



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «РАДИОЭЛЕКТРОНИКА И ЛАЗЕРНАЯ ТЕХНИКА»
КАФЕДРА «ТЕХНОЛОГИИ ПРИБОРОСТРОЕНИЯ» (РЛ6)

ОТЧЕТ

по домашнему заданию №2

на тему «Микроконтроллер stm32 (GPIOy, TIMx, FLASH, USART, DMA)»

и по лабораторной работе №2

по дисциплине «Цифровые устройства и микропроцессоры»

Студент: Шатовкин Константин Романович
(фамилия, имя, отчество)

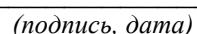

(подпись, дата)

Группа: РЛ6-81

Преподаватель:

Доцент кафедры РЛ6
(должность)

Семеренко Д.А.
(фамилия и.о.)


(подпись, дата)

Оценка: _____

2024 г.

Оглавление

1 Оборудование	4
1.1 stm32f051	4
1.2 Дополнительное оборудование	5
2 Задание по вариантам и условие ЛР№2	6
2.1 Вариант 1 – Цель.....	6
2.2 Вариант 1 – Условия.....	6
Параллельный интерфейс (чтение)	6
Последовательный интерфейс (запись)	6
2.3 Вариант 2 – Цель.....	6
2.4 Вариант 2 – Условия.....	6
Параллельный интерфейс (запись).....	6
Последовательный интерфейс (чтение)	6
2.5 Общие замечания	7
2.6 Условие лабораторной работы №2	8
3 Выполнение.....	9
3.1 Предисловие	9
3.2 Создание проекта	9
3.3 Код.....	17
stm32f0xx.h.....	17
buffer.h	21
buffer.c	22
FLASH.h	23
FLASH.c	23
Variables.h	25
Variables.c.....	25
Button_init.h.....	26
Button_init.c.....	26
Button_cmd.h	27
Button_cmd.c	28
Periphery_for_transfer_and_receive_init.h.....	31
Periphery_for_transfer_and_receive_init.c.....	31
USART_DMA_setup.h.....	33
USART_DMA_setup.c	33
Serial_receive_and_parallel_transfer.h.....	35

Serial_receive_and_parallel_transfer.c	36
main.c	42
3.4 Сборка и компиляция	42
3.5 Обновление драйверов платы.....	46
3.6 Выполнение 2-го варианта перед отладкой	48
3.7 Отладка	50
STM32CubeProgrammer	50
STM32CubeMonitor	52
Какая из программ оказалась лучше для отладки	59

1 Оборудование.

Перед началом работы необходимо определиться с оборудованием, которое будет использоваться. По условию, выполнение работы допустимо с применением микроконтроллеров stm32f0 и stm32f4. В данной работе я буду использовать 2 платы stm32f051 с микроконтроллерами stm32f051R8T6.

1.1 stm32f051

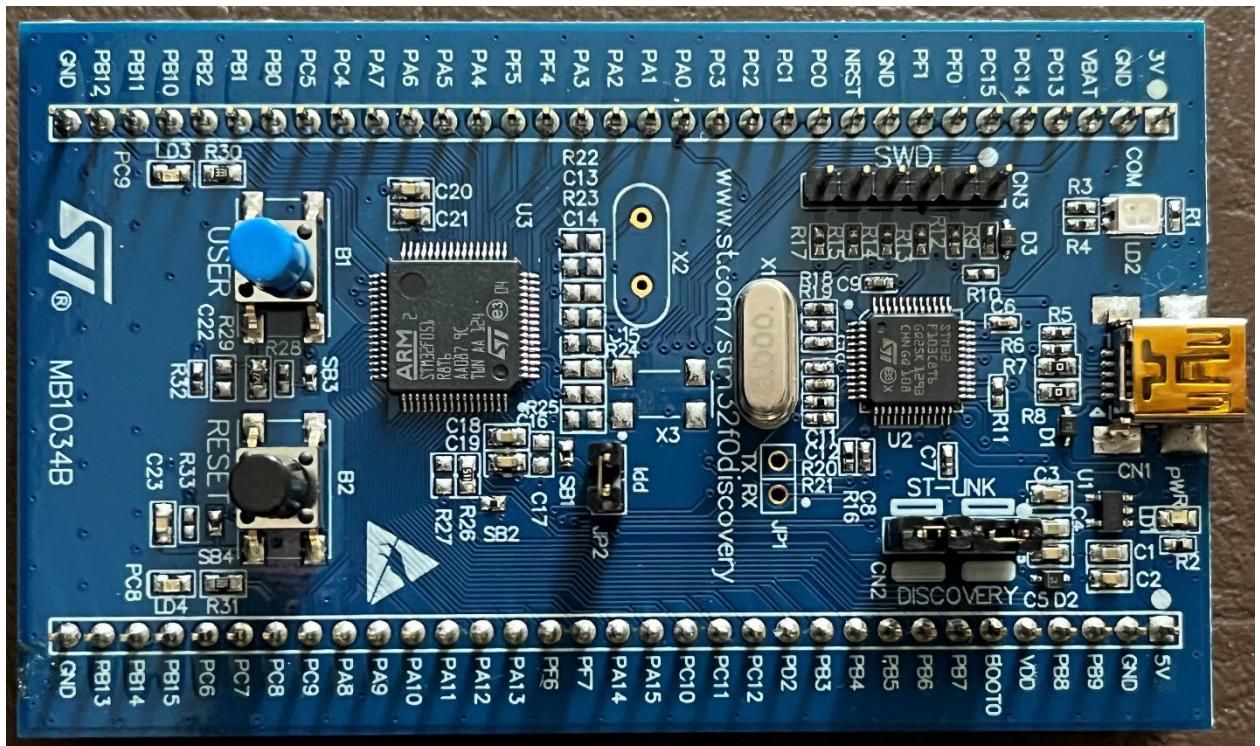


Рисунок 1.1 – stm32f051

Figure 1. Block diagram

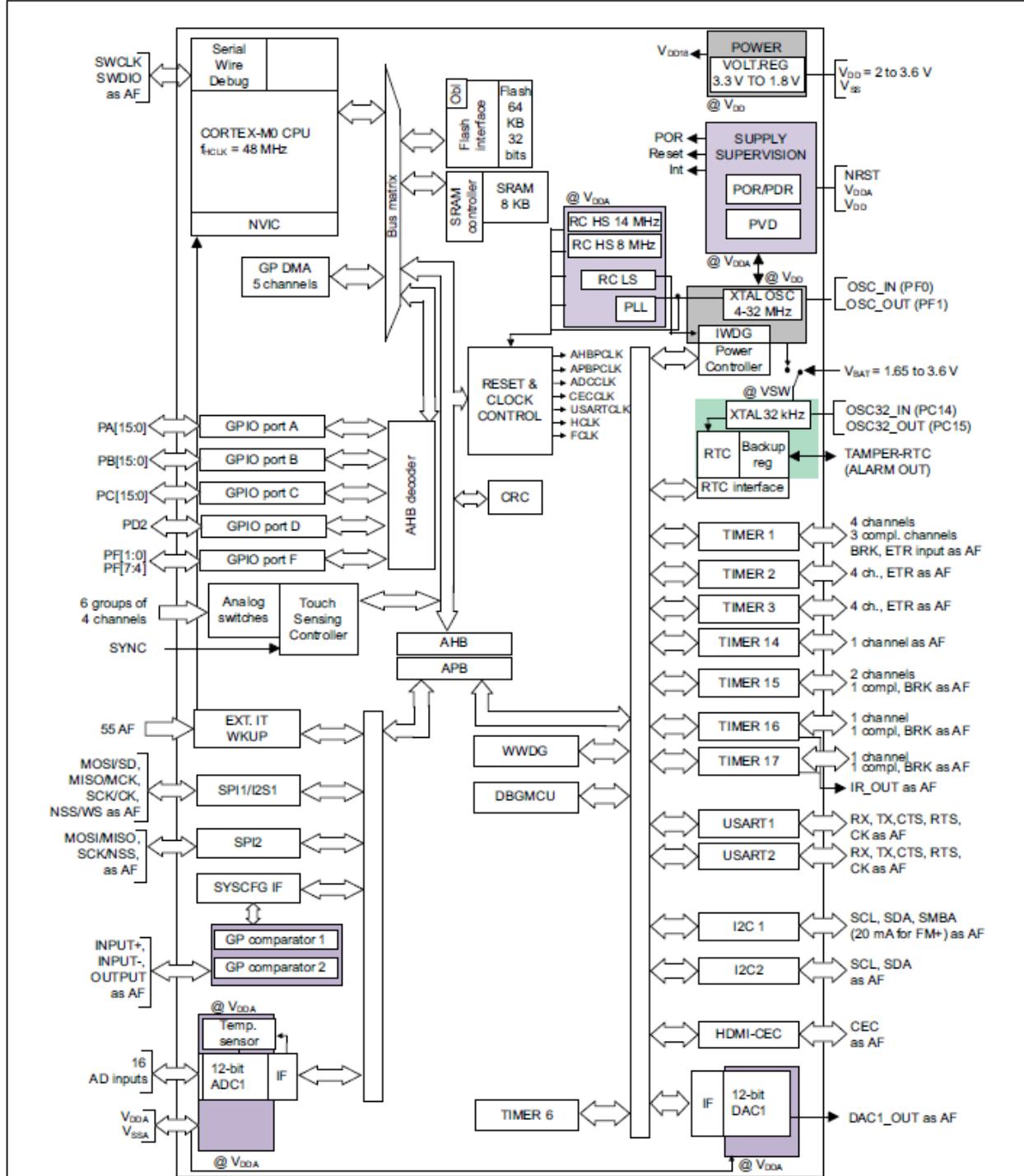


Рисунок 1.2 – Схема чипа

1.2 Дополнительное оборудование

Платы соединялись между собой 13-ю проводами мама-мама, подключались к ПК 2-мя мультимедийными кабелями USB2.0 А вилка – Mini USB вилка.

2 Задание по вариантам и условие ЛР№2.

2.1 Вариант 1 – Цель

Выполнить передачу данных по последовательному интерфейсу, полученных по параллельному интерфейсу.

2.2 Вариант 1 – Условия

Параллельный интерфейс (чтение)

Данные по параллельному интерфейсу могут быть считаны после установки лог 1 на выводе микроконтроллера «en», после считывания необходимо установить лог 1 на выводе «ready» после этого сигнал «en» устанавливается в ноль. Временные диаграммы показаны на рисунках 2.1 и 2.2.

Последовательный интерфейс (запись)

На рисунке 2.3 показана схема подключения сигналов МК. Частота передачи данных по последовательному интерфейсу определяется установкой соответствующих логических сигналов на входы контроллера f0 … f2. Частота передачи данных меняется дискретно от 100Гц до 10кГц (шаг дискретизации выбирается самостоятельно). Предусмотреть возникновение ситуации получения данных по параллельному интерфейсу быстрее, чем их передача по последовательному.

2.3 Вариант 2 – Цель

Выполнить передачу данных по последовательному интерфейсу, полученных по параллельному интерфейсу.

2.4 Вариант 2 – Условия

Параллельный интерфейс (запись)

После считывания данных по последовательному интерфейсу их необходимо передать по параллельному интерфейсу. После установления данных на выводах микроконтроллера (параллельный интерфейс) устанавливается лог 1 на выводе «en». Этот логический сигнал устанавливается в лог 0, после того как данные были считаны, т. е. после перехода уровня сигнала «ready» из лог 1 в лог 0.

Последовательный интерфейс (чтение)

Считывание данных осуществляется после установления сигнала «d_send» в лог 0. Как показано на рисунках 2.1 и 2.2. Длительность лог 0 и лог 1 определяется входами f0 … f2.

На рисунке 2.3 показана схема подключения сигналов МК. Частота передачи данных по последовательному интерфейсу определяется установкой соответствующих логических сигналов на входы контроллера f0… f2. Частота передачи данных меняется дискретно от 100Гц до 10кГц (шаг дискретизации выбирается

самостоятельно). Предусмотреть возникновение ситуации получения данных по последовательному интерфейсу быстрее, чем их считывание по параллельному.

