

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ  
ОБРАЗОВАТЕЛЬНОЕ  
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
МОСКОВСКИЙ ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ  
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)  
ФИЗТЕХ-ШКОЛА РАДИОТЕХНИКИ И КИБЕРНЕТИКИ

# Работа с семисегментным индикатором. Устранение дребезга.

Работу выполнили:

Державин Андрей Андреевич, группа Б01-901

Хайдари Фарид Гулович, группа Б01-901

Шурыгин Антон Алексеевич, группа Б01-909

Лирисман Карина Сергеевна, группа Б03-001

Долгопрудный, 2021

# Содержание

<b>1</b>	<b>Постановка задачи проекта . . . . .</b>	<b>3</b>
<b>2</b>	<b>Кнопка: . . . . .</b>	<b>4</b>
2.1	Принцип работы . . . . .	4
2.2	Дребезг контактов . . . . .	6
<b>3</b>	<b>Семисегментный индикатор . . . . .</b>	<b>9</b>
3.1	Принцип работы . . . . .	9
<b>4</b>	<b>Программная реализация нашего проекта на языке C с применением интегрированной среды разработки IAR for ARM . . . . .</b>	<b>11</b>
4.1	Бегущая строка . . . . .	11
4.2	Счетчик нажатий кнопки . . . . .	19

# 1 Постановка задачи проекта

Работа с портами ввода-вывода отладочной платы; применение знаний о системе тактирования портов, которые позволят "начать общение" с внешним миром. В нашем случае под "внешним миром" пониманием обработку нажатия на кнопку, а так же подключение семисегментного индикатора.

## **Оборудование:**

Семисегментный индикатор, резисторы номиналом 300 Ом, соединительные провода, макетная плата.

## **Отладочная плата для STM32F0DISCOVERY:**

микроконтроллер STM32F051R8T6 до 48MHz, 64kBFlash, 8kB Ram, корпус LQFP64, встроенный эмулятор-отладчик ST – LINK/V2.

Питание платы способно осуществляться двумя способами: через USB, через внешний источник питания. Напряжение питания от внешнего источника 3 V и 5 V.

Отчет по проекту разделён на три логические части: описание задачи с кнопкой, с семисегментным индикатором, а так же программная реализация.

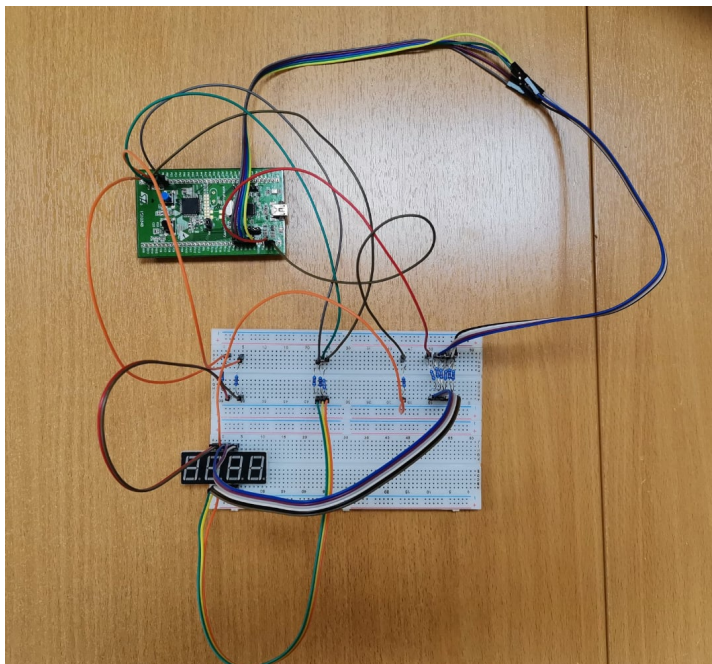


Рис. 1

## 2 Кнопка:

### 2.1 Принцип работы

На данном микроконтроллере существует 6 различных портов:

