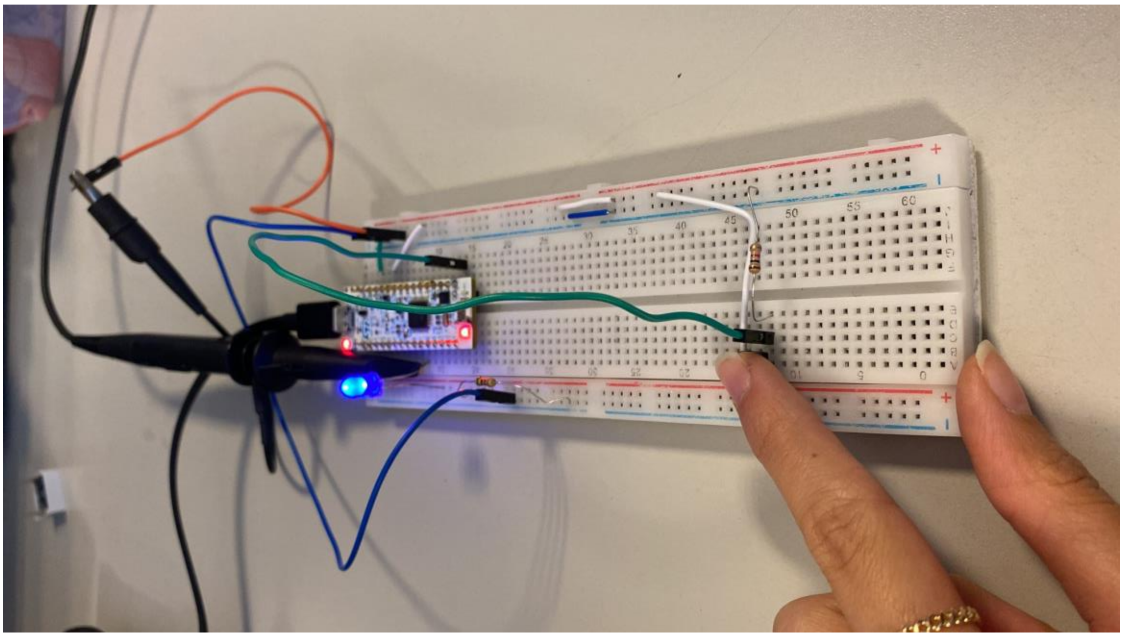


Figure 9:Circuit in Breadboard (mode 1)

No interrupt is used in this section, so polling will be observed. When the code is put into operation at -O2, the time intervals are accelerated. In the laboratory environment, two optimization cases were also investigated.

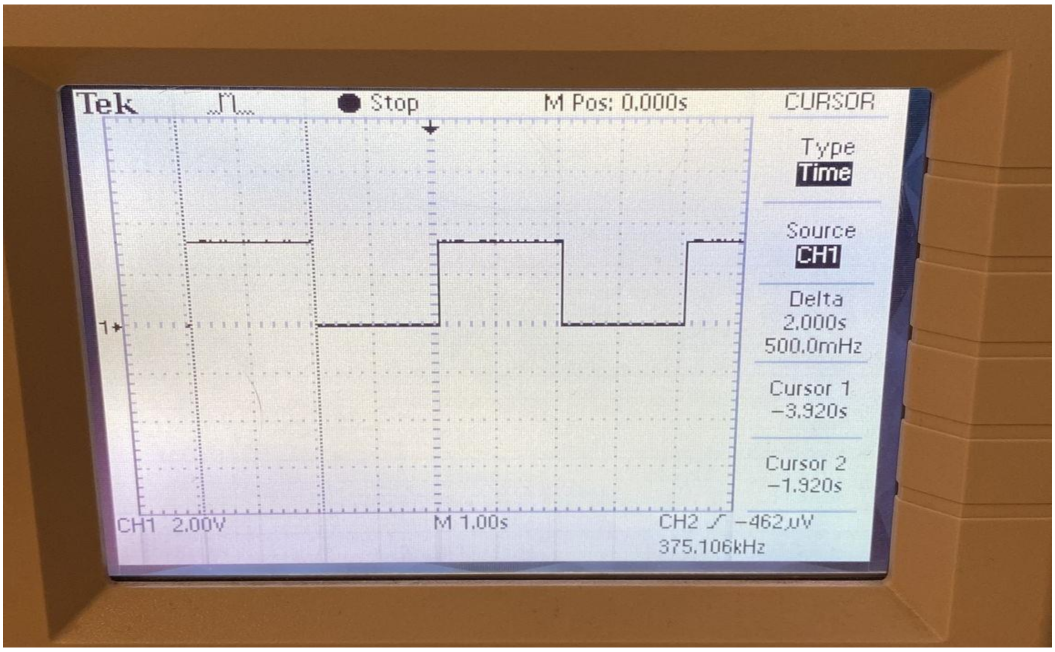


Figure 10: MODE 1 (2 sec)

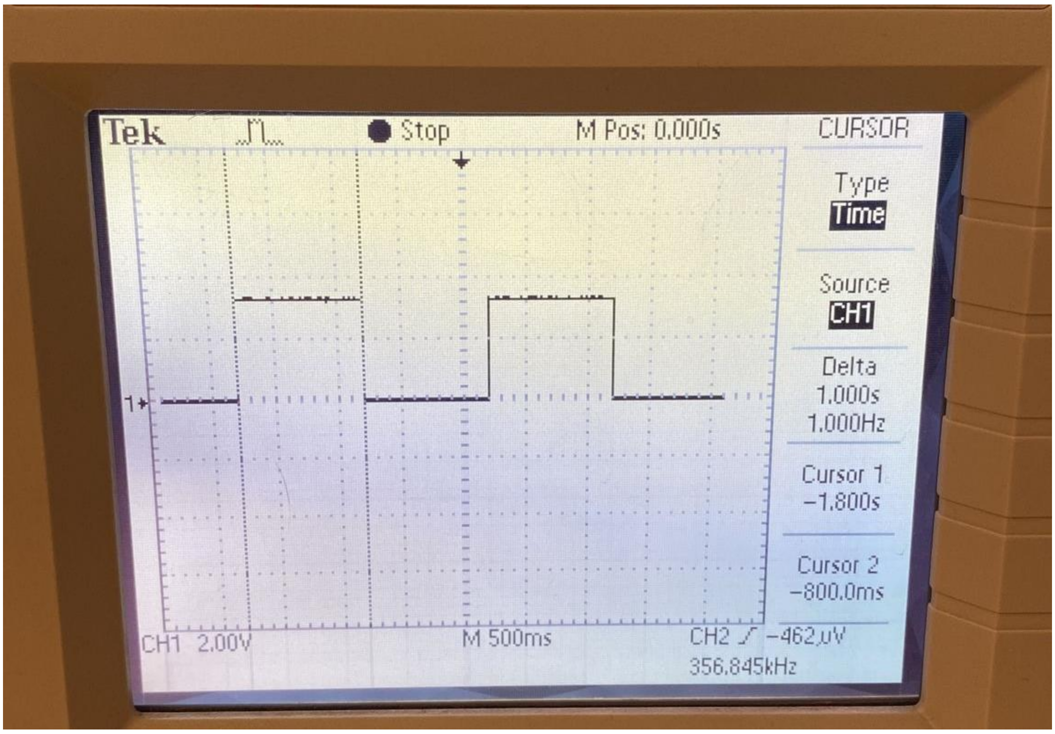


Figure 11: : MODE 1 (1 sec)

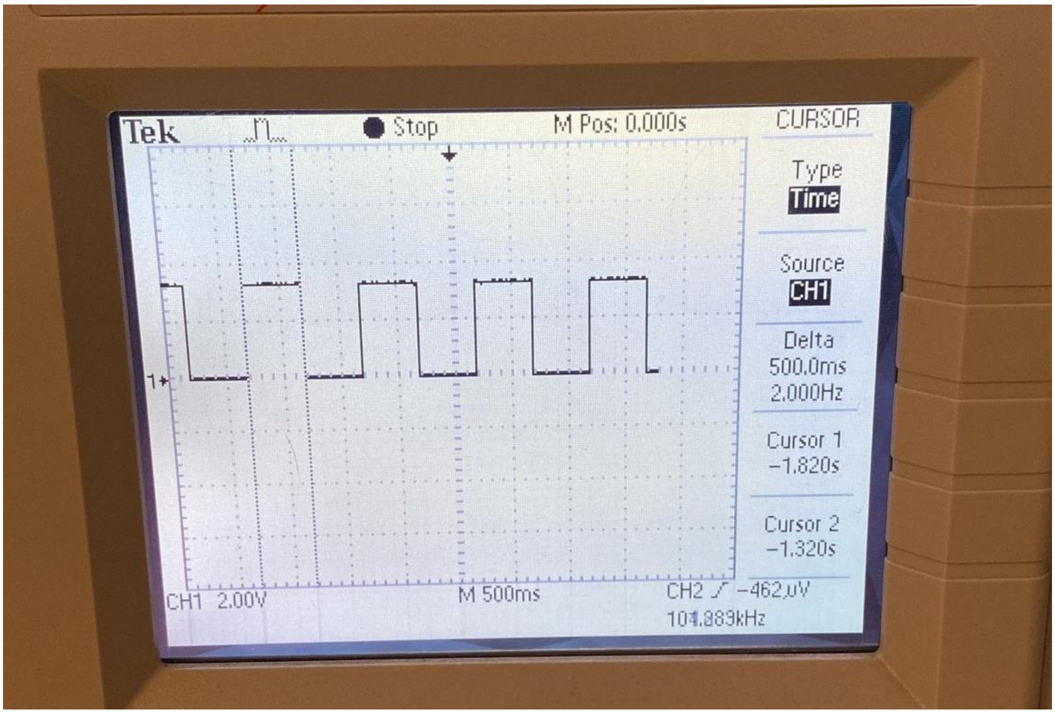


Figure 12: : MODE 1 (0.5 sec)

An example of the first three modes is observed on the oscilloscope screen. The results are as expected. **The same measurements were taken at -O2. The oscilloscope results are as given below.**