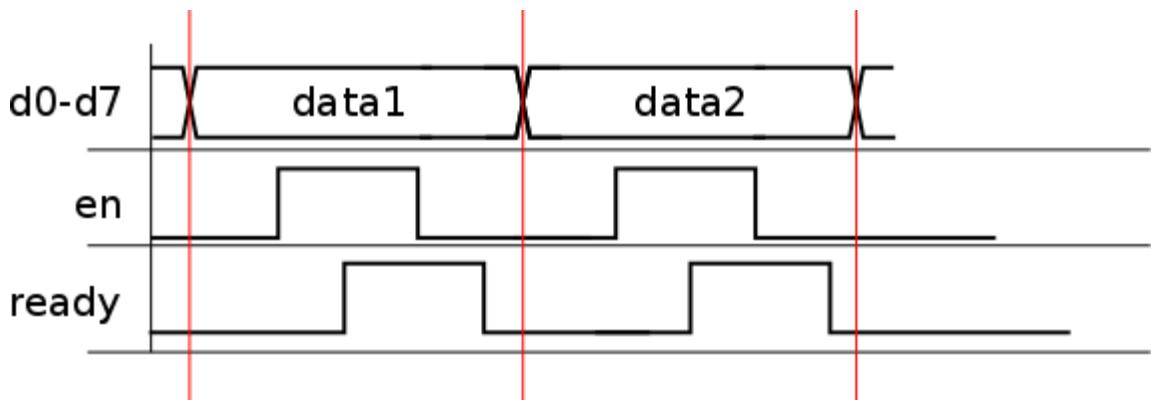


Рисунок 2.1 – Временные диаграммы установления сигналов на входах микроконтроллера

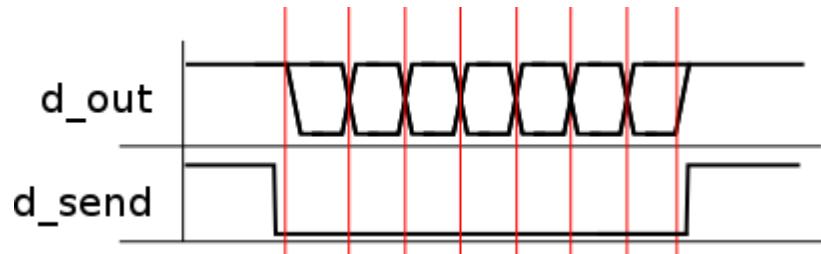


Рисунок 2.2 – Временные диаграммы установления сигналов на выходах микроконтроллера

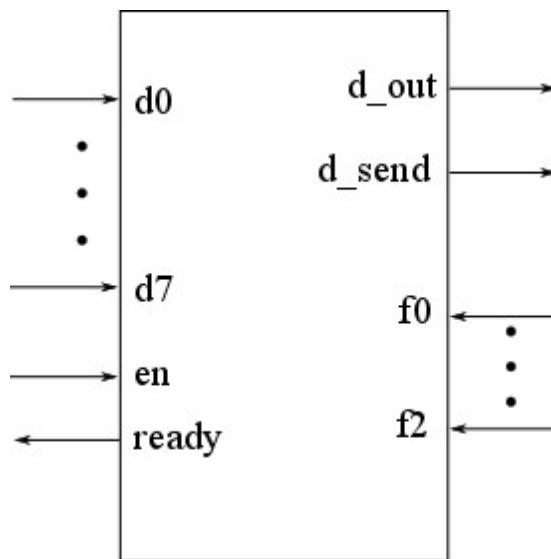


Рисунок 2.3 – Схема выводов микроконтроллера

2.5 Общие замечания

1. Данные передаются по нажатию кнопки.
2. Данные передаются пакетом по 1024 байта, которые хранятся во FLASH памяти микроконтроллера.
3. Запись данных выполняется тоже в область FLASH памяти, при этом они не перезаписывают данные, предназначенные для передачи данных.

4. При длительном нажатии кнопки происходит вход в режим системных настроек (как вариант), после длительного нажатия начинает мигать светодиод, после чего необходимо кратковременным нажатием на кнопку выполнить команду:
одно нажатие – запись данных во FLASH;
два нажатия – включить/выключить светодиод на втором МК;
три нажатия – установить частоту микроконтроллера.

5. Обновление данных для тестирования работы программы выполняется по последовательной передаче данных USART.

2.6 Условие лабораторной работы №2

Настроить USART для реализации получения данных платой с ПК и передачи данных платой на ПК.

3 Выполнение.

3.1 Предисловие

Для выполнения задания будем использовать линейку программ STM32Cube (IDE, Programmer, Monitor), а также STM32 ST-LINK Utility. Код будем писать сразу для обоих вариантов, однако, чтобы он работал, необходимо будет закомментировать противоположный вариант, иначе будет выдаваться ошибка о повторе используемых функций.

3.2 Создание проекта

Создание проекта и написание кода будет происходить в STM32CubeIDE. Для более быстрой проверки работы кода на платах в будущем создадим сразу 2 проекта в 2-х разных директориях (сейчас – один, после этого – второй) с идентичным кодом, в которых будут различаться лишь закомментированные части.

Итак, запускаем программу. Сразу после запуска нам предложат выбрать директорию для работы. Если директория будет “чужой” (т.е. файловое расположение, заложенное в ранее созданный проект, будет отличаться от того, которое он имеет сейчас), то сборщик проекта не будет видеть конечной цели и будет отказываться работать. Он даже ошибки высвечивать не будет.

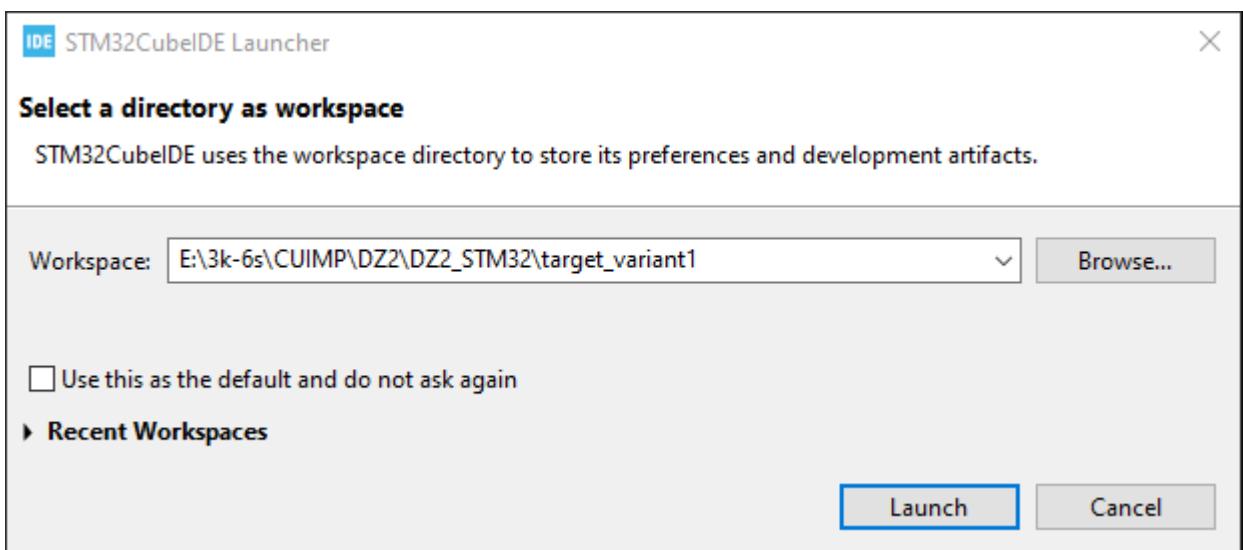


Рисунок 3.1 – Выбор директории для работы

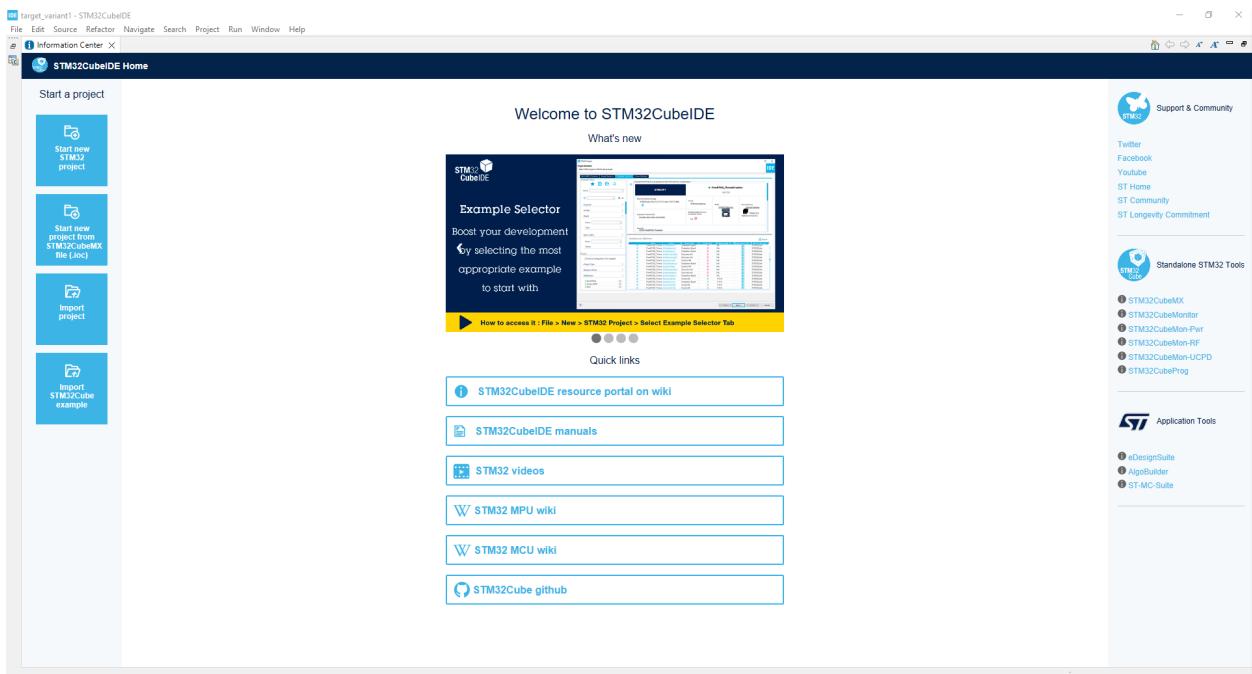


Рисунок 3.2 – Окно после выбора директории запуска STM32CubeIDE

Нажимаем File --> New --> STM32 Project. Если всплывает иконка вопроса с настройкой параметров интернет-соединения, то нажимаем No.

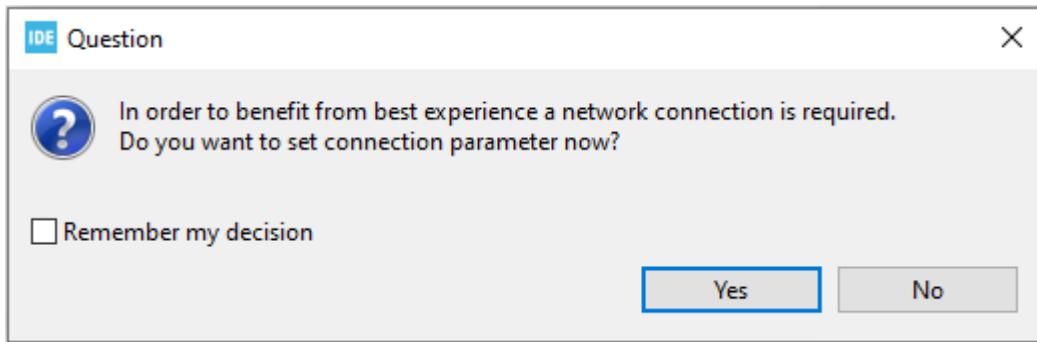


Рисунок 3.3 – Иконка с вопросом

Во всплывшем окне слева сверху выбираем Board Selector и выбираем свою (или максимально идентичную) плату.

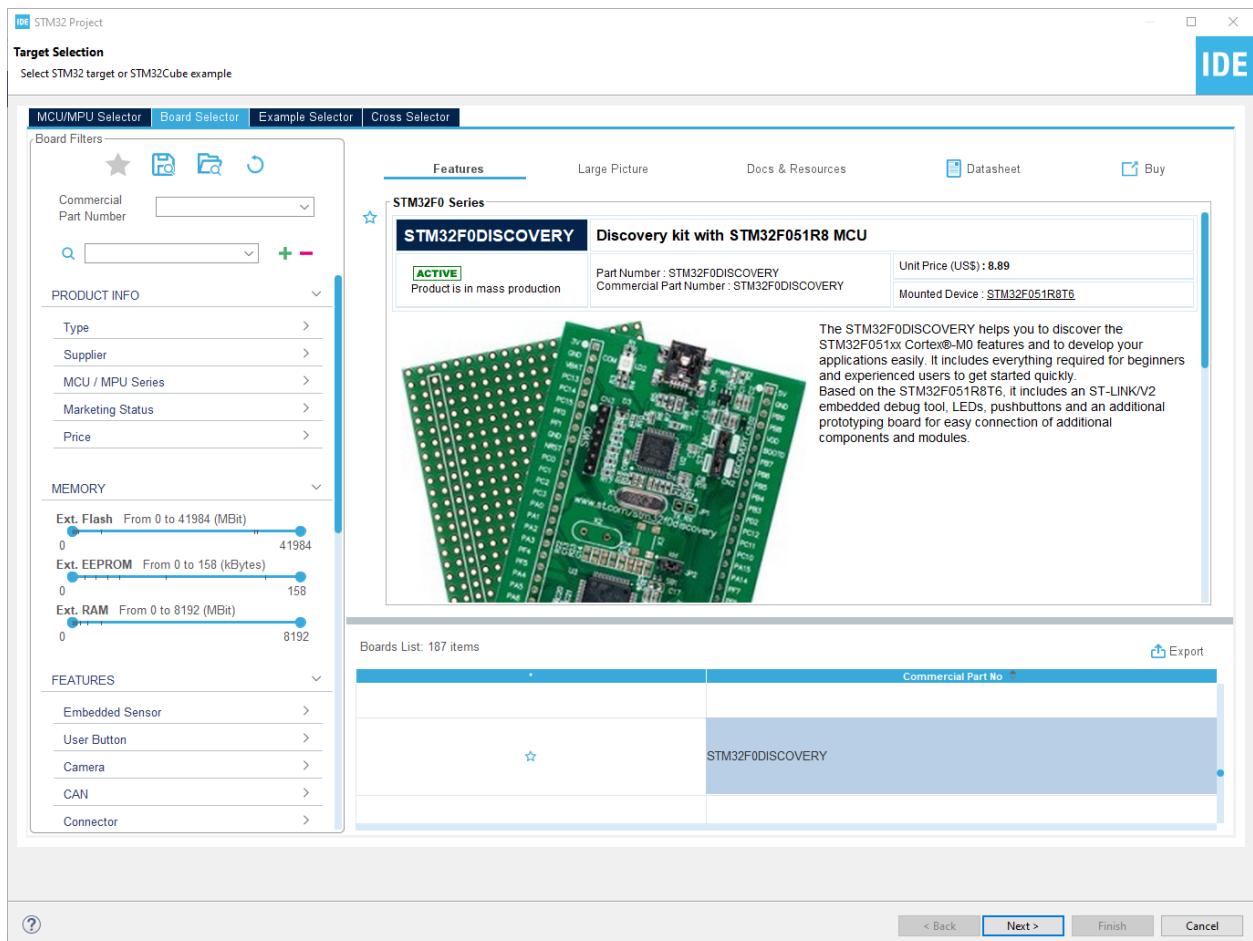


Рисунок 3.4 – Выбор платы во всплывшем окне

В следующем всплывшем окне набираем название нашего проекта, переключаем галочку в Targeted Project Type на Empty и жмём Finish.

