

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»
КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ
И ПРОГРАММНОЙ ИНЖЕНЕРИИ (КАФЕДРА №43)

КУРСОВАЯ РАБОТА (ПРОЕКТ)
ЗАЩИЩЕНА С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ:

Доцент, к.т.н	/	/	/	А. А. Попов
(должность, учёная степень, звание)		(подпись)	(дата защиты)	(инициалы, фамилия)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К КУРСОВОЙ РАБОТЕ (ПРОЕКТУ)

Гирлянда из 8-ми светодиодов

ПО КУРСУ: «ПРОГРАММИРОВАНИЕ ВСТРОЕННЫХ ПРИЛОЖЕНИЙ»

РАБОТУ ВЫПОЛНИЛ (А) СТУДЕНТ (КА) ГРУППЫ:	4936	/	Белов Е. И. Крутов А. В. Черкасов Е. Б.
			(инициалы, фамилия)

/	/
(подпись студента)	(дата отчета)

Санкт-Петербург 2022

Содержание

1. Задание на курсовое проектирование	3
2. Техническое задание на прибор	4
2.1 Диаграмма вариантов использования	4
2.2 Основные требования	5
3. Схемы и алгоритмы работы	6
4. Тестирование	13
5. Экономическая оценка	21
6. Заключение	21
7. Список используемой литературы	22
Приложение А. Код программы	23
Функция main	23
Код обработки прерываний	24
Функции для режима с одним светодиодом	25
Функции для режима с двумя светодиодами	29

1. Задание на курсовое проектирование

Реализовать движение огонька влево, вправо с заданной скоростью (переключение линий поочередно), двух огней.

Для плавного зажигания и потухания светодиодов использовать ШИМ сигнал. Скорость и направление движения задавать, используя Keypad 4x4.

Какие кнопки использовать, и какими функциями наделить решить самостоятельно.

2. Техническое задание на прибор

В рамках разрабатываемой модели гирлянды выделяются следующие исполнители (актеры):

- Пользователь;
- Светодиоды;
- Микроконтроллер (как управляющее устройство).

Основные функции гирлянды:

По гирлянде движется светодиод (два светодиода) в одну или другую сторону.

У гирлянды есть 3 кнопки для изменения режима работы:

- 1)Изменение скорости движения светодиодов (4 скорости – зажигание и потухание 1с, 0.5, 0.25, 0.04)
- 2)Изменение направления движения (по часовой стрелки и против)
- 3)Изменение режима (1 или 2 огня)

Ограничения:

При нажатии нескольких кнопок одновременно, сработает обработчик на последнюю отжатую кнопку

2.1 Диаграмма вариантов использования



Рисунок 1 - диаграмма вариантов использования

2.2 Основные требования

Входы

3 кнопки:

- изменение скорости движения светодиодов
- изменение направления движения
- изменение режима (1 или 2 огня)

Выходы

8 светодиодов

Функции

По гирлянде движется светодиод (два светодиода) в одну или другую сторону. Для плавного зажигания и затухания используется ШИМ сигнал. С помощью кнопок можно изменять режимы работы гирлянды (1 или 2 огня), направления движения, скорость движения светодиодов.

Особенности

Отсутствуют

Питание

Питание от сети переменного тока через стандартный блок питания (USB адаптер).

Размеры и вес

Достаточно маленькие, чтобы использовать на рабочем столе

Стоимость производства

Стоимость отладочной платы STM32F103C8 - 83 рубля.

Стоимость 4-кнопочной клавиатуры – 44 рубля.

Светодиоды, провода и корпус – 100 рублей.

Сборка и тестирование - 50 рублей.

Таким образом стоимость производства изделия должна составить не более 300 рублей.

3. Схемы и алгоритмы работы

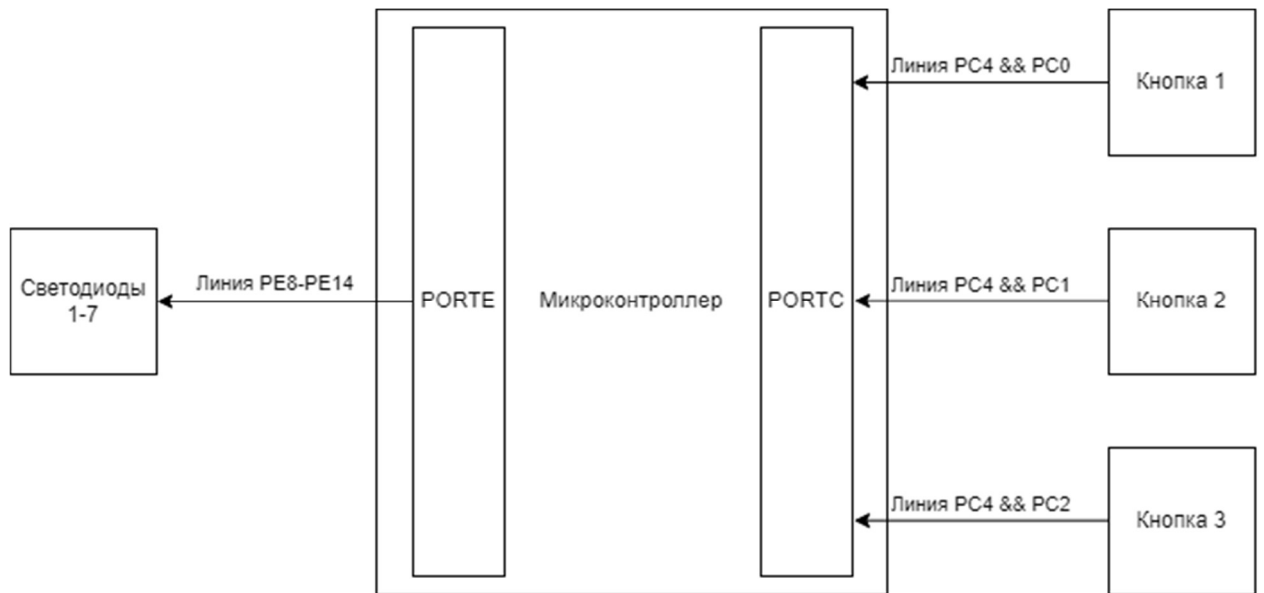


Рисунок 2 - Структурная схема подключения периферийных устройств к МК

Настройка тактирования:

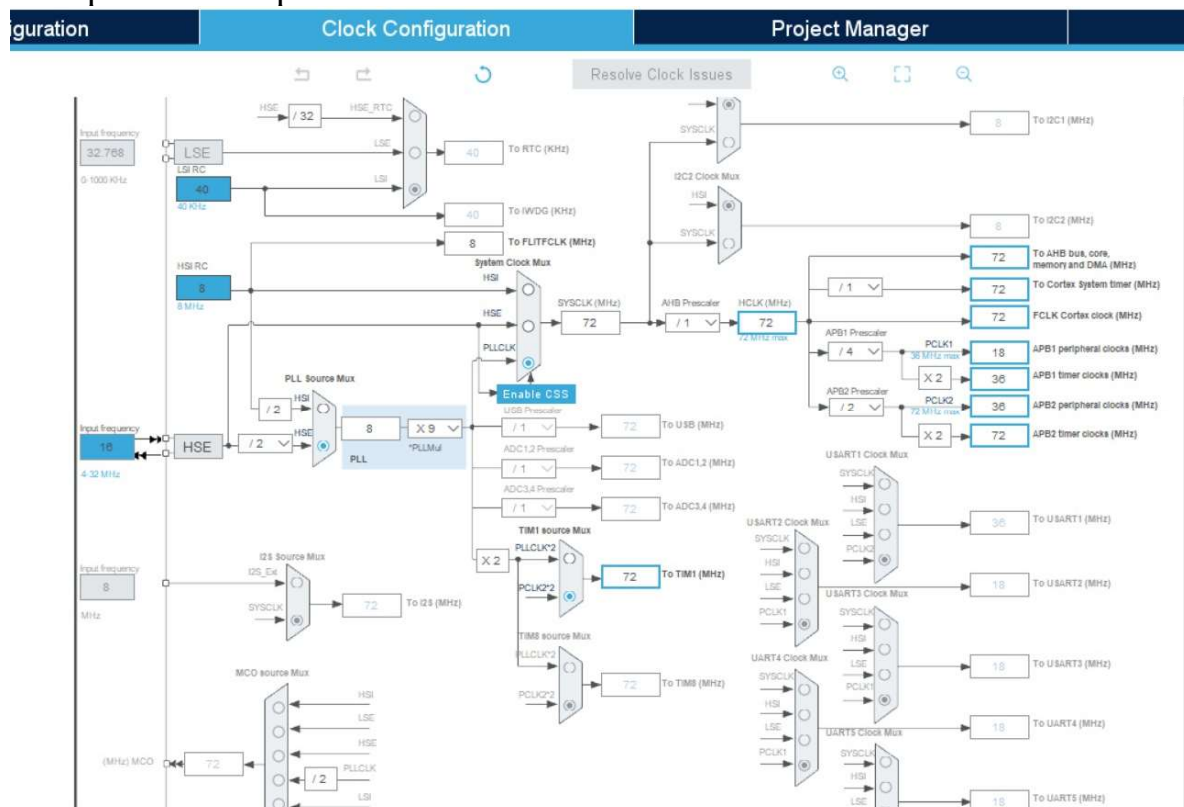


Рисунок 3 - Настройка тактирования в программе STM32CubeMX

Для работы кнопок использованы соответствующие линии согласно схеме, представленной ниже:

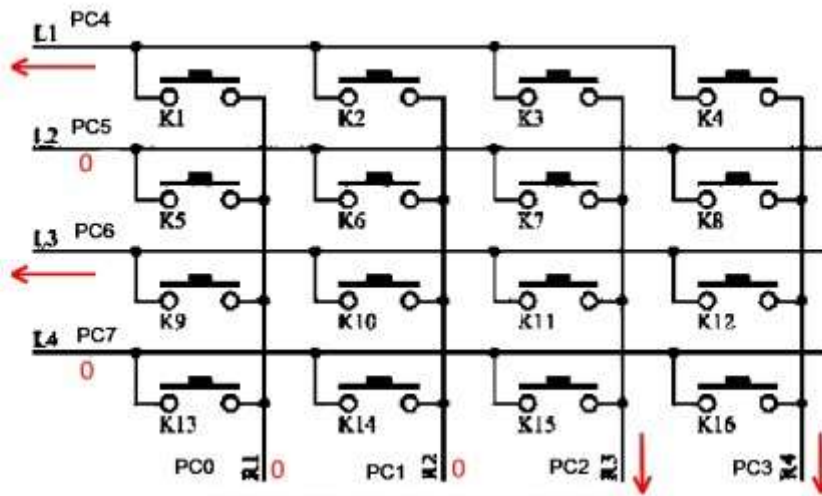


Рисунок 4 - Схема портов матричной клавиатуры 4x4

Для кнопок использовали обработчики внешних прерываний EXTI

Включаем шим для ch1/1n до ch4 с использованием первого таймера - TIM1 процедурой HAL_TIM_PWM_Start (&htim1, TIM_CHANNEL_1);

Задание значения шима осуществляется с помощью TIM1-CCR1 для двух каналов комплементарного и прямого на каждом из которых находится светодиод.

Так как присвоение значения шима происходит сразу для двух каналов, то возникает проблема присвоения значения для отдельного канала с одним светодиодом, данная проблема решается путем очистки битов канала, который не будет использоваться, пример:

```
TIM1->CCER &= ~TIM_CCER_CC3NE; (ВЫКЛЮЧЕНИЕ 3-его комплементарного канала)
```

Также можно включить канал после выключения

```
TIM1->CCER |= TIM_CCER_CC1NE; (включения первого компл. канала)
```

Для режима с двумя светодиодами мы использовали полярность, после изменения бита полярности, все новые присвоенные биты будут инвертироваться.

Для работы системы будем использовать модель прерываний, поскольку модель опроса состояний усложнит программу.

Изначально, после сброса, система находится в неинициализированном состоянии. Далее происходит автоматическая инициализация и система

переходит в состояние горения светодиодов в режим, установленный по умолчанию, и ожидает поступления внешнего сигнала (см. рис. 5)



Рисунок 5 - Диаграмма состояний

При поступлении внешнего сигнала (ввод с клавиатуры) система переходит в состояние управления светодиодом, в котором происходит смена режима скорости либо изменение направления на противоположное (по часовой/против часов стрелки, либо изменение режима (1 или 2 огня). После завершения обработки прерывания система возвращается в состояние горения внешних светодиодов.

Рассмотрим работу системы на обобщенной диаграмме последовательности взаимодействия:

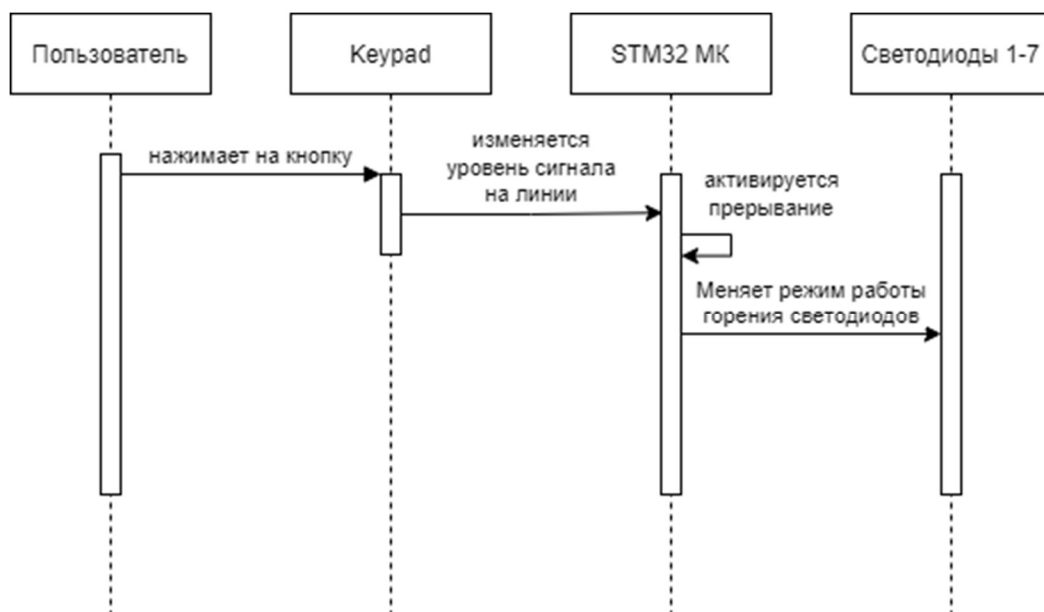


Рисунок 6 - Диаграмма последовательности взаимодействия

Выводы по диаграмме:

- 1) Будем использовать прерывания для инициации обработки нажатия кнопок. Каждая кнопка (PC0-PC2) будет присоединена к соответствующей линии. Система прерываний микроконтроллера STM32F103C8 позволяет адресовать до 20 прерываний от входных линий, что позволит создать прерывание для каждой кнопки.
- 2) Для работы шим необходим таймер. Для этого будем использовать системный таймер, счетчик задержки и обработчик прерываний системного таймера.
- 3) Для работы таймера необходима настройка прерываний системного таймера SysTick_IRQn и прерываний обработки ввода.