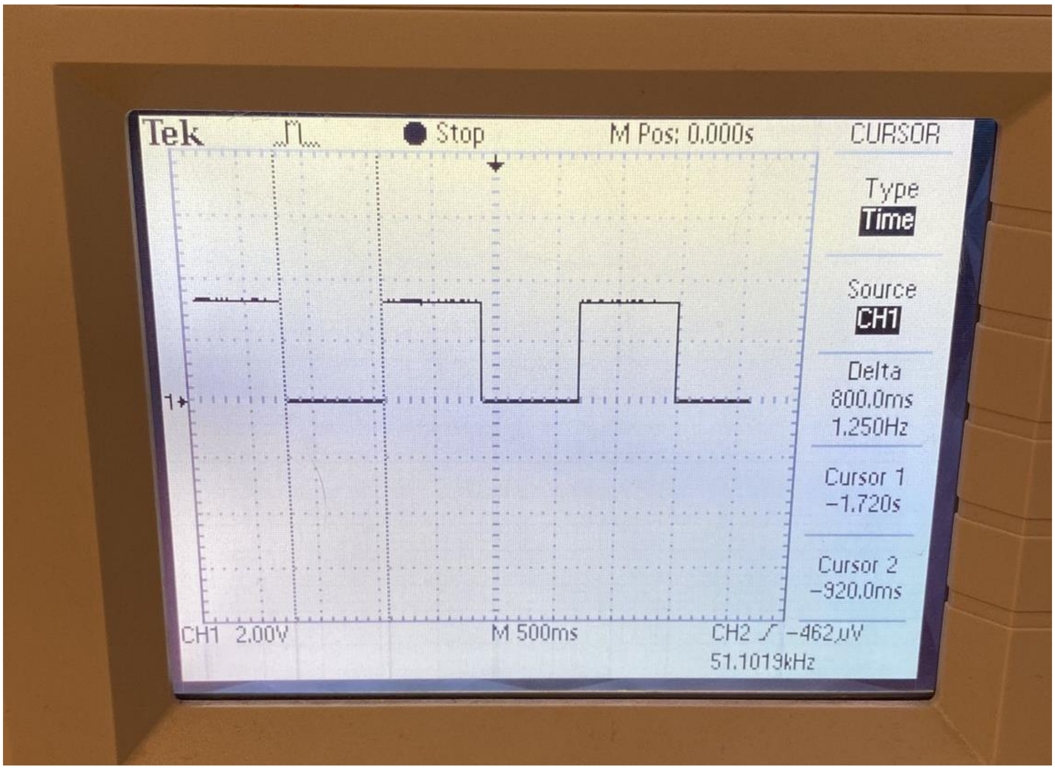


Figure 13: Optimizasyon (Mode 1)

As can be seen, time interval decreased 800ms.

2.2 FLOWCHART (PROBLEM 2)

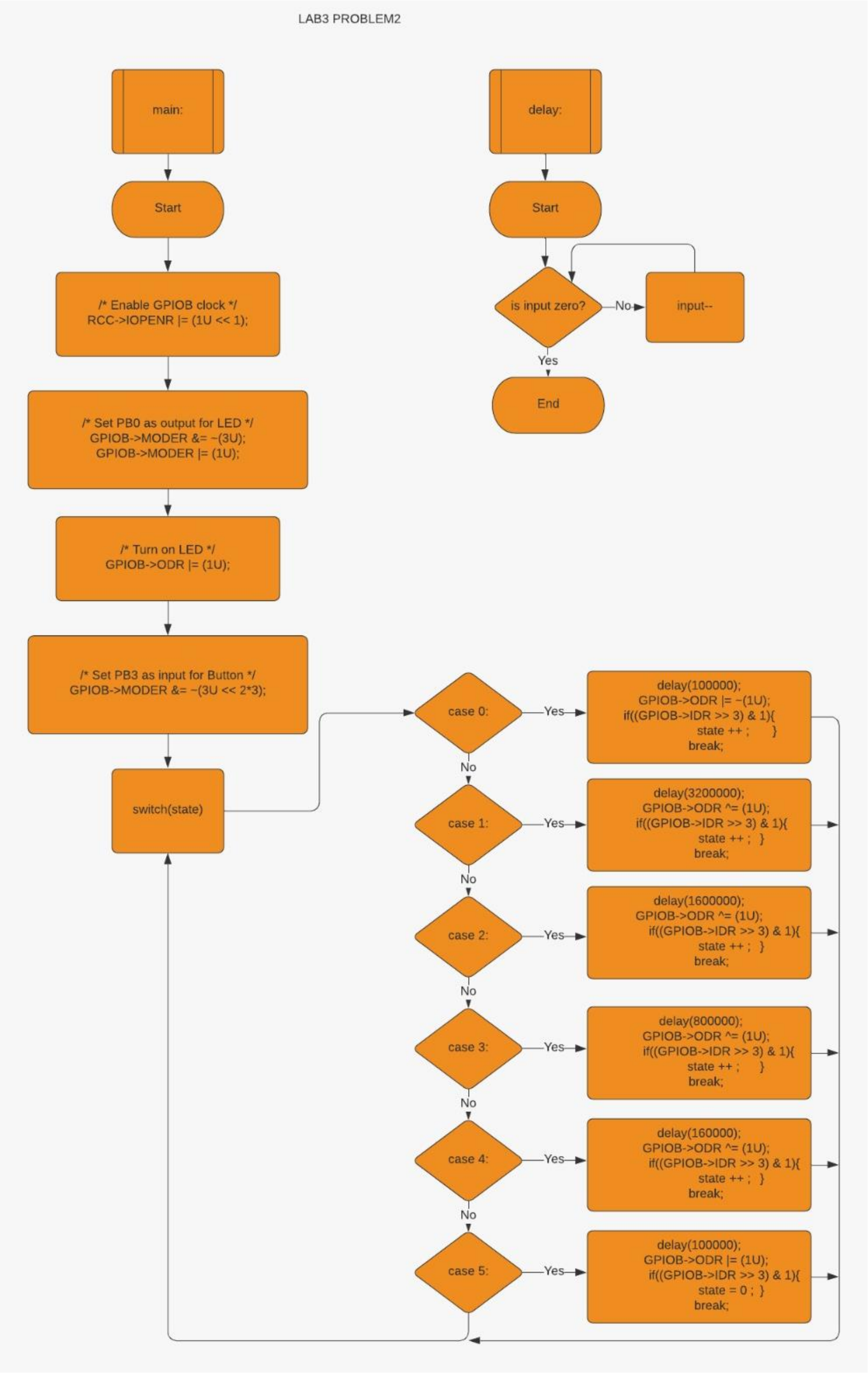


Figure 14: Flowchart Problem 2

2.3 BLOCK DIAGRAM (PROBLEM 2)

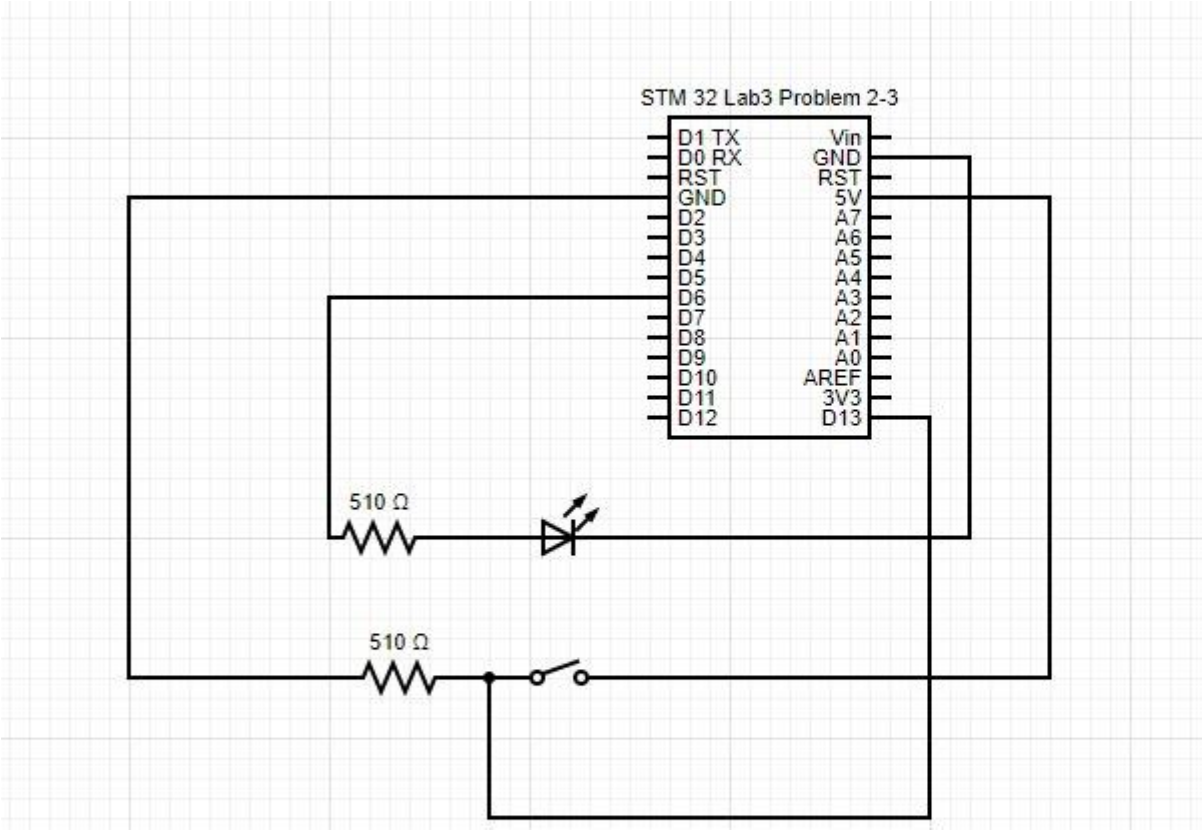


Figure 15: Block Diagram Problem 2

2.4 CODES (PROBLEM 2)

```
#include "stm32g0xx.h"

void delay(volatile uint32_t);

enum mode{mode0, mode1, mode2, mode3, mode4, mode5};
enum mode state;

int main(void) {

    /* Enable GPIOB clock */

    RCC->IOPENR |= (1U << 1);

    /* Setup PB0 as output for led*/
    GPIOB->MODER &= ~(3U);
    GPIOB->MODER |= (1U);

    /* Turn on LED */
      /* GPIOB->ODR |= (1U); */

    /* Setup PB3 as input for button*/
    GPIOB->MODER &= ~(3U << 2*3);
```

```
while(1) {
  switch(state){

  case 0:
         //Led is of no toggling
         GPIOB->ODR |= ~(1U);
         delay(1000000);

         if((GPIOB->IDR >> 3) & 1){
               state ++ ;   }
         break;

  case 1:

         GPIOB->ODR ^= (1U);
         delay(3200000);

         if((GPIOB->IDR >> 3) & 1){
             state ++ ;     }
         break;

  case 2:

         GPIOB->ODR ^= (1U);
         delay(1600000);

         if((GPIOB->IDR >> 3) & 1){
             state ++ ;     }
         break;

  case 3:

      GPIOB->ODR ^= (1U);
      delay(800000);

      if((GPIOB->IDR >> 3) & 1){
          state ++ ;  }
      break;

  case 4:

         GPIOB->ODR ^= (1U);
         delay(160000);

         if((GPIOB->IDR >> 3) & 1){
             state ++ ;     }
         break;

  case 5:
         //Led is on no toggling
         GPIOB->ODR |= (1U);
         delay(1000000);

         if((GPIOB->IDR >> 3) & 1){
             state = 0 ;    }
         break;
  }
```

```
        /* Toggle LED */

    }

    return 0;
}

void delay(volatile uint32_t s) {
    for(; s>0; s--);
}
```

3 PROBLEM 3

3.1 DEFINITON (PROBLEM 3)

In this problem, the operation in the second problem is repeated using interrupt and default handler. The oscilloscope image for mode one is as follows. There is no change.