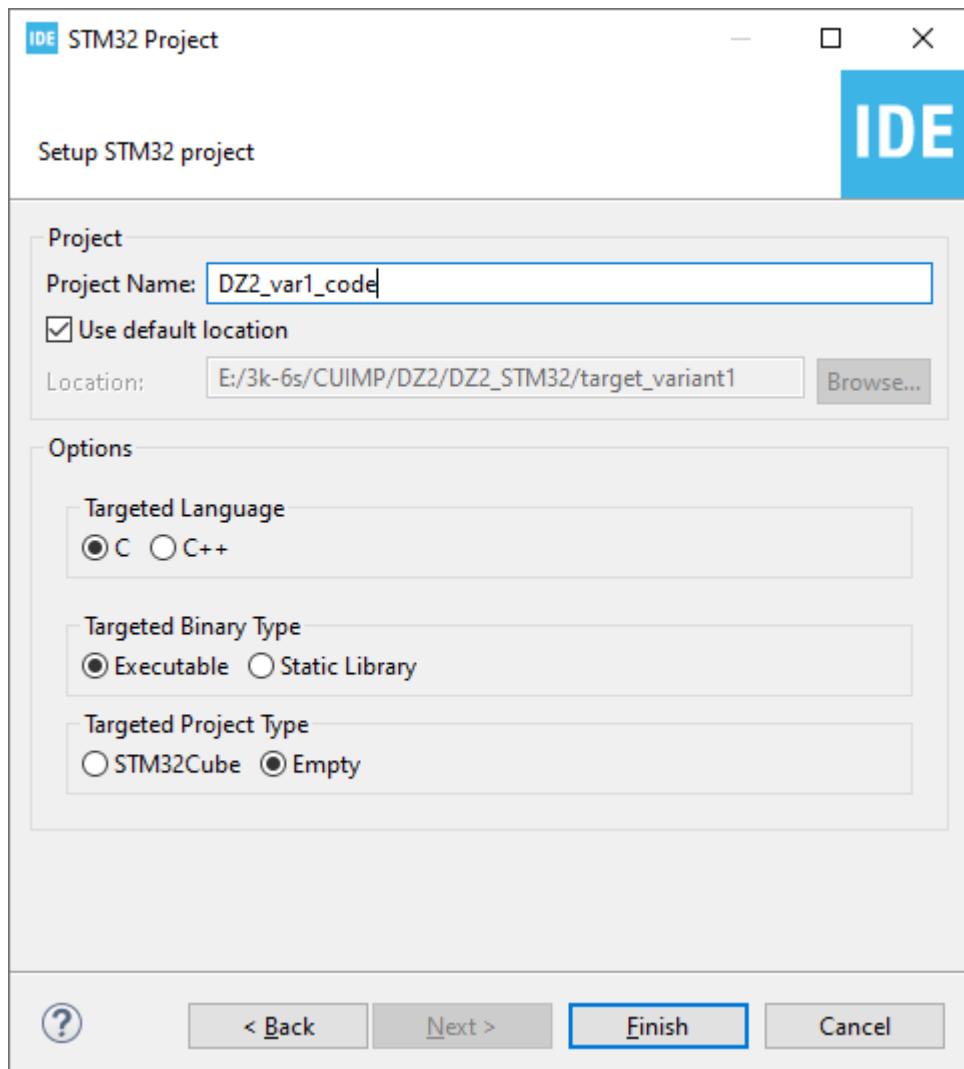


Рисунок 3.5 – Называние проекта и смена галочки в новом всплывшем окне

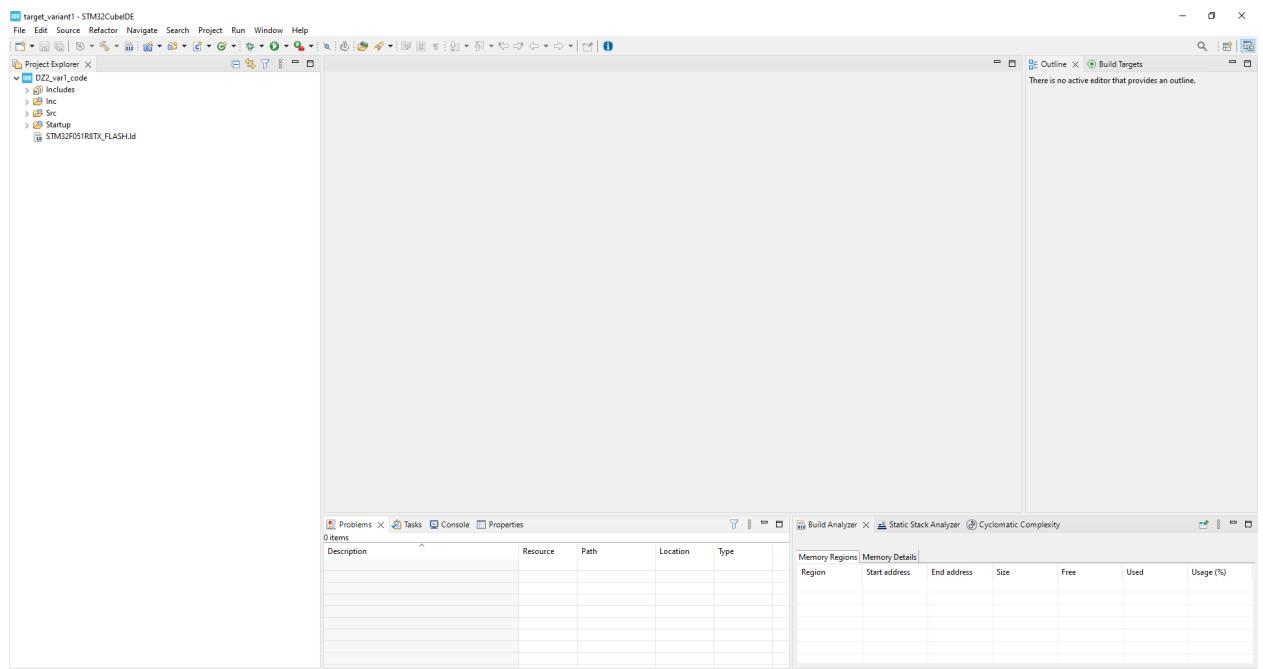
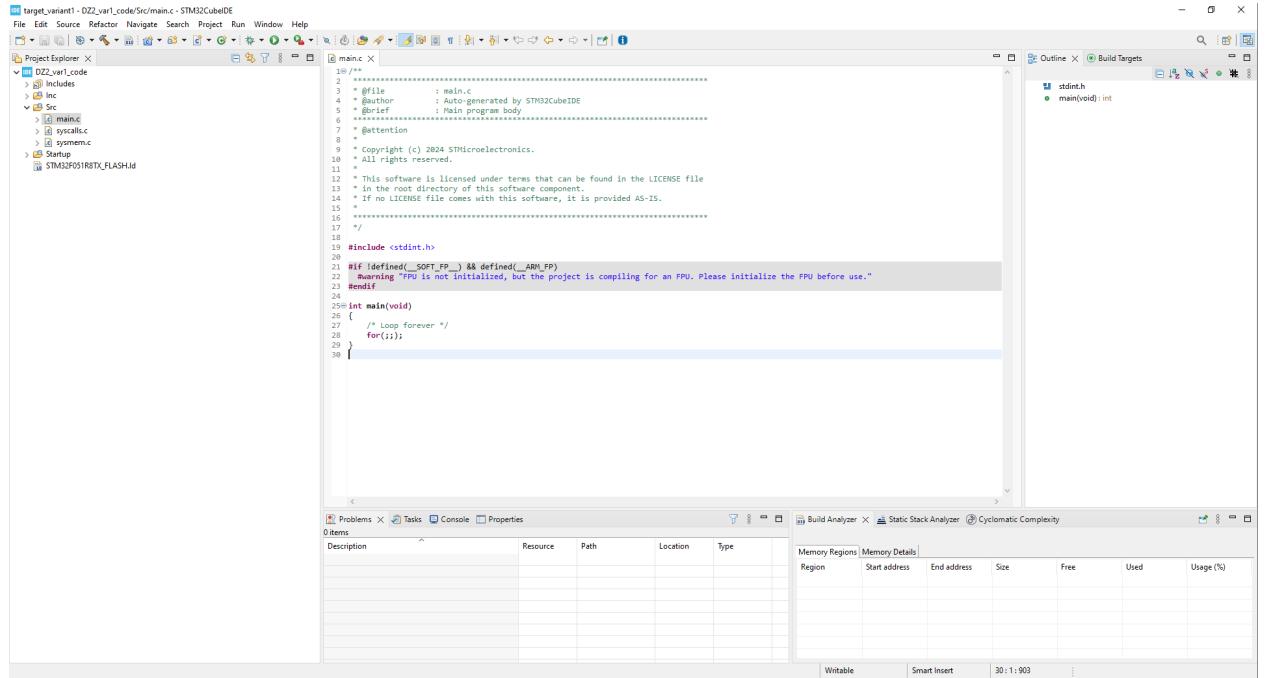


Рисунок 3.6 – Окно после создания проекта

Все скрипты, которые мы будем писать, должны находиться в папке Src. Заходим в неё и нажимаем 2 раза ЛКМ по main.c.



The screenshot shows the STM32CubeIDE interface. The Project Explorer on the left lists a project named 'target_variant1 - D22_var1_code' with a 'Src' folder containing 'main.c', 'syscalls.c', and 'systemc'. The main window displays the content of 'main.c'. The code starts with a header guard, copyright notice, and a warning about the FPU. It includes the `<cstdlib.h>` header and defines a `main(void)` function that loops forever. The bottom part of the interface shows the Problems, Tasks, Console, Properties, Build Analyzer, Static Stack Analyzer, and Cyclomatic Complexity tabs.

```
2
3 * @file      : main.c
4 * @author    : Auto generated by STM32CubeIDE
5 * @brief     : Main program body
6 ****
7 * @attention
8 *
9 * Copyright (c) 2024 STMicroelectronics.
10 * All rights reserved.
11 *
12 * This software is licensed under terms that can be found in the LICENSE file
13 * in the root directory of this software component.
14 * If no LICENSE file comes with this software, it is provided AS-IS.
15 *
16 ****
17 */
18
19 #include <cstdlib.h>
20
21 #if defined(__SOFT_FPU__) && defined(__ARM_FPU__)
22 #warning "FPU is not initialized, but the project is compiling for an FPU. Please initialize the FPU before use."
23#endif
24
25int main(void)
26{
27    /* Loop forever */
28    for(;;)
29}
```

Рисунок 3.7 – Открытие файла main.c

Далее заходим в Project --> Properties. Теперь в C/C++ Build --> Settings --> Tool Settings --> MCU Post build outputs и ставим галочку напротив Convert to Intel Hex file (-O ihex). Нажимаем Apply.

