MINISTRY OF SCIENCE AND HIGHER EDUCATION OF THE RUSSIAN FEDERATION
Federal State Autonomous Educational Institution
higher education
"SAINT PETERSBURG STATE UNIVERSITY
AEROSPACE INSTRUMENT MAKING»

DEPARTMENT OF COMPUTER TECHNOLOGIES AND SOFTWARE ENGINEERING (DEPARTMENT №43)

COURSE WORK (PROJECT)
DEFENDED WITH ASSESSMENT

SUPERVISOR:

Associate Professor, Ph.D.	/	/	/ A. A. Popov
(position, academic degree, title)	(signature)	(date of protection)	(initials, surname)

EXPLANATORY NOTE
TO COURSE WORK (PROJECT)

Garland of 8 LEDs

COURSE: "PROGRAMMING OF BUILT-IN APPLICATIONS"

THE WORK IS PERFORMED BY (A) STUDENT (KA) OF THE GROUP: 4936 / Belov E. I.
Krutov A.V.
Cherkasov E. B.
(initials, surname)

/ / (student's signature) (date of report)

Saint Petersburg 2022

Content

1. Assignment for course design

Implement the movement of the light to the left, to the right at a given speed (switching lines alternately), two lights.

To smoothly turn on and off the LEDs, use a PWM signal. Set the speed and direction of movement using the Keypad 4x4.

Which buttons to use, and what functions to give to decide on your own.

2. Terms of reference for the device

Within the framework of the developed model of the garland, the following performers (actors) are distinguished:

– User;
– LEDs;
– Microcontroller (as a control device).

The main functions of the garland:
An LED (two LEDs) moves along the garland in one direction or the other.

The garland has 3 buttons to change the operating mode:
1) Changing the speed of the LEDs (4 speeds - ignition and decay 1s, 0.5, 0.25, 0.04)
2) Changing the direction of movement (clockwise and counterclockwise)
3) Mode change (1 or 2 lights)
Restrictions:
When pressing several buttons at the same time, the handler for the last pressed button will work

2.1 Use case diagram

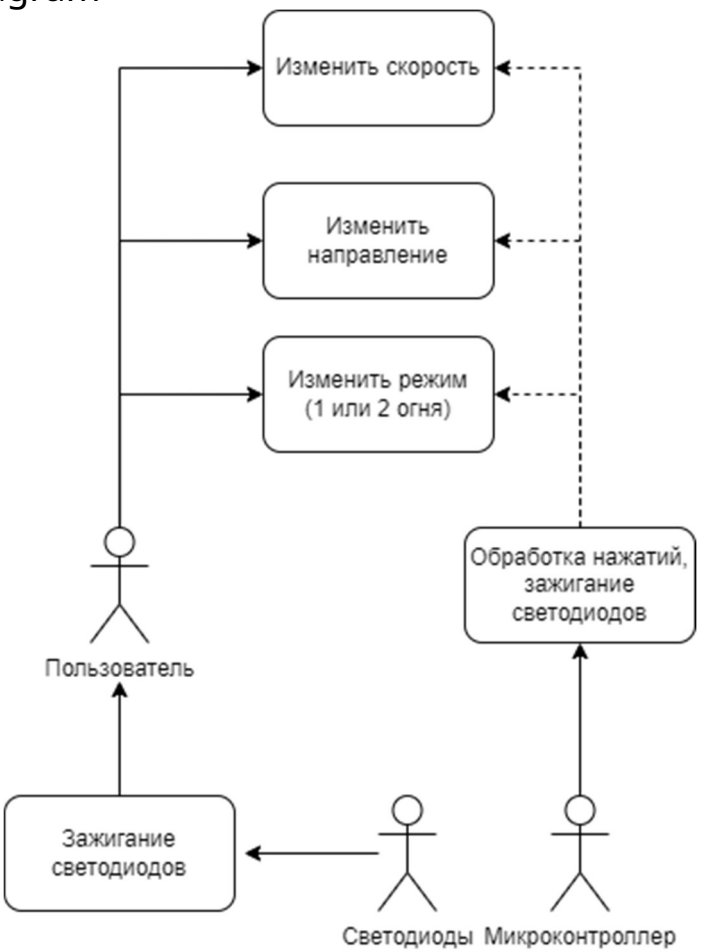

Figure 1 - use case diagram

2.2 Basic requirements

Inputs

3 buttons:

changing the speed of the LEDs changing
the direction of the movement
mode change (1 or 2 lights)

exits

8 LEDs

Functions

An LED (two LEDs) moves along the garland in one direction or the other. A PWM signal is used for smooth ignition and decay. Using the buttons, you can change the operating modes of the garland (1 or 2 lights), the direction of movement, the speed of the LEDs.

Peculiarities

Missing

Food

AC power via standard power supply (USB adapter).

Dimensions and weight

Small enough to use on a desktop

production cost

The cost of the STM32F103C8 debug board is 83 rubles.
The cost of a 4-button keyboard is 44 rubles. LEDs,
wires and housing - 100 rubles.
Assembly and testing - 50 rubles.

Thus, the cost of production of the product should be
no more than 300 rubles.

3. Schemes and algorithms of work

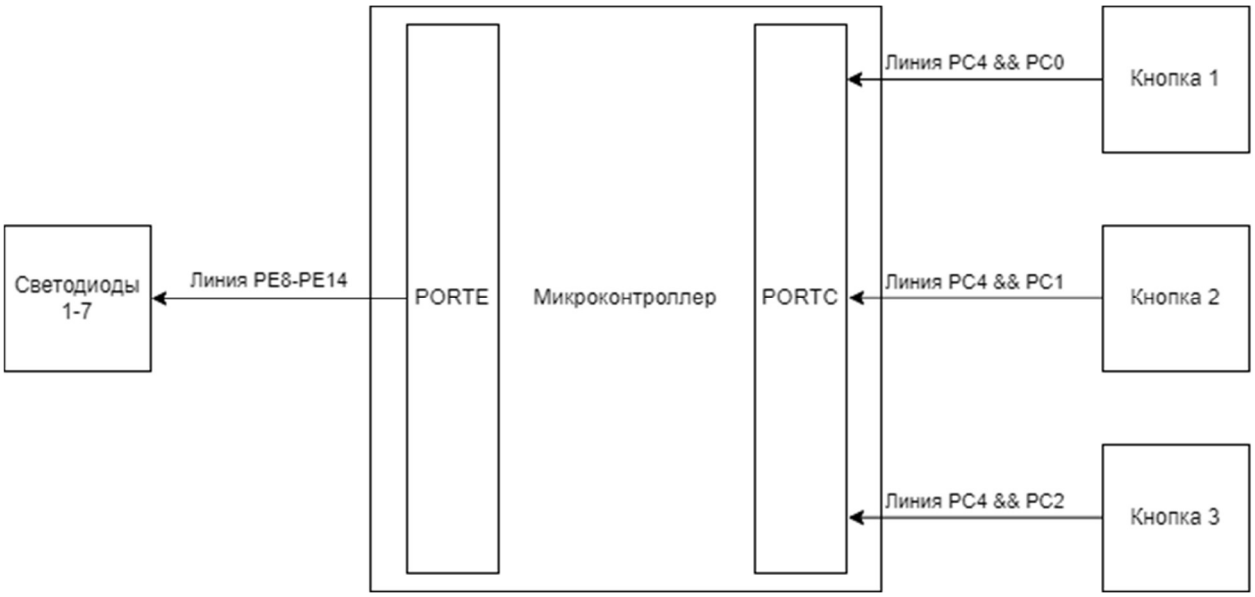

Figure 2 - Structural diagram of connecting peripheral devices to the MK

Clock setting:

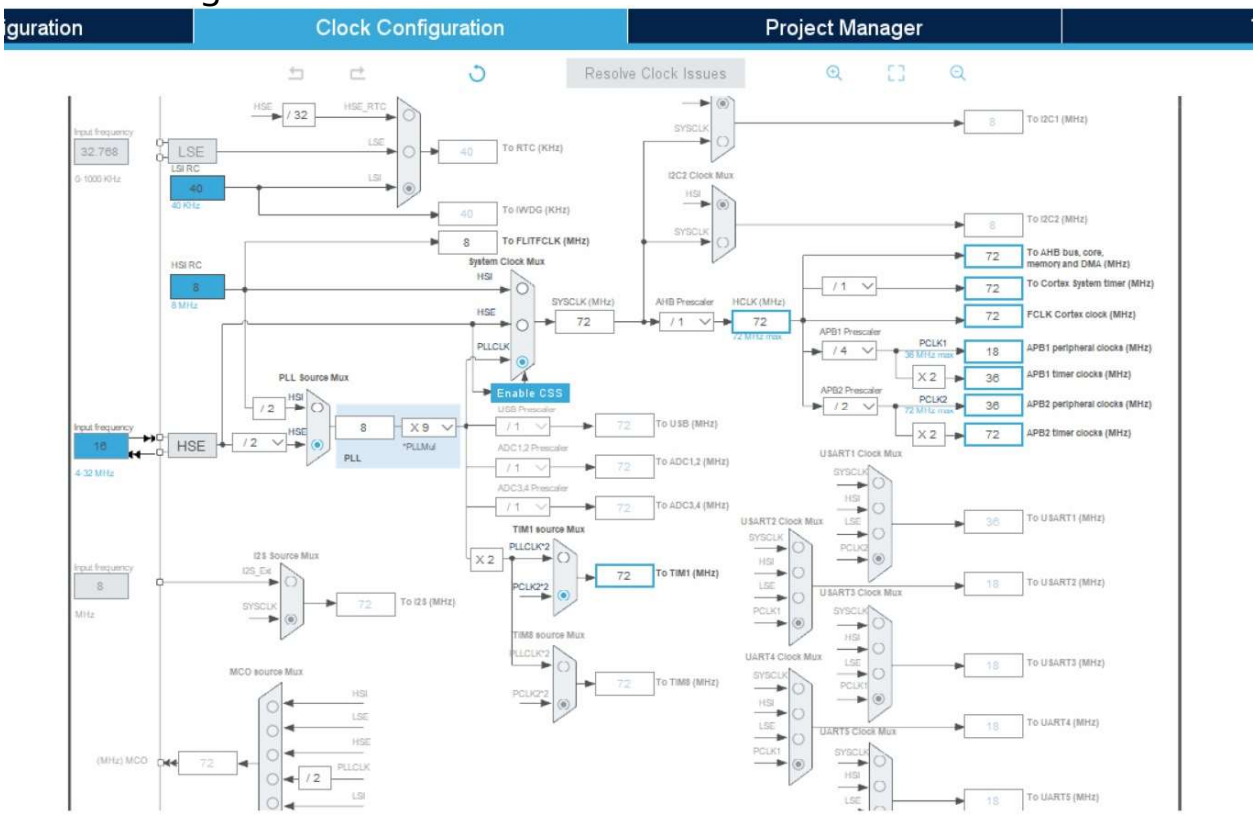

Figure 3 - Setting the clock in the STM32CubeMX program

For the operation of the buttons, the corresponding lines are used according to the diagram below:

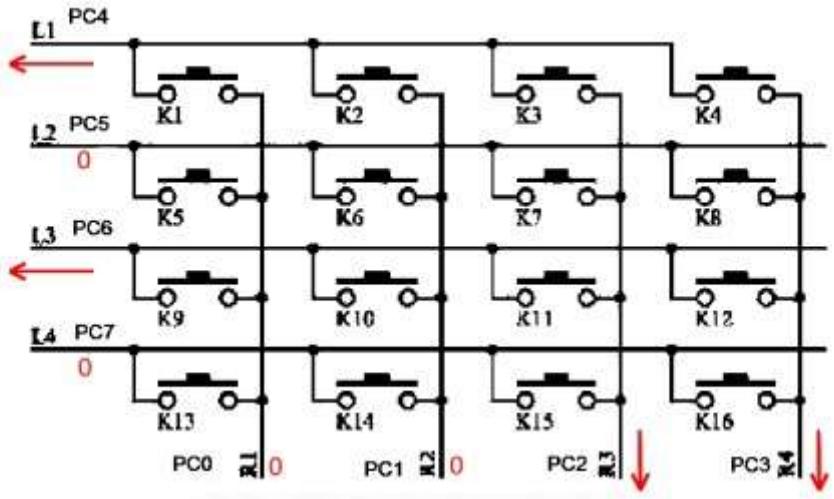

Figure 4 - 4x4 Matrix Keyboard Port Diagram

EXTI external interrupt handlers were used for buttons

Enable PWM for ch1 / 1n to ch4 using the first timer - TIM1

procedure HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);

The setting of the shim value is carried out using TIM1-CCR1 for two channels, complementary and direct, each of which has an LED.