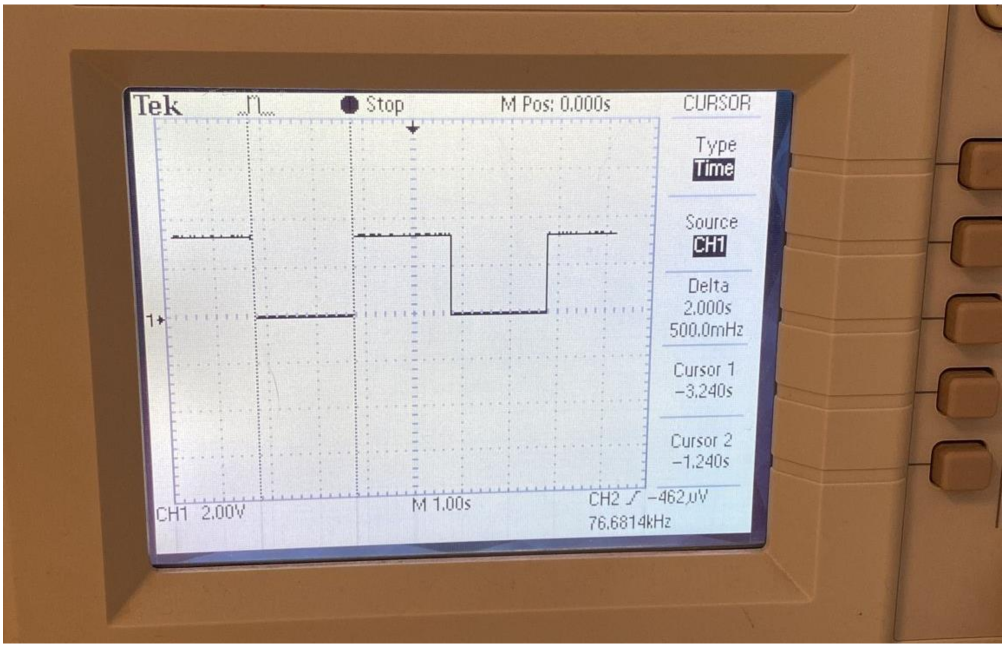


Figure 16: Mode 1 Problem 3 (2sec)