- GPIOA
- GPIOB
- GPIOC
- GPIOD
- GPIOE

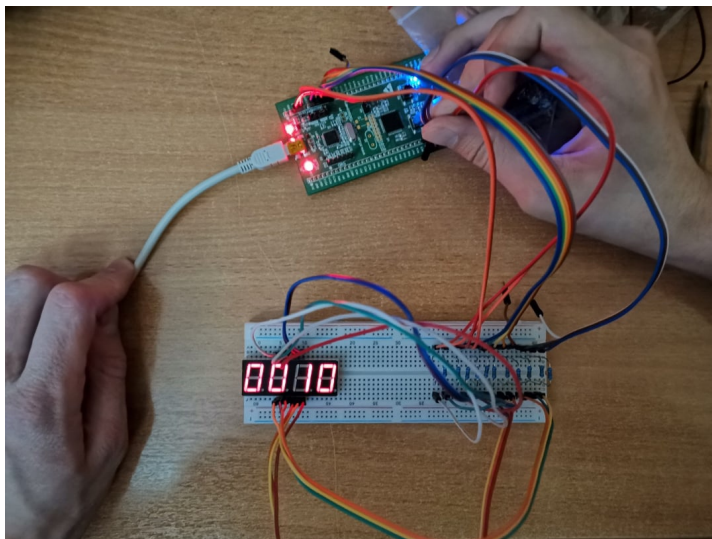


Рис. 2

- GPIOF

У каждого порта по 16 пинов. Однако в большинстве случаев в виду ограниченности ножек микроконтроллера не все порты реализуются полностью. В случае нашей платы: три пина заняты под два светодиода и одну кнопку, два пина заняты для соединения с программатором.

Управление с пинами осуществляем с помощью LL функций.

```
//обычная функция для записи в регистр
LL_GPIO_SetPinMode(GPIOx, LL_GPIO_PIN_x, Regime) ;
//За один раз данная функция конфигурирует только один пин.

//Настройка тип цифрового выхода
LL_GPIO_SetPinOutputMode(GPIOx, LL_GPIO_PIN_x, Regime);
// Аналогично только один пин.

//Функции для изменения выходного состояния
```

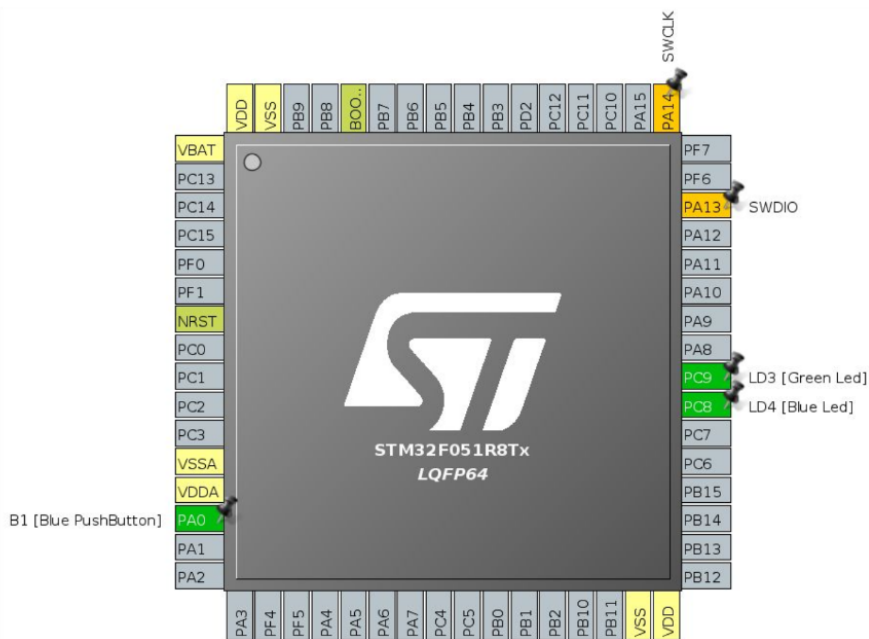


Рис. 3 Устройства подключения кнопки и график с дребезгом

```
LL_GPIO_WriteOutputPort(GPIOx, output_value); -> ODR
```

```
LL_GPIO_WriteOutputPin(GPIOx, bits_of_pins); -> BSRR
```

```
LL_GPIO_ResetOutputPin(GPIOx, bits_of_pins); -> BRR
```

//Можно писать в несколько пинов!

## 2.2 Дребезг контактов

На рис.4 показано подключение кнопки USER на нашей отладочной плате. Синяя кнопка заземлена, так же подключена в пину PA0.