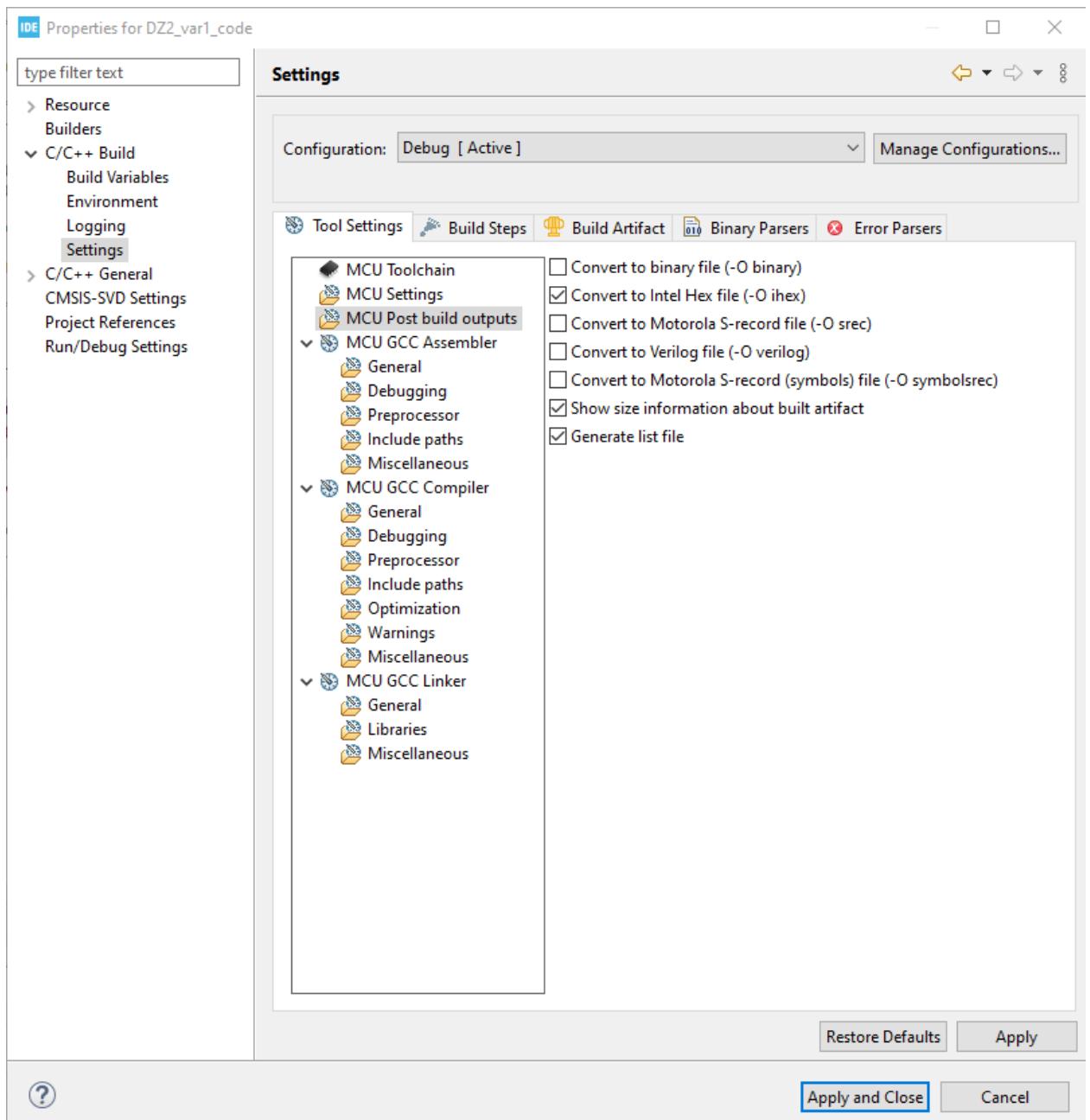


Рисунок 3.8 – Ставим галочку там, где нужно

Далее заходим в C/C++ Build --> Settings --> Tool Settings --> MCU Settings и ставим там две галочки снизу (если они не проставлены). Нажимаем Apply.

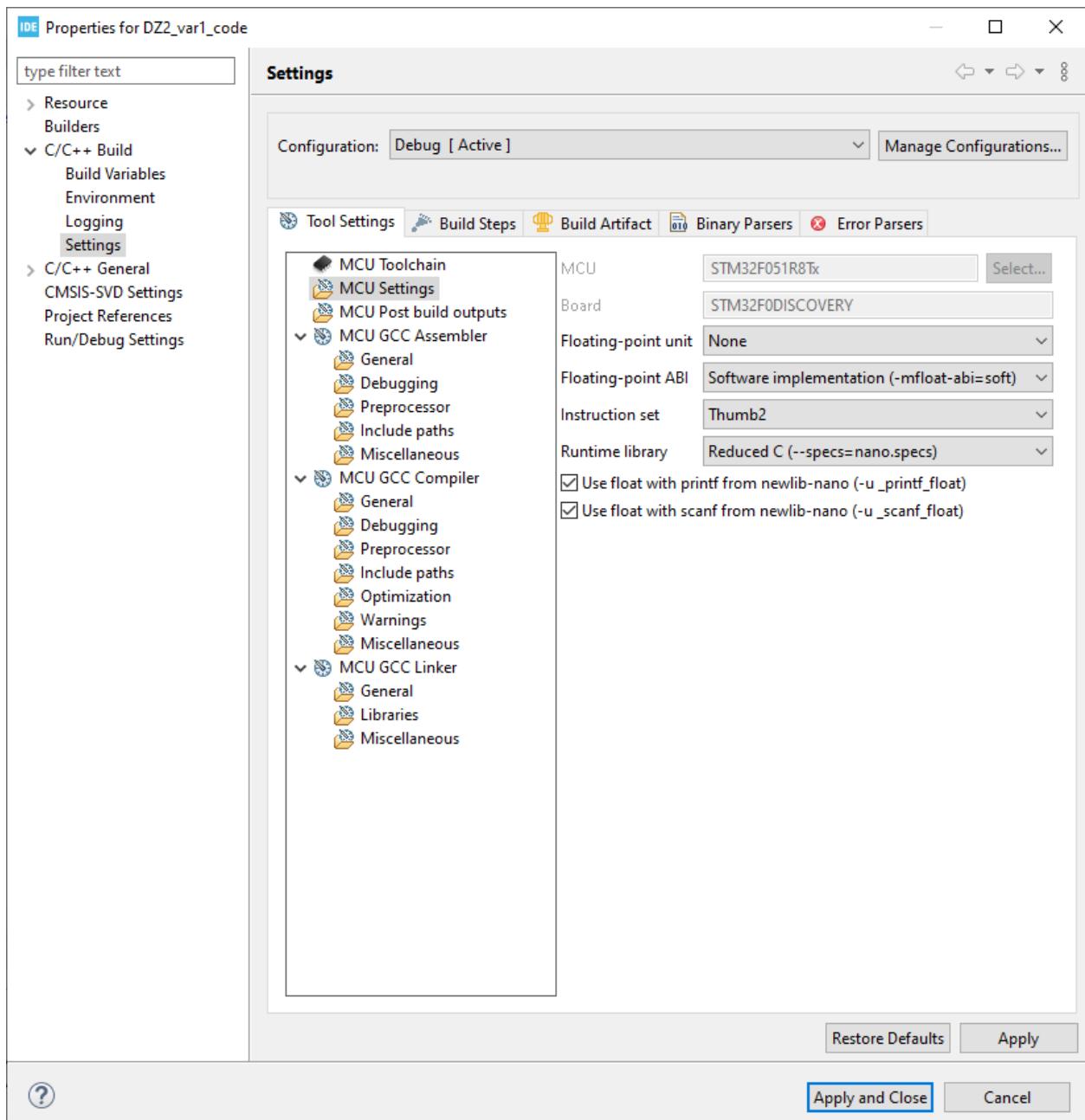


Рисунок 3.9 – Ставим две галочки там, где нужно

Далее переходим по пути C/C++ Build --> Settings --> MCU GCC Compiler --> Miscellaneous и напротив Other flags нажимаем на лист бумаги с зелёным плюсиком. В появившемся окне вводим -fcommon. Если этого не сделать, то компилятор будет выдавать кучу ошибок с multiple definition (якобы одна и та же переменная объявляется в двух разных объектных файлах). Нажимаем Apply.

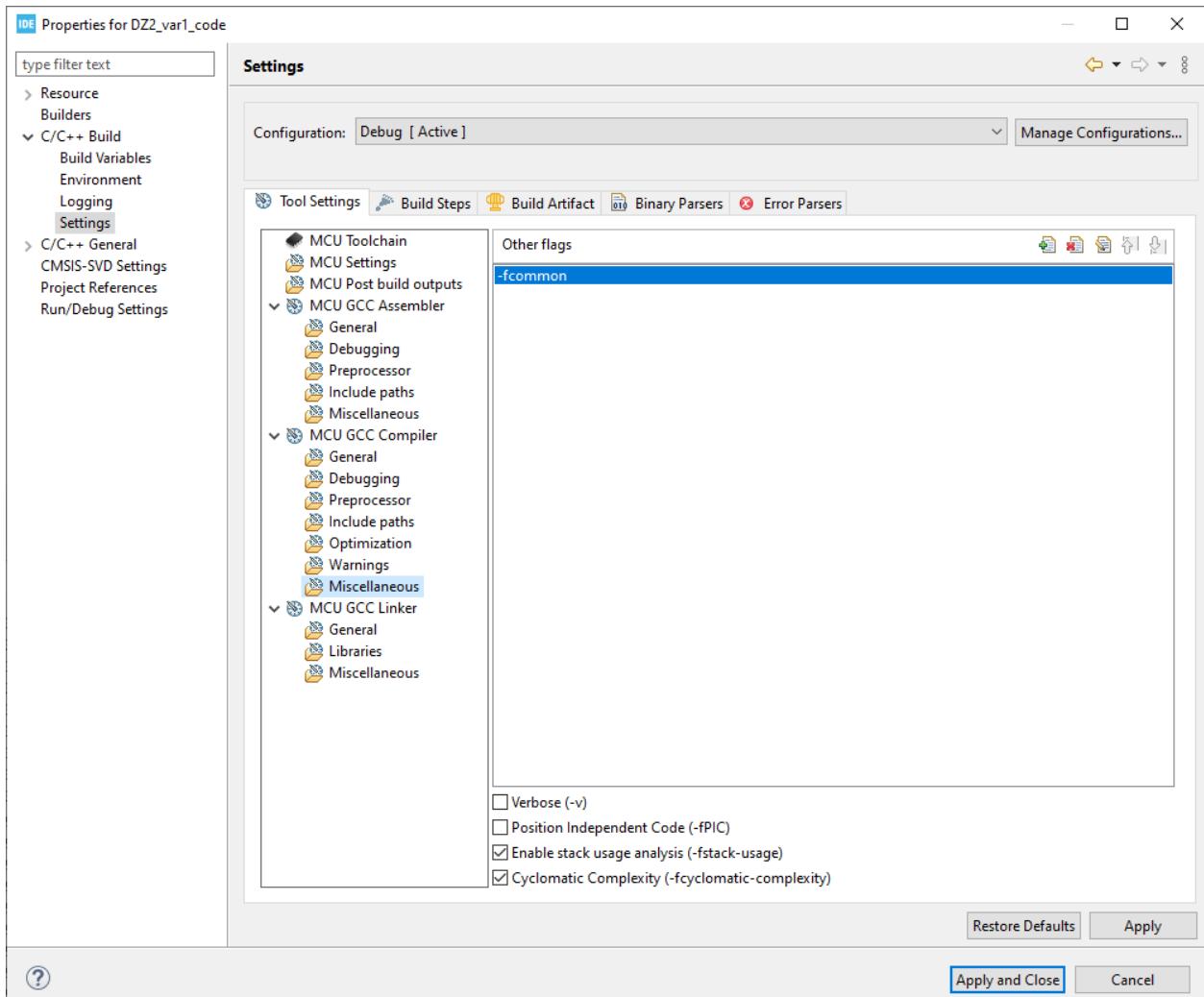


Рисунок 3.10 – Введение очень важной команды куда надо

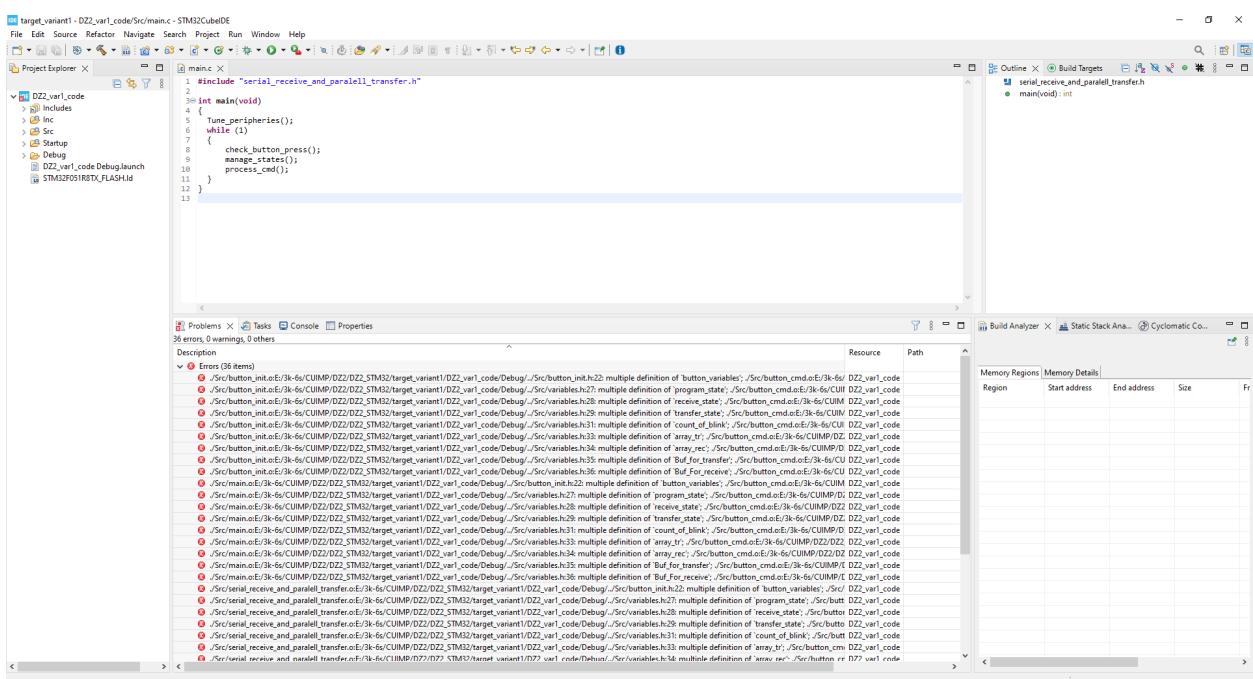


Рисунок 3.11 – Вот что будет, если комманду не ввести – 36 ошибок после сборки

Далее нажимаем Apply and Close.