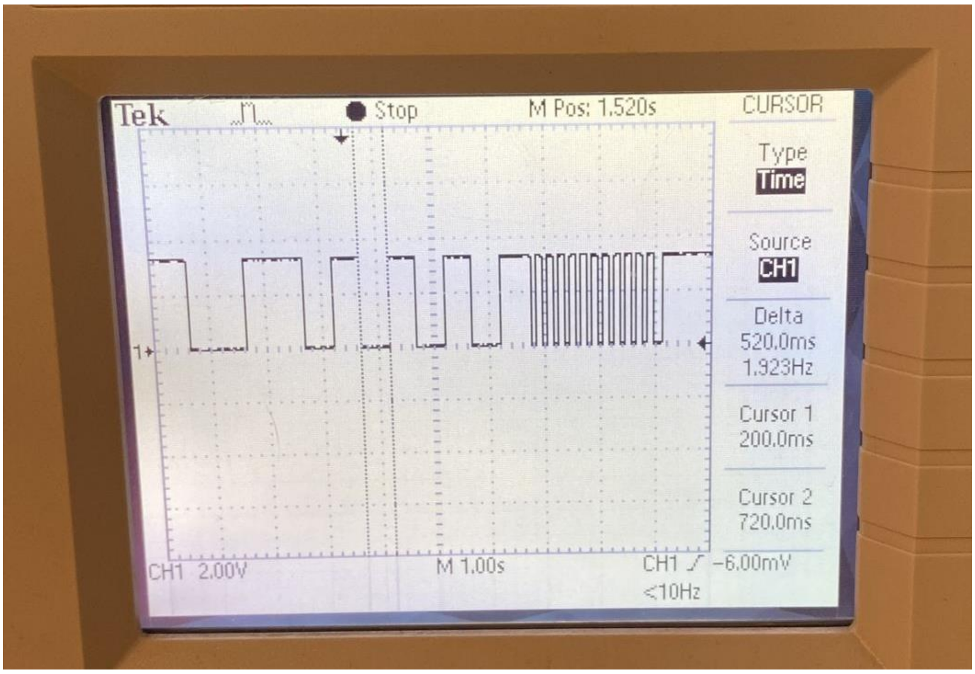


Figure 17: All Of The Modes

3.2 FLOWCHART (PROBLEM 3)

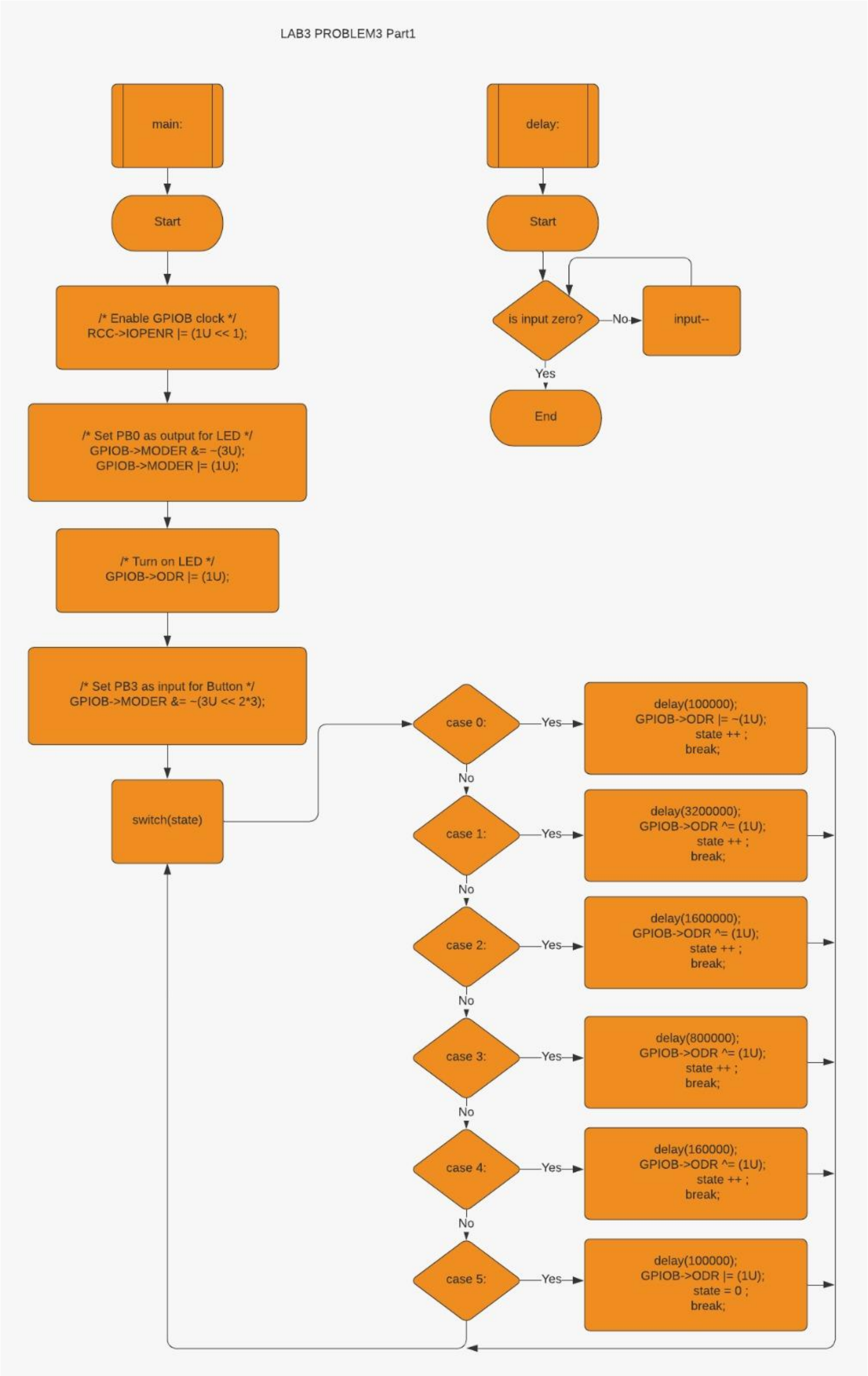


Figure 18: Flowchart for one

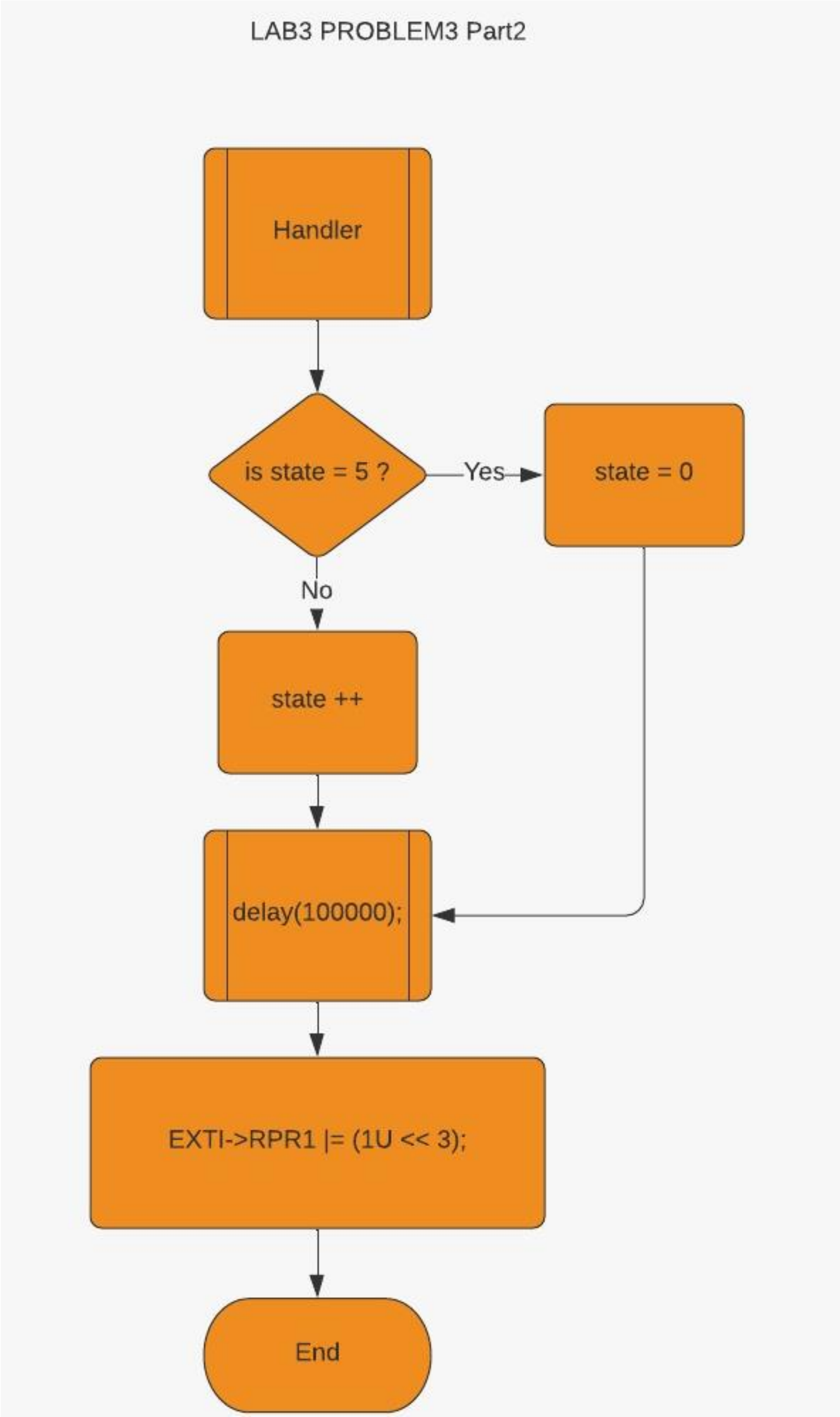


Figure 19: Flowchart part 2

3.3 BLOCK DIAGRAM (PROBLEM 3)

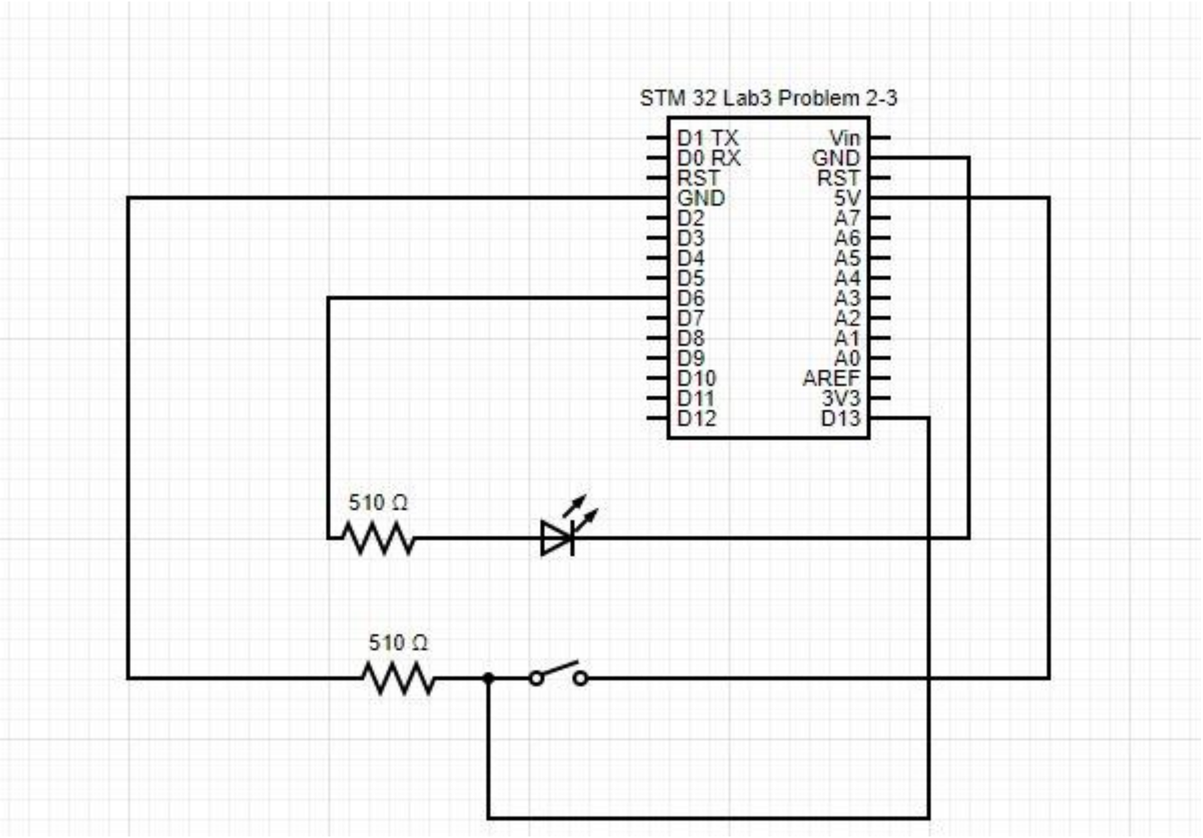


Figure 20: Block Diagram (Problem 3)

3.4 CODES (PROBLEM 3)

```
#include "stm32g0xx.h"


void delay(volatile uint32_t);
enum mode{mode0, mode1, mode2, mode3, mode4, mode5};
volatile enum mode state;

void EXTI2_3_IRQHandler(void) {
	if (state != 5) {
		state ++;
	}
	else {
		state = 0;
	}

	delay(100000);
	EXTI->RPR1 |= (1U << 3);

}

int main(void) {
    /* Enable GPIOB clock */
    RCC->IOPENR |= (1U << 1);

    /* Setup PB3 as input */
    GPIOB->MODER &= ~(3U << 2*3);
    /* Setup PB0 as output */
    GPIOB->MODER &= ~(3U);
    GPIOB->MODER |= (1U);

    EXTI->RTSR1 |= (1U << 3); // B3
    EXTI->EXTICR[0] |= (1U << 8*3);
    EXTI->IMR1 |= (1U << 3);

    NVIC_SetPriority(EXTI2_3_IRQn, 0);
    NVIC_EnableIRQ(EXTI2_3_IRQn);


    while(1) {
      switch(state){
      case 0:
            GPIOB->BRR |= (1U); // turn off led
            break;
      case 1:
            GPIOB->ODR ^= (1U);
            delay(3200000);
            break;
      case 2:
            GPIOB->ODR ^= (1U);
            delay(1600000);
            break;
      case 3:
                  GPIOB->ODR ^= (1U);
                  delay(800000);
            break;
      case 4:
            GPIOB->ODR ^= (1U);
            delay(160000);
            break;
      case 5:
            GPIOB->ODR |= (1U); // turn on led
            break;
      }
    }

    return 0;
}

void delay(volatile uint32_t s) {
    for(; s>0; s--);
}
```

4 PROBLEM 4

4.1 DEFINITION (PROBLEM 4)

Connect the keypad to the microcontroller and detect button presses using external interrupts. An SSD is used to display the pressed button. The circuit is as given below.

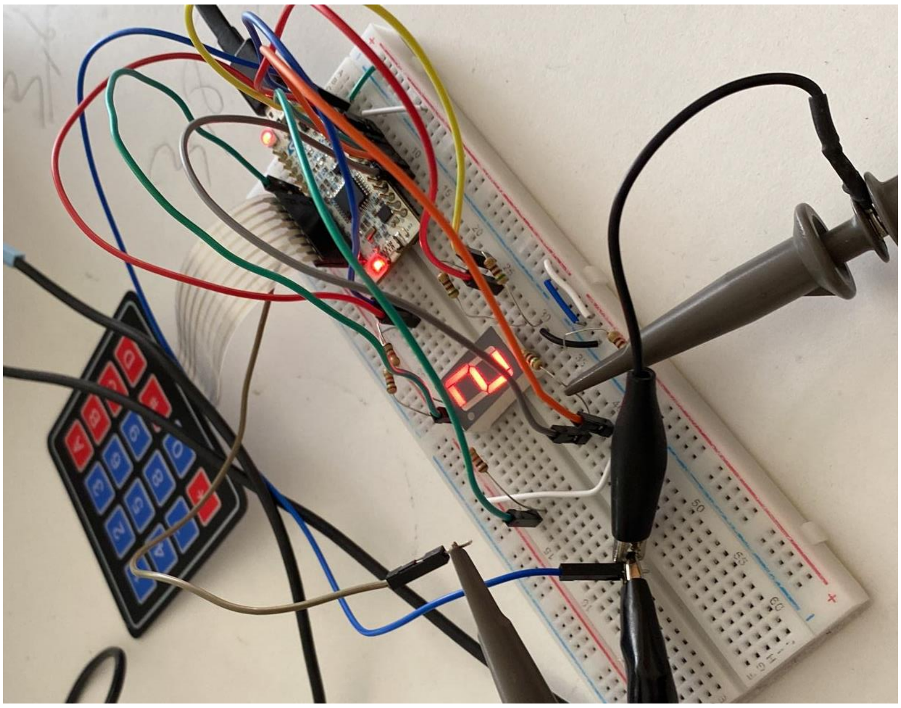


Figure 21: Circuit In Breadboard

The delay times are observed on the oscilloscope screen as follows.