Детализируем состояния системы на диаграмме активности функции main:

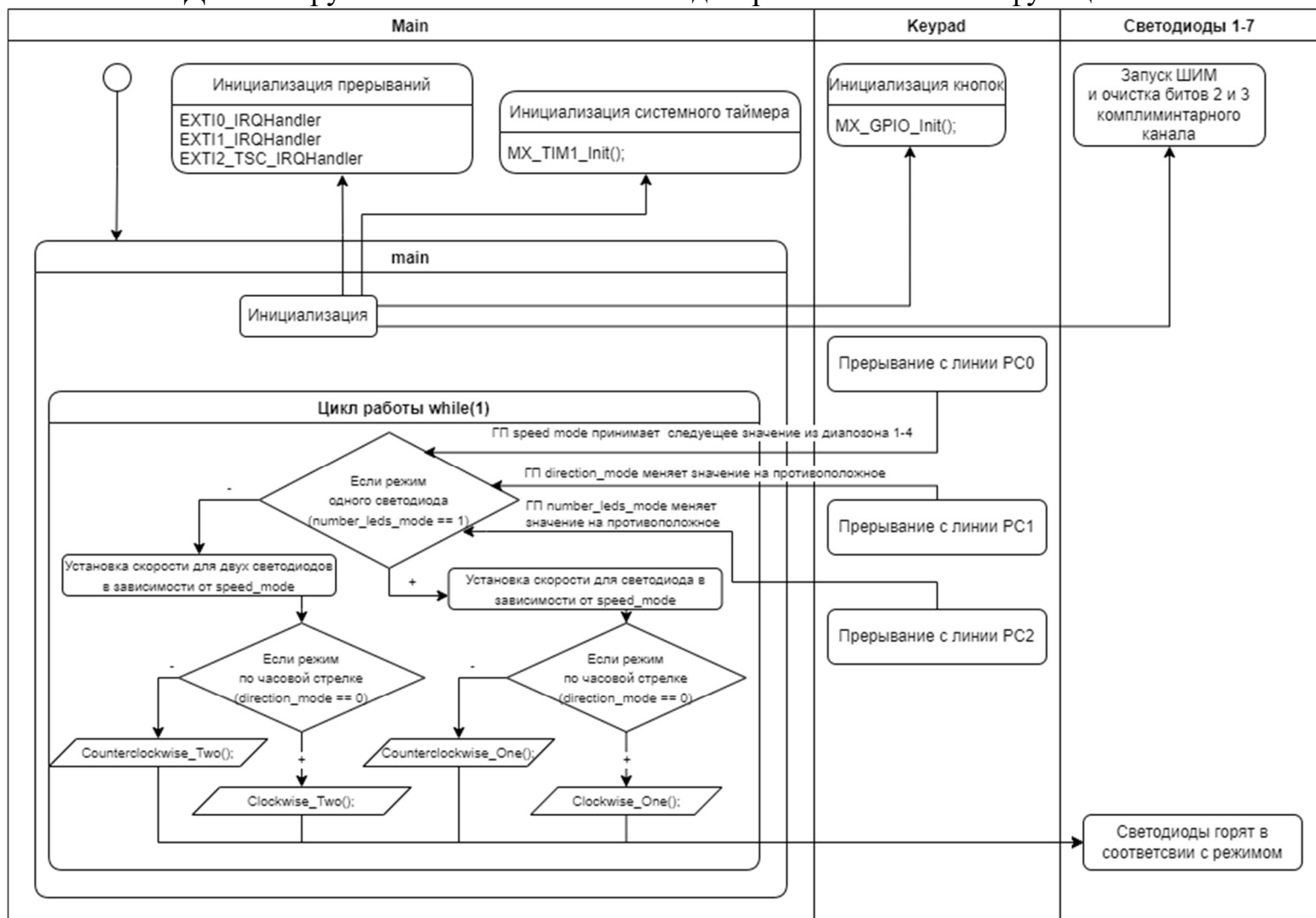


Рисунок 7 - Диаграмма активности функции main (ГП – глобальная переменная)

Отсюда следует выделить следующие основные управляющие структуры:

- 1) Функция `main()` с бесконечным циклом.
- 2) Функция `MX_TIM1_Init()` для инициализации системного таймера.
- 3) Функция `MX_GPIO_Init()`, реализующая инициализацию кнопок (разрешение работы порта GPIO C, настройка линий PC0-PC2, настройка EXTI).
- 4) Функции `Clockwise_One()`, `Counterclockwise_One()`, `Clockwise_Two()`, `Counterclockwise_Two()`, реализующие работу светодиодов (линии PE8-PE14).
- 5) Функции `EXTI0_IRQHandler()`, `EXTI1_IRQHandler()`, `EXTI2_TSC_IRQHandler()`, реализующие настройку прерываний.

Пример функции `main` в Приложении А

Рассмотрим алгоритм обработки прерываний `EXTI0_IRQHandler()` на рис. 8:

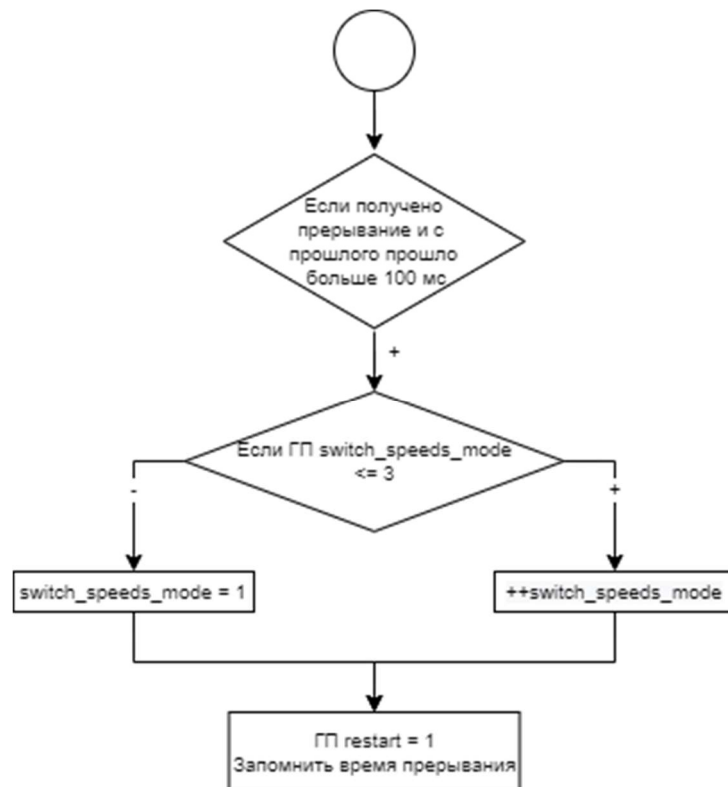


Рисунок 8 - Алгоритм прерываний

Аналогично устроены и `EXTI1_IRQHandler()` и `EXTI2_TSC_IRQHandler()`, только вместо изменения глобальной переменной `switch_speeds_mode`, они инвертируют значение переменных `direction_mode` и `number_leds_mode` соответственно.

Пример функций обработки прерываний в Приложении А

Рассмотрим алгоритм работы гирлянды для одного светодиода:

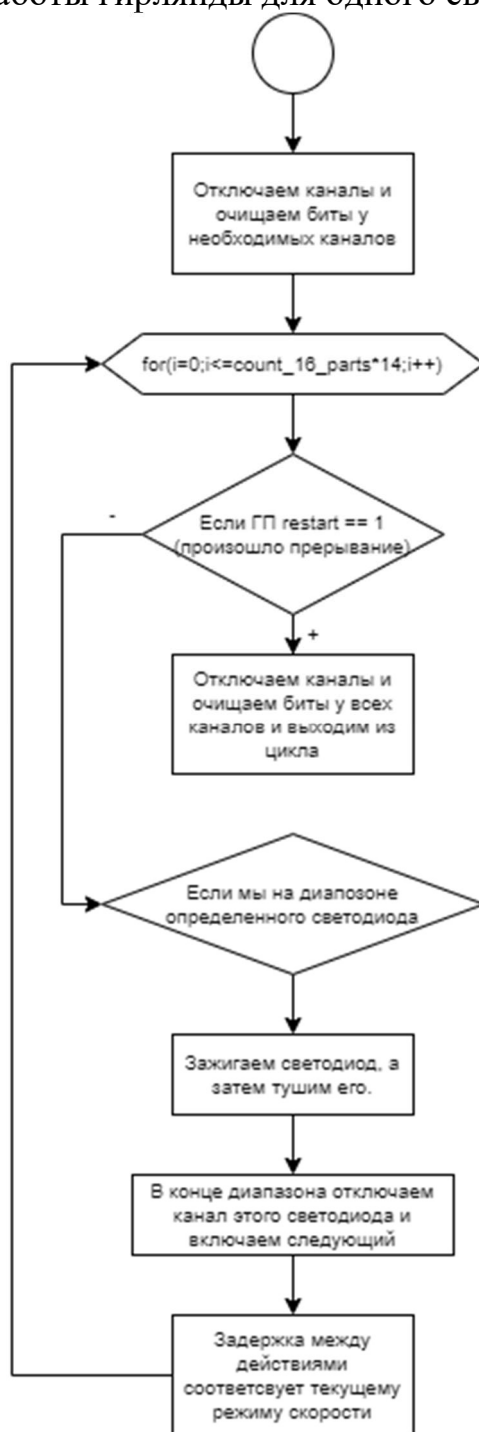


Рисунок 9 - Алгоритм работы гирлянды для одного светодиода

Пример [функций для режима с одним светодиодом](#) в Приложении А

Рассмотрим алгоритм работы гирлянды для двух светодиодов:

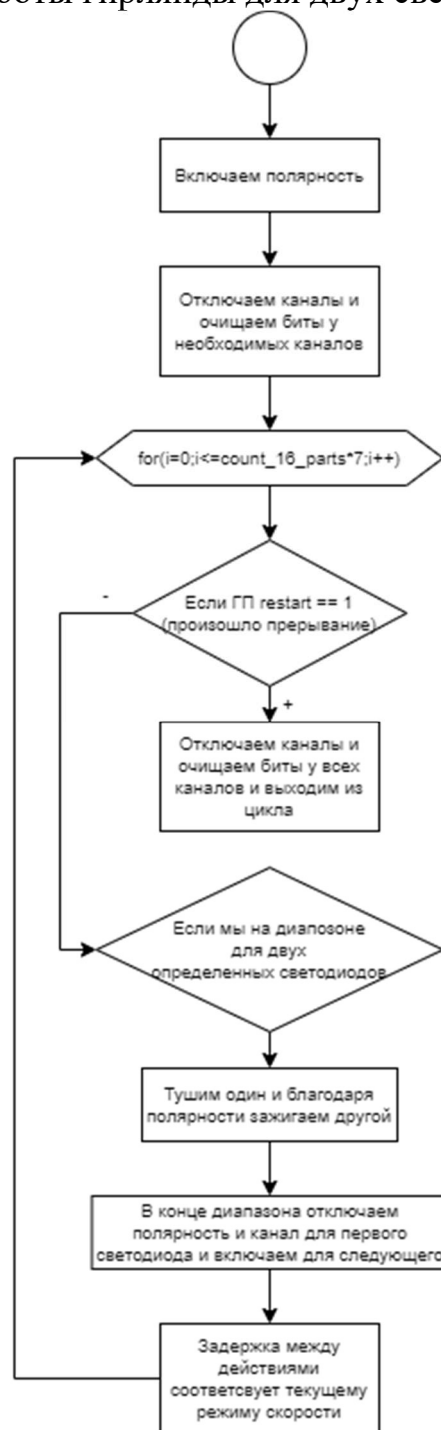


Рисунок 10 - Алгоритм работы гирлянды для двух светодиодов

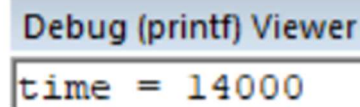
Пример функций для режима с двумя светодиодами в Приложении А

4. Тестирование

Тесты для контроля соответствия прибора техническому заданию:

Все тесты для направления движения по часовой стрелке

- Режим одного диода для скорости - 1 сек. на потухание и зажигание:



```
Debug (printf) Viewer
time = 14000
```

Рисунок 11 - Время на весь круг = 14000 миллисекунд

$$|14000 - 1 \text{ сек.} \cdot 1000 \cdot 2 \cdot 7 \text{ диодов}| = 0 \text{ миллисекунд погрешность}$$

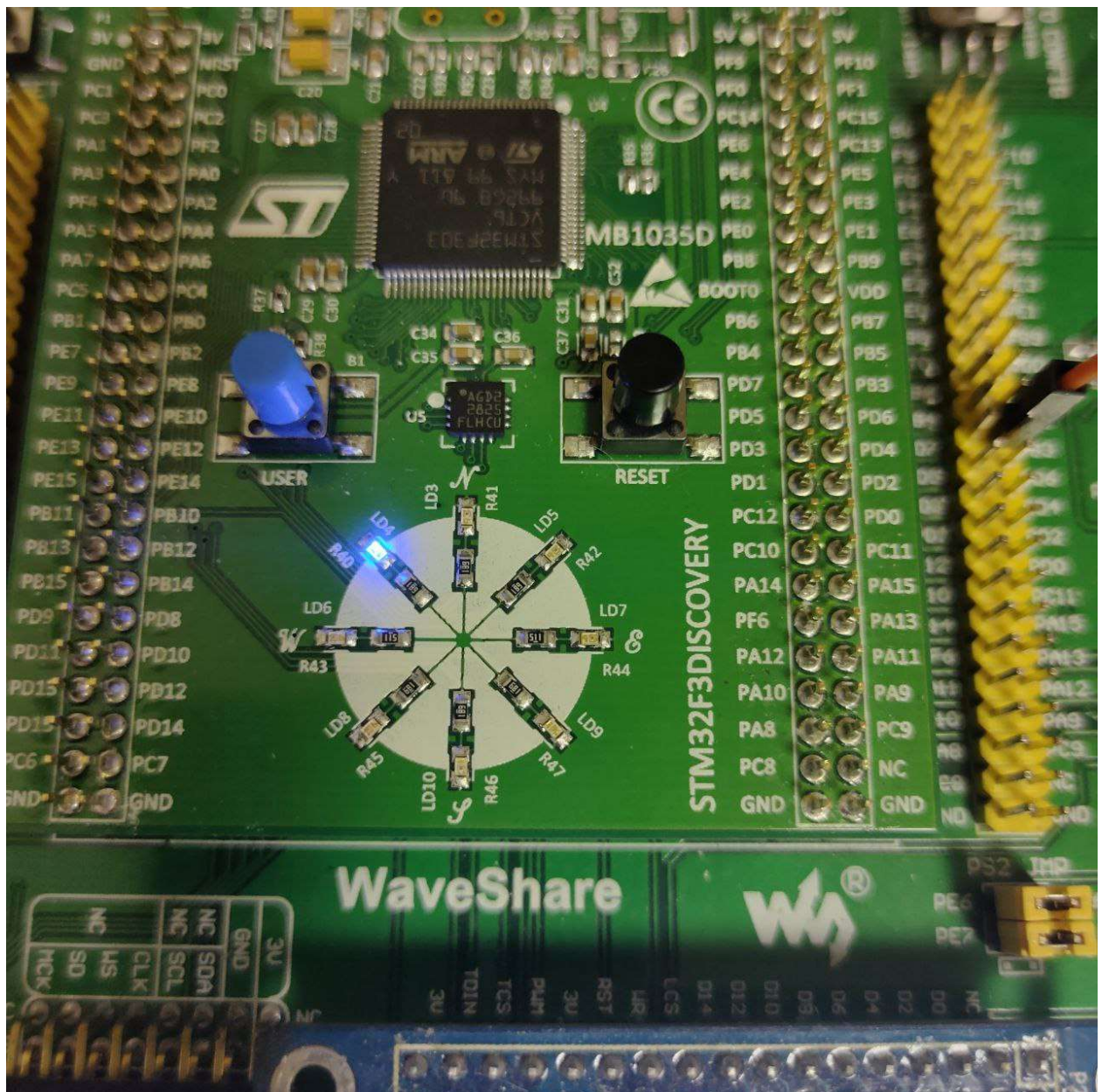


Рисунок 12 - Режим одного диода для скорости - 1 сек. на потухание и зажигание

- Режим одного диода для скорости – 0.5 сек. на потухание и зажигание:

Debug (printf) Viewer
time = 7006

Рисунок 13 - Время на весь круг = 7006 миллисекунд

$|7006 - 0.5\text{сек.} \cdot 1000 \cdot 2 \cdot 7\text{диодов}| = 6\text{ миллисекунд погрешность}$