3.3 Код

Приступаем к написанию кода.

Отдельно подчеркну, что без 8 скриптов проект компилироваться не будет. Это: "stm32f0xx.h", "stm32f051x8.h" (или другой скрипт с похожим названием, в зависимости от вашей платы), "core_cm0.h", "core_cmFunc.h", "core_cmInstr.h", "cmsis_gcc.h", "system_stm32f0xx.h", "system_stm32f0xx.c". Я взял их из общего доступа. Они должны находиться в папке Src вместе с другими скриптами (которые будут написаны уже нами). Здесь приведу лишь 1 скрипт "stm32f0xx.h", в который я внёс правки, раскомментировав include с названием своей платы.

Замечу, что если в файловой системе (н-р, Проводник или Total Commander) перенести откуда-то файлы скриптов в папку Src вашего проекта, то в открытой программе STM32CubeIDE нужно просто в Project Explorer (слева сверху) нажать ПКМ по названию вашего проекта, а затем нажать на Refresh. Тогда добавленные скрипты станут видны и в программе.

stm32f0xx.h

```
/**  
*****  
* @file      stm32f0xx.h  
* @author    MCD Application Team  
* @version   V2.2.3  
* @date     29-January-2016  
* @brief    CMSIS STM32F0xx Device Peripheral Access Layer Header File.  
*  
*          The file is the unique include file that the application programmer  
*          is using in the C source code, usually in main.c. This file contains:  
*          - Configuration section that allows to select:  
*              - The STM32F0xx device used in the target application  
*              - To use or not the peripheral's drivers in application code(i.e.  
*                  code will be based on direct access to peripheral's registers  
*                  rather than drivers API), this option is controlled by  
*                  "#define USE_HAL_DRIVER"  
*  
*****  
* @attention  
*  
* <h2><center>&copy; COPYRIGHT(c) 2016 STMicroelectronics</center></h2>  
*  
* Redistribution and use in source and binary forms, with or without modification,  
* are permitted provided that the following conditions are met:  
*   1. Redistributions of source code must retain the above copyright notice,  
*      this list of conditions and the following disclaimer.  
*   2. Redistributions in binary form must reproduce the above copyright notice,  
*      this list of conditions and the following disclaimer in the documentation  
*      and/or other materials provided with the distribution.  
*   3. Neither the name of STMicroelectronics nor the names of its contributors  
*      may be used to endorse or promote products derived from this software  
*      without specific prior written permission.  
*  
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"  
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE  
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
```

```

* DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE
* FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
* DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
* SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
* CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
***** */
*/
/** @addtogroup CMSIS
* @{
*/
/** @addtogroup stm32f0xx
* @{
*/
#ifndef __STM32F0xx_H
#define __STM32F0xx_H

#ifdef __cplusplus
extern "C" {
#endif /* __cplusplus */

/** @addtogroup Library_configuration_section
* @{
*/
/*
* @brief STM32 Family
*/
#if !defined (STM32F0)
#define STM32F0
#endif /* STM32F0 */

/* Uncomment the line below according to the target STM32 device used in your
application
*/
#if !defined (STM32F030x6) && !defined (STM32F030x8) &&
!defined (STM32F031x6) && !defined (STM32F038xx) &&
!defined (STM32F042x6) && !defined (STM32F048xx) && !defined (STM32F070x6) &&
!defined (STM32F051x8) && !defined (STM32F058xx) &&
!defined (STM32F071xB) && !defined (STM32F072xB) && !defined (STM32F078xx) &&
!defined (STM32F070xB) && \
    !defined (STM32F091xC) && !defined (STM32F098xx) && !defined (STM32F030xC)
/* #define STM32F030x6 */ /*!< STM32F030x4, STM32F030x6 Devices (STM32F030xx microcontrollers where the Flash memory ranges between 16 and 32 Kbytes) */
/* #define STM32F030x8 */ /*!< STM32F030x8 Devices (STM32F030xx microcontrollers where the Flash memory is 64 Kbytes) */
/* #define STM32F031x6 */ /*!< STM32F031x4, STM32F031x6 Devices (STM32F031xx microcontrollers where the Flash memory ranges between 16 and 32 Kbytes) */
/* #define STM32F038xx */ /*!< STM32F038xx Devices (STM32F038xx microcontrollers where the Flash memory is 32 Kbytes) */
/* #define STM32F042x6 */ /*!< STM32F042x4, STM32F042x6 Devices (STM32F042xx microcontrollers where the Flash memory ranges between 16 and 32 Kbytes) */
*/

```

```

/* #define STM32F048x6 */ /*!< STM32F048xx Devices (STM32F042xx microcontrollers
where the Flash memory is 32 Kbytes) */
#define STM32F051x8 /*!< STM32F051x4, STM32F051x6, STM32F051x8 Devices
(STM32F051xx microcontrollers where the Flash memory ranges between 16 and 64 Kbytes)
*/
/* #define STM32F058xx */ /*!< STM32F058xx Devices (STM32F058xx microcontrollers
where the Flash memory is 64 Kbytes) */
/* #define STM32F070x6 */ /*!< STM32F070x6 Devices (STM32F070x6 microcontrollers
where the Flash memory ranges between 16 and 32 Kbytes) */
/* #define STM32F070xB */ /*!< STM32F070xB Devices (STM32F070xB microcontrollers
where the Flash memory ranges between 64 and 128 Kbytes) */
/* #define STM32F071xB */ /*!< STM32F071x8, STM32F071xB Devices (STM32F071xx mi-
crocontrollers where the Flash memory ranges between 64 and 128 Kbytes)
*/
/* #define STM32F072xB */ /*!< STM32F072x8, STM32F072xB Devices (STM32F072xx mi-
crocontrollers where the Flash memory ranges between 64 and 128 Kbytes)
*/
/* #define STM32F078xx */ /*!< STM32F078xx Devices (STM32F078xx microcontrollers
where the Flash memory is 128 Kbytes) */
/* #define STM32F030xC */ /*!< STM32F030xC Devices (STM32F030xC microcontrollers
where the Flash memory is 256 Kbytes) */
/* #define STM32F091xC */ /*!< STM32F091xB, STM32F091xC Devices (STM32F091xx mi-
crocontrollers where the Flash memory ranges between 128 and 256 Kbytes)
*/
/* #define STM32F098xx */ /*!< STM32F098xx Devices (STM32F098xx microcontrollers
where the Flash memory is 256 Kbytes) */
#endif

/* Tip: To avoid modifying this file each time you need to switch between these
devices, you can define the device in your toolchain compiler preprocessor.
*/
#ifndef !defined (USE_HAL_DRIVER)
/**
 * @brief Comment the line below if you will not use the peripherals drivers.
In this case, these drivers will not be included and the application code will
be based on direct access to peripherals registers
*/
/*#define USE_HAL_DRIVER */
#endif /* USE_HAL_DRIVER */

/**
 * @brief CMSIS Device version number V2.2.3
*/
#define __STM32F0_DEVICE_VERSION_MAIN      (0x02) /*!< [31:24] main version */
#define __STM32F0_DEVICE_VERSION_SUB1     (0x02) /*!< [23:16] sub1 version */
#define __STM32F0_DEVICE_VERSION_SUB2     (0x03) /*!< [15:8]  sub2 version */
#define __STM32F0_DEVICE_VERSION_RC       (0x00) /*!< [7:0]   release candidate */
#define __STM32F0_DEVICE_VERSION          (((__STM32F0_DEVICE_VERSION_MAIN << 24) \
| (__STM32F0_DEVICE_VERSION_SUB1 << 16) \
| (__STM32F0_DEVICE_VERSION_SUB2 << 8 ) \
| (__STM32F0_DEVICE_VERSION_RC))

/**
 * @}
*/
/** @addtogroup Device_Included
 * @{
*/
#if defined(STM32F030x6)

```

```

#include "stm32f030x6.h"
#elif defined(STM32F030x8)
    #include "stm32f030x8.h"
#elif defined(STM32F031x6)
    #include "stm32f031x6.h"
#elif defined(STM32F038xx)
    #include "stm32f038xx.h"
#elif defined(STM32F042x6)
    #include "stm32f042x6.h"
#elif defined(STM32F048xx)
    #include "stm32f048xx.h"
#elif defined(STM32F051x8)
    #include "stm32f051x8.h"
#elif defined(STM32F058xx)
    #include "stm32f058xx.h"
#elif defined(STM32F070x6)
    #include "stm32f070x6.h"
#elif defined(STM32F070xB)
    #include "stm32f070xb.h"
#elif defined(STM32F071xB)
    #include "stm32f071xb.h"
#elif defined(STM32F072xB)
    #include "stm32f072xb.h"
#elif defined(STM32F078xx)
    #include "stm32f078xx.h"
#elif defined(STM32F091xC)
    #include "stm32f091xc.h"
#elif defined(STM32F098xx)
    #include "stm32f098xx.h"
#elif defined(STM32F030xC)
    #include "stm32f030xc.h"
#else
    #error "Please select first the target STM32F0xx device used in your application (in stm32f0xx.h file)"
#endif

/**
 * @}
 */

/** @addtogroup Exported_types
 * @{
 */
typedef enum
{
    RESET = 0,
    SET = !RESET
} FlagStatus, ITStatus;

typedef enum
{
    DISABLE = 0,
    ENABLE = !DISABLE
} FunctionalState;
#define IS_FUNCTIONAL_STATE(STATE) (((STATE) == DISABLE) || ((STATE) == ENABLE))

typedef enum
{
    ERROR = 0,
    SUCCESS = !ERROR
} ErrorStatus;

```

```

/**
 * @}
 */

/** @addtogroup Exported_macros
 * @{
 */
#define SET_BIT(REG, BIT)      ((REG) |= (BIT))
#define CLEAR_BIT(REG, BIT)    ((REG) &= ~(BIT))
#define READ_BIT(REG, BIT)     ((REG) & (BIT))
#define CLEAR_REG(REG)         ((REG) = (0x0))
#define WRITE_REG(REG, VAL)   ((REG) = (VAL))
#define READ_REG(REG)          ((REG))

#define MODIFY_REG(REG, CLEARMASK, SETMASK)  WRITE_REG((REG), (((READ_REG(REG)) &
(~(CLEARMASK))) | (SETMASK)))

/**
 * @}
 */

#if defined (USE_HAL_DRIVER)
#include "stm32f0xx_hal.h"
#endif /* USE_HAL_DRIVER */

#ifndef __cplusplus
}
#endif /* __cplusplus */

#endif /* __STM32F0xx_H */
/** @}
 */

/*
 * @}
 */

***** (C) COPYRIGHT STMicroelectronics *****END OF FILE****/

```

buffer.h

Буфер предназначен для временного хранения данных, а потому используется при записи данных во FLASH, а также при их передаче на другой МК и их получении с другого МК.

```
#ifndef BUFFER_H_
#define BUFFER_H_
```

```

#include "stm32f0xx.h"

#define BUFFER_SIZE 1024           // задание значения константы размера буфера
#define DATA_SIZE 8               // задание значения константы размера данных

typedef struct RingBuf_c {
    uint8_t *buf;                // адрес, где физически хранится буфер
    uint16_t write_point;         // точка записи данных
    uint16_t read_point;          // точка считывания данных
    uint16_t buf_size;            // размер буфера
    uint16_t amount_data_in_buf;  // количество байт в буфере
} buf;

// функции
void InitBuffer(buf* buffer, uint8_t* address, uint16_t size); // инициализация буфера
void Buffer_add_to_end(buf* buffer, uint8_t data);               // добавление данных в
конец буфера, тем временем write_point++
uint8_t Buffer_get_from_front(buf* buffer);                      // получение данных из
передней части буфера, тем временем read_point++
uint8_t Buffer_empty(buf* buffer);                                // проверка, пуст ли
буфер (возвращение 1, если пуст, 0 - если нет)
uint8_t Buffer_is_full(buf* buffer);                               // проверка, заполнен
ли буфер (возвращение 1, если заполнен, 0 - если нет)
void Clear_buffer(buf* buffer);                                  // очистка буфера

#endif /* BUFFER_H_ */

```

buffer.c

```

#include "buffer.h"

void InitBuffer(buf* buffer, uint8_t* address, uint16_t size) { // инициализация бу-
фера
    buffer->buf = address; // установка адреса buf
    buffer->read_point = buffer->write_point = 0; // буфер пуст, поэтому
read_point и write_point находятся в начале
    buffer->buf_size = size; // установка размера буфера, в нашем случае это 1024
байта
    buffer->amount_data_in_buf = 0; // буфер пуст, поэтому количество байт в бу-
фере = 0
}

void Buffer_add_to_end(buf* buffer, uint8_t data) { // добавление данных
    в конец буфера, тем временем write_point++
    if(buffer->amount_data_in_buf < BUFFER_SIZE) {
        buffer->buf[buffer->write_point] = data; // запись данных до конца
        buffer->write_point++; // сдвиг write_point
        buffer->amount_data_in_buf++; // увеличение объема данных на 1

        if(buffer->write_point >= BUFFER_SIZE) // если write_point больше
BUFFER_SIZE или равна ему
            buffer->write_point = 0; // то установить
write_point в начало буфера
    }
}

uint8_t Buffer_get_from_front(buf* buffer) { // получение данных
    из передней части буфера, тем временем read_point++
    if(buffer->amount_data_in_buf != 0) { // если есть данные
        uint8_t data = buffer->buf[buffer->read_point]; // получить данные