Since the assignment of the shim value occurs for two channels at once, then there is a problem of assigning a value for a separate channel with one LED, this problem is solved by clearing the bits of the channel that will not be used, for example:

TIM1->CCER &= ~TIM_CCER_CC3NE; (OFF 3rd Complementary Channel)

You can also turn on the channel after turning it off
TIM1->CCER |= TIM_CCER_CC1NE; (inclusion of the first set of channels)

For the dual LED mode we used polarity, after changing the polarity bit, all newly assigned bits will be inverted.

For the operation of the system, we will use the interrupt model, since the state polling model will complicate the program.

Initially, after a reset, the system is in an uninitialized state. Next comes automatic initialization and the system

turns on the LEDs to the default mode and waits for an external signal (see Fig. 5)

Figure 5 - State Diagram

When an external signal (keyboard input) is received, the system enters the LED control state, in which the speed mode is changed, or the direction is reversed (clockwise / counterclockwise, or the mode is changed (1 or 2 lights). After the interrupt processing is completed, the system returns to the external LEDs.

Consider the operation of the system on a generalized interaction sequence diagram:

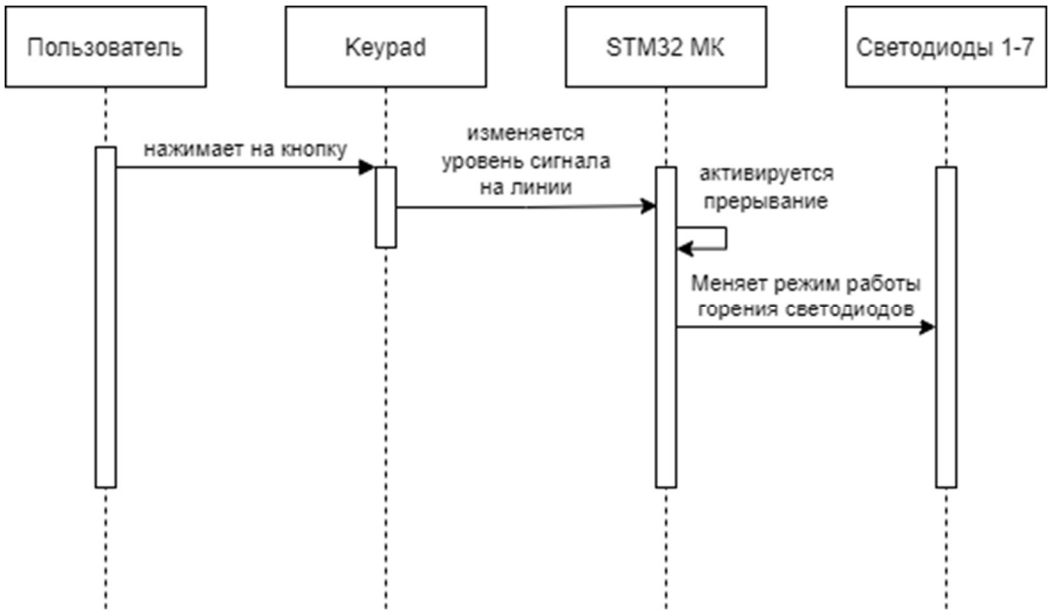

Figure 6 - Interaction Sequence Diagram

Conclusions on the diagram:

1) We will use interrupts to initiate the processing of button presses. Each button (PC0-PC2) will be connected to the corresponding line. The interrupt system of the STM32F103C8 microcontroller allows you to address up to 20 interrupts from input lines, which will allow you to create an interrupt for each button.

2) PWM requires a timer to work. To do this, we will use the system timer, the delay counter and the system timer interrupt handler.

3) For the timer to work, it is necessary to configure the SysTick_IRQn system timer interrupts and input processing interrupts.

We detail the system states in the activity diagram of the main function:

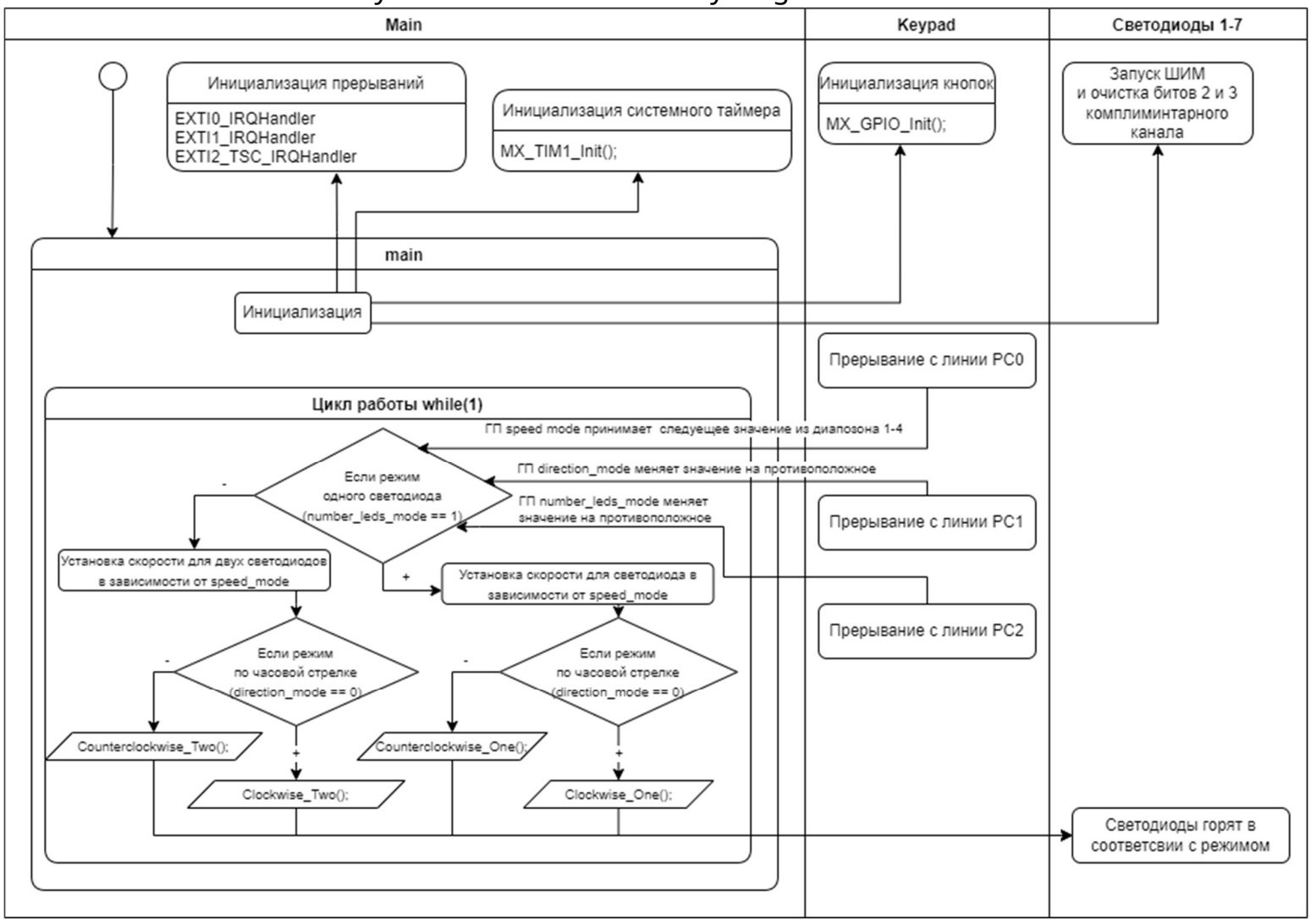

Figure 7 - Activity diagram of the main function (GP - global variable)

From here, the following main control structures should be distinguished:

1) The main() function with an infinite loop.

2) Function MX_TIM1_Init() to initialize the system timer.

3) The MX_GPIO_Init() function, which implements the initialization of the buttons (enabling the operation of the GPIO C port, setting the PC0-PC2 lines, setting EXTI).

4) Clockwise_One(), Counterclockwise_One(), Clockwise_Two(), Counterclockwise_Two() functions that implement the operation of LEDs (PE8-PE14 lines).

5) EXTI0_IRQHandler(), EXTI1_IRQHandler(), EXTI2_TSC_IRQHandler() functions that implement interrupt settings.

Examplemain functions in Appendix A

Consider the EXTI0_IRQHandler() interrupt handling algorithm in Fig. eight:

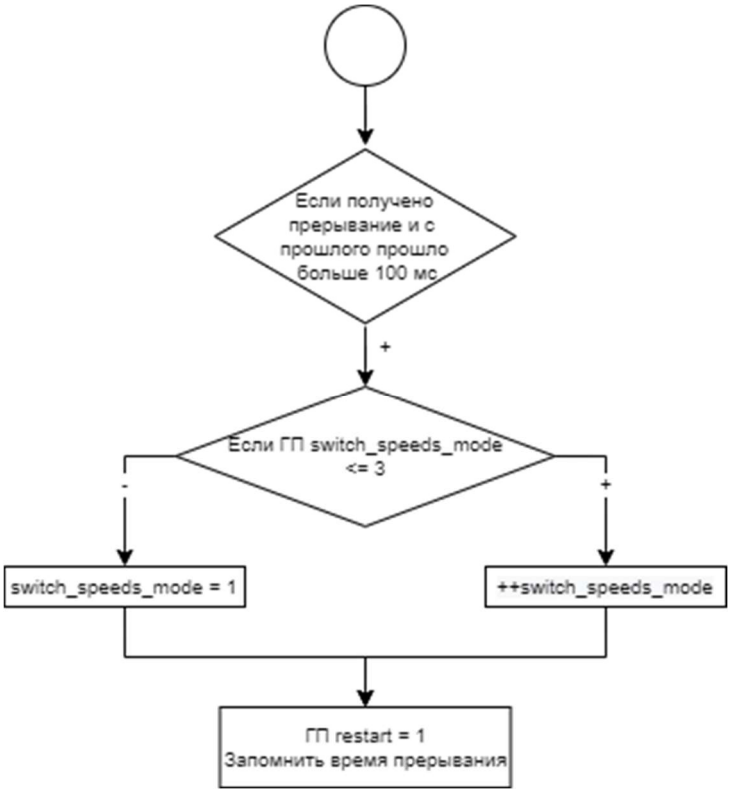

Figure 8 - Interrupt algorithm

EXTI1_IRQHandler() and EXTI2_TSC_IRQHandler() are similar, but instead of changing the global switch_speeds_mode variable, they invert the value of the direction_mode and number_leds_mode variables, respectively.
Exampleinterrupt handling functions in Appendix A