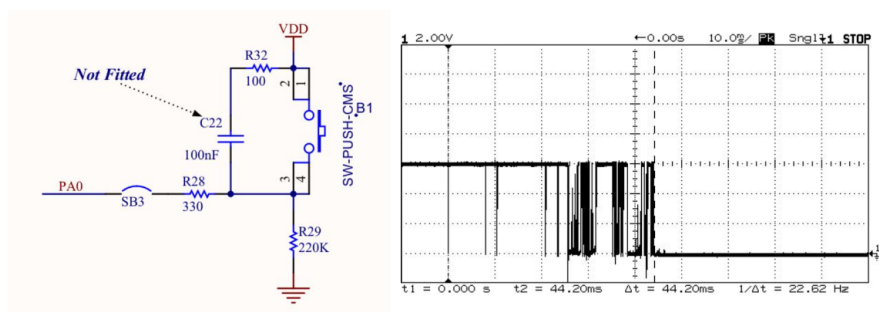


Рис. 4 Устройства подключения кнопки и график с дребезгом

Проблема дребезга заключается в том, что при нажатии на кнопку происходит на самом деле не только лишь одно нажатие, а еще и большое количество ложных переключений, которые приводят к ложным положительным срабатываниям. Нативным образом график этого процесса в реальном и в идеальном случаях показан на рис. 4 и рис.5.

Методы борьбы с дребезгом.

**Аналоговый:**

Установка низкочастотный фильтр для фильтрации дребезга, который мы представляем как высокочастотный сигнал.

**Программные методы:**

Создание некоторой задержки, после которой мы снова опрашиваем кнопку, чтобы удостовериться, что кнопка действительно была нажата.

В нашей работе мы пытались бороться с дребезгом различными программными методами: наивным образом, а так же более совершенным. Ниже приводим код наивной и более совершенной реализации:

```
// Naive realisation
// 1 - кнопка нажата; 0 - отжата
status = LL_GPIO_IsInputPinSet(GPIOA, LL_GPIO_PIN_0); // check regis
```

```
if (status == 1)
{
    is_on = 1;
    delay_0.1ms();

    if (is on)
        LL_GPIO_ResetOutputPin(GPIOC, LL_GPIO_PIN_8)
    else
        LL_GPIO_SetOutputPin(GPIOC, LL_GPIO_PIN_8);
}

// Modern realisation
// check register cond
status = LL_GPIO_IsInputPinSet(GPIOA, LL_GPIO_PIN_0);

if (status)
{
    counter++;    // counter starting for sure
    delay_10ms();
}

if (counter >= 5)
{
    LL_GPIO_ResetOutputPin(GPIOC, LL_GPIO_PIN_8);

    while(1)
        Show();    // function for output smth
}

else
    LL_GPIO_SetOutputPin(GPIOC, LL_GPIO_PIN_8);
```



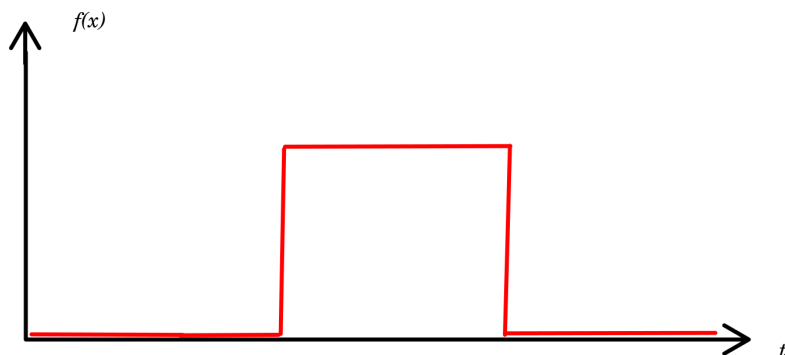


Рис. 5 Идеальный график сигнала

### 3 Семисегментный индикатор

#### 3.1 Принцип работы

Семисегментный светодиодный индикатор — устройство отображения цифровой информации. Это — наиболее простая реализация индикатора, который может отображать арабские цифры. Для отображения букв используются более сложные многосегментные и матричные индикаторы.

Семисегментный светодиодный индикатор, как говорит его название, состоит из семи элементов индикации (сегментов), включающихся и выключающихся по отдельности. Включая их в разных комбинациях, из них можно составить упрощённые изображения арабских цифр.