```

```

        buffer->read_point++; // сдвиг read_point

        if(buffer->read_point >= BUFFER_SIZE) // если read_point больше
BUFFER_SIZE или равна ему
            buffer->read_point = 0; // то установить read_point в
начале буфера

        buffer->amount_data_in_buf--; // уменьшение объёма данных на 1
        return data;
    }
    return -1;
}

uint8_t Buffer_empty(buf* buffer) { // проверка, пуст ли
буфер (возвращение 1, если пуст, 0 - если нет)
    return ((buffer->read_point == buffer->write_point) && (buffer-
>amount_data_in_buf == 0));
}

uint8_t Buffer_is_full(buf* buffer) { // проверка, запол-
нен ли буфер (возвращение 1, если заполнен, 0 - если нет)
    return (buffer->amount_data_in_buf == BUFFER_SIZE);
}

void Clear_buffer(buf* buffer) { // очистка буфера
    buffer->read_point = 0;
    buffer->write_point = 0;
    buffer->amount_data_in_buf = 0;
}

```

FLASH.h

Во FLASH записывается передаваемая/получаемая с ПК и полученная с другого микроконтроллера информация.

```

#ifndef FLASH_H_
#define FLASH_H_

#define PAGE_FOR_TRANSFER (uint32_t)0x0800F800 // страница для пересылки (62)
#define PAGE_FOR_RECEIVE (uint32_t)0x0800FC00 // страница для получения (63)

#include "stm32f0xx.h"

void Set_protection_of_flash(); // установка защиты FLASH
void Flash_unlock(); // разблокировка FLASH
void Erase_Page(uint32_t Page_for_erase); // удаление страницы
void Write_data_to_flash(uint32_t Page_adress, uint8_t* data, uint16_t data_size);
// запись данных на FLASH
void ReadFromFlash(uint32_t Page_adress, uint8_t* data, uint16_t data_size); // чте-
ние данных из FLASH

#endif /* FLASH_H_ */

```

FLASH.c

```

#include "FLASH.h"

void Set_protection_of_flash() { // установка защиты FLASH
    FLASH->CR |= FLASH_CR_LOCK;
}

```

```

void Flash_unlock() { // разблокировка FLASH
    while( (FLASH->SR & FLASH_SR_BSY) == FLASH_SR_BSY); // проверка, не занята ли
FLASH
    if( (FLASH->CR & FLASH_CR_LOCK ) == FLASH_CR_LOCK) { // разблокировка FLASH,
если она была заблокирована
        FLASH->KEYR = FLASH_KEY1;
        FLASH->KEYR = FLASH_KEY2;
    }
}

void Erase_Page(uint32_t Page_for_erase) { // удаление страницы
    Flash_unlock();
    while( (FLASH->SR & FLASH_SR_BSY) == FLASH_SR_BSY); // проверка, не занята ли
FLASH
    FLASH->CR |= FLASH_CR_PER; // регистрация для удаления страницы
    FLASH->AR = Page_for_erase; // установка адреса страницы для удаления
    FLASH->CR |= FLASH_CR_STRT; // начало удаления
    // ожидание, пока страница не будет удалена
    while( (FLASH->SR & FLASH_SR_BSY) == FLASH_SR_BSY);
    // сброс бита "завершение операции"
    if( (FLASH->SR & FLASH_SR_EOP) == FLASH_SR_EOP )
        FLASH->SR |= FLASH_SR_EOP;
    FLASH->CR &= ~FLASH_CR_PER; // сброс регистрации для удаления страницы
    Set_protection_of_flash();
}

void Write_data_to_flash(uint32_t Page_adress, uint8_t* data, uint16_t data_size) {
// запись данных на FLASH
    Erase_Page(Page_adress);
    Flash_unlock();
    FLASH->CR |= FLASH_CR_PG; // установка программирующего бита
    // запись данных на FLASH-адрес, в нашем случае - 1064 байта (1 страница)
    uint16_t data_for_write = 0;
    for (int i = 0; i < data_size / 2; i++) {
        data_for_write = 0;
        data_for_write |= data[i * 2];
        data_for_write |= ((data[i * 2 + 1]) << 8);
        *(uint16_t*)(Page_adress + i*2) = data_for_write;
        while( (FLASH->SR & FLASH_SR_BSY) == FLASH_SR_BSY);
    }
    // ожидание, пока будут записаны данные
    // сброс бита "завершение операции"
    if( (FLASH->SR & FLASH_SR_EOP) == FLASH_SR_EOP )
        FLASH->SR |= FLASH_SR_EOP;
    FLASH->CR &= ~FLASH_CR_PG; // удаление программирующего бита
    Set_protection_of_flash();
}

void ReadFromFlash(uint32_t Page_adress, uint8_t* data, uint16_t data_size){ // чтение
данных из FLASH
    uint16_t data_for_read = 0;
    for(int i = 0; i < data_size / 2; i++) {
        data_for_read = *(uint16_t*)(Page_adress + i*2);
        while( (FLASH->SR & FLASH_SR_BSY) == FLASH_SR_BSY);
        data[i * 2] = data_for_read & 0x00FF;
        data[i * 2 + 1] = (data_for_read >> 8) & 0x00FF;
    }
}

```

Variables.h

В данном файле размещены переменные, необходимые для отслеживания состояния выполнения программы (т.е. флаги), ключ для мигания диодом на другом МК, а также буферы для передаваемых и получаемых данных.

```
#ifndef VARIABLES_H_
#define VARIABLES_H_

#include "buffer.h"

#define KEY_FOR_BLINK_LED ('B' + 'L' + 'I' + 'N' + 'K')

typedef enum ProgramStates {
    WAITING_FOR_ACTION,
    SETUP_MODE,
    TRANSFER_DATA,
    TRANSFER_BIT,
    TRANSFER_8_BIT,
    CATCH_SIG_READY,
    WRITE_DATA_INTO_FLASH,
    BLINK_LED_ON_OTHER_MK,
    KEY_TRANSFER,
    BLINK_MY_LED,
    RECEIVE_BIT,
    RECEIVE_8_BIT,
    RECEIVE_COMPLETE,
    PROGRAMMING,
    RECEIVE_DATA_FROM_PC,
    TRANSFER_DATA_TO_PC
} ProgramStates;

ProgramStates program_state;
ProgramStates receive_state;
ProgramStates transfer_state;

uint8_t count_of_blink; // переменная для подсчёта количества миганий светодиода при получении соответствующей команды

uint8_t array_tr[BUFFER_SIZE];
uint8_t array_rec[BUFFER_SIZE];
buf Buf_for_transfer;
buf Buf_For_receive;

#endif /* VARIABLES_H_ */
```

Variables.c

```
#include "variables.h"

ProgramStates program_state = WAITING_FOR_ACTION;
ProgramStates receive_state = WAITING_FOR_ACTION;
ProgramStates transfer_state = WAITING_FOR_ACTION;

uint8_t count_of_blink = 0; // переменная для подсчёта количества миганий светодиода при получении соответствующей команды
```

Button_init.h

В данных файлах размещены переменные для контролирования текущего состояния кнопки и нажатий, а также проинициализирована вся периферия для корректной работы кнопки.

```
#ifndef BUTTON_INIT_H_
#define BUTTON_INIT_H_

#include "variables.h"

typedef enum ButtonStates {
    WAITING,
    CHECK_AMOUNT_OF_PRESS,
    BUTTON_PRESSED,
    BUTTON_UNPRESSED,
    CONTROL_RATTLE_ON,
    CONTROL_RATTLE_OFF,
} ButtonStates;

typedef struct Variables_for_button_functioning {
    ButtonStates button_state;
    ButtonStates rattle_check;
    ButtonStates press_check;
    uint8_t count_short_press;
} Variables_for_button_functioning;

Variables_for_button_functioning button_variables;

// функции
void init_GPIO_for_Button(); // инициализация GPIOA для кнопки на PA0
void init_TIM6_for_rattle_eliminating(); // таймер задержки для устранения дребезжания кнопки
void init_TIM2_for_long_pressure_check(); // таймер для проверки длительности нажатия (нажатие 3 секунды)
void init_TIM3_for_check_result_of_press(); // Каждую секунду проверяется, была ли нажата кнопка или нет. Если была, то происходит выбор последующих действий.
void init_TIM14_for_blinkLed(); // таймер для мигания светодиода в режиме настройки
#endif /* BUTTON_INIT_H_ */
```

Button_init.c

```
#include "button_init.h"

void init_GPIO_for_Button() { // инициализация GPIOA для кнопки на PA0
    RCC->AHBENR |= RCC_AHBENR_GPIOAEN;
    RCC->AHBENR |= RCC_AHBENR_GPIOCEN;
    GPIOA->MODER &= ~GPIO_MODER_MODERO; // режим ввода на PA0
    GPIOC->MODER |= GPIO_MODER_MODER8_0 | GPIO_MODER_MODER9_0; // режим вывода
    для светодиодов PC8 и PC9
    EXTI->IMR |= EXTI_IMR_IM0; // регистрация маски прерывания
    EXTI->RTSR |= EXTI_RTSR_RT0; // выбор повышающего триггера
    EXTI->FTSR |= EXTI_FTSR_FT0; // выбор поникающего триггера
    NVIC_EnableIRQ(EXTI0_1_IRQn); // настройка NVIC для EXTI0
    NVIC_SetPriority(EXTI0_1_IRQn, 5);
}

void init_TIM6_for_rattle_eliminating () { // таймер задержки для устранения дребезжания кнопки
```

```

RCC->APB1ENR |= RCC_APB1ENR_TIM6EN;
TIM6->ARR = 4000; // настройка TIM6 на 25ms
TIM6->PSC = 50;
// включение прерывания
TIM6->DIER |= TIM_DIER_UIE;
NVIC_EnableIRQ(TIM6_DAC_IRQn);
NVIC_SetPriority(TIM6_DAC_IRQn, 5);
TIM6->CR1 |= TIM_CR1_CEN;
}

void init_TIM2_for_long_pressure_check() {
    RCC->APB1ENR |= RCC_APB1ENR_TIM2EN;
    TIM2->ARR = 12000; // настройка TIM2 на 3sec
    TIM2->PSC = 2000;
    // включение прерывания
    TIM2->DIER |= TIM_DIER_UIE;
    NVIC_EnableIRQ(TIM2_IRQn);
    NVIC_SetPriority(TIM2_IRQn, 5);
    TIM2->CR1 |= TIM_CR1_CEN;
}

void init_TIM3_for_check_result_of_press() {
    RCC->APB1ENR |= RCC_APB1ENR_TIM3EN;
    TIM3->ARR = 8000; // настройка TIM3 на 1sec
    TIM3->PSC = 1000;
    // включение прерывания
    TIM3->DIER |= TIM_DIER_UIE;
    NVIC_EnableIRQ(TIM3_IRQn);
    NVIC_SetPriority(TIM3_IRQn, 5);
    TIM3->CR1 |= TIM_CR1_CEN;
}

void init_TIM14_for_blinkLed() {
    RCC->APB1ENR |= RCC_APB1ENR_TIM14EN;
    TIM14->ARR = 4000; // настройка TIM14 на 0.5sec
    TIM14->PSC = 1000;
    // включение прерывания
    TIM14->DIER |= TIM_DIER_UIE;
    NVIC_EnableIRQ(TIM14_IRQn);
    NVIC_SetPriority(TIM14_IRQn, 5);
}

```

Button_cmd.h

В данных файлах размещены функции для осуществления процесса обработки нажатий и последующих действий, связанных с ними.

```

#ifndef BUTTON_CMD_H_
define BUTTON_CMD_H_

#include "stm32f0xx.h"
#include "button_init.h"
#include "FLASH.h"

// прерывания:
void EXTI0_1_IRQHandler(void);
void TIM6_DAC_IRQHandler(void);
void TIM2_IRQHandler(void);
void TIM3_IRQHandler(void);
void TIM14_IRQHandler(void);

```

```

// функции
void Timers_turn_on(); // включить все таймеры для работы кнопки
void Timers_turn_off(); // выключить все таймеры для работы кнопки
void Button_reset_to_be_ready_for_work(); // перезапуск переменных у кнопки по порядку для получения нового нажатия
void Button_tune(); // содержит все функции для инициализации
void check_button_press(); // каждую секунду проверяет, как много нажатий было выполнено
// включение или выключение светодиодов (только для визуального контроля программы)
void Turn_on_Led_PC8();
void Turn_off_Led_PC8();
void Turn_on_Led_PC9();
void Turn_off_Led_PC9();

#endif /* BUTTON_CMD_H_ */