Consider the algorithm of the garland for one LED:

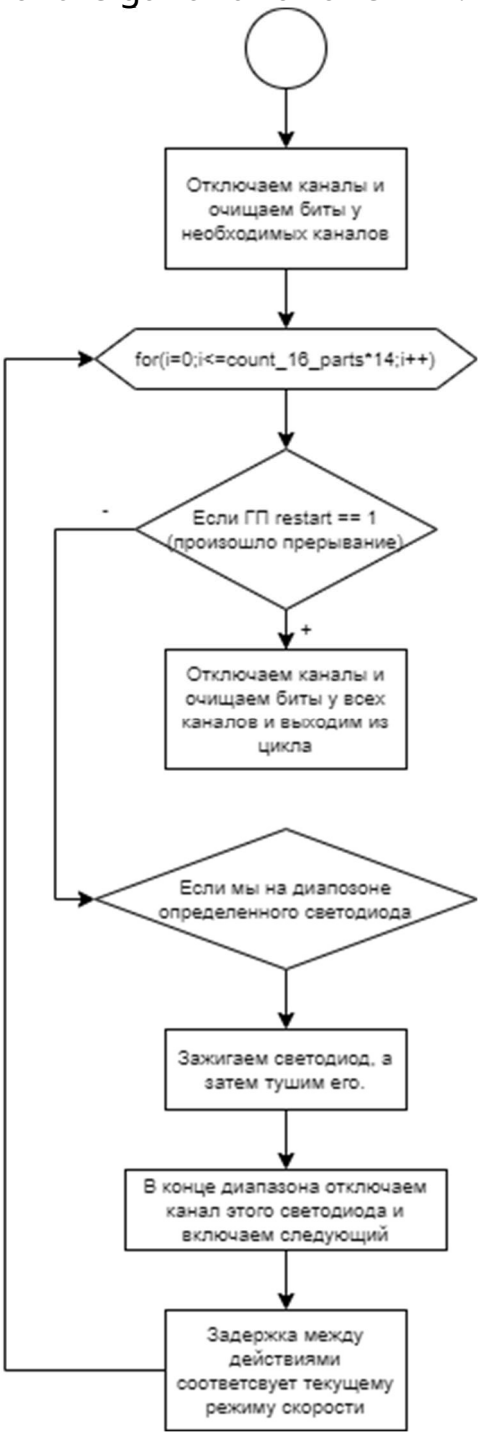

Figure 9 - Algorithm for the operation of a garland for one LED

Example[functions for single LED mode](#) in Appendix A

Consider the algorithm of the garland for two LEDs:

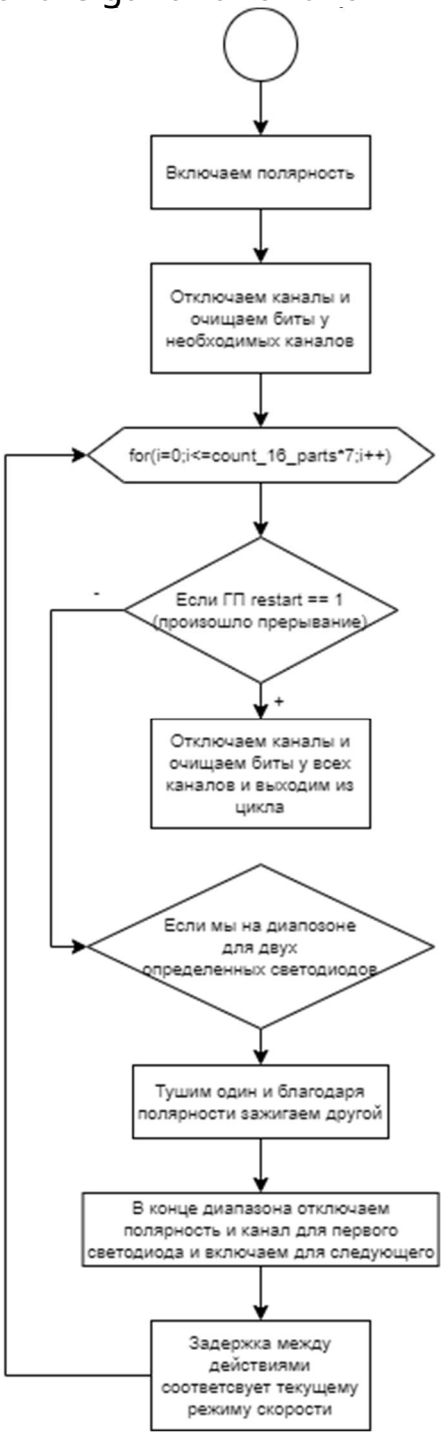

Figure 10 - The algorithm of the garland for two LEDs

Examplefunctions for dual LED mode in Appendix A

4. Testing

Tests to control the compliance of the device with the technical specifications:

All tests for clockwise movement direction

- Single diode mode for speed -1 sec. for extinguishing and ignition:

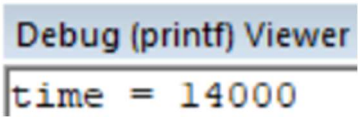

Figure 11 - Time for the whole circle = 14000 milliseconds

|14000 – 1sec.*1000 * 2 * 7diodes| = 0 millisecond error

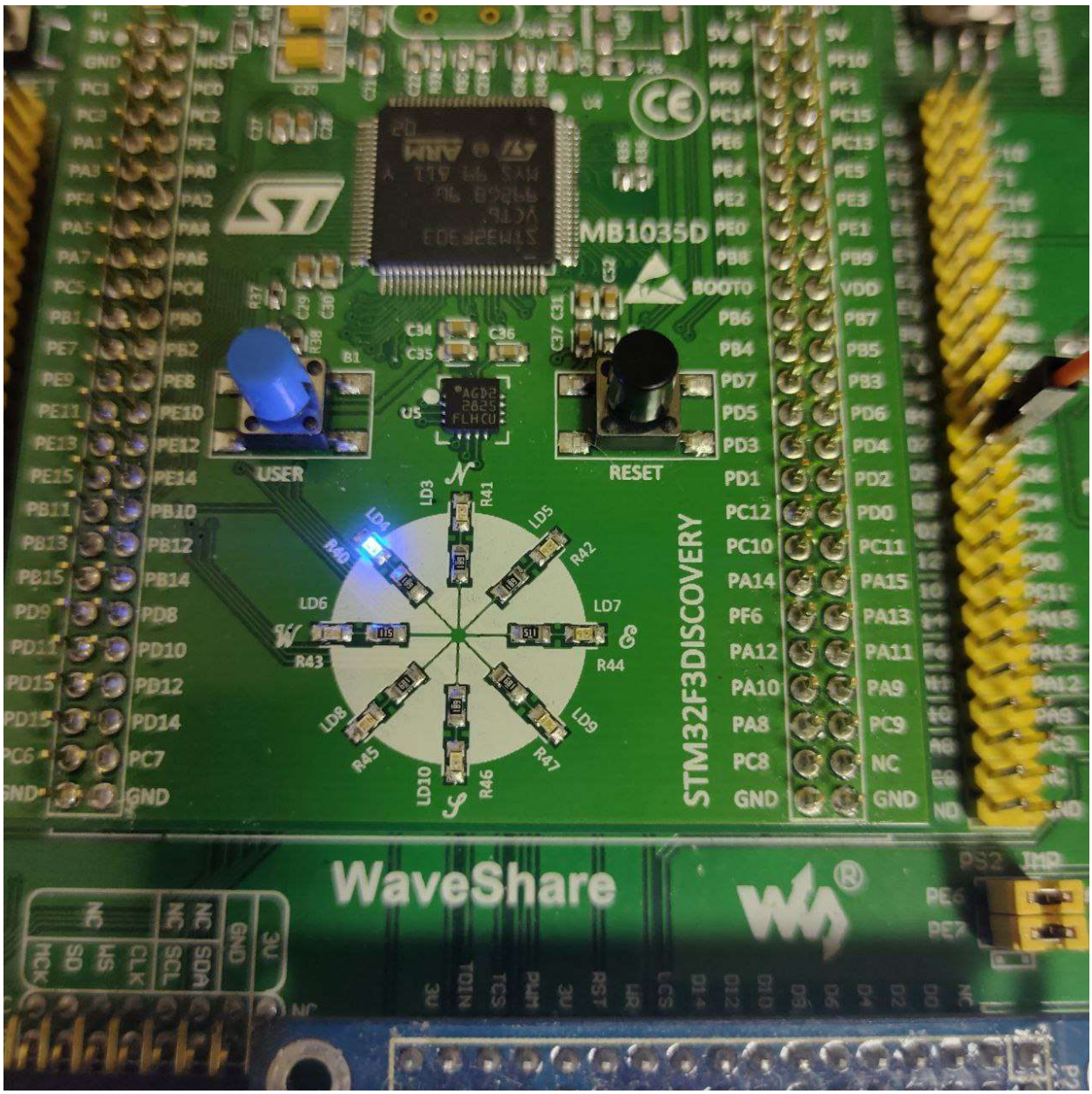

Figure 12 - Single diode mode for speed - 1 sec. for extinguishing and ignition

- Single diode mode for speed -0.5 sec for extinguishing and ignition:

Debug (printf) Viewer

time = 7006

Figure 13 - Time for the whole circle = 7006 milliseconds

|7006 – 0.5sec*1000 * 2 * 7diodes|= 6 millisecond error

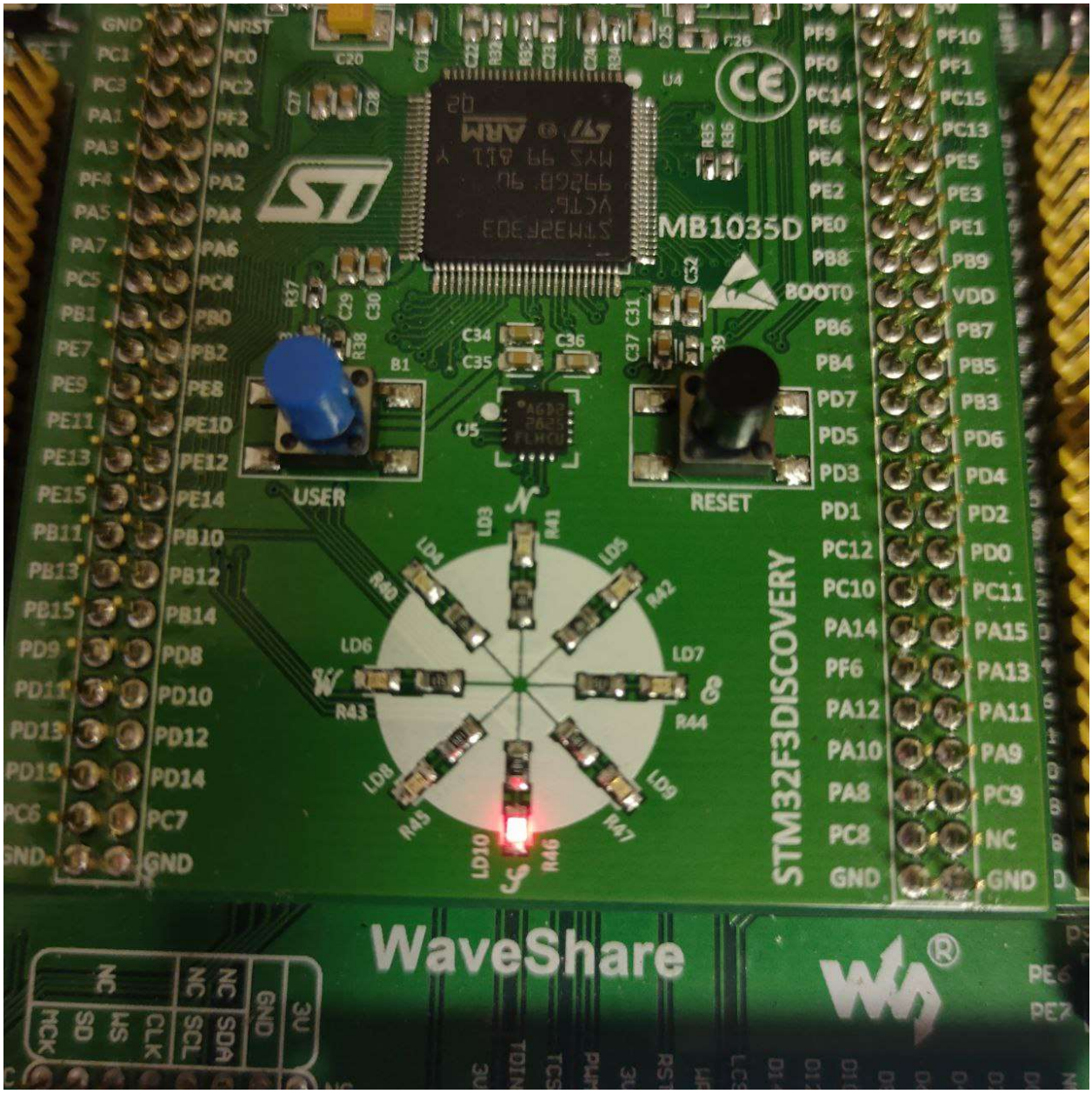

Figure 14 - Single diode mode for speed - 0.5 sec. for extinguishing and ignition

- Single diode mode for speed -0.25 s for extinguishing and ignition:

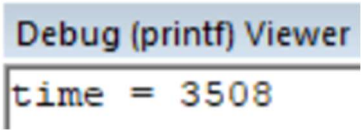

Figure 15 - Time for the whole circle = 3508 milliseconds

|3508 – 0.25s*1000 * 2 * 7diodes| = 8 millisecond error

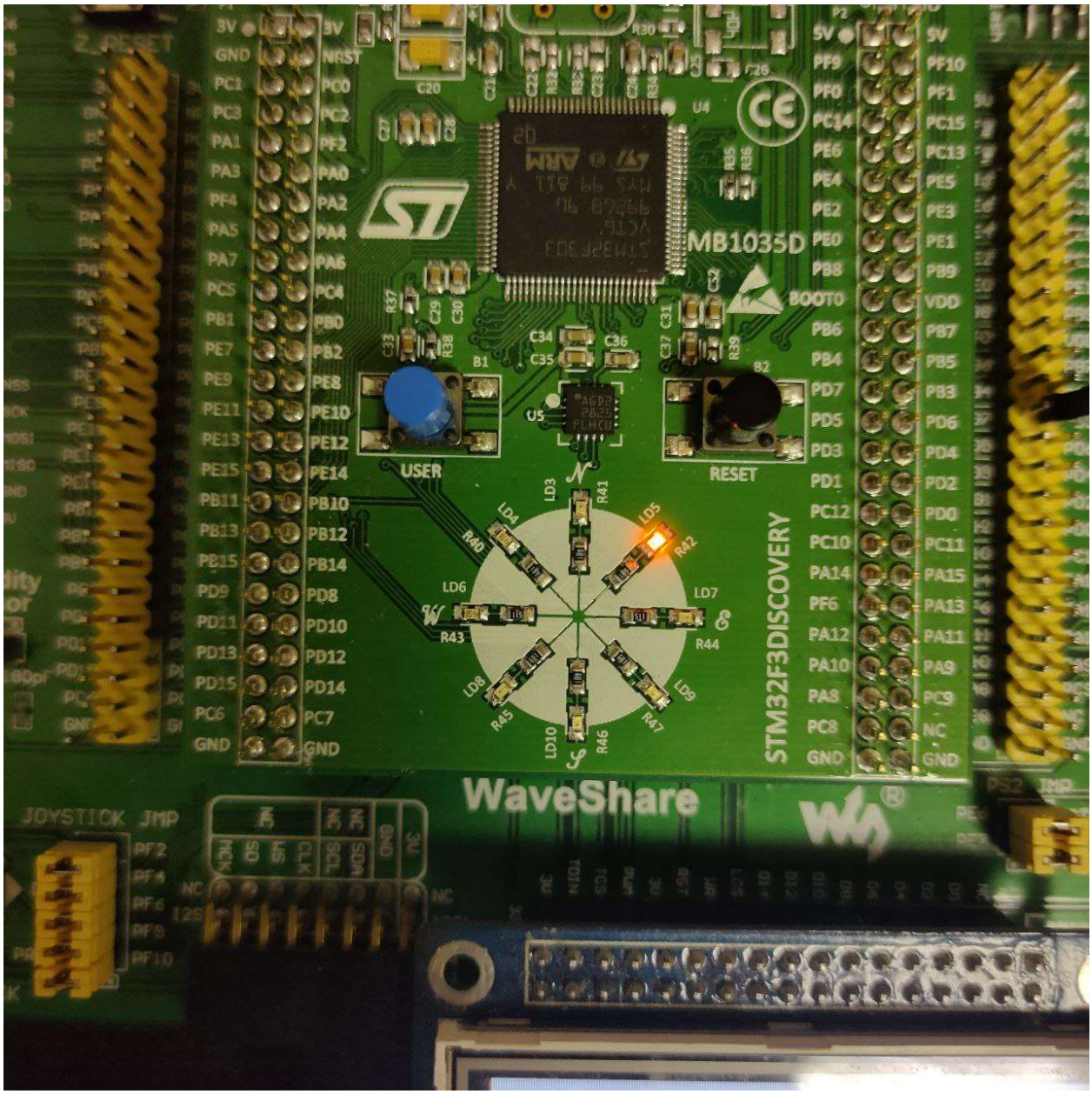

Figure 16 - Single diode mode for speed - 0.25 s. for extinguishing and ignition

- Single diode mode for speed -0.04 s for extinguishing and ignition:

Debug (printf) Viewer

time = 554

Figure 17 - Time for the whole circle = 554 milliseconds

|554 – 0.04sec.*1000 * 2 * 7diodes| = 4 millisecond error

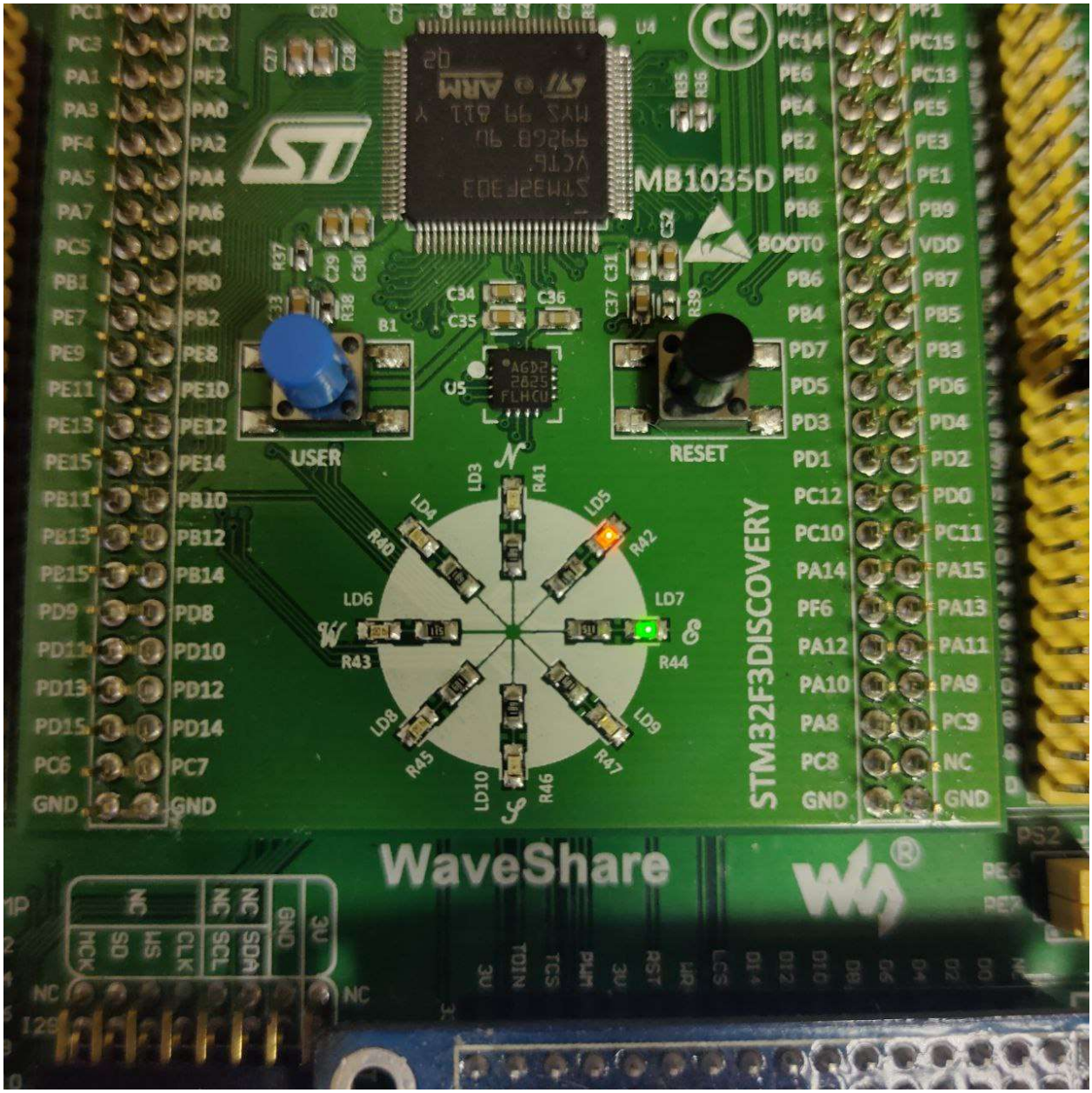

Figure 18 - Single diode mode for speed - 0.04 s. for extinguishing and ignition:

- Dual diode mode for speed -1 sec. for extinguishing and ignition:

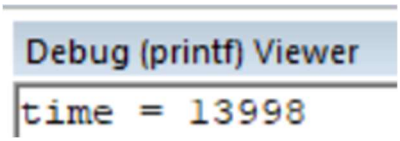

Figure 19 - Time for the whole circle = 13998 milliseconds

|13998 – 1sec*1000 * 2 * 7diodes| = 2 millisecond error

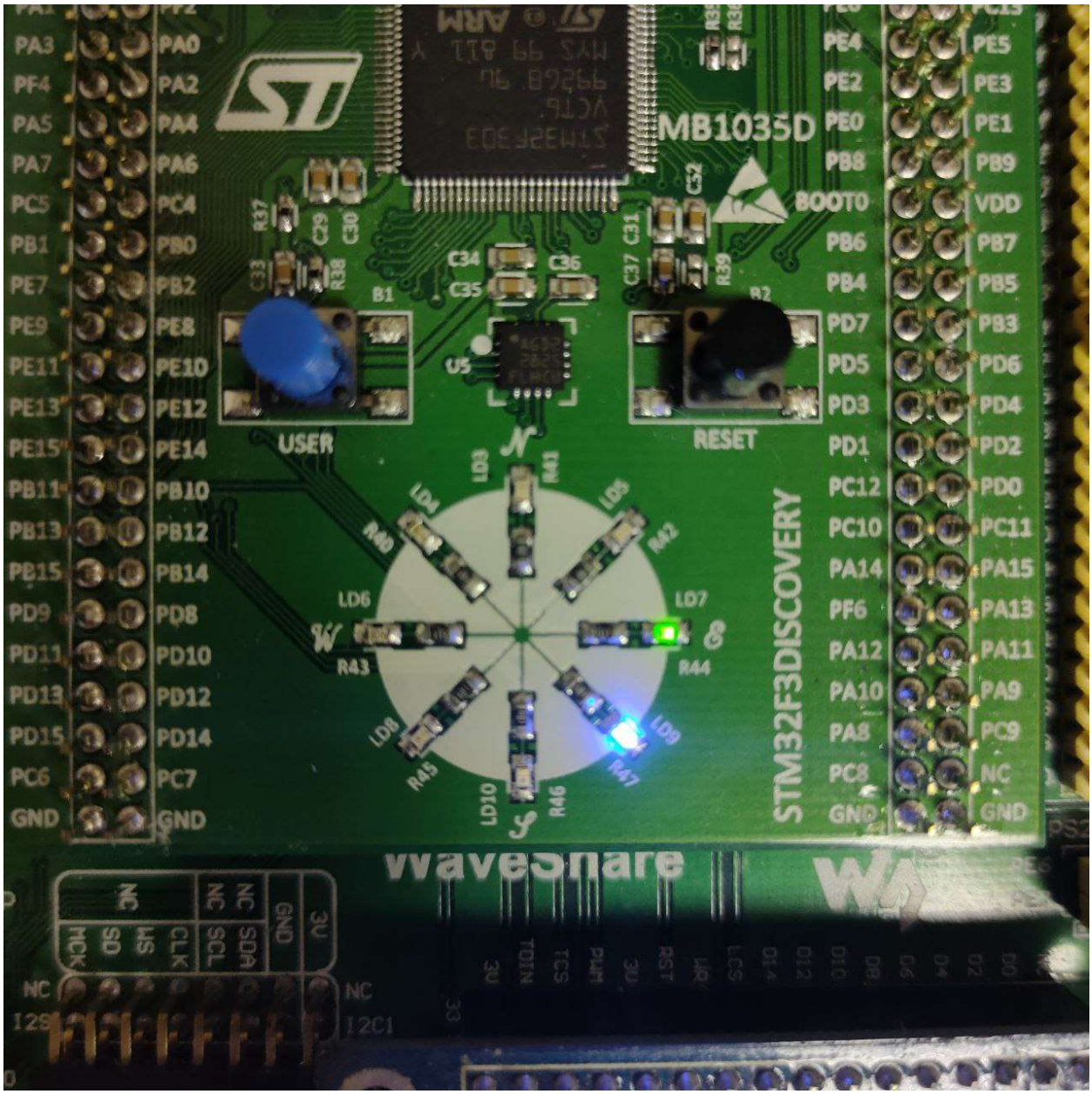

Figure 20 - Dual diode mode for speed - 1 sec. for extinguishing and ignition

- Dual diode mode for speed -0.5 sec for extinguishing and ignition:

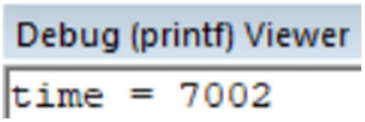

Figure 21 - Full lap time = 7002 milliseconds

|7002 – 0.5s*1000 * 2 * 7diodes| = 2 millisecond error

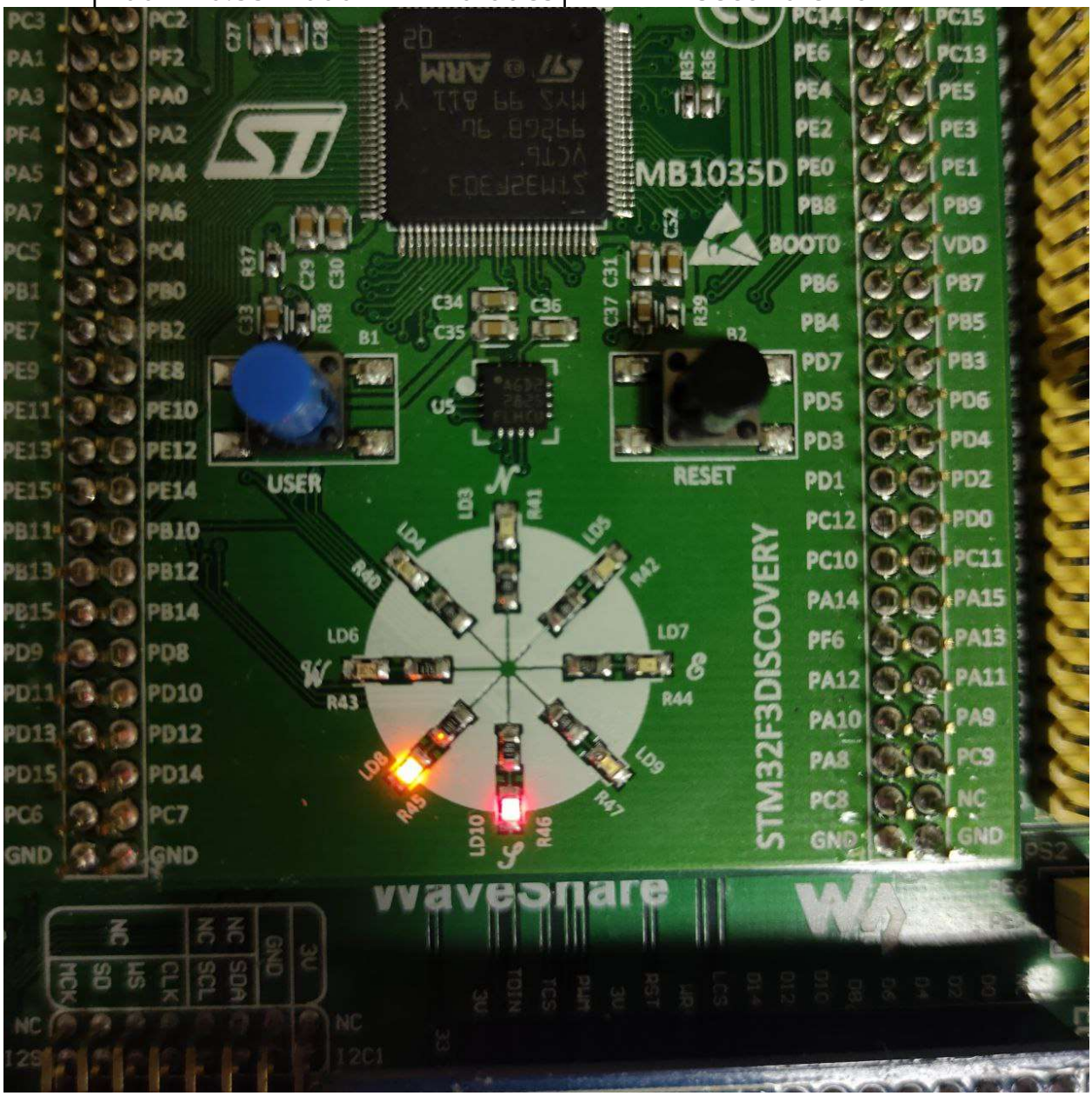

Figure 22 - Dual diode mode for speed - 0.5 sec. for extinguishing and ignition

- Dual diode mode for speed -0.25 s for extinguishing and ignition:

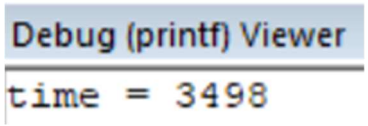

Figure 23 - Time for the whole circle = 3498 milliseconds

|3498 – 0.25s*1000 * 2 * 7diodes| = 2 millisecond error

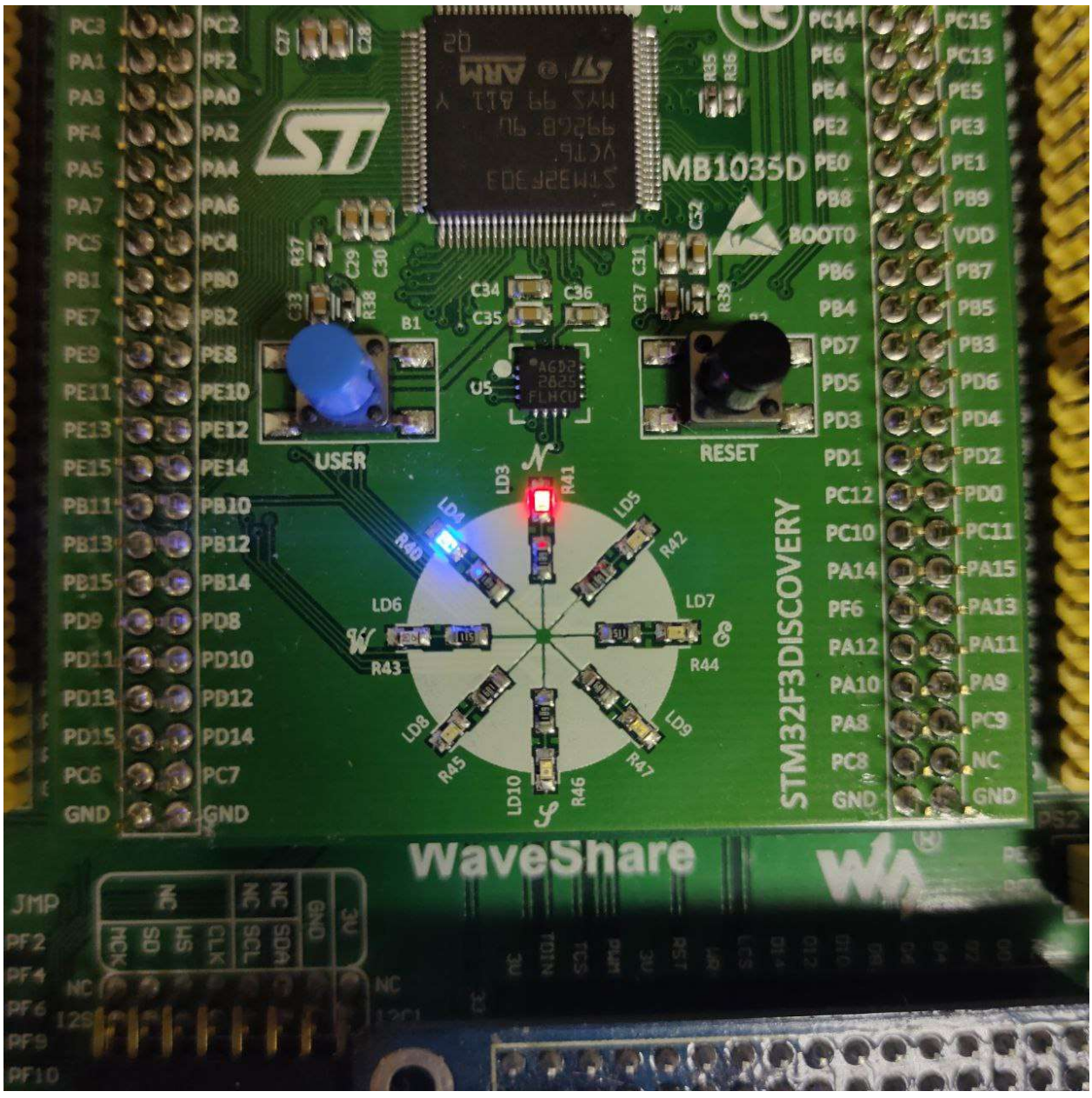

Figure 24 - Dual diode mode for speed - 0.25 sec. for extinguishing and ignition

- Dual diode mode for speed -0.04 s for extinguishing and ignition:

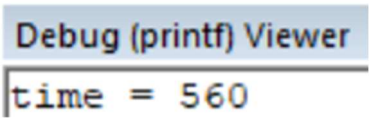

Figure 25 - Time for the whole circle = 560 milliseconds

|560 – 0.04sec.*1000 * 2 * 7diodes| = 0 millisecond error

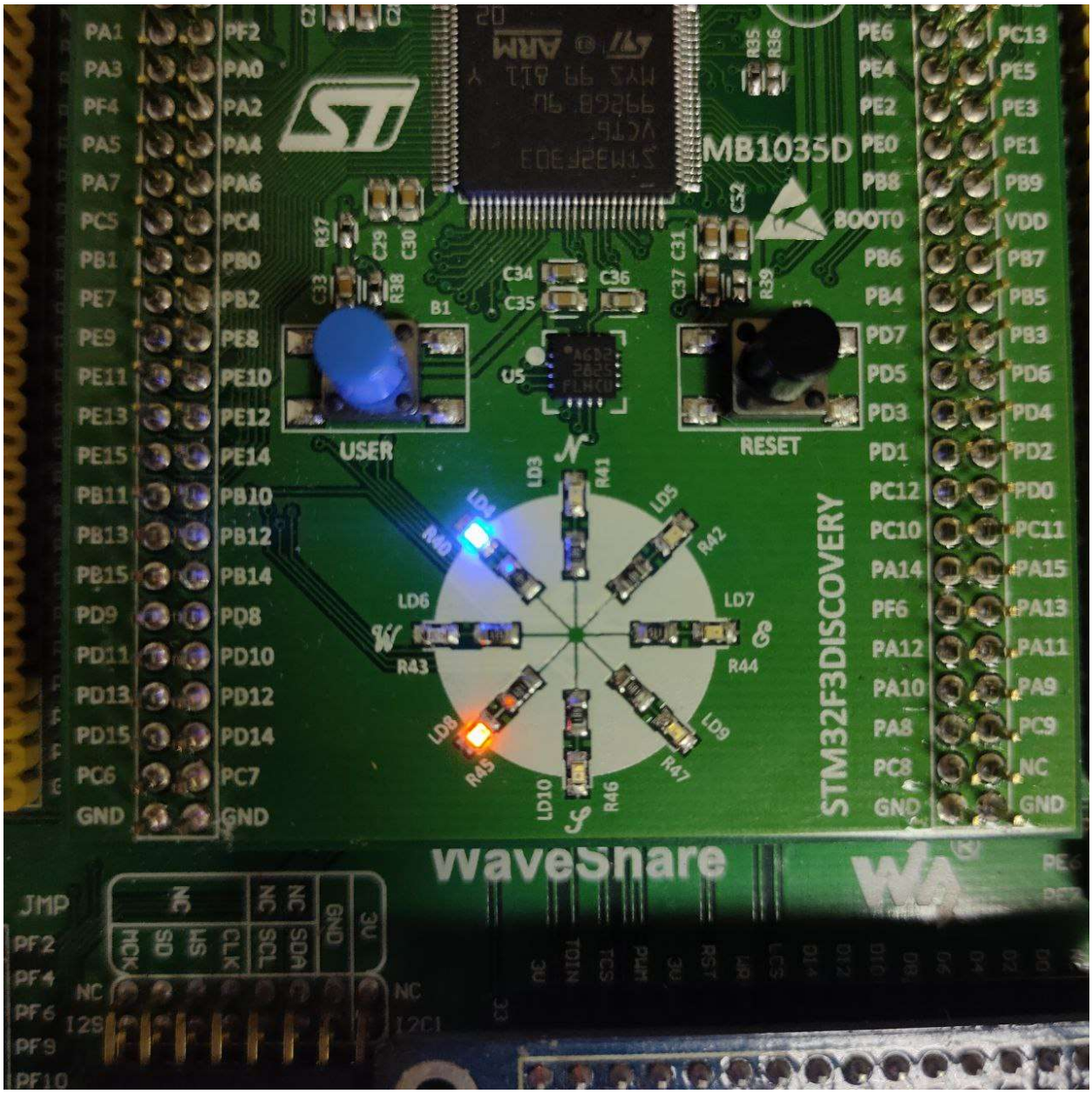

Figure 26 - Dual diode mode for speed - 0.04 sec. for extinguishing and ignition

Similarly for the counter-clockwise direction, the results are exactly the same.

5. Economic evaluation

Component	Price for aliexpress	Aliexpress Link
microcontroller	116 rub.	STM32F103C8T6
4-key keyboard	44 rub.	Keyboard Matrix for Arduino UNO MEGA2560 DUE button
8 LEDs	5.44 RUB	100pcs 5mm LED diodes F5, assorted set
Outcome:	165 rub	Excluding assembly, wires and case.

In mass production, the price of components will be lower due to bulk purchases, which is one way to reduce the price.

6. Conclusion

As a result of the implementation of this project, the design of a device based on a microcontroller was carried out. Skills were obtained in the field of architecture and software of embedded systems, as well as knowledge about the structure, functions and basics of programming microcontrollers, which allow solving analysis issues
functioning of embedded systems software.

7. List of used literature

1. Microcontrollers. Development of Embedded Applications: Tutorial / A.E. Vasiliev; St. Petersburg State Polytechnic University - St. Petersburg. : Publishing House of St. Petersburg State Polytechnical University, 2003. - 211 p.

2. The Definitive Guide to ARM Cortex-M3 and Cortex-M4 Processors. third edition. Joseph Yiu. ARM Ltd., Cambridge, UK. [electronic resource] // URL: https://www.pdfdrive.com/the-definitive-guide-to-arm-cortex-m3-and-cortex-m4-processors-e187111520.html

3. Joseph Yu. ARM's Cortex-M3 core. Complete guide. 2012. ISBN: 978-5-94120-243-0. [electronic resource] // URL: https://bok.xyz/book/2373589/b5c3ad

4.RM0008. Reference manual STM32F101xx, STM32F102xx, STM32F103xx, STM32F105xx and STM32F107xx advanced Arm®-based 32-bit MCUs [electronic resource] // URL: https://www.st.com/resource/en/reference_manual/cd00171190-stm32f101xxstm32f102xx-stm32f103xx-stm32f105xx-and-stm32f107xx-advanced-armbased-32bit-mcus-stmicroelectronics.pdf

5. Datasheet STM32F103x8 STM32F103xB Medium-density performance line ARM®- based 32-bit MCU with 64 or 128 KB Flash, USB, CAN, 7 timers, 2 ADCs, 9 com. interfaces. [electronic resource] // URL: https://www.st.com/resource/en/datasheet/stm32f103c8.pdf

6. Martin M. STM32 Insider Guide [electronic resource] // URL:https://istarik.ru/file/STM32.pdf

7. Recommendation ITU-R M.1677-1 International Morse code. International Telecommunication Union 2009 [electronic resource] //URL: https://www.itu.int/dms_pubrec/itu-r/rec/m/R-REC-M.1677-1-200910-I!!PDF-R.pdf

Appendix A. Program Code

For convenience, the project implementation is in an open repository on GitHub:

https://github.com/Krutov777/stm32_garland

main function

```
int main(void)
{
  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* Configure the system clock */
  SystemClock_Config();

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_TIM1_Init();

  /* USER CODE BEGIN 2 */
  HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1); // start shim on channel 1(ch1)
  HAL_TIMEx_PWMN_Start (&htim1, TIM_CHANNEL_1);// start shim on complimentary channel 1(ch1n)
  HAL_TIM_PWM_Start (&htim1, TIM_CHANNEL_2);
  HAL_TIMEx_PWMN_Start(&htim1, TIM_CHANNEL_2);
  HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_3);
  HAL_TIMEx_PWMN_Start(&htim1, TIM_CHANNEL_3);
  HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_4);
  GPIO_InitTypeDef GPIO_InitStruct = {0};
  GPIO_InitStruct.Pin = GPIO_PIN_15;
  GPIO_InitStruct.Mode = GPIO_MODE_AF_OD;
  GPIO_InitStruct.Pull = GPIO_PULLDOWN;
  GPIO_InitStruct.Speed   = GPIO_SPEED_FREQ_LOW;
  GPIO_InitStruct.Alternate = GPIO_AF2_TIM1;
  HAL_GPIO_Init(GPIOE, &GPIO_InitStruct);
  __HAL_TIM_ENABLE_IT(&htim1, TIM_IT_BREAK);

  TIM1->CCER &= ~TIM_CCER_CC2NE; // clear bits in ch2n and ch3n that is disable this channels
  TIM1->CCER &= ~TIM_CCER_CC3NE;
          /* USER CODE END 2 */

          /* infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE BEGIN 3 */
                    if(number_leds_mode == 1) {
                            switch_speeds_one_led(switch_speeds_mode);
                            if(direction_mode == 0) {
                                    Clockwise_One();
                                    printf("time = %i\n", HAL_GetTick()*2);
                            }
                            else {
                                    Counterclockwise_One(); printf("time =
                                    %i\n", HAL_GetTick()*2);
                            }
                    }
```

```
                else if(number_leds_mode == 2) {
                        switch_speeds_two_led(switch_speeds_mode);
                        if(direction_mode == 0) {
                                Clockwise_Two();
                                printf("time = %i\n", HAL_GetTick()*2);
                        }
                        else {
                                Counterclockwise_Two(); printf("time =
                                %i\n", HAL_GetTick()*2);
                        }
                }
 }
 /* USER CODE END 3 */
        /* USER CODE END WHILE */
}
```