Сегменты обозначаются буквами от А до G; восьмой сегмент — десятичная точка (decimal point, DP), предназначенная для отображения дробных чисел.

Устройство имеет 10 выводов, центральный вывод в каждом ряду — это общий анод/катод в зависимости от типа индикатора. Остальные вы-

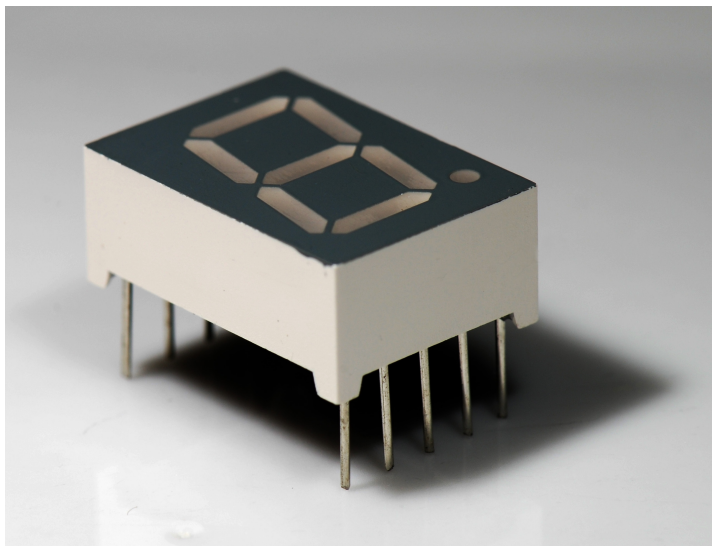


Рис. 6 Фото индикатора

воды подключаются непосредственно к каждому из сегментов a,b,c,d,e,f,g,dp.

При подключении индикатора к микроконтроллеру чем, что на каждый сегмент необходимо последовательно подключить резистор номиналом 200 – 300 Ом.