```

Button_cmd.c

```

#include <button_cmd.h>

void EXTI0_1_IRQHandler(void) {
    if( (EXTI->PR & EXTI_PR_PIF0) == EXTI_PR_PIF0) {
        EXTI->PR |= EXTI_PR_PIF0;
        // если проверка для RATTLE или program_state = PROGRAMMING -> return
        if(program_state == PROGRAMMING || button_variables.rattle_check == CONTROL_RATTLE_ON)
            return;
        // если кнопка не нажата, то установить CONTROL_RATTLE_ON для устранения дребезга кнопки
        if(button_variables.button_state == BUTTON_UNPRESSED) {
            button_variables.button_state = BUTTON_PRESSED; // кнопка нажата
            button_variables.rattle_check = CONTROL_RATTLE_ON; // контроль за дребезгом кнопки включен
            button_variables.count_short_press++; // увеличение количества нажатий
            Timers_turn_on();
            return;
        }
        if(button_variables.button_state == BUTTON_PRESSED) { // если кнопка была не нажата
            button_variables.button_state = BUTTON_UNPRESSED; // то это был спадающий фронт прерывания
            button_variables.rattle_check = CONTROL_RATTLE_ON; // контроль за дребезгом кнопки включен
            Timers_turn_on();
        }
    }
}

void TIM6_DAC_IRQHandler(void) {
    TIM6->SR &= ~TIM_SR UIF;
    TIM6->CR1 &= ~TIM_CR1_CEN;
    button_variables.rattle_check = CONTROL_RATTLE_OFF; // если таймер досчитал до 25 мс, то отключить проверку дребезга
}

void TIM2_IRQHandler(void) {
    TIM2->SR &= ~TIM_SR UIF;
    TIM2->CR1 &= ~TIM_CR1_CEN;
}

```

```

//если таймер досчитал до 3 секунд, то установить состояние длительного нажа-
тия
    if(button_variables.button_state == BUTTON_PRESSED) {
        button_variables.count_short_press = 0; // сброс значения до нуля, чтобы
        подсчитать количество нажатий
        program_state = SETUP_MODE; // установка режима настройки
        TIM14->CR1 |= TIM_CR1_CEN; // включение таймера для мигания светодиода
    }
}

void TIM3_IRQHandler(void) {
    TIM3->SR &= ~TIM_SR UIF;
    button_variables.press_check = CHECK_AMOUNT_OF_PRESS;
}

void check_button_press() {
    if(button_variables.press_check != CHECK_AMOUNT_OF_PRESS || button_varia-
bles.count_short_press == 0)
        return;
    button_variables.press_check = WAITING;
    // если это было длительное нажатие, то выбрать одну из двух функций
    if(program_state == SETUP_MODE && button_variables.button_state == BUTTON_UN-
PRESSED) {
        switch(button_variables.count_short_press) {
        case 1:
            program_state = WRITE_DATA_INTO_FLASH;
            Turn_on_Led_PC8(); // если это было 1 короткое нажатие
            Turn_off_Led_PC9(); // то записать данные во FLASH
            Timers_turn_off();
            break;
        case 2:
            program_state = BLINK_LED_ON_OTHER_MK; // если это были 2 коротких
нажатия
            Turn_off_Led_PC8(); // запустить мигание светодиода на втором МК
            Turn_off_Led_PC9();
            Timers_turn_off();
            break;
        default:
            Turn_off_Led_PC8();
            Turn_off_Led_PC9();
            Timers_turn_off();
            Button_reset_to_be_ready_for_work(); // если было выполнено бо-
льше 2-х нажатий -> перезапустить кнопку
            break;
        }
        return;
    }
    // в противном случае проверить, нет ли коротких нажатий
    if(button_variables.button_state == BUTTON_UNPRESSED) {
        switch(button_variables.count_short_press) {
        case 1:
            program_state = TRANSFER_DATA;
            Turn_on_Led_PC8(); // если это было однократное короткое
нажатие
            Turn_on_Led_PC9(); // то передать данные
            Timers_turn_off();
            break;
        case 2:
            program_state = RECEIVE_DATA_FROM_PC; // если это были 2 ко-
ротких нажатия
            Turn_on_Led_PC8(); // то получить данные с ПК через USART
        }
    }
}

```

```

        Timers_turn_off();
        break;
    case 3:
        program_state = TRANSFER_DATA_TO_PC; // если это были 3 коротких нажатия
        Turn_on_Led_PC9(); // передать данные на ПК через USART
        Timers_turn_off();
        break;
    default:
        Timers_turn_off();
        Button_reset_to_be_ready_for_work(); // если было выполнено более 3-х нажатий -> перезапустить кнопку
        break;
    }
}
}

void TIM14_IRQHandler(void) {
    if( (TIM14->DIER & TIM_DIER_UIE) == TIM_DIER_UIE ) {
        TIM14->SR &= ~TIM_SR UIF; // таймер для мигания светодиода каждые 0,5 с
        GPIOC->ODR ^= GPIO_ODR_8; // в режиме настройки
        GPIOC->ODR ^= GPIO_ODR_9;
        if(program_state == BLINK_MY_LED)
            count_of_blink++;
    }
}

void Button_tune() {
    init_GPIO_for_Button();
    init_TIM2_for_long_pressure_check();
    init_TIM3_for_check_result_of_press();
    init_TIM6_for_rattle_eliminating();
    init_TIM14_for_blinkLed();
    Button_reset_to_be_ready_for_work();
}

void Timers_turn_on() {
    TIM2->CNT = 0;
    TIM3->CNT = 0;
    TIM6->CNT = 0;
    TIM14->CNT = 0;
    //если режим НАСТРОЙКИ уже установлен, то нет необходимости включать timer2 для проверки длительного нажатия
    if(program_state != SETUP_MODE)
        TIM2->CR1 |= TIM_CR1_CEN;
    TIM3->CR1 |= TIM_CR1_CEN;
    TIM6->CR1 |= TIM_CR1_CEN;
}

void Timers_turn_off() {
    TIM2->CR1 &= ~TIM_CR1_CEN;
    TIM3->CR1 &= ~TIM_CR1_CEN;
    TIM6->CR1 &= ~TIM_CR1_CEN;
    TIM14->CR1 &= ~TIM_CR1_CEN;
}

void Button_reset_to_be_ready_for_work() {
    button_variables.button_state = BUTTON_UNPRESSED;
    button_variables.rattle_check = CONTROL_RATTLE_OFF;
    button_variables.count_short_press = 0;
}

```

```

}

void Turn_on_Led_PC8() {
    GPIOC->ODR |= GPIO_ODR_8;
}
void Turn_off_Led_PC8() {
    GPIOC->ODR &= ~GPIO_ODR_8;
}
void Turn_on_Led_PC9(){
    GPIOC->ODR |= GPIO_ODR_9;
}
void Turn_off_Led_PC9(){
    GPIOC->ODR &= ~GPIO_ODR_9;
}

```

Periphery_for_transfer_and_receive_init.h

В данный файлах проинициализирована периферия для параллельной передачи и последовательного приёма данных (вариант 2), а также для последовательной передачи и параллельного приёма данных (вариант 1).

```

#ifndef PERIPHERY_FOR_TRANSFER_AND_RECEIVE_INIT_H_
#define PERIPHERY_FOR_TRANSFER_AND_RECEIVE_INIT_H_

#include "stm32f0xx.h"

//#define PAR_TRANSFER_AND_SERIAL_RECEIVE // Вариант 2
#define SER_TRANSFER_AND_PAR_RECEIVE // Вариант 1

/*
#endif PAR_TRANSFER_AND_SERIAL_RECEIVE
// PA1 - d_send, PA2 - enable, PA3 - ready, PA4 - clock, PA5 - serial data, PB1-PB8 -
par data
void init_GPIO_for_transfer_and_receive_data();
#endif
*/
#endif SER_TRANSFER_AND_PAR_RECEIVE
// PA1 - d_send, PA2 - enable, PA3 - ready, PA4 - clock, PA5 - serial data, PB1-PB8 -
par data
void init_GPIO_for_transfer_and_receive_data();
// инициализация timer15 для генерации тактового сигнала
void init_TIM15_for_clock_sig();
#endif

#endif /* PERIPHERY_FOR_TRANSFER_AND_RECEIVE_INIT_H_ */

```

Periphery_for_transfer_and_receive_init.c

```

#include "Periphery_for_transfer_and_receive_init.h"
/*
#endif PAR_TRANSFER_AND_SERIAL_RECEIVE
void init_GPIO_for_transfer_and_receive_data() {
    // RCC включен
    RCC->AHBENR |= RCC_AHBENR_GPIOAEN | RCC_AHBENR_GPIOBEN;
    RCC->APB2ENR |= RCC_APB2ENR_SYSCFGCOMPEN;
    // включение внешнего прерывания для PA3-PA5
    SYSCFG->EXTICR[0] |= SYSCFG_EXTICR1_EXTI1_PA | SYSCFG_EXTICR1_EXTI3_PA;
    SYSCFG->EXTICR[1] |= SYSCFG_EXTICR2_EXTI4_PA | SYSCFG_EXTICR2_EXTI5_PA;
    // PB1-PB8 переведены в режим вывода
}

```

```

        GPIOB->MODER |= GPIO_MODER_MODER1_0 | GPIO_MODER_MODER2_0 |
        GPIO_MODER_MODER3_0 | GPIO_MODER_MODER4_0 |
                    GPIO_MODER_MODER5_0 | GPIO_MODER_MODER6_0 |
        GPIO_MODER_MODER7_0 | GPIO_MODER_MODER8_0;
        // PA1 переведён в режим ввода в качестве сигнала d_send от другого МК
        GPIOA->MODER &= ~GPIO_MODER_MODER1;
        // PA2 переведён в режим вывода в качестве сигнала en на другой МК
        GPIOA->MODER |= GPIO_MODER_MODER2_0;
        // PA3 переведён в режим ввода по мере готовности сигнала от другого МК
        GPIOA->MODER &= ~GPIO_MODER_MODER3;
        // PA4 для приёма тактовых импульсов от другого МК
        GPIOA->MODER &= ~GPIO_MODER_MODER4;
        // PA5 для приёма последовательных данных
        GPIOA->MODER &= ~GPIO_MODER_MODER5;

        // прерывание включено для Px2, Px3 и Px4
        EXTI->IMR |= EXTI_IMR_IM2 | EXTI_IMR_IM3 | EXTI_IMR_IM4;
        // снижающийся фронт для Px3 и Px4
        EXTI->FTSR |= EXTI_FTSR_FT3 | EXTI_FTSR_FT4;

        NVIC_EnableIRQ(EXTI2_3_IRQHandler);
        NVIC_SetPriority(EXTI2_3_IRQHandler, 9);
        NVIC_EnableIRQ(EXTI4_15_IRQHandler);
        NVIC_SetPriority(EXTI4_15_IRQHandler, 1);
    }
#endif
*/
#endif SER_TRANSFER_AND_PAR_RECEIVE
void init_GPIO_for_transfer_and_receive_data() {
    // RCC включен
    RCC->AHBENR |= RCC_AHBENR_GPIOAEN;
    RCC->AHBENR |= RCC_AHBENR_GPIOBEN;
    RCC->APB2ENR |= RCC_APB2ENR_SYSCFGCOMPEN;
    SYSCFG->EXTICR[0] |= SYSCFG_EXTICR1_EXTI2_PA;
    // PB1-PB8 переведены в режим ввода для приёма параллельных данных
    GPIOB->MODER &= ~GPIO_MODER_MODER1;
    GPIOB->MODER &= ~GPIO_MODER_MODER2;
    GPIOB->MODER &= ~GPIO_MODER_MODER3;
    GPIOB->MODER &= ~GPIO_MODER_MODER4;
    GPIOB->MODER &= ~GPIO_MODER_MODER5;
    GPIOB->MODER &= ~GPIO_MODER_MODER6;
    GPIOB->MODER &= ~GPIO_MODER_MODER7;
    GPIOB->MODER &= ~GPIO_MODER_MODER8;
    // PA1 установлен в режим вывода как сигнал d_send
    GPIOA->MODER |= GPIO_MODER_MODER1_0;
    // PA2 переведён в режим ввода в качестве сигнала en от другого МК
    GPIOA->MODER &= ~GPIO_MODER_MODER2;
    // PA3 переведён в режим вывода в качестве сигнала ready от другого МК
    GPIOA->MODER |= GPIO_MODER_MODER3_0;
    // PA4 для генерации тактового сигнала
    GPIOA->MODER |= GPIO_MODER_MODER4_0;
    // PA5 для передачи данных
    GPIOA->MODER |= GPIO_MODER_MODER5_0;

    // включено прерывание для Px2 и Px4
    EXTI->IMR |= EXTI_IMR_IM2 | EXTI_IMR_IM4;
    // поднимающийся фронт в en sig для считывания данных и clock sig для установки данных
    EXTI->RTSR |= EXTI_RTSR_RT2 | EXTI_RTSR_RT4;

    // сигналу d_send изначально присвоено значение "1"
}

```

```

GPIOA->ODR |= GPIO_ODR_1;

NVIC_EnableIRQ(EXTI4_15 IRQn);
NVIC_SetPriority(EXTI4_15 IRQn, 1);
NVIC_EnableIRQ(EXTI2_3 IRQn);
NVIC_SetPriority(EXTI2_3 IRQn, 9);
}

void init_TIM15_for_clock_sig() {
    // RCC включен
    RCC->APB2ENR |= RCC_APB2ENR_TIM15EN;
    // настройка TIM14 на частоту 1000 Гц
    TIM15->ARR = 700;
    TIM15->PSC = 4;
    // прерывание включено
    TIM15->DIER |= TIM_DIER_UIE;
    NVIC_EnableIRQ(TIM15 IRQn);
    NVIC_SetPriority(TIM15 IRQn, 4);
}
#endif