Interrupt Handling Code

```
// handler for k0 button with external interrupt - speed
mode void EXTI0_IRQHandler(void)
{
        // chatter protection with 100ms time
        if(!flag_key1_press && (HAL_GetTick() - time_key1_press) > 100) {

                flag_key1_press = 1;
        }
        if(HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_0) == GPIO_PIN_RESET && flag_key1_press) {

                flag_key1_press = 0;
                // click action
                if(switch_speeds_mode <= 3)
                        ++ switch_speeds_mode;
                else
                        switch_speeds_mode = 1;
                printf("speed = %i\n", switch_speeds_mode);
                restart=1;
                time_key1_press = HAL_GetTick();
        }
 HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_0);
}

/**
 * @brief This function handles EXTI line1 interrupt.
 */
// handler for k1 button with external interrupt - direction mode
void EXTI1_IRQHandler(void)
{
        // chatter protection with 100ms time
        if(!flag_key1_press && (HAL_GetTick() - time_key1_press) > 100) {

                flag_key1_press = 1;
        }
        if(HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_1) == GPIO_PIN_RESET && flag_key1_press) {

                flag_key1_press = 0;
                if(direction_mode == 1)
                        direction_mode = 0;
                else direction_mode = 1; printf("direction =
                %i\n", direction_mode); restart=1;

                time_key1_press = HAL_GetTick();
        }
```

```
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_1);
}

/**
  * @brief This function handles EXTI line2 and Touch Sense controller.
  */
// handler for k2 button with external interrupt - number leds
mode void EXTI2_TSC_IRQHandler(void)
{
  // chatter protection with 100ms time
        if(!flag_key1_press && (HAL_GetTick() - time_key1_press) > 100) {

                flag_key1_press = 1;
        }
        if(HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_2) == GPIO_PIN_RESET && flag_key1_press) {

                flag_key1_press = 0;
                if(number_leds_mode == 1)
                        ++ number_leds_mode;
                else number_leds_mode = 1;
                printf("number of leds = %i\n", number_leds_mode);
                restart=1; // flag for reset mode
                time_key1_press = HAL_GetTick();
        }
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_2);
}
```

Functions for Single LED Mode

```
void Clockwise_One(void) {

        uint32_t count_16_parts = count/16;
        TIM1->CCER &= ~TIM_CCER_CC4E;// clear bits in ch4 that is disable this channel
        TIM1->CCR4 = 0;                 // set zero value for ch4, so that the LED does not light up

        TIM1->CCER |= TIM_CCER_CC1NE;// set bits for ch1n that is enable this channel
        TIM1->CCER &= ~TIM_CCER_CC1E;// disable ch1
        for(i=0;i<=count_16_parts*14;i++) {

                if (restart) // if flag restart = 1 then clear all channels {

                        TIM1->CCER &= ~(TIM_CCER_CC1E | TIM_CCER_CC1NE |
                        TIM_CCER_CC2E | TIM_CCER_CC2NE
                        | TIM_CCER_CC3E | TIM_CCER_CC3NE |
                        TIM_CCER_CC4E);
                        TIM1->CCR1 = 0;
                        TIM1->CCR2 = 0;
                        TIM1->CCR3 = 0;
                        TIM1->CCR4 = 0;
                        restart=0;
                        break;
                }
                if(i<count_16_parts)
                {
                        TIM1->CCR1 = 65535 * i / (count_16_parts);
                }
                else if((i>count_16_parts - 1)&&(i<count_16_parts*2)) {

                        TIM1->CCR1 = 65535 * (count_16_parts*2-i) / count_16_parts;
                }
                else if (i == count_16_parts*2) {

                        TIM1->CCER |= TIM_CCER_CC1E;
```

```
            TIM1->CCER &= ~TIM_CCER_CC1NE;
}
else if((i>count_16_parts*2 - 1)&&(i<count_16_parts*3)) {

            TIM1->CCR1 = 65535 * (i-count_16_parts*2) / count_16_parts;
}
else if((i>count_16_parts*3 - 1)&&(i<count_16_parts*4)) {

            TIM1->CCR1 = 65535 * (count_16_parts*4-i) / count_16_parts;
}
else if (i == count_16_parts*4) {
            TIM1->CCER &= ~TIM_CCER_CC1E;

            TIM1->CCER |= TIM_CCER_CC2NE;
            TIM1->CCER &= ~TIM_CCER_CC2E;
}
else if((i>count_16_parts*4 - 1)&&(i<count_16_parts*5)) {

            TIM1->CCR2 = 65535 * (i-count_16_parts*4) / count_16_parts;
}
else if((i>count_16_parts*5 - 1)&&(i<count_16_parts*6)) {

            TIM1->CCR2 = 65535 * (count_16_parts*6-i) / count_16_parts;
}
else if (i == count_16_parts*6) {

            TIM1->CCER |= TIM_CCER_CC2E; TIM1-
            >CCER &= ~TIM_CCER_CC2NE;
}
else if((i>count_16_parts*6 - 1)&&(i<count_16_parts*7)) {

            TIM1->CCR2 = 65535 * (i-count_16_parts*6) / count_16_parts;
}
else if((i>count_16_parts*7 - 1)&&(i<count_16_parts*8)) {

            TIM1->CCR2 = 65535 * (count_16_parts*8-i) / count_16_parts;
}
else if (i == count_16_parts*8) {

            TIM1->CCER &= ~TIM_CCER_CC2E;

            TIM1->CCER |= TIM_CCER_CC3NE;
            TIM1->CCER &= ~TIM_CCER_CC3E;
}
else if((i>count_16_parts*8 - 1)&&(i<count_16_parts*9)) {

            TIM1->CCR3 = 65535 * (i-count_16_parts*8) / count_16_parts;
}
else if((i>count_16_parts*9 - 1)&&(i<count_16_parts*10)) {

            TIM1->CCR3 = 65535 * (count_16_parts*10-i) / count_16_parts;
}
else if (i == count_16_parts*10) {

            TIM1->CCER |= TIM_CCER_CC3E; TIM1-
            >CCER &= ~TIM_CCER_CC3NE;
}
else if((i>count_16_parts*10 - 1)&&(i<count_16_parts*11)) {

            TIM1->CCR3 = 65535 * (i-count_16_parts*10) / count_16_parts;
}
else if((i>count_16_parts*11 - 1)&&(i<count_16_parts*12)) {
```

```
                TIM1->CCR3 = 65535 * (count_16_parts*12-i) / count_16_parts;
            }
            else if (i == count_16_parts*12) {

                TIM1->CCER &= ~TIM_CCER_CC3E;

                TIM1->CCER |= TIM_CCER_CC4E;
            }
            else if((i>count_16_parts*12 - 1)&&(i<count_16_parts*13)) {

                TIM1->CCR4 = 65535 * (i-count_16_parts*12) / count_16_parts;
            }
            else if((i>count_16_parts*13 - 1)&&(i<count_16_parts*14)) {

                TIM1->CCR4 = 65535 * (count_16_parts*14-i) / count_16_parts;
            }
            //delay
            for(d=0;d<delay;d++)
            {
            }
        }
}

void Counterclockwise_One(void) {

        uint32_t count_16_parts = count/16;
        TIM1->CCER &= ~TIM_CCER_CC1NE; // disable ch1n
        TIM1->CCR1 = 0;
        TIM1->CCER |= TIM_CCER_CC4E; // set bits for ch4nthat is enable this channel
        for(i=0;i<=count_16_parts*14;i++)
        {
            if (restart) { // if flag restart = 1 then clear all channels
                TIM1->CCER &= ~(TIM_CCER_CC1E | TIM_CCER_CC1NE |
                TIM_CCER_CC2E | TIM_CCER_CC2NE
                | TIM_CCER_CC3E | TIM_CCER_CC3NE |
                TIM_CCER_CC4E);
                TIM1->CCR1 = 0;
                TIM1->CCR2 = 0;
                TIM1->CCR3 = 0;
                TIM1->CCR4 = 0;
                restart=0;
                break;
            }
            if(i<count_16_parts)
            {
                TIM1->CCR4 = 65535 * i / (count_16_parts);
            }
            else if((i>count_16_parts - 1)&&(i<count_16_parts*2)) {

                TIM1->CCR4 = 65535 * (count_16_parts*2-i) / count_16_parts;
            }
            else if (i == count_16_parts * 2) {

                TIM1->CCER &= ~TIM_CCER_CC4E;

                TIM1->CCER |= TIM_CCER_CC3E; TIM1-
                >CCER &= ~TIM_CCER_CC3NE;
            }
            else if((i>count_16_parts*2 - 1)&&(i<count_16_parts*3)) {

                TIM1->CCR3 = 65535 * (i-count_16_parts*2) / count_16_parts;
            }
            else if((i>count_16_parts*3 - 1)&&(i<count_16_parts*4))
```

```
{
        TIM1->CCR3 = 65535 * (count_16_parts*4-i) / count_16_parts;
}
else if (i == count_16_parts * 4) {

        TIM1->CCER |= TIM_CCER_CC3NE;
        TIM1->CCER &= ~TIM_CCER_CC3E;
}
else if((i>count_16_parts*4 - 1)&&(i<count_16_parts*5)) {

        TIM1->CCR3 = 65535 * (i-count_16_parts*4) / count_16_parts;
}
else if((i>count_16_parts*5 - 1)&&(i<count_16_parts*6)) {

        TIM1->CCR3 = 65535 * (count_16_parts*6-i) / count_16_parts;
}
else if (i == count_16_parts * 6) {

        TIM1->CCER &= ~TIM_CCER_CC3NE;

        TIM1->CCER |= TIM_CCER_CC2E; TIM1-
        >CCER &= ~TIM_CCER_CC2NE;
}
else if((i>count_16_parts*6 - 1)&&(i<count_16_parts*7)) {

        TIM1->CCR2 = 65535 * (i-count_16_parts*6) / count_16_parts;
}
else if((i>count_16_parts*7 - 1)&&(i<count_16_parts*8)) {

        TIM1->CCR2 = 65535 * (count_16_parts*8-i) / count_16_parts;
}
else if (i == count_16_parts * 8) {

        TIM1->CCER |= TIM_CCER_CC2NE;
        TIM1->CCER &= ~TIM_CCER_CC2E;
}
else if((i>count_16_parts*8 - 1)&&(i<count_16_parts*9)) {

        TIM1->CCR2 = 65535 * (i-count_16_parts*8) / count_16_parts;
}
else if((i>count_16_parts*9 - 1)&&(i<count_16_parts*10)) {

        TIM1->CCR2 = 65535 * (count_16_parts*10-i) / count_16_parts;
}
else if (i == count_16_parts * 10) {

        TIM1->CCER &= ~TIM_CCER_CC2NE;

        TIM1->CCER |= TIM_CCER_CC1E; TIM1-
        >CCER &= ~TIM_CCER_CC1NE;
}
else if((i>count_16_parts*10 - 1)&&(i<count_16_parts*11)) {

        TIM1->CCR1 = 65535 * (i-count_16_parts*10) / count_16_parts;
}
else if((i>count_16_parts*11 - 1)&&(i<count_16_parts*12)) {

        TIM1->CCR1 = 65535 * (count_16_parts*12-i) / count_16_parts;
}
else if (i == count_16_parts * 12) {

        TIM1->CCER |= TIM_CCER_CC1NE;
        TIM1->CCER &= ~TIM_CCER_CC1E;
```

```
                }
                else if((i>count_16_parts*12 - 1)&&(i<count_16_parts*13)) {

                        TIM1->CCR1 = 65535 * (i-count_16_parts*12) / count_16_parts;
                }
                else if((i>count_16_parts*13 - 1)&&(i<count_16_parts*14)) {

                        TIM1->CCR1 = 65535 * (count_16_parts*14-i) / count_16_parts;
                }
                //delay
                for(d=0;d<delay;d++)
                {
                }
        }
}
```