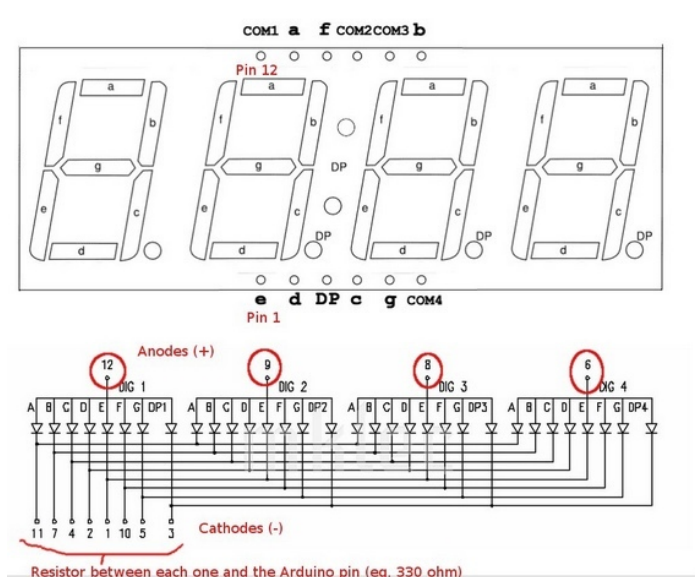


Рис. 7 Описание устройства

## 4 Программная реализация нашего проекта на языке C с применением интегрированной среды разработки IAR for ARM

### 4.1 Бегущая строка

```
#include "libs/stm32f0xx_ll_rcc.h"
#include "libs/stm32f0xx_ll_system.h"
#include "libs/stm32f0xx_ll_bus.h"
#include "libs/stm32f0xx_ll_gpio.h"
```

```
void delay()
{
```

```
    for (int i = 0; i < 600000; ++i)
    {}
    return;
}

void delay_10ms()
{
    for (int i = 0; i < 6000; ++i)
    {}
    return;
}

uint16_t sgmnts[7] =
{
    LL_GPIO_PIN_0, // E - segment 0
    LL_GPIO_PIN_1, // D - segment 1
    LL_GPIO_PIN_3, // C - segment 2
    LL_GPIO_PIN_4, // G - segment 3
    LL_GPIO_PIN_6, // B - segment 4
    LL_GPIO_PIN_9, // F - segment 5
    LL_GPIO_PIN_10, // A - segment 6
};

uint16_t Show_digit(uint32_t digit)
{
    switch (digit)
    {
        case 0: return (sgmnts[0] | sgmnts[1] | sgmnts[2] |
                        sgmnts[4] | sgmnts[6] | sgmnts[5]);

        case 1: return (sgmnts[4] | sgmnts[2]);

        case 2: return (sgmnts[6] | sgmnts[4] | sgmnts[3] |
                        sgmnts[0] | sgmnts[1]);
```

```
case 3: return (sgmnts[4] | sgmnts[2] | sgmnts[6] |
               sgmnts[3] | sgmnts[1]);

case 4: return (sgmnts[4] | sgmnts[2] | sgmnts[5] |
               sgmnts[3]);

case 5: return (sgmnts[6] | sgmnts[2] | sgmnts[5] |
               sgmnts[3] | sgmnts[1]);

case 6: return (sgmnts[6] | sgmnts[5] | sgmnts[0] |
               sgmnts[3] | sgmnts[2] | sgmnts[1]);

case 7: return (sgmnts[4] | sgmnts[2] | sgmnts[6]);

case 8: return (sgmnts[1] | sgmnts[2] | sgmnts[3] |
               sgmnts[4] | sgmnts[5] | sgmnts[6] |
               sgmnts[0]);

case 9: return (sgmnts[1] | sgmnts[2] | sgmnts[3] |
               sgmnts[4] | sgmnts[5] | sgmnts[6]);

default: return 0; //disable all segments

}

}

void dyn_display(uint16_t num)
{
    while(1)
    {
```

```
uint16_t result[4] =
{
    [0] = Show_digit(num % 10) | 0x0980,
    [1] = Show_digit((num / 10) % 10) | 0x0920,
    [2] = Show_digit((num / 100) % 10) | 0x08A0,
    [3] = Show_digit((num / 1000) % 10) | 0x01A0,

};

static int digit_num = 0;

LL_GPIO_WriteOutputPort(GPIOB, result[digit_num]);
delay();

digit_num = (digit_num + 1) % 4;
}
return;
}

uint16_t Hello_people(uint8_t symbol)
{
    switch(symbol)
    {
        case 'H': return sgmnts[5] | sgmnts[0] | sgmnts[3]
                        | sgmnts[4] | sgmnts[2];

        case 'E': return sgmnts[6] | sgmnts[3] | sgmnts[1]
                        | sgmnts[0] | sgmnts[5];

        case 'L': return sgmnts[0] | sgmnts[1] | sgmnts[5];

        case 'O': return (sgmnts[0] | sgmnts[1] | sgmnts[2] |
                        sgmnts[4] | sgmnts[6] | sgmnts[5]);
    }
}
```

```
case ',': return LL_GPIO_PIN_2;

case 'P': return sgmnets[6] | sgmnets[3] | sgmnets[4]
           | sgmnets[0] | sgmnets[5];

case ' ': return 0x0000;
}
}

void Show_str( uint32_t cnt )
{
    static int i = 0;

    uint8_t str[] = "HELLO PEOPLE ";

    uint16_t scr_pos[] = { 0x01A0, 0x08A0, 0x0920, 0x0980};
    uint16_t size = sizeof(str) / sizeof(char) - 1;

    uint8_t sym_pos = (cnt + i) % size;

    uint16_t pos_mask = scr_pos[i];

    LL_GPIO_WriteOutputPort(GPIOB, Hello_people(str[sym_pos]) | pos_mask);

    i = (i + 1) % 4;
}

void gpio_config()
{
    LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOA);
    // enable tact port
    LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOC);
```

```
LL_GPIO_SetPinMode(GPIOC, LL_GPIO_PIN_8, LL_GPIO_MODE_OUTPUT);
LL_GPIO_SetPinMode(GPIOC, LL_GPIO_PIN_0, LL_GPIO_MODE_INPUT);
//enable digital port

LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOB);

LL_GPIO_SetPinMode(GPIOB, LL_GPIO_PIN_0, LL_GPIO_MODE_OUTPUT);
LL_GPIO_SetPinMode(GPIOB, LL_GPIO_PIN_1, LL_GPIO_MODE_OUTPUT);
LL_GPIO_SetPinMode(GPIOB, LL_GPIO_PIN_2, LL_GPIO_MODE_OUTPUT);
LL_GPIO_SetPinMode(GPIOB, LL_GPIO_PIN_3, LL_GPIO_MODE_OUTPUT);
LL_GPIO_SetPinMode(GPIOB, LL_GPIO_PIN_4, LL_GPIO_MODE_OUTPUT);
LL_GPIO_SetPinMode(GPIOB, LL_GPIO_PIN_5, LL_GPIO_MODE_OUTPUT);
LL_GPIO_SetPinMode(GPIOB, LL_GPIO_PIN_6, LL_GPIO_MODE_OUTPUT);
LL_GPIO_SetPinMode(GPIOB, LL_GPIO_PIN_7, LL_GPIO_MODE_OUTPUT);
LL_GPIO_SetPinMode(GPIOB, LL_GPIO_PIN_8, LL_GPIO_MODE_OUTPUT);
LL_GPIO_SetPinMode(GPIOB, LL_GPIO_PIN_9, LL_GPIO_MODE_OUTPUT);
LL_GPIO_SetPinMode(GPIOB, LL_GPIO_PIN_10, LL_GPIO_MODE_OUTPUT);
LL_GPIO_SetPinMode(GPIOB, LL_GPIO_PIN_11, LL_GPIO_MODE_OUTPUT);

return;
}

void Show()
{
    static uint32_t n = 0;

    for (int i = 0; i < 20000; ++i)
        Show_str(n);

    n++;
}

int main()
{
    uint32_t status = 0, on_ = 0;
```



```
uint32_t counter = 0;
uint32_t buf[1] = {0};

gpio_config();

while (1)
{
    status = LL_GPIO_IsInputPinSet(GPIOA, LL_GPIO_PIN_0);
    // check register IDR cond

    if (status)
    {
        counter++;
        delay_10ms();
    }

    if (counter >= 5)
    {
        LL_GPIO_ResetOutputPin(GPIOC, LL_GPIO_PIN_8);

#ifdef 0
        LL_GPIO_WriteOutputPort(GPIOB, Hello_people('L'));
#endif

        while(1)
            Show();
    }

    else
        LL_GPIO_SetOutputPin(GPIOC, LL_GPIO_PIN_8);
}

return 0;
```