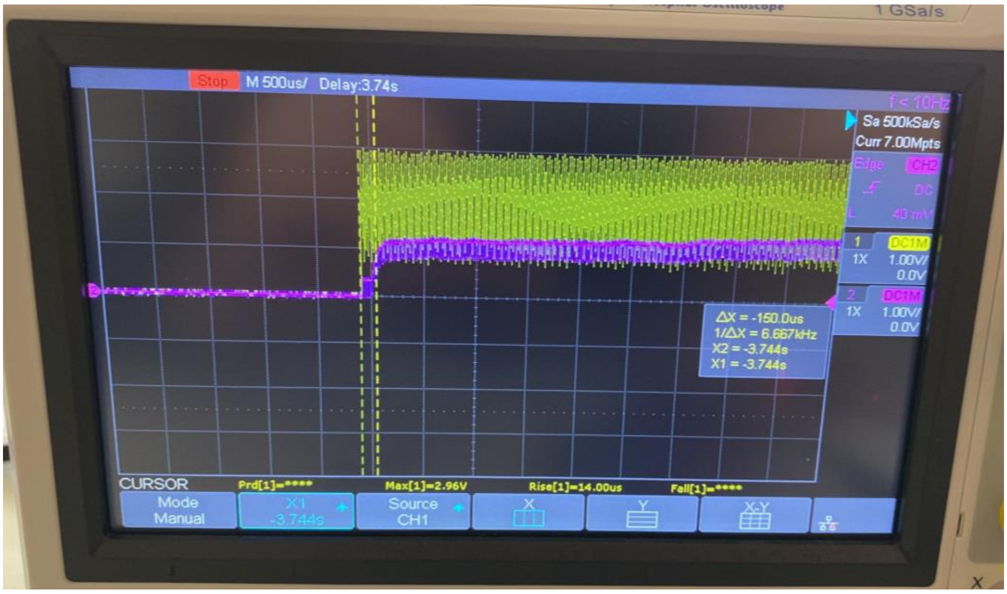


Figure 22: Time Delay

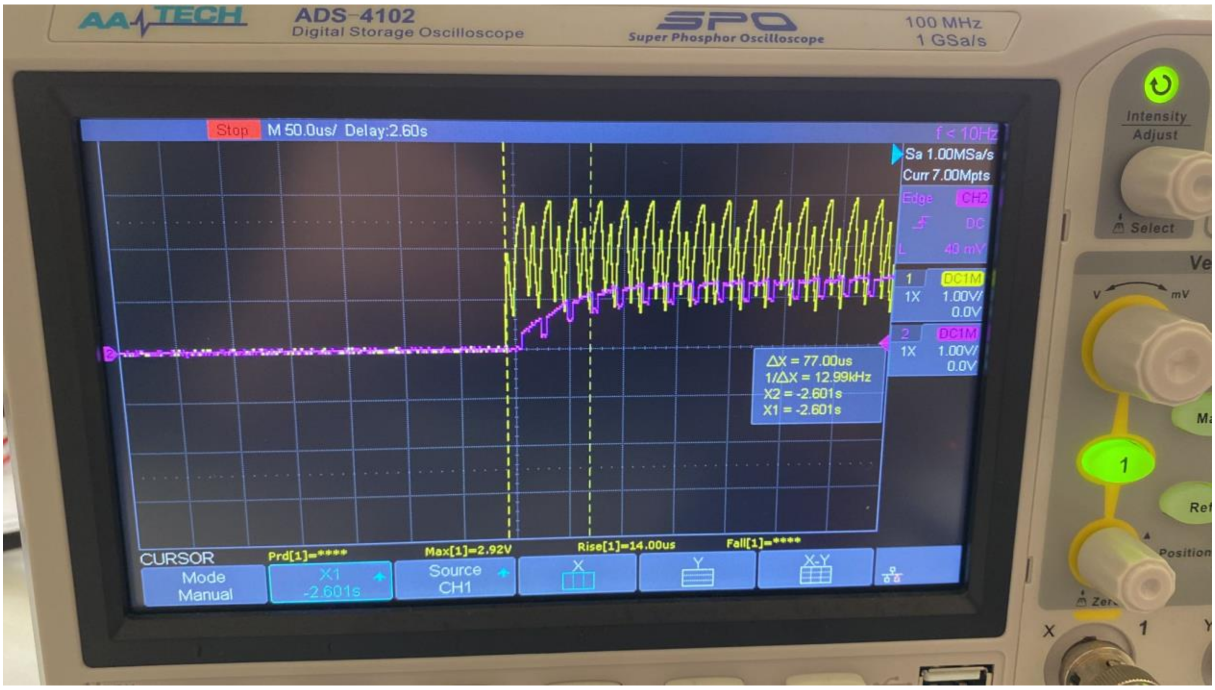


Figure 23: Time Delay

The oscilloscope image of the **holding time** is as follows.

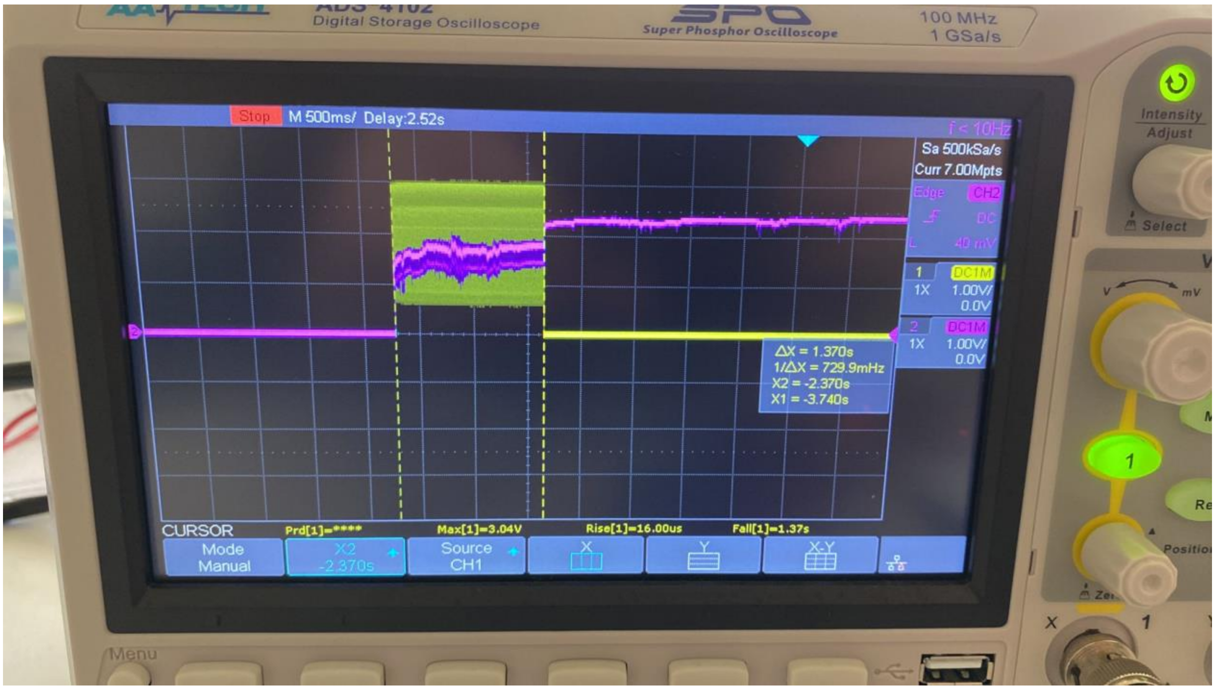
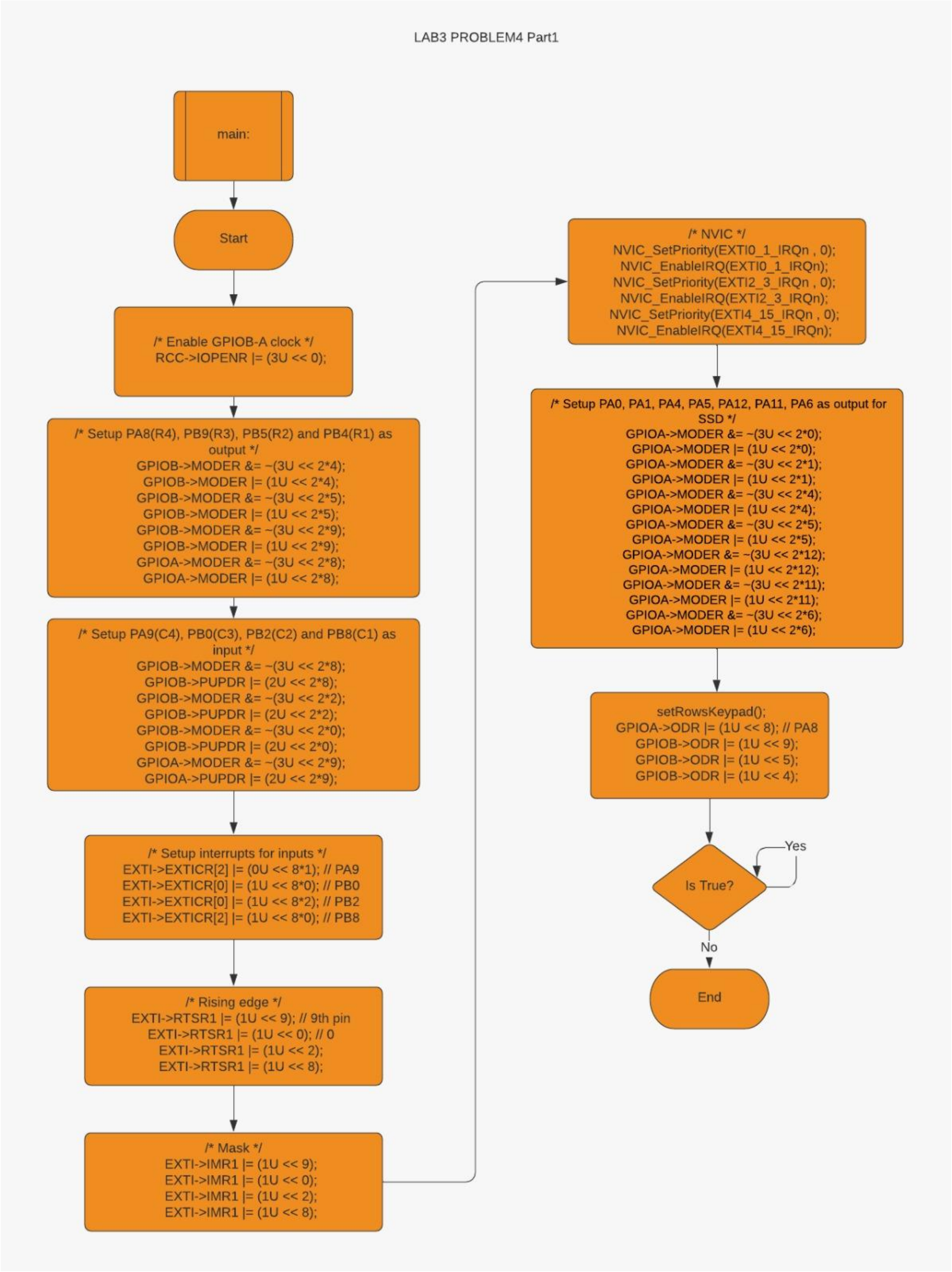


Figure 24: Holding Time

4.2 FLOWCHART (PROBLEM 4)



```
/* Enable GPIOB-A clock */
RCC->IOPENR |= (3U << 0);
```

```
/* Setup PA8(R4), PB9(R3), PB5(R2) and PB4(R1) as output */
GPIOB->MODER &= ~(3U << 2*4);
GPIOB->MODER |= (1U << 2*4);
GPIOB->MODER &= ~(3U << 2*5);
GPIOB->MODER |= (1U << 2*5);
GPIOB->MODER &= ~(3U << 2*9);
GPIOB->MODER |= (1U << 2*9);
GPIOA->MODER &= ~(3U << 2*8);
GPIOA->MODER |= (1U << 2*8);
```

```
/* Setup PA9(C4), PB0(C3), PB2(C2) and PB8(C1) as input */
GPIOB->MODER &= ~(3U << 2*8);
GPIOB->PUPDR |= (2U << 2*8);
GPIOB->MODER &= ~(3U << 2*2);
GPIOB->PUPDR |= (2U << 2*2);
GPIOB->MODER &= ~(3U << 2*0);
GPIOB->PUPDR |= (2U << 2*0);
GPIOA->MODER &= ~(3U << 2*9);
GPIOA->PUPDR |= (2U << 2*9);
```

```
/* Setup interrupts for inputs */
EXTI->EXTICR[2] |= (0U << 8*1); // PA9
EXTI->EXTICR[0] |= (1U << 8*0); // PB0
EXTI->EXTICR[0] |= (1U << 8*2); // PB2
EXTI->EXTICR[2] |= (1U << 8*0); // PB8
```

```
/* Rising edge */
EXTI->RTSR1 |= (1U << 9); // 9th pin
EXTI->RTSR1 |= (1U << 0); // 0
EXTI->RTSR1 |= (1U << 2);
EXTI->RTSR1 |= (1U << 8);
```

```
/* Mask */
EXTI->IMR1 |= (1U << 9);
EXTI->IMR1 |= (1U << 0);
EXTI->IMR1 |= (1U << 2);
EXTI->IMR1 |= (1U << 8);
```

```
/* NVIC */
NVIC_SetPriority(EXTI0_1_IRQn , 0);
NVIC_EnableIRQ(EXTI0_1_IRQn);
NVIC_SetPriority(EXTI2_3_IRQn , 0);
NVIC_EnableIRQ(EXTI2_3_IRQn);
NVIC_SetPriority(EXTI4_15_IRQn , 0);
NVIC_EnableIRQ(EXTI4_15_IRQn);
```

```
/* Setup PA0, PA1, PA4, PA5, PA12, PA11, PA6 as output for SSD */
GPIOA->MODER &= ~(3U << 2*0);
GPIOA->MODER |= (1U << 2*0);
GPIOA->MODER &= ~(3U << 2*1);
GPIOA->MODER |= (1U << 2*1);
GPIOA->MODER &= ~(3U << 2*4);
GPIOA->MODER |= (1U << 2*4);
GPIOA->MODER &= ~(3U << 2*5);
GPIOA->MODER |= (1U << 2*5);
GPIOA->MODER &= ~(3U << 2*12);
GPIOA->MODER |= (1U << 2*12);
GPIOA->MODER &= ~(3U << 2*11);
GPIOA->MODER |= (1U << 2*11);
GPIOA->MODER &= ~(3U << 2*6);
GPIOA->MODER |= (1U << 2*6);
```

```
setRowsKeypad();
GPIOA->ODR |= (1U << 8); // PA8
GPIOB->ODR |= (1U << 9);
GPIOB->ODR |= (1U << 5);
GPIOB->ODR |= (1U << 4);
```