Functions for Dual LED Mode

```
void Clockwise_Two()
{
        uint32_t count_16_parts = count/16; TIM1-
   >CCER &= ~TIM_CCER_CC1NP; // polarity on
        TIM1->CCER |= TIM_CCER_CC1NE | TIM_CCER_CC1E; // channel on
        for(i=0;i<=count_16_parts*7;i++)
        {
                if (restart){
                        TIM1->CCER &= ~(TIM_CCER_CC1E | TIM_CCER_CC1NE |
                        TIM_CCER_CC2E | TIM_CCER_CC2NE
                        | TIM_CCER_CC3E | TIM_CCER_CC3NE |
                        TIM_CCER_CC4E);
                        TIM1->CCER &= ~(TIM_CCER_CC1P | TIM_CCER_CC1NP |
                        TIM_CCER_CC2P | TIM_CCER_CC2NP
                        | TIM_CCER_CC3P | TIM_CCER_CC3NP |
                        TIM_CCER_CC4P);
                        TIM1->CCR1 = 0;
                        TIM1->CCR2 = 0;
                        TIM1->CCR3 = 0;
                        TIM1->CCR4 = 0;
                        restart=0;
                        break;
                }
                else if(i<count_16_parts) {

                        TIM1->CCR1 = 65535 * i / (count_16_parts);
                }
                else if(i == count_16_parts) {

                        TIM1->CCER &= ~TIM_CCER_CC1NP; // polarity on TIM1-
                        >CCER |= TIM_CCER_CC1E; // channel on TIM1->CCER &=
                        ~TIM_CCER_CC1NE; // comp. channel off

                        TIM1->CCER |= TIM_CCER_CC2NP; // polarity off ?? TIM1-
                        >CCER |= TIM_CCER_CC2NE; // comp. channel on TIM1-
                        >CCER &= ~TIM_CCER_CC2E; // channel off
                }
                else if((i>count_16_parts {   - 1)&&(i<count_16_parts*2))

                        TIM1->CCR1 = 65535 * (count_16_parts*2-i)/(count_16_parts);
                        TIM1->CCR2 = 65535 * (count_16_parts*2-i)/(count_16_parts);
                }
                else if(i == count_16_parts*2) {

                        TIM1->CCR1 = 0;
```

```
                TIM1->CCER &= ~TIM_CCER_CC2NP; // polarity on
                TIM1->CCER |= TIM_CCER_CC2E; // channel on
        }
        else if((i>count_16_parts*2 - 1)&&(i<count_16_parts*3)) {

                TIM1->CCR2 = 65535 * (i-count_16_parts*2)/(count_16_parts);
        }
        else if(i == count_16_parts*3) {

                TIM1->CCER &= ~TIM_CCER_CC2NP;
                TIM1->CCER |= TIM_CCER_CC2E; TIM1-
                >CCER &= ~TIM_CCER_CC2NE;

                TIM1->CCER |= TIM_CCER_CC3NP;
                TIM1->CCER |= TIM_CCER_CC3NE;
                TIM1->CCER &= ~TIM_CCER_CC3E;
        }
        else if((i>count_16_parts*3 - 1)&&(i<count_16_parts*4)) {

                TIM1->CCR2 = 65535 * (count_16_parts*4-i)/(count_16_parts);
                TIM1->CCR3 = 65535 * (count_16_parts*4-i)/(count_16_parts);
        }
        else if(i == count_16_parts*4) {

                TIM1->CCR2 = 0;
                TIM1->CCER &= ~TIM_CCER_CC3NP;
                TIM1->CCER |= TIM_CCER_CC3E;
        }
        else if((i>count_16_parts*4 - 1)&&(i<count_16_parts*5)) {

                TIM1->CCR3 = 65535 * (i-count_16_parts*4)/(count_16_parts);
        }
else if(i == count_16_parts*5) {

                TIM1->CCER &= ~TIM_CCER_CC3NP;
                TIM1->CCER |= TIM_CCER_CC3E; TIM1-
                >CCER &= ~TIM_CCER_CC3NE;

                TIM1->CCER |= TIM_CCER_CC4P;
                TIM1->CCER |= TIM_CCER_CC4E;
}
        else if((i>count_16_parts*5 - 1)&&(i<count_16_parts*6)) {

                TIM1->CCR3 = 65535 * (count_16_parts*6-i)/(count_16_parts);
                TIM1->CCR4 = 65535 * (count_16_parts*6-i)/(count_16_parts);
        }
else if(i == count_16_parts*6) {

                TIM1->CCR3 = 0;
                TIM1->CCER |= TIM_CCER_CC1NE;

                TIM1->CCER &= ~TIM_CCER_CC1NP;
                TIM1->CCER |= TIM_CCER_CC1NE;
                TIM1->CCER &= ~TIM_CCER_CC1E;
}
        else if((i>count_16_parts*6 - 1)&&(i<count_16_parts*7)) {

                TIM1->CCR4 = 65535 * (i-count_16_parts*6)/(count_16_parts);
                TIM1->CCR1 = 65535 * (i-count_16_parts*6)/(count_16_parts);
        }
        else if(i == count_16_parts*7) {

                TIM1->CCR4 = 65535;
```

```
                        TIM1->CCER &= ~TIM_CCER_CC1NP;
                        TIM1->CCER |= TIM_CCER_CC1E; TIM1-
                        >CCER |= TIM_CCER_CC1NE;
                }

                for(d=0;d<delay;d++)
                {
                }
        }
}

void Counterclockwise_Two(void) {

        uint32_t count_16_parts = count/16; //
        on 7, off 7, on 6, off 1, CCR increases
        TIM1->CCER &= ~TIM_CCER_CC1NE; // disable complimentary channel 1 (1) (in case it's not the first
round)

        TIM1->CCER |= TIM_CCER_CC4P;      // enable polarity on forward channel 4 (7) //
        TIM1->CCER |= TIM_CCER_CC4E;      enable direct channel 4 (7)

        TIM1->CCER |= TIM_CCER_CC3E; // enable direct channel 3 (6) TIM1->CCER &=
        ~TIM_CCER_CC3NE; // disable complimentary channel 3 (5) for (i = 0;i <=
        count_16_parts * 7;i++)
        {
                if (restart){ //disable all
                        TIM1->CCER &= ~(TIM_CCER_CC1E | TIM_CCER_CC1NE |
                        TIM_CCER_CC2E | TIM_CCER_CC2NE
                        | TIM_CCER_CC3E | TIM_CCER_CC3NE |
                        TIM_CCER_CC4E);
                        TIM1->CCER &= ~(TIM_CCER_CC1P | TIM_CCER_CC1NP |
                        TIM_CCER_CC2P | TIM_CCER_CC2NP
                        | TIM_CCER_CC3P | TIM_CCER_CC3NP |
                        TIM_CCER_CC4P);
                        TIM1->CCR1 = 0;
                        TIM1->CCR2 = 0;
                        TIM1->CCR3 = 0;
                        TIM1->CCR4 = 0;
                        restart=0;
                        break;
                }
                if (i < count_16_parts) {

                        TIM1->CCR4 = 65535 * i / (count_16_parts);
                        TIM1->CCR3 = 65535 * i / (count_16_parts);
                }
                //on 6, off 6, on 5, off 7, CCR decreases
                else if (i == count_16_parts)
                {
                        TIM1->CCER &= ~TIM_CCER_CC4E; // disable direct channel 4 (7

                        TIM1->CCER &= ~TIM_CCER_CC3NP; // enable polarity on complementary channel 3
(5)
                        TIM1->CCER |= TIM_CCER_CC3NE; // enable complimentary channel 3 (5)
                }
                else if ((i > count_16_parts - 1) && (i < count_16_parts * 2)) {

                        TIM1->CCR3 = 65535 * (count_16_parts * 2 - i) / (count_16_parts);
                }

                //on 5, off 5, on 4, off 6, CCR increases
                else if (i == count_16_parts * 2)
```

```
{
	TIM1->CCER &= ~TIM_CCER_CC3E; // disable direct channel 3 (6)

	TIM1->CCER |= TIM_CCER_CC2E; // enable direct channel 2 (4) TIM1->CCER |=
	TIM_CCER_CC3NP; // disable polarity on complementary channel 3
(5)
	TIM1->CCER &= ~TIM_CCER_CC2NE;// disable complimentary channel 2 (3)
}
else if ((i > count_16_parts * 2 - 1) && (i < count_16_parts * 3)) {

	TIM1->CCR3 = 65535 * (i - count_16_parts * 2) / (count_16_parts);
	TIM1->CCR2 = 65535 * (i - count_16_parts * 2) / (count_16_parts);
}
//on 4, off 4, on 3, off 5, CCR decreases
else if (i == count_16_parts * 3)
{
	TIM1->CCER &= ~TIM_CCER_CC3NE; // disable complimentary channel 3 (5)

	TIM1->CCER &= ~TIM_CCER_CC2NP; // enable polarity on complementary channel 2
(3)
	TIM1->CCER |= TIM_CCER_CC2NE; // enable complimentary channel 2 (3)
}
else if ((i > count_16_parts * 3 - 1) && (i < count_16_parts * 4)) {

	TIM1->CCR2 = 65535 * (count_16_parts * 4 - i) / (count_16_parts);
}
//on 3, off 3, on 2, off 4, CCR increases
else if (i == count_16_parts * 4)
{
	TIM1->CCER &= ~TIM_CCER_CC2E; // disable direct channel 2 (4)

	TIM1->CCER |= TIM_CCER_CC1E; // enable direct channel 1 (2) TIM1->CCER |=
	TIM_CCER_CC2NP; // disable polarity on complementary channel 2
(3)
	TIM1->CCER &= ~TIM_CCER_CC1NE; // disable complimentary channel 1 (1)
}
else if ((i > count_16_parts * 4 - 1) && (i < count_16_parts * 5)) {

	TIM1->CCR2 = 65535 * (i - count_16_parts * 4) / (count_16_parts);
	TIM1->CCR1 = 65535 * (i - count_16_parts * 4) / (count_16_parts);
}
//on 2, off 2, on 1, off 3, CCR decreases
else if (i == count_16_parts * 5)
{
	TIM1->CCER &= ~TIM_CCER_CC2NE; // disable complimentary channel 2 (3)

	TIM1->CCER &= ~TIM_CCER_CC1NP; // enable polarity on complementary channel 1
(one)
	TIM1->CCER |= TIM_CCER_CC1NE; // enable complimentary channel 1 (1)
}
else if ((i > count_16_parts * 5 - 1) && (i < count_16_parts * 6)) {

	TIM1->CCR1 = 65535 * (count_16_parts * 6 - i) / (count_16_parts);
}
//on 1, off 1, on 7, off 2, CCR increases
else if (i == count_16_parts * 6)
{
	TIM1->CCER &= ~TIM_CCER_CC1E; // disable direct channel 1 (2)

	TIM1->CCER |= TIM_CCER_CC1NP; // disable polarity on complementary channel 1
(one)

	TIM1->CCER &= ~TIM_CCER_CC4P; // disable polarity on forward channel 4 (7) (in
```

```
case it's not the first circle)
                                TIM1->CCER |= TIM_CCER_CC4E; // enable direct channel 4 (7)
                        }
                        else if ((i > count_16_parts * 6 - 1) && (i < count_16_parts * 7)) {

                                TIM1->CCR1 = 65535 * (i - count_16_parts * 6) / (count_16_parts);
                                TIM1->CCR4 = 65535 * (i - count_16_parts * 6) / (count_16_parts);
                        }
                        else if (i == count_16_parts * 7) {

                        }

                        for (d = 0;d < delay;d++) {

                        }
                }
}
```