}

## 4.2 Счетчик нажатий кнопки

Рассмотрим написанную нами программу. Для начала требуется подключить необходимые заголовочные файлы:

```
#include "stm32f0xx_ll_rcc.h"
#include "stm32f0xx_ll_system.h"
#include "stm32f0xx_ll_bus.h"
#include "stm32f0xx_ll_gpio.h"
```

Также для работы потребуются функции, организующие задержки. Наиболее простой вариант - реализовать простой холостой цикл. Главное отключить оптимизации компилятора, во избежание выкидывания им этого цикла.

```
void delay( void )
{
    for (int i = 0; i < 600000; i++);
}
```

```
void delay10( void )
{
    for (int i = 0; i < 6000; i++);
}
```

Напишем функцию, реализующую конфигурацию пинов GPIO:

```
void gpio_config( void )
{
    LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOA);
    LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOC);

    LL_GPIO_SetPinMode(GPIOC, LL_GPIO_PIN_9, LL_GPIO_MODE_OUTPUT);
    LL_GPIO_SetPinMode(GPIOC, LL_GPIO_PIN_8, LL_GPIO_MODE_OUTPUT);

    // GPIOB init
    LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOB);
```

```
LL_GPIO_SetPinMode(GPIOB, LL_GPIO_PIN_0, LL_GPIO_MODE_OUTPUT);
LL_GPIO_SetPinMode(GPIOB, LL_GPIO_PIN_1, LL_GPIO_MODE_OUTPUT);
LL_GPIO_SetPinMode(GPIOB, LL_GPIO_PIN_2, LL_GPIO_MODE_OUTPUT);
LL_GPIO_SetPinMode(GPIOB, LL_GPIO_PIN_3, LL_GPIO_MODE_OUTPUT);
LL_GPIO_SetPinMode(GPIOB, LL_GPIO_PIN_4, LL_GPIO_MODE_OUTPUT);
LL_GPIO_SetPinMode(GPIOB, LL_GPIO_PIN_5, LL_GPIO_MODE_OUTPUT);
LL_GPIO_SetPinMode(GPIOB, LL_GPIO_PIN_6, LL_GPIO_MODE_OUTPUT);
LL_GPIO_SetPinMode(GPIOB, LL_GPIO_PIN_7, LL_GPIO_MODE_OUTPUT);
LL_GPIO_SetPinMode(GPIOB, LL_GPIO_PIN_8, LL_GPIO_MODE_OUTPUT);
LL_GPIO_SetPinMode(GPIOB, LL_GPIO_PIN_9, LL_GPIO_MODE_OUTPUT);
LL_GPIO_SetPinMode(GPIOB, LL_GPIO_PIN_10, LL_GPIO_MODE_OUTPUT);
LL_GPIO_SetPinMode(GPIOB, LL_GPIO_PIN_11, LL_GPIO_MODE_OUTPUT);
}
```