Figure 25: Flowchart Problem 4

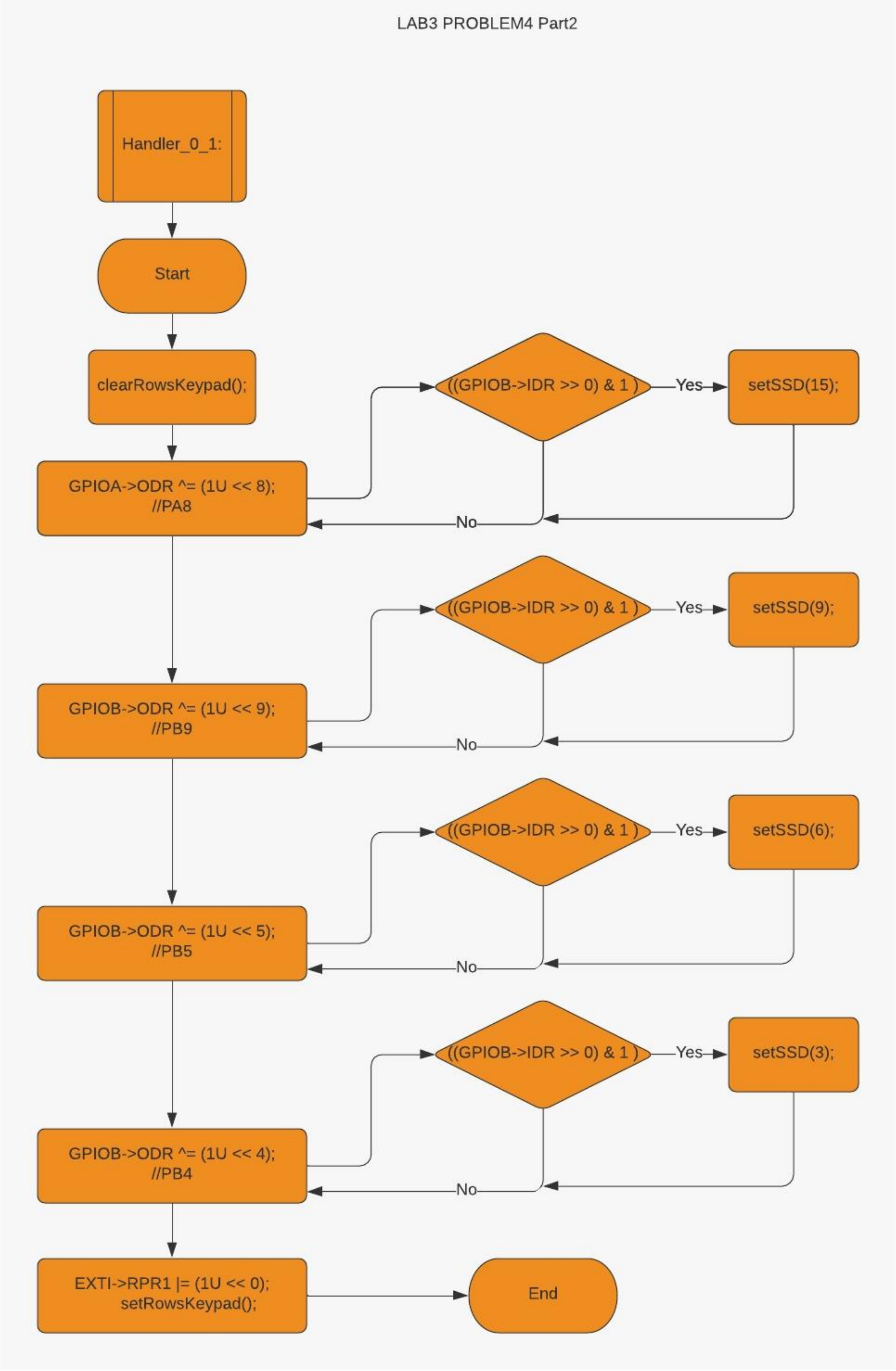


Figure 26: Flowchart Problem 4

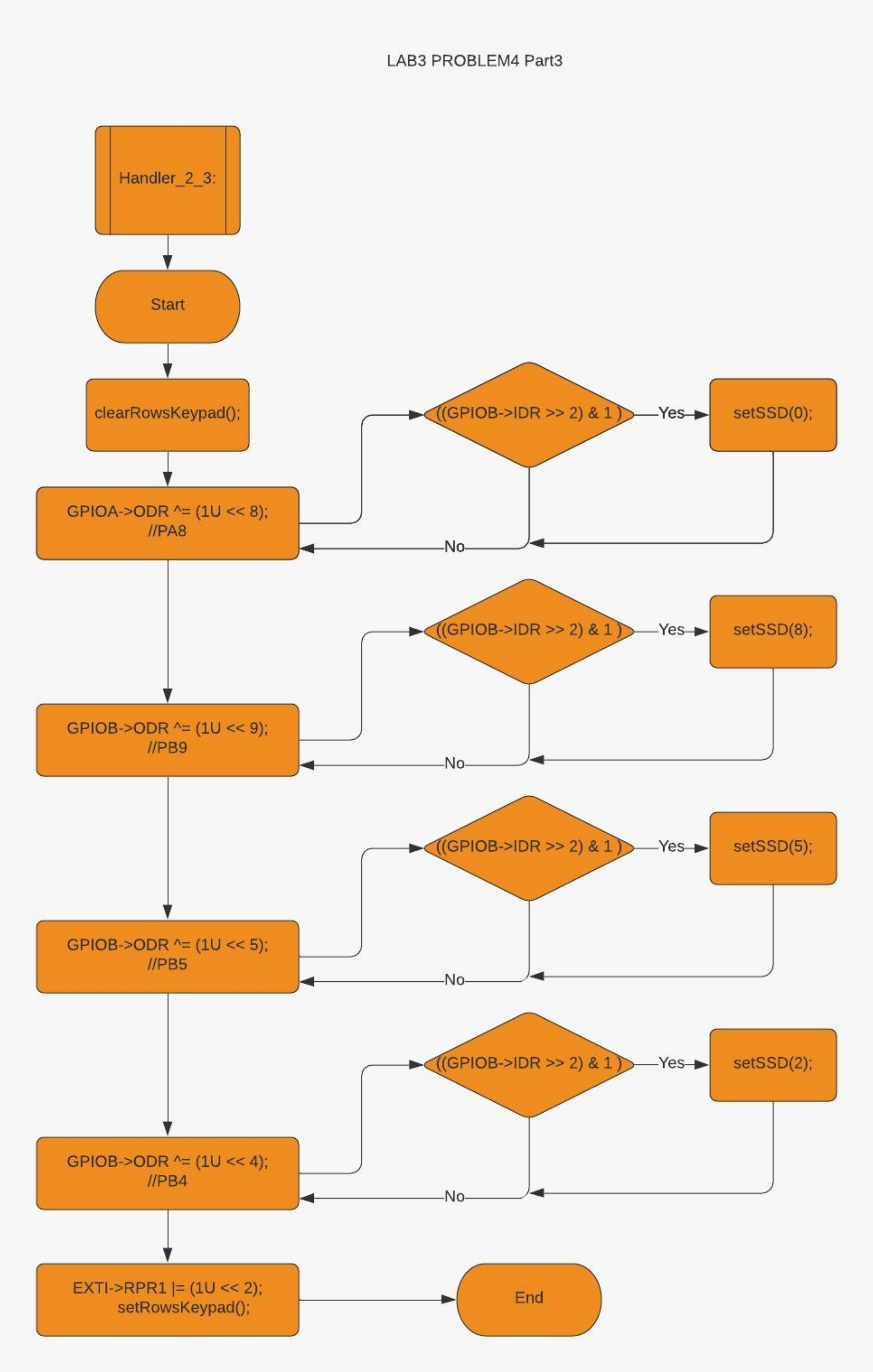


Figure 27: Flowchart Problem 4

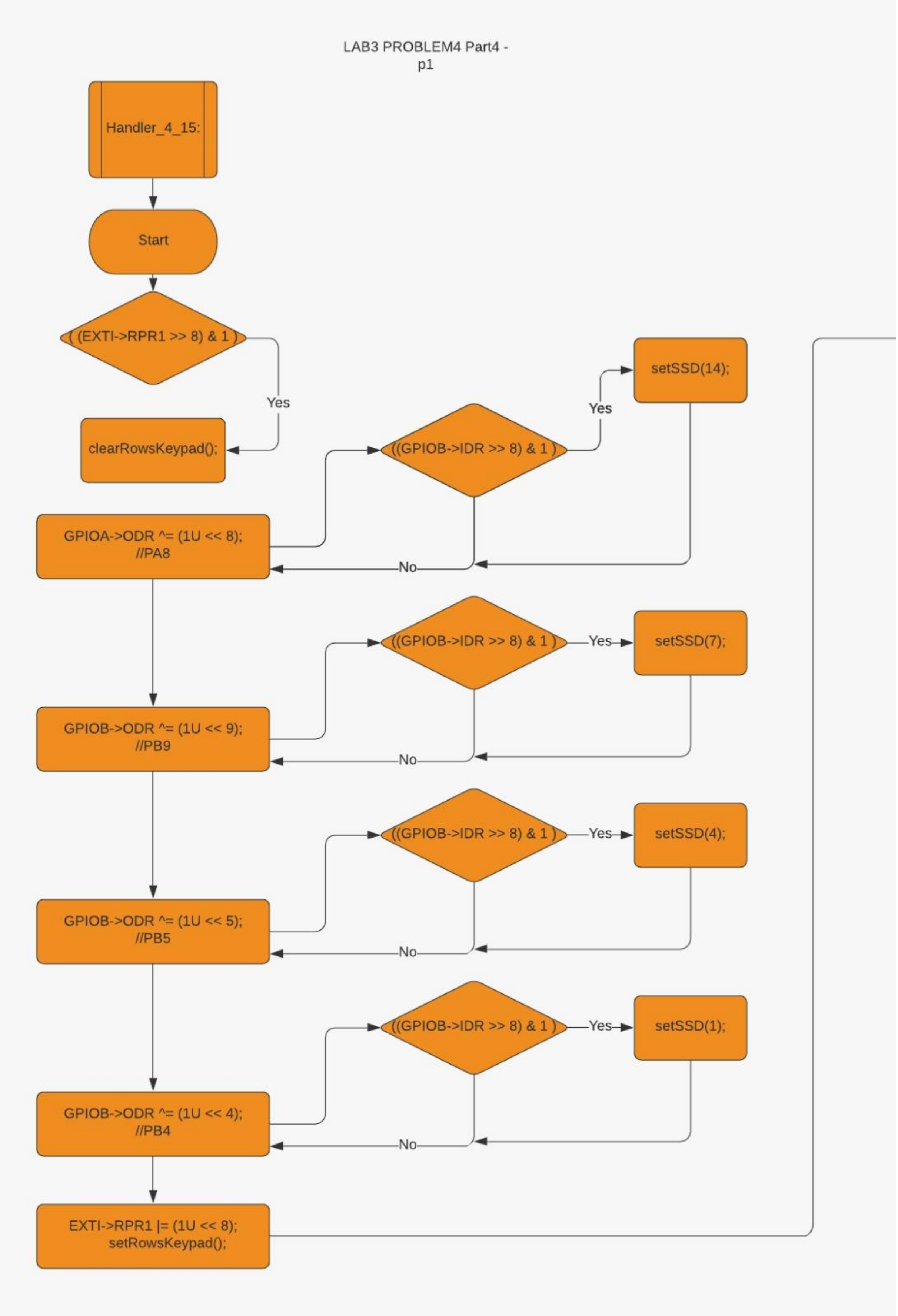


Figure 28: Flowchart Problem 4

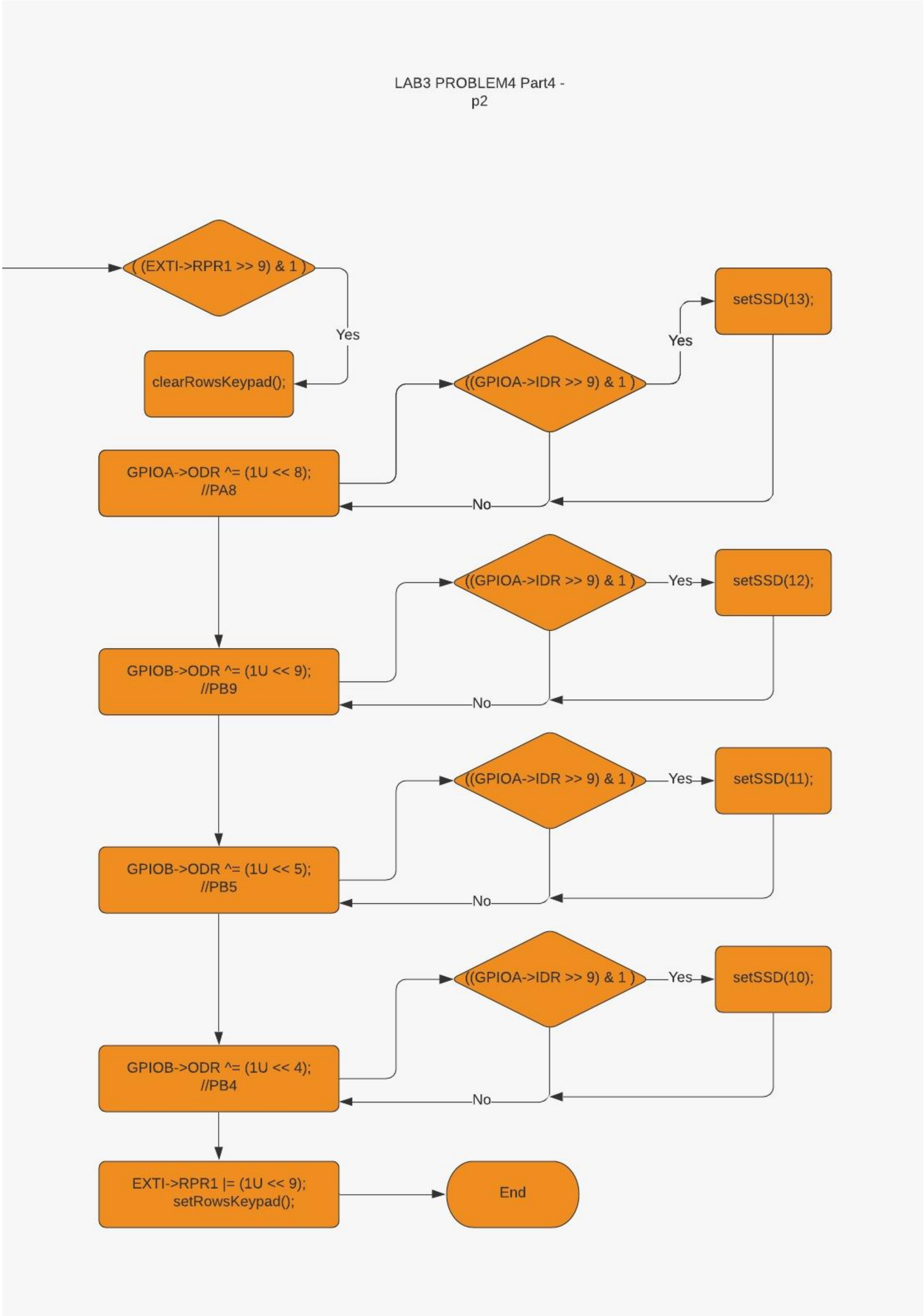


Figure 29: Flowchart Problem 4

4.3 BLOCK DIAGRAM (PROBLEM 4)

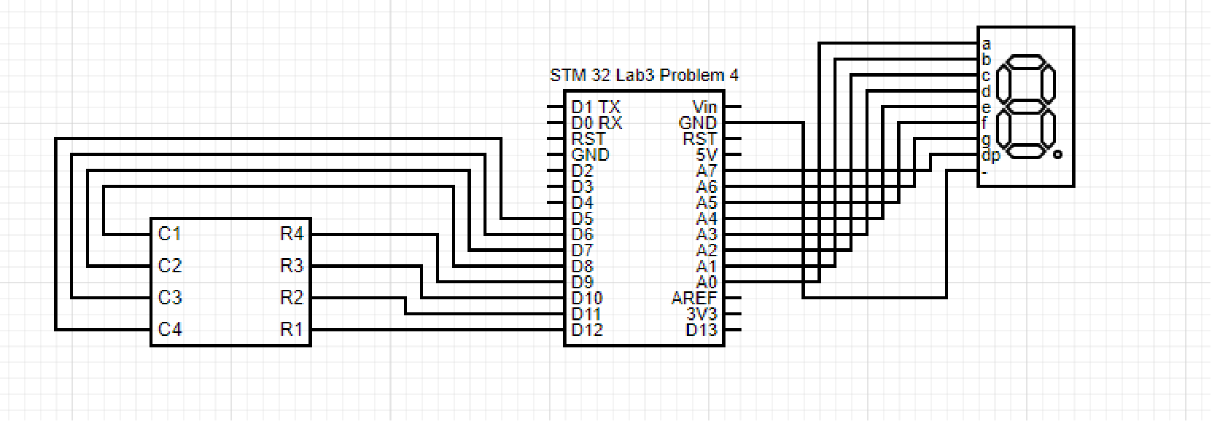


Figure 30: Block Diagram

4.4 CODES (PROBLEM 4)

```
#include "stm32g0xx.h"

#define LEDDELAY    1600000

void delay(volatile uint32_t);

void clearRowsKeypad(void);
void setRowsKeypad(void);

void setSSD(int x);
void clearSSD(void);

void EXTI0_1_IRQHandler(void){ //PB0 (C3)
    clearRowsKeypad();

    GPIOA->ODR ^= (1U << 8); //PA8
    if ((GPIOB->IDR >> 0) & 1 ) { // read PB0
        setSSD(15);
    }
    GPIOA->ODR ^= (1U << 8); //PA8

    GPIOB->ODR ^= (1U << 9); //PB9
    if ((GPIOB->IDR >> 0) & 1 ) { // read PB0
        setSSD(9);
    }
    GPIOB->ODR ^= (1U << 9); //PB9

    GPIOB->ODR ^= (1U << 5); //PB5
    if ((GPIOB->IDR >> 0) & 1 ) { // read PB0
        setSSD(6);
    }
    GPIOB->ODR ^= (1U << 5); //PB5

    GPIOB->ODR ^= (1U << 4); //PB4
    if ((GPIOB->IDR >> 0) & 1 ) { // read PB0
        setSSD(3);
    }
```

```
	GPIOB->ODR ^= (1U << 4); //PB4

	EXTI->RPR1 |= (1U << 0);
	setRowsKeypad();
}

void EXTI2_3_IRQHandler(void){ //PB2 (C2)
	clearRowsKeypad();

	GPIOA->ODR ^= (1U << 8); //PA8
	if ((GPIOB->IDR >> 2) & 1 ) { // read PB2
		setSSD(0);
	}
	GPIOA->ODR ^= (1U << 8); //PA8

	GPIOB->ODR ^= (1U << 9); //PB9
	if ((GPIOB->IDR >> 2) & 1 ) { // read PB2
		setSSD(8);
	}
	GPIOB->ODR ^= (1U << 9); //PB9

	GPIOB->ODR ^= (1U << 5); //PB5
	if ((GPIOB->IDR >> 2) & 1 ) { // read PB2
		setSSD(5);
	}
	GPIOB->ODR ^= (1U << 5); //PB5

	GPIOB->ODR ^= (1U << 4); //PB4
	if ((GPIOB->IDR >> 2) & 1 ) { // read PB2
		setSSD(2);
	}
	GPIOB->ODR ^= (1U << 4); //PB4

	EXTI->RPR1 |= (1U << 2);
	setRowsKeypad();
}

void EXTI4_15_IRQHandler(void){ //PB8 (C1) and PA9 (C4)

	if ( (EXTI->RPR1 >> 8) & 1 ) {
		clearRowsKeypad();

		GPIOA->ODR ^= (1U << 8); //PA8
		if ((GPIOB->IDR >> 8) & 1 ) { // read PB8
			setSSD(14);
		}
		GPIOA->ODR ^= (1U << 8); //PA8

		GPIOB->ODR ^= (1U << 9); //PB9
		if ((GPIOB->IDR >> 8) & 1 ) { // read PB8
			setSSD(7);
		}
		GPIOB->ODR ^= (1U << 9); //PB9

		GPIOB->ODR ^= (1U << 5); //PB5
		if ((GPIOB->IDR >> 8) & 1 ) { // read PB8
			setSSD(4);
		}
		GPIOB->ODR ^= (1U << 5); //PB5
```

```
             GPIOB->ODR ^= (1U << 4); //PB4
             if ((GPIOB->IDR >> 8) & 1 ) { // read PB8
                    setSSD(1);
             }
             GPIOB->ODR ^= (1U << 4); //PB4

             EXTI->RPR1 |= (1U << 8);
             setRowsKeypad();
      }
      if ( (EXTI->RPR1 >> 9) & 1 ) {
             clearRowsKeypad();

             GPIOA->ODR ^= (1U << 8); //PA8
             if ((GPIOA->IDR >> 9) & 1 ) { // read PA9
                    setSSD(13);
             }
             GPIOA->ODR ^= (1U << 8); //PA8

             GPIOB->ODR ^= (1U << 9); //PB9
             if ((GPIOA->IDR >> 9) & 1 ) { // read PA9
                    setSSD(12);
             }
             GPIOB->ODR ^= (1U << 9); //PB9

             GPIOB->ODR ^= (1U << 5); //PB5
             if ((GPIOA->IDR >> 9) & 1 ) { // read PA9
                    setSSD(11);
             }
             GPIOB->ODR ^= (1U << 5); //PB5

             GPIOB->ODR ^= (1U << 4); //PB4