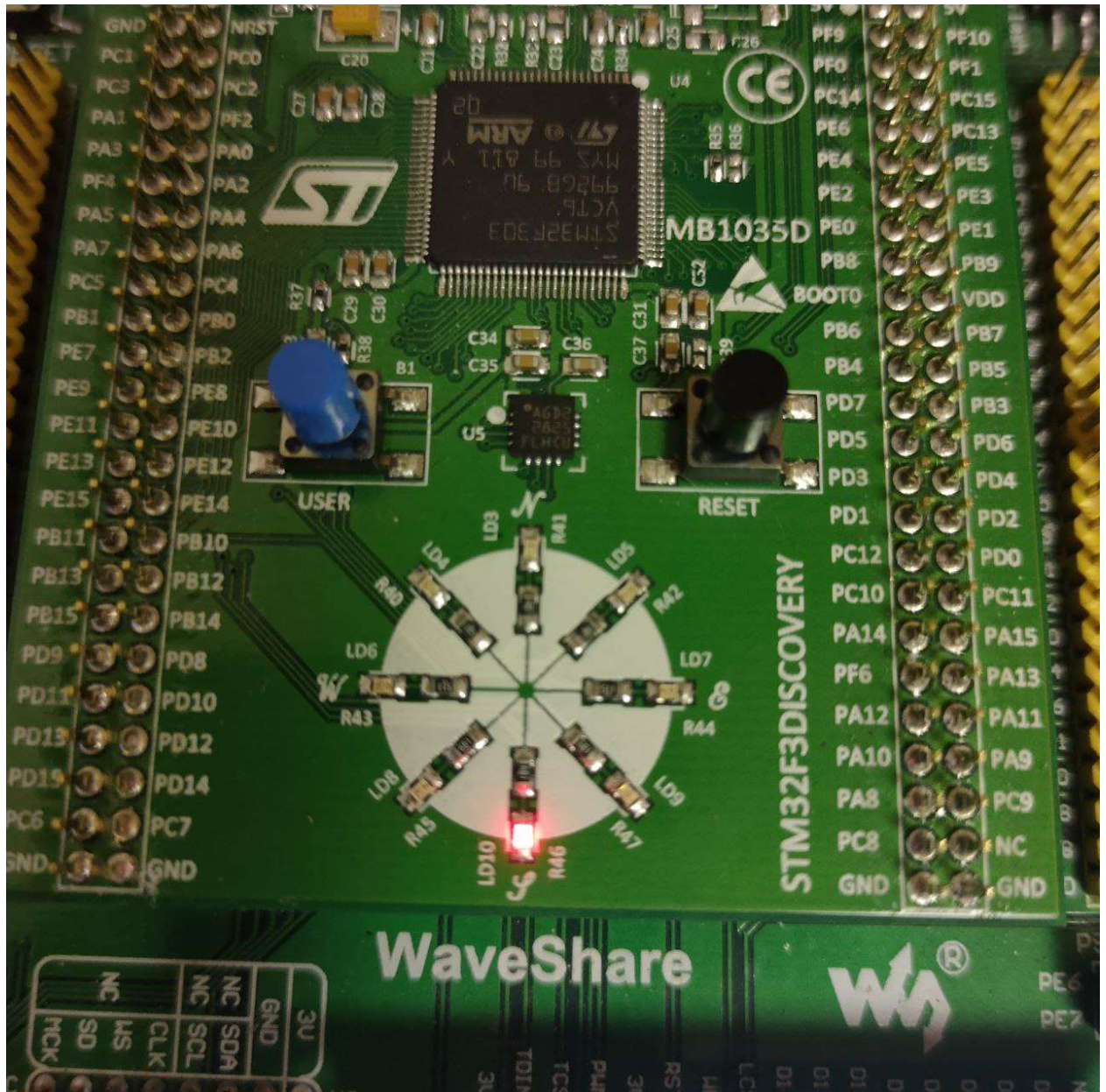


Рисунок 14 - Режим одного диода для скорости - 0.5 сек. на потухание и зажигание

- Режим одного диода для скорости – 0.25 с. на потухание и зажигание:

Debug (printf) Viewer

time = 3508

Рисунок 15 - Время на весь круг = 3508 миллисекунд

$$|3508 - 0.25\text{сек.} \cdot 1000 \cdot 2 \cdot 7\text{диодов}| = 8 \text{ миллисекунд погрешность}$$

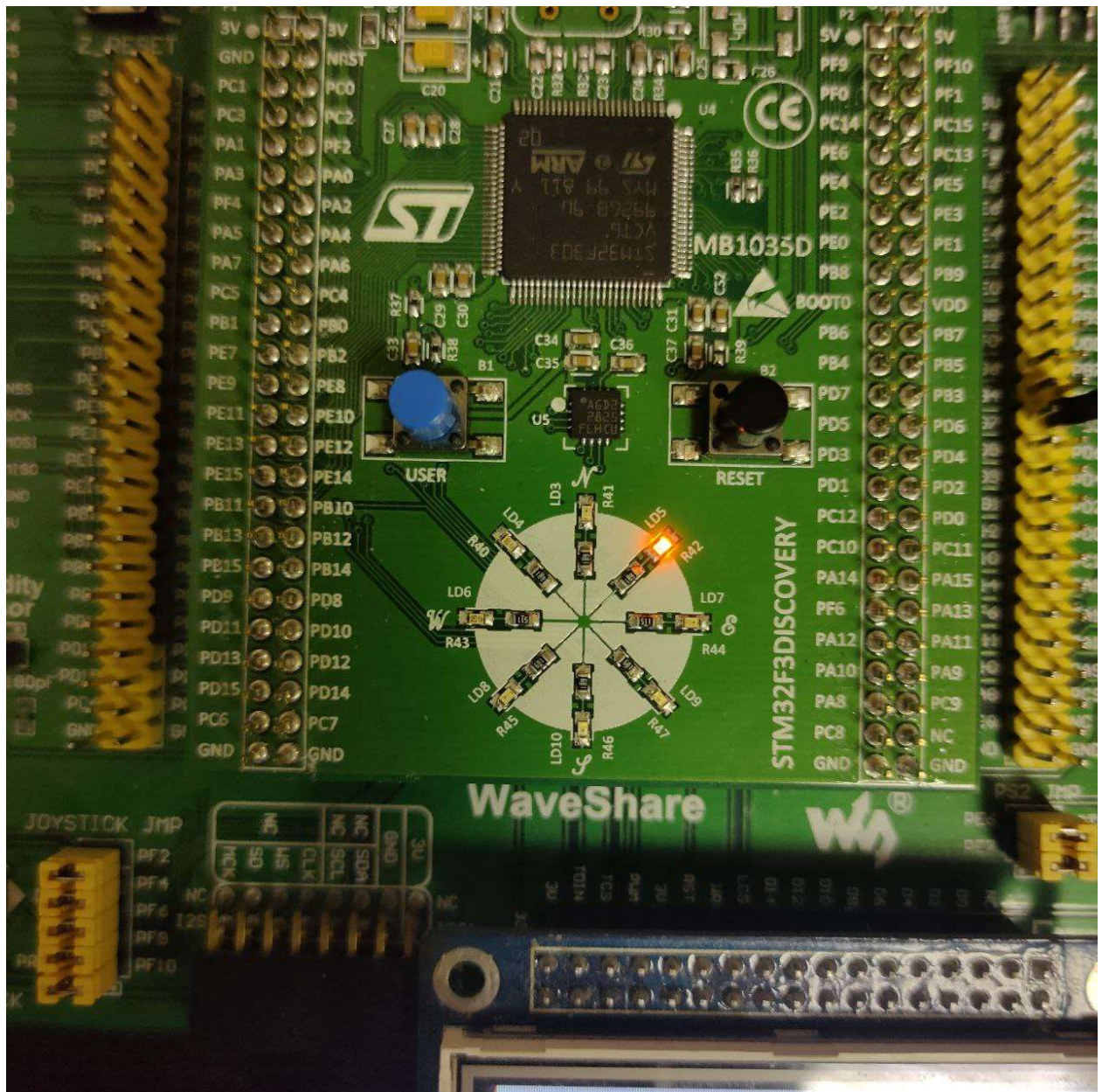


Рисунок 16 - Режим одного диода для скорости – 0.25 с. на потухание и зажигание

- Режим одного диода для скорости – 0.04 с. на потухание и зажигание:

Debug (printf) Viewer
time = 554

Рисунок 17 - Время на весь круг = 554 миллисекунды

$$|554 - 0.04\text{сек.} * 1000 * 2 * 7\text{диодов}| = 4 \text{ миллисекунды погрешность}$$

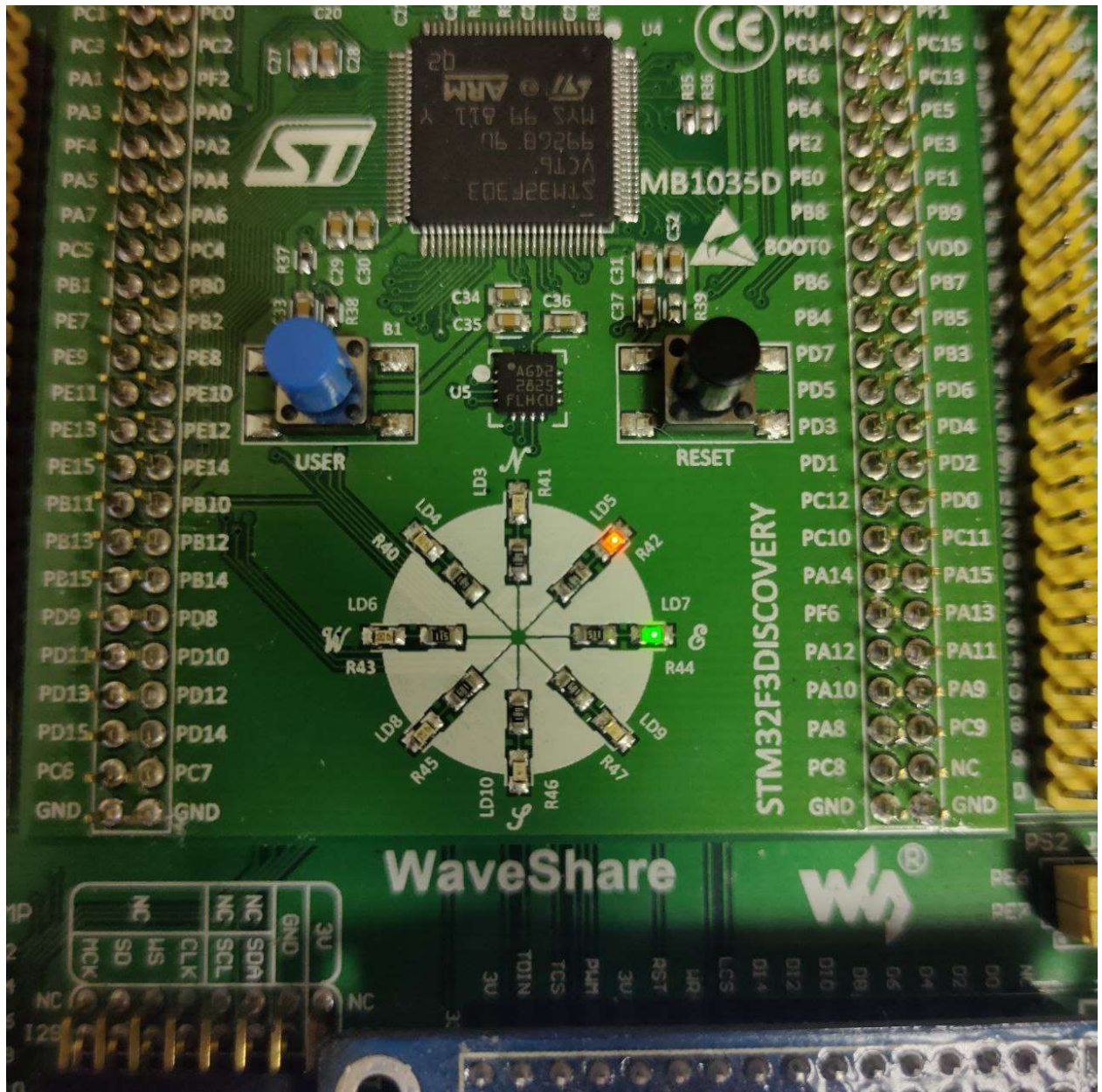


Рисунок 18 - Режим одного диода для скорости – 0.04 с. на потухание и зажигание:

- Режим двух диодов для скорости - 1 сек. на потухание и зажигание:

Debug (printf) Viewer

time = 13998

Рисунок 19 - Время на весь круг = 13998 миллисекунд

$$|13998 - 1\text{сек.} \cdot 1000 \cdot 2 \cdot 7\text{диодов}| = 2 \text{ миллисекунды погрешность}$$

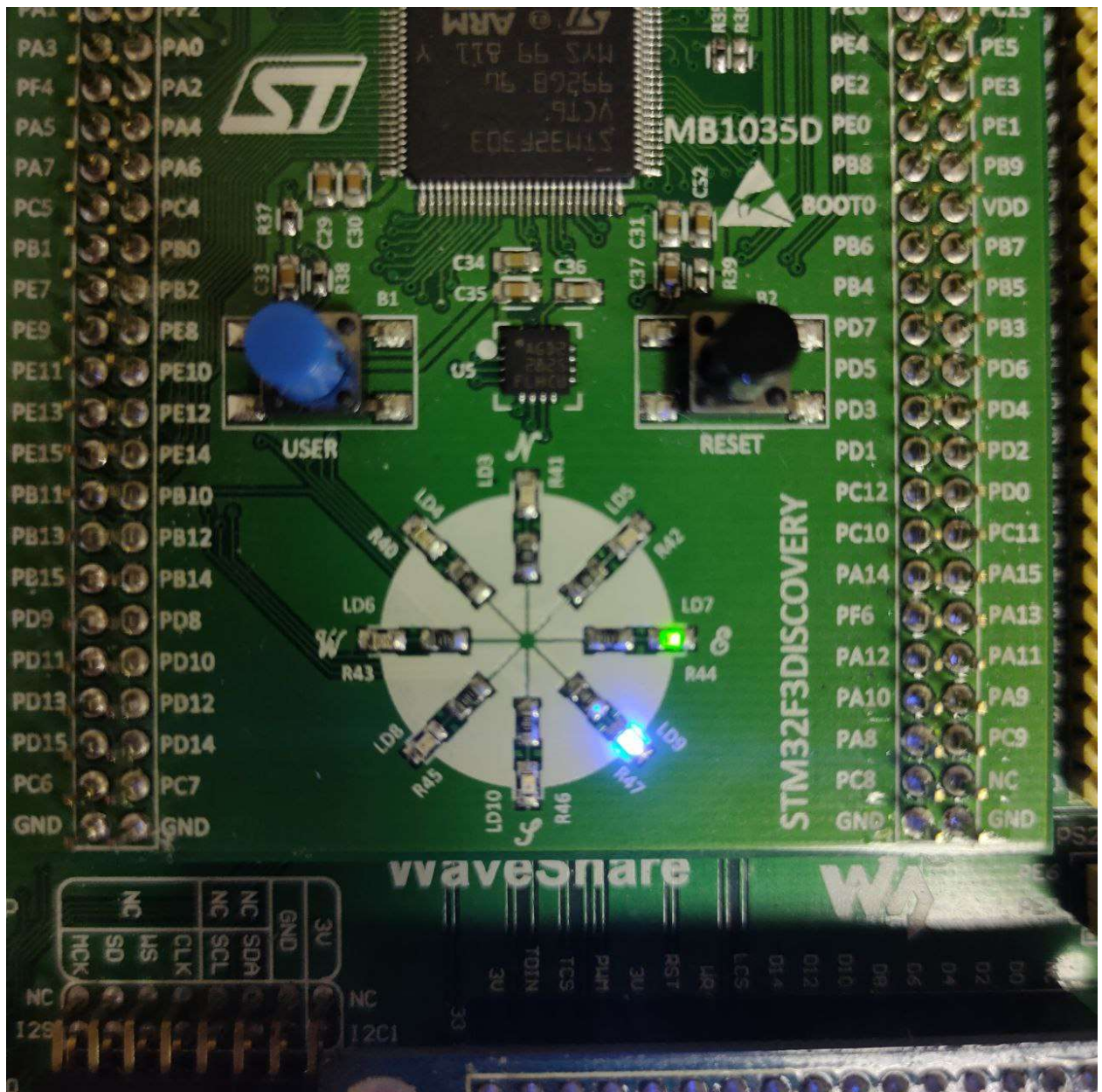


Рисунок 20 - Режим двух диодов для скорости - 1 сек. на потухание и зажигание

- Режим двух диодов для скорости – 0.5 сек. на потухание и зажигание:

Debug (printf) Viewer

time = 7002

Рисунок 21- Время на весь круг = 7002 миллисекунд

$$|7002 - 0.5\text{сек.} * 1000 * 2 * 7\text{диодов}| = 2 \text{ миллисекунды погрешность}$$

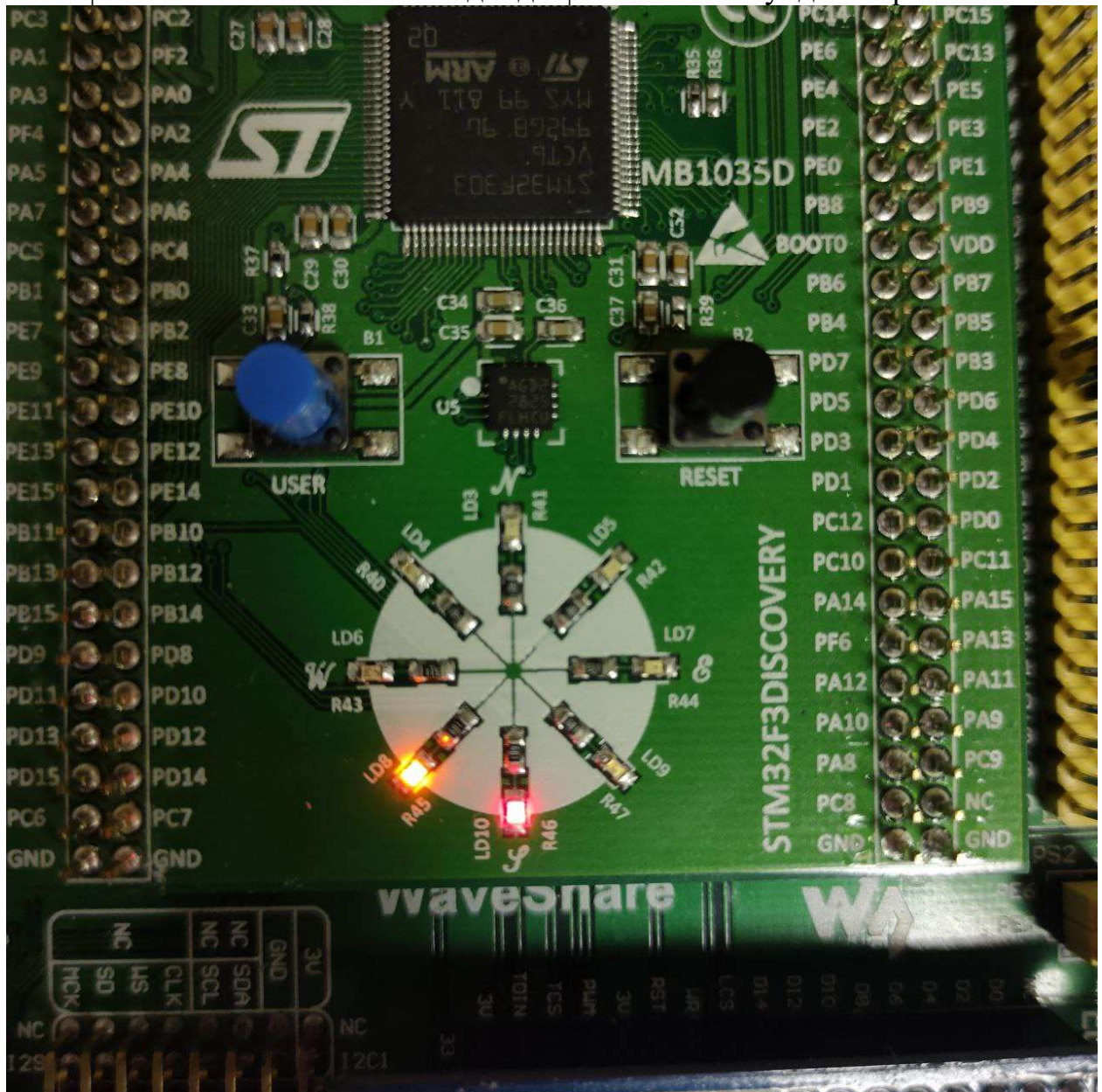


Рисунок 22 - Режим двух диодов для скорости - 0.5 сек. на потухание и зажигание

- Режим двух диодов для скорости – 0.25 с. на потухание и зажигание:

Debug (printf) Viewer
time = 3498

Рисунок 23 - Время на весь круг = 3498 миллисекунд

$$|3498 - 0.25\text{сек.} * 1000 * 2 * 7\text{диодов}| = 2 \text{ миллисекунды погрешность}$$

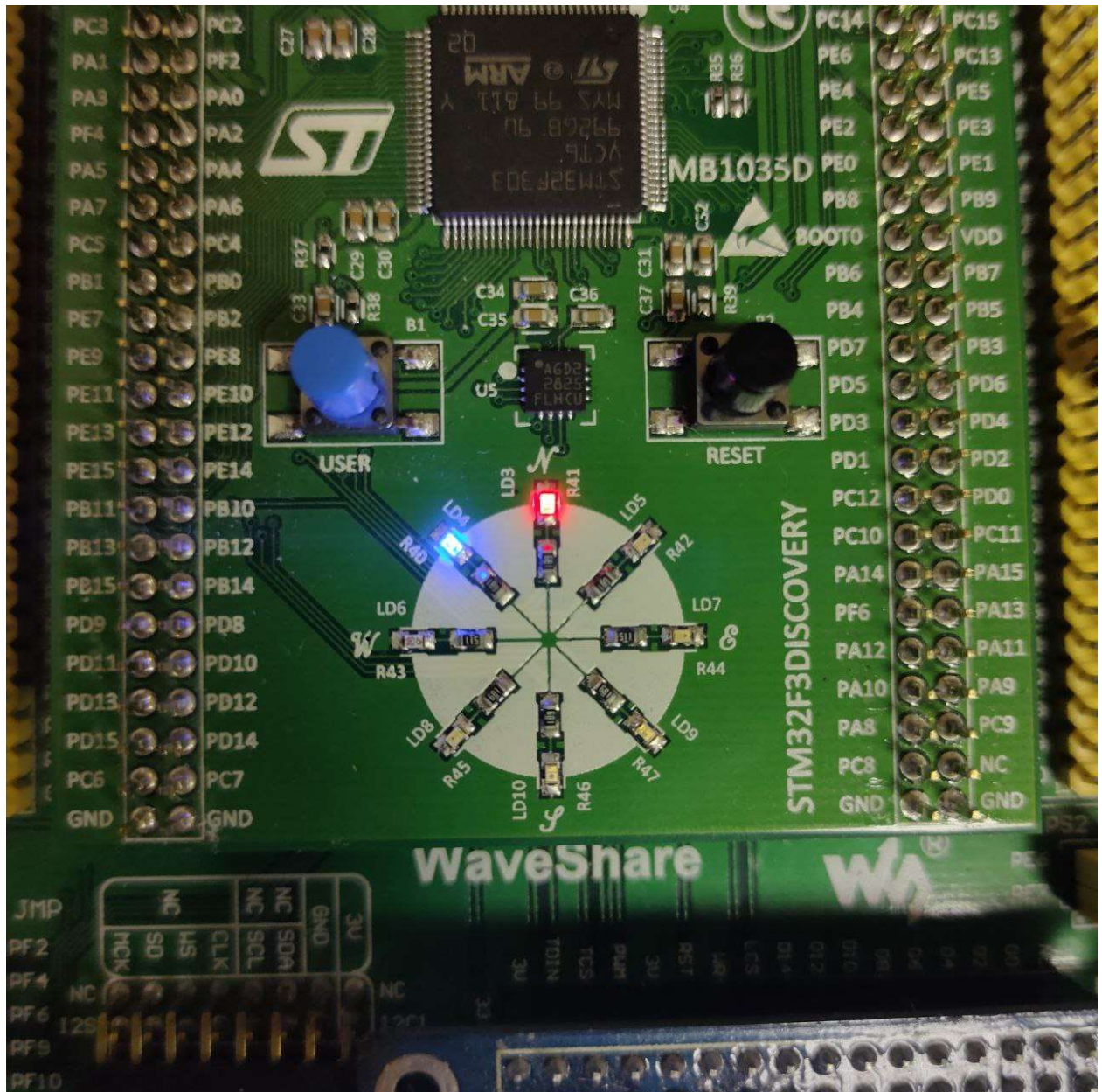


Рисунок 24 - Режим двух диодов для скорости - 0.25 сек. на потухание и зажигание

- Режим двух диодов для скорости – 0.04 с. на потухание и зажигание:

Debug (printf) Viewer

time = 560

Рисунок 25 - Время на весь круг = 560 миллисекунд

$$|560 - 0.04\text{сек.} * 1000 * 2 * 7\text{диодов}| = 0 \text{ миллисекунд погрешность}$$

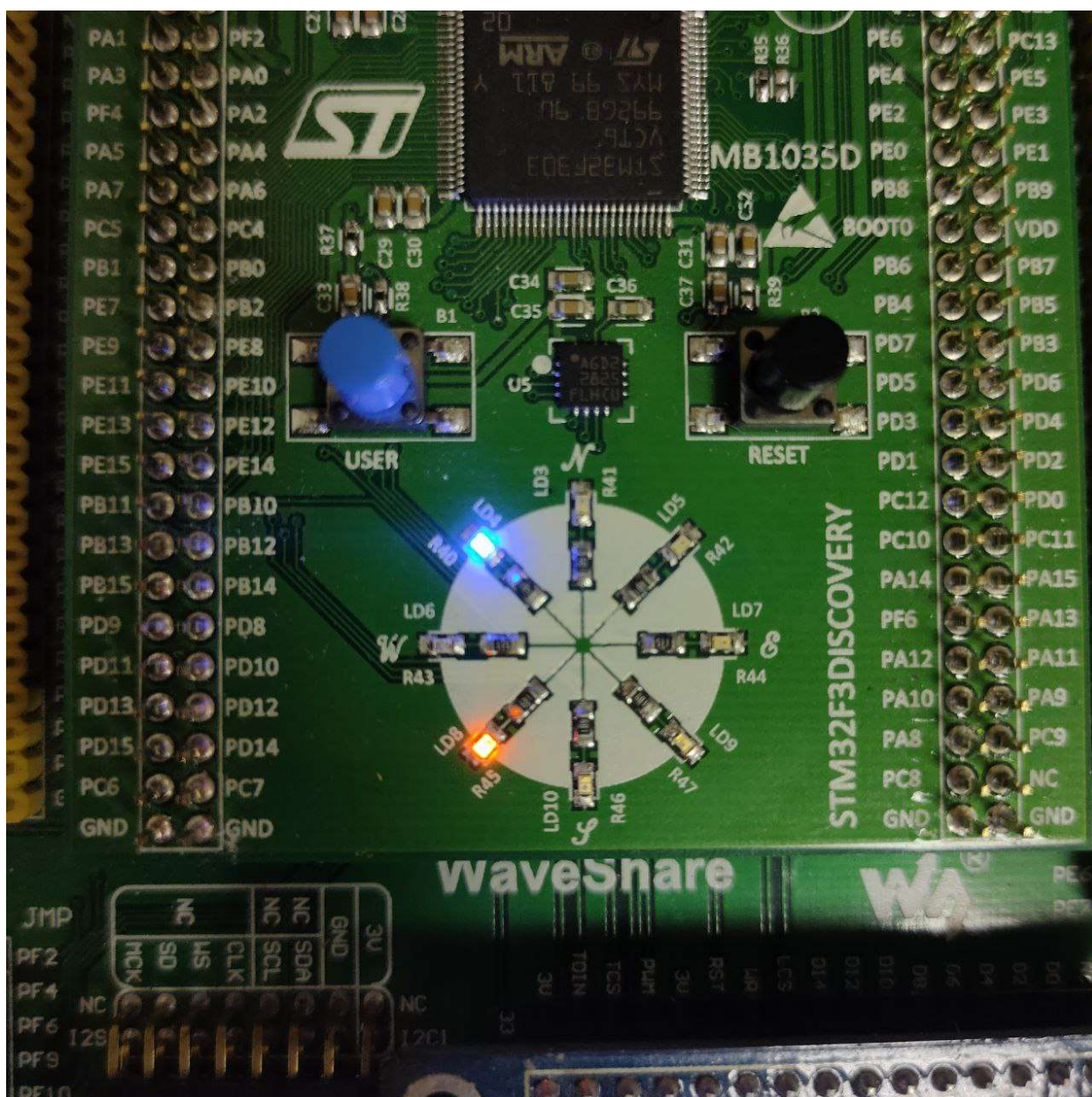


Рисунок 26 - Режим двух диодов для скорости - 0.04 сек. на потухание и зажигание

Аналогично для направления движения против часовой стрелки, результаты в точности совпадают.

5. Экономическая оценка

Компонент	Цена на Aliexpress	Ссылка на Aliexpress
Микроконтроллер	116 руб.	STM32F103C8T6
4-х кнопочная клавиатура	44 руб.	Матрица для клавиатуры для Arduino UNO MEGA2560 DUE button
8 светодиодов	5,44 руб	100 шт., 5 мм Светодиодные диоды F5, набор в ассортименте
Итог:	165 руб	Без учета сборки, проводов и корпуса.

При массовом производстве, цена на компоненты будет ниже из-за оптовых покупок, что является одним из путей снижения цены.

6. Заключение

В результате выполнения данного проекта было осуществлено проектирование устройства на базе микроконтроллера. Были получены навыки в области архитектуры и программного обеспечения встроенных систем, а также знания о структуре, функциях и основах программирования микроконтроллеров, позволяющих решать вопросы анализа функционирования программного обеспечения встраиваемых систем.

7. Список используемой литературы

1. Микроконтроллеры. Разработка встраиваемых приложений: учебное пособие / А.Е. Васильев; С.-Петербургский государственный политехнический ун-т. - СПб. : Изд-во СПбГПУ, 2003. - 211 с.
2. The Definitive Guide to ARM Cortex-M3 and Cortex-M4 Processors. Third Edition. Joseph Yiu. ARM Ltd., Cambridge, UK. [электронный ресурс] // URL: <https://www.pdfdrive.com/the-definitive-guide-to-arm-cortex-m3-and-cortex-m4-processors-e187111520.html>
3. Джозеф Ю. Ядро Cortex-M3 компании ARM. Полное руководство. 2012. ISBN: 978- 5-94120-243-0. [электронный ресурс] // URL: <https://bok.xyz/book/2373589/b5c3ad>
4. RM0008. Reference manual STM32F101xx, STM32F102xx, STM32F103xx, STM32F105xx and STM32F107xx advanced Arm®-based 32-bit MCUs [электронный ресурс] // URL: https://www.st.com/resource/en/reference_manual/cd00171190-stm32f101xx-stm32f102xx-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. [электронный ресурс] // URL: <https://www.st.com/resource/en/datasheet/stm32f103c8.pdf>
6. Мартин М. Инсайдерское руководство по STM32 [электронный ресурс] // URL: <https://istarik.ru/file/STM32.pdf>
7. Рекомендация МСЭ-R М.1677-1 Международный код Морзе. Международный союз электросвязи 2009 [электронный ресурс] //URL: https://www.itu.int/dms_pubrec/itu-r/rec/m/R-REC-M.1677-1-200910-I!!PDF-R.pdf

Приложение А. Код программы

Для удобства реализация проекта находится в открытом репозитории на GitHub:

https://github.com/Krutov777/stm32_garland

Функция main

```
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 cannel 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 */
}

```

Код обработки прерываний

// 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);
}

```

Функции для режима с одним светодиодом

```

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 ch4n that 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++)
    {
    }
}
}

```

Функции для режима с двумя светодиодами

```

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

            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
(1)
    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
(1)
    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++)
    {
    }
    }
}

```