Для отображения определённой цифры на семисегментном индикаторе, нам потребуется битовая маска. Для её вычисления напомним специальную функцию:

```
uint16_t show_digit( uint32_t digit )
{
    uint16_t s[] = {
        0x0400, // segment A - 0
        0x0040, // segment B - 1
        0x0008, // segment C - 2
        0x0002, // segment D - 3
        0x0001, // segment E - 4
        0x0200, // segment F - 5
        0x0010, // segment G - 6
    };

    switch (digit)
    {
    case 0:
        return s[0] | s[1] | s[2] | s[3] | s[4] | s[5];
    case 1:
        return s[1] | s[2];
    case 2:
```

```
    return s[0] | s[1] | s[6] | s[4] | s[3];
case 3:
    return s[0] | s[1] | s[2] | s[3] | s[6];
case 4:
    return s[1] | s[2] | s[5] | s[6];
case 5:
    return s[0] | s[2] | s[3] | s[5] | s[6];
case 6:
    return s[0] | s[2] | s[3] | s[4] | s[5] | s[6];
case 7:
    return s[0] | s[1] | s[2];
case 8:
    return s[0] | s[1] | s[2] | s[3] | s[4] | s[5] | s[6];
case 9:
    return s[0] | s[1] | s[2] | s[3] | s[5] | s[6];
default:
    return 0;
}
}
```

Перейдём к написанию главной функции. Она будет осуществлять вывод четырёхзначного числа на экран семисегментного индикатора. Так как семисегментный индикатор может вывести только одно число за раз, то вывод четырёх цифр будет происходить путём многократного последовательного вывода всех цифр числа.

```
void show_number( uint32_t num )
{
    uint16_t res[4] =
    {
        [0] = show_digit(num % 10) | 0x0980,
        [1] = show_digit(num / 10 % 10) | 0x0920,
        [2] = show_digit(num / 100 % 10) | 0x08A0,
        [3] = show_digit(num / 1000 % 10) | 0x01A0,
    };

    static int i = 0;
```

```
LL_GPIO_WriteOutputPort(GPIOB, res[i]);

i = (i + 1) % 4;
}
```

Главная функция программы с реализацией антидребезга:

```
int main( void )
{
    gpio_config();
    uint32_t counter = 0, is_pressed = 0, is_on = 0;
    uint32_t n = 0;

    while (1)
    {
        // 1 - on, 0 - off
        /* Check if button pressed */
        if (LL_GPIO_IsInputPinSet(GPIOA, LL_GPIO_PIN_0))
        {
            is_pressed = 1;
            counter = 0; /* reset counter */
        }

        if (is_pressed)
        {
            counter++;
            delay10();
        }

        if (counter >= 5)
        {
            if (is_on)
            {
                LL_GPIO_ResetOutputPin(GPIOC, LL_GPIO_PIN_9);
                LL_GPIO_SetOutputPin(GPIOC, LL_GPIO_PIN_8);
            }
        }
    }
}
```

```
else
{
    LL_GPIO_SetOutputPin(GPIOC, LL_GPIO_PIN_9);
    LL_GPIO_ResetOutputPin(GPIOC, LL_GPIO_PIN_8);
}
// change state
is_on = 1 - is_on;

//LL_GPIO_WriteOutputPort(GPIOB, 0x0000);
++;

is_pressed = 0;
counter = 0;
}

show_number(n);
}
}
```