             if ((GPIOA->IDR >> 9) & 1 ) { // read PA9
                    setSSD(10);
             }
             GPIOB->ODR ^= (1U << 4); //PB4

             EXTI->RPR1 |= (1U << 9);
             setRowsKeypad();
      }


}

int main(void) {
    /* Enable GPIOB-A clock */
    RCC->IOPENR |= (3U << 0);

    /* Setup PA8(R4), PB9(R3), PB5(R2) and PB4(R1) as output */
    GPIOB->MODER &= ~(3U << 2*4);
    GPIOB->MODER |= (1U << 2*4);

    GPIOB->MODER &= ~(3U << 2*5);
    GPIOB->MODER |= (1U << 2*5);

    GPIOB->MODER &= ~(3U << 2*9);
    GPIOB->MODER |= (1U << 2*9);

    GPIOA->MODER &= ~(3U << 2*8);
```

```
GPIOA->MODER |= (1U << 2*8);

/* Setup PA9(C4), PB0(C3), PB2(C2) and PB8(C1) as input */
GPIOB->MODER &= ~(3U << 2*8);
GPIOB->PUPDR |= (2U << 2*8);

GPIOB->MODER &= ~(3U << 2*2);
GPIOB->PUPDR |= (2U << 2*2);

GPIOB->MODER &= ~(3U << 2*0);
GPIOB->PUPDR |= (2U << 2*0);

GPIOA->MODER &= ~(3U << 2*9);
GPIOA->PUPDR |= (2U << 2*9);

/* Setup interrupts for inputs */
EXTI->EXTICR[2] |= (0U << 8*1); // PA9
EXTI->EXTICR[0] |= (1U << 8*0); // PB0
EXTI->EXTICR[0] |= (1U << 8*2); // PB2
EXTI->EXTICR[2] |= (1U << 8*0); // PB8

/* Rising edge */
EXTI->RTSR1 |= (1U << 9); // 9th pin
EXTI->RTSR1 |= (1U << 0); // 0
EXTI->RTSR1 |= (1U << 2);
EXTI->RTSR1 |= (1U << 8);

/* Mask */
EXTI->IMR1 |= (1U << 9);
EXTI->IMR1 |= (1U << 0);
EXTI->IMR1 |= (1U << 2);
EXTI->IMR1 |= (1U << 8);

/* NVIC */
NVIC_SetPriority(EXTI0_1_IRQn , 0);
NVIC_EnableIRQ(EXTI0_1_IRQn);

NVIC_SetPriority(EXTI2_3_IRQn , 0);
NVIC_EnableIRQ(EXTI2_3_IRQn);

NVIC_SetPriority(EXTI4_15_IRQn , 0);
NVIC_EnableIRQ(EXTI4_15_IRQn);



/* Setup PA0, PA1, PA4, PA5, PA12, PA11, PA6 as output for SSD */
GPIOA->MODER &= ~(3U << 2*0);
GPIOA->MODER |= (1U << 2*0);

GPIOA->MODER &= ~(3U << 2*1);
GPIOA->MODER |= (1U << 2*1);

GPIOA->MODER &= ~(3U << 2*4);
GPIOA->MODER |= (1U << 2*4);

GPIOA->MODER &= ~(3U << 2*5);
GPIOA->MODER |= (1U << 2*5);

GPIOA->MODER &= ~(3U << 2*12);
```

```
    GPIOA->MODER |= (1U << 2*12);

    GPIOA->MODER &= ~(3U << 2*11);
    GPIOA->MODER |= (1U << 2*11);

    GPIOA->MODER &= ~(3U << 2*6);
    GPIOA->MODER |= (1U << 2*6);

    /* Set all rows */
    setRowsKeypad();
    GPIOA->ODR |= (1U << 8); // PA8
    GPIOB->ODR |= (1U << 9);
    GPIOB->ODR |= (1U << 5);
    GPIOB->ODR |= (1U << 4);

    while(1) {
      //Do nothing
    }

    return 0;
}

void delay(volatile uint32_t s) {
    for(; s>0; s--);
}

void clearRowsKeypad(void){
        GPIOA->BRR |= (1U << 8); // PA8
        GPIOB->BRR |= (1U << 9);
        GPIOB->BRR |= (1U << 5);
        GPIOB->BRR |= (1U << 4);
}

void setRowsKeypad(void){
        GPIOA->ODR |= (1U << 8); // PA8
        GPIOB->ODR |= (1U << 9);
        GPIOB->ODR |= (1U << 5);
        GPIOB->ODR |= (1U << 4);
}

void setSSD(int x){
      clearSSD();
      switch(x) {
      case 0:
            GPIOA->ODR |= (1U << 0); // A
            GPIOA->ODR |= (1U << 1); // B
            GPIOA->ODR |= (1U << 4); // C
            GPIOA->ODR |= (1U << 5); // D
            GPIOA->ODR |= (1U << 12); // E
            GPIOA->ODR |= (1U << 11); // F
            break;
      case 1:
            GPIOA->ODR |= (1U << 1); // B
            GPIOA->ODR |= (1U << 4); // C
            break;
      case 2:
            GPIOA->ODR |= (1U << 0); // A
            GPIOA->ODR |= (1U << 1); // B
            GPIOA->ODR |= (1U << 5); // D
            GPIOA->ODR |= (1U << 12); // E
            GPIOA->ODR |= (1U << 6); // G

            break;
      case 3:
            GPIOA->ODR |= (1U << 0); // A
            GPIOA->ODR |= (1U << 1); // B
            GPIOA->ODR |= (1U << 4); // C
            GPIOA->ODR |= (1U << 5); // D
            GPIOA->ODR |= (1U << 6); // G
            break;
      case 4:
            GPIOA->ODR |= (1U << 1); // B
            GPIOA->ODR |= (1U << 4); // C
            GPIOA->ODR |= (1U << 11); // F
            GPIOA->ODR |= (1U << 6); // G
            break;
      case 5:
            GPIOA->ODR |= (1U << 0); // A
            GPIOA->ODR |= (1U << 4); // C
            GPIOA->ODR |= (1U << 5); // D
            GPIOA->ODR |= (1U << 11); // F
            GPIOA->ODR |= (1U << 6); // G
            break;
      case 6:
            GPIOA->ODR |= (1U << 0); // A
            GPIOA->ODR |= (1U << 4); // C
            GPIOA->ODR |= (1U << 5); // D
            GPIOA->ODR |= (1U << 12); // E
            GPIOA->ODR |= (1U << 11); // F
            GPIOA->ODR |= (1U << 6); // G
            break;
      case 7:
            GPIOA->ODR |= (1U << 0); // A
            GPIOA->ODR |= (1U << 1); // B
            GPIOA->ODR |= (1U << 4); // C
            break;
      case 8:
            GPIOA->ODR |= (1U << 0); // A
            GPIOA->ODR |= (1U << 1); // B
            GPIOA->ODR |= (1U << 4); // C
            GPIOA->ODR |= (1U << 5); // D
            GPIOA->ODR |= (1U << 12); // E
            GPIOA->ODR |= (1U << 11); // F
            GPIOA->ODR |= (1U << 6); // G
            break;
      case 9:
            GPIOA->ODR |= (1U << 0); // A
            GPIOA->ODR |= (1U << 1); // B
            GPIOA->ODR |= (1U << 4); // C
            GPIOA->ODR |= (1U << 11); // F
            GPIOA->ODR |= (1U << 6); // G
            break;
      case 10: // A
            GPIOA->ODR |= (1U << 0); // A
            GPIOA->ODR |= (1U << 1); // B
            GPIOA->ODR |= (1U << 4); // C
            GPIOA->ODR |= (1U << 12); // E
            GPIOA->ODR |= (1U << 11); // F
            GPIOA->ODR |= (1U << 6); // G
            break;
        case 11: // B
            GPIOA->ODR |= (1U << 4); // C
            GPIOA->ODR |= (1U << 5); // D
            GPIOA->ODR |= (1U << 12); // E
            GPIOA->ODR |= (1U << 11); // F
            GPIOA->ODR |= (1U << 6); // G
            break;
        case 12: // C
            GPIOA->ODR |= (1U << 0); // A
            GPIOA->ODR |= (1U << 5); // D
            GPIOA->ODR |= (1U << 12); // E
            GPIOA->ODR |= (1U << 11); // F
            break;
        case 13: // D
            GPIOA->ODR |= (1U << 1); // B
            GPIOA->ODR |= (1U << 4); // C
            GPIOA->ODR |= (1U << 5); // D
            GPIOA->ODR |= (1U << 12); // E
            GPIOA->ODR |= (1U << 6); // G
            break;
        case 14: // *
            GPIOA->ODR |= (1U << 6); // G
            break;
        case 15: // #
            GPIOA->ODR |= (1U << 1); // B
            GPIOA->ODR |= (1U << 4); // C
            GPIOA->ODR |= (1U << 12); // E
            GPIOA->ODR |= (1U << 11); // F
            GPIOA->ODR |= (1U << 6); // G
            break;
        }
}

// clear PA0, PA1, PA4, PA5, PA12, PA11, PA6 for SSD
void clearSSD(void) {
        GPIOA->BRR |= (1U << 0); // A
        GPIOA->BRR |= (1U << 1); // B
        GPIOA->BRR |= (1U << 4); // C
        GPIOA->BRR |= (1U << 5); // D
        GPIOA->BRR |= (1U << 12); // E
        GPIOA->BRR |= (1U << 11); // F
        GPIOA->BRR |= (1U << 6); // G
}
```