```

USART_DMA_setup.h

В данных файлах проинициализирована периферия для передачи данных на ПК и их получения с ПК.

```

#ifndef USART_DMA_SETUP_H_
#define USART_DMA_SETUP_H_

#include "stm32f0xx.h"
#include "buffer.h"

void DMA1_Channel2_3_IRQHandler(void);

void init_GPIO_for_USART(); // инициализация GPIO для USART
void init_USART1(); // инициализация USART1 для приема данных с ПК и передачи данных на ПК
void init_DMA_to_work_with_USART(uint32_t transmit_page, uint32_t receive_page); // инициализация DMA для работы с USART и буферами
void setup_USART_DMA(uint32_t transmit_page, uint32_t receive_page); // содержит все вышеперечисленные функции
void start_receive_data_from_PC(); // включение DMA_channel3 для приёма данных с ПК
void start_transmit_data_to_PC(); // включение DMA_channel2 для передачи данных на ПК

#endif /* USART_DMA_SETUP_H_ */

```

USART_DMA_setup.c

```

#include "USART_DMA_setup.h"

void DMA1_Channel2_3_IRQHandler(void) {
    // проверка, в каком канале происходит прерывание
    if ((DMA1->ISR & DMA_ISR_TCIF2) == DMA_ISR_TCIF2) {
        DMA1->IFCR |= DMA_IFCR_CTCIF2; // снятие флага для прерывания
        DMA1_Channel2->CCR &= ~DMA_CCR_EN; // выключение канала DMA
    }
    if ((DMA1->ISR & DMA_ISR_TCIF3) == DMA_ISR_TCIF3) {
        DMA1->IFCR |= DMA_IFCR_CTCIF3; // снятие флага для прерывания
    }
}

```

```

        DMA1_Channel3->CCR &= ~DMA_CCR_EN; // выключение канала DMA
    }

void init_GPIO_for_USART(){
    RCC->AHBENR |= RCC_AHBENR_GPIOAEN;
    // USART1_TX на PA9 -> установка MODER на AF
    GPIOA->MODER |= GPIO_MODER_MODER9_1;
    // USART1_RX на PA10 -> установка MODER на AF
    GPIOA->MODER |= GPIO_MODER_MODER10_1;
    // установка AF1 для PA9 и PA10
    GPIOA->AFR[1] |= 0x01 << GPIO_AFRH_AFRH1_Pos;
    GPIOA->AFR[1] |= 0x01 << GPIO_AFRH_AFRH2_Pos;
}

void init_USART1() {
    // RCC включен
    RCC->APB2ENR |= RCC_APB2ENR_USART1EN;
    // приёмник на PA10
    USART1->CR1 |= USART_CR1_RE;
    // передатчик PA9
    USART1->CR1 |= USART_CR1_TE;
    USART1->BRR = SystemCoreClock / 115200;
    // инициализация DMA для работы с передатчиком и приёмником
    USART1->CR3 |= USART_CR3_DMAT | USART_CR3_DMAR;
    // USART1 включен
    USART1->CR1 |= USART_CR1_UE;
}

void init_DMA_to_work_with_USART(uint32_t transmit_page, uint32_t receive_page) {
    RCC->AHBENR |= RCC_AHBENR_DMA1EN;
    // режим увеличения объёма памяти
    DMA1_Channel2->CCR |= DMA_CCR_MINC;
    DMA1_Channel3->CCR |= DMA_CCR_MINC;
    // dir передачи данных
    DMA1_Channel2->CCR |= DMA_CCR_DIR; // из памяти
    DMA1_Channel3->CCR &= ~DMA_CCR_DIR; // из периферии
    // циклический режим включен
    DMA1_Channel2->CCR |= DMA_CCR_CIRC;
    DMA1_Channel3->CCR |= DMA_CCR_CIRC;
    // размер данных в байтах
    DMA1_Channel2->CNDTR = BUFFER_SIZE;
    DMA1_Channel3->CNDTR = BUFFER_SIZE;
    // адрес периферии
    DMA1_Channel2->CPAR = (uint32_t)(&(USART1->TDR));
    DMA1_Channel3->CPAR = (uint32_t)(&(USART1->RDR));
    // адрес данных
    DMA1_Channel2->CMAR = (uint32_t)(receive_page);
    DMA1_Channel3->CMAR = (uint32_t)(transmit_page);
    // прерывание включено
    DMA1_Channel2->CCR |= DMA_CCR_TCIE;
    DMA1_Channel3->CCR |= DMA_CCR_TCIE;
    NVIC_EnableIRQ(DMA1_Channel2_3_IRQn);
    NVIC_SetPriority(DMA1_Channel2_3_IRQn, 3);
}

void start_receive_data_from_PC(){
    DMA1_Channel3->CCR |= DMA_CCR_EN;
}

void start_transmit_data_to_PC() {

```

```

        DMA1_Channel12->CCR |= DMA_CCR_EN;
    }

void setup_USART_DMA(uint32_t transmit_page, uint32_t receive_page) {
    init_GPIO_for_USART();
    init_USART1();
    init_DMA_to_work_with_USART(transmit_page, receive_page);
}

```

Serial_receive_and_parallel_transfer.h

В данных файлах расположены основные функции для передачи и получения данных, а также исполнения дополнительных функций, таких как мигание диодом на другом МК, получение данных с ПК, передача данных на ПК, запись данных во FLASH.

```

#ifndef SERIAL_RECEIVE_AND_PARALLEL_TRANSFER_H_
#define SERIAL_RECEIVE_AND_PARALLEL_TRANSFER_H_

#define AMOUNT_RECEIVE_DATA 8
#include "Periphery_for_transfer_and_receive_init.h"
#include "button_cmd.h"
#include "USART_DMA_setup.h"

#ifdef SER_TRANSFER_AND_PAR_RECEIVE
void TIM15_IRQHandler(void); // timer15 для генерации тактовых импульсов для последовательной передачи данных
#endif
void EXTI4_15_IRQHandler(void); // опережение тактовой частоты генерирует прерывание в EXTI4 -> здесь установлен флаг для чтения данных из GPIO
void EXTI2_3_IRQHandler(void); // PA2 - сигнал en получен с этого МК, PA3 - готовый сигнал с другого МК. В прерывании EXTI2 установлен флаг для передачи данных.

// функции
void Setup_flash_with_data(); // инициализация буфера и его запись во FLASH
void Tune_peripheries(); // инициализация всех периферийных устройств, необходимых для работы программы
void write_to_flash(); // запись данных во FLASH с помощью FLASH.h и FLASH.c
void manage_states(); // запуск программы зависит от того, какое действие выбрано
void check_key_for_blink_led(); // проверка во время приёма данных, были ли первые 16 бит ключом для мигания светодиода на текущем МК
void blink_led_3_times(); // если ключ был получен, мигание светодиодом 3 раза
void process_cmd(); // действия для передачи 8 бит, получения бита и перехвата готового сигнала с другого МК

#ifdef SER_TRANSFER_AND_PAR_RECEIVE
void Restart_work_after_transfer(); // возвращение кнопки в исходное состояние
void transfer_bit(); // передача 1 бита на другой МК
void check_byte_transfer_complete(); // выполнение действий при передаче байтов данных
void receive_8bit(); // получение 8 битов от другого МК
void check_byte_receive_complete(); // если байт получен -> записать в buf -> начать получать новый байт данных
void check_receive_complete(); // если все данные получены -> перезапустить работу кнопки
void blink_led_on_other_mk(); // если кнопка была нажата 2 раза в режиме настройки - > ключ переключения для мигания светодиода на другом МК
#endif
/*

```

```

#ifndef PAR_TRANSFER_AND_SERIAL_RECEIVE
void Restart_work_after_transfer(); // возвращение кнопки в исходное состояние
void receive_bit(); // получение 1 бита от другого МК
void check_byte_receive_complete(); // выполнение действий при получении байтов данных
void check_receive_complete(); // проверка, заполнен ли buf с полученными данными
void set_data_on_GPIO(); // передача 8 бит на другом МК
#endif
*/
#endif /* SERIAL_RECEIVE_AND_PARALELL_TRANSFER_H */

```

Serial_receive_and_parallel_transfer.c

```

#include "serial_receive_and_parallel_transfer.h"

static uint8_t count_bit_for_tr = 0;
static uint8_t count_bit_for_rec = 0;
static uint8_t data_for_tr = 0;
static uint8_t data_for_rec = 0;

void EXTI2_3_IRQHandler(void) {
    if( (EXTI->PR & EXTI_PR_PIF2) == EXTI_PR_PIF2 ) {
        EXTI->PR |= EXTI_PR_PIF2; /*
#ifndef PAR_TRANSFER_AND_SERIAL_RECEIVE
        transfer_state = TRANSFER_8_BIT;
#endif */
#ifndef SER_TRANSFER_AND_PAR_RECEIVE
        receive_state = RECEIVE_8_BIT;
#endif
    }
    if( (EXTI->PR & EXTI_PR_PIF3) == EXTI_PR_PIF3 ) {
        EXTI->PR |= EXTI_PR_PIF3; /*
#ifndef PAR_TRANSFER_AND_SERIAL_RECEIVE
        transfer_state = CATCH_SIG_READY;
#endif */
    }
}

void EXTI4_15_IRQHandler(void) {
    if( (EXTI->PR & EXTI_PR_PIF4) == EXTI_PR_PIF4 ) {
        EXTI->PR |= EXTI_PR_PIF4; /*
#ifndef PAR_TRANSFER_AND_SERIAL_RECEIVE
        receive_state = RECEIVE_BIT;
#endif */
#ifndef SER_TRANSFER_AND_PAR_RECEIVE
        transfer_state = TRANSFER_BIT;
#endif
    }
}

#ifndef SER_TRANSFER_AND_PAR_RECEIVE
void TIM15_IRQHandler(void) { // в прерываниях timer15 генерируется тактовый сигнал
    if( (TIM15->DIER & TIM_DIER_UIE) == TIM_DIER_UIE ) {
        TIM15->SR &= ~TIM_SR UIF;
        GPIOA->ODR ^= GPIO_ODR_4; // генерация тактового сигнала на PA4
    }
}

void Restart_work_after_transfer() {
    GPIOA->ODR &= ~GPIO_ODR_4; // генерация спадающего фронта для получения последнего бита в другом МК
}

```

```

        for(int i = 0; i < 20; i++); // задержка для того, чтобы sig d_send успел до-
стичь другого МК
        TIM15->CR1 &= ~TIM_CR1_CEN; // а затем установить для d_send значение "1".
        GPIOA->ODR |= GPIO_ODR_1; // Значение PA1 равно 1 (сигнал d_send), по-
скольку передача завершена
        // Buf для повторной передачи данных, заполненный данными
        Buf_for_transfer.amount_data_in_buf = BUFFER_SIZE;
        Turn_off_Led_PC8();
        Turn_off_Led_PC9();
        Button_reset_to_be_ready_for_work();
        program_state = WAITING_FOR_ACTION;
    }

void transfer_bit() {
    // если count_bit_for_tr = 0, то получаем байт данных из буфера
    if(count_bit_for_tr == 0)
        data_for_tr = Buffer_get_from_front(&Buf_for_transfer);
    GPIOA->ODR &= ~GPIO_ODR_5; // очищаем бит для передачи данных
    // запись 1 бита в PA5, для этого - получение определенного бита из байта и
    // перемещение его в ODR5 (<< 5)
    GPIOA->ODR |= (( (data_for_tr >> count_bit_for_tr) & 0x0001 ) << 5);
    count_bit_for_tr++;
}

void check_byte_transfer_complete() {
    // если передаётся байт данных, то значение count_bit_for_tr сбрасывается до
    // нуля, чтобы
    if(count_bit_for_tr == DATA_SIZE) {
        count_bit_for_tr = 0; // осуществилась передача следующего байта данных
        // если это был последний элемент в буфере передачи, то передача завер-
        // шена
        if(Buffer_empty(&Buf_for_transfer))
            Restart_work_after_transfer();
    }
}

void receive_8bit() {
    data_for_rec |= ((GPIOB->IDR >> 1) & 0x00FF); // получение 8 бит от PB1-PB9
}

void check_byte_receive_complete() {
    // если было получено 8 бит, то осуществить запись данных в buf и обнулить пе-
    // ременные для получения данных
    Buffer_add_to_end(&Buf_For_receive, data_for_rec);
    count_bit_for_rec = 0;
    data_for_rec = 0;
    check_key_for_blink_led(); // проверка, были ли первые 16 бит ключом для
    // миграции светодиода
}

void check_receive_complete() {
    if(Buffer_is_full(&Buf_For_receive)) {
        receive_state = RECEIVE_COMPLETE; // если buf для получения заполнен,
        // то получение завершено
        Clear_buffer(&Buf_For_receive); // искусственно созданный buf снова
        пуст
    }
}

void blink_led_on_other_mk() {
    GPIOA->ODR &= ~GPIO_ODR_5; // очистка бита для передачи данных
}

```

```

    // запись 1 бита в PA5, для этого - получение определенного бита из байта и
    // перемещение его в ODR5 (<< 5)
    GPIOA->ODR |= (( (KEY_FOR_BLINK_LED >> count_bit_for_tr) & 0x0001 ) << 5);
    count_bit_for_tr++;
    if(count_bit_for_tr == (DATA_SIZE * 2) ) {
        count_bit_for_tr = 0;
        Restart_work_after_transfer();
    }
}

#endif
/*
#endif PAR_TRANSFER_AND_SERIAL_RECEIVE
void receive_bit() {
    // получение бита из GPIOA->IDR PA5 и запись в переменную temp
    data_for_rec |= (((GPIOA->IDR >> 5) & 0x0001 ) << count_bit_for_rec);
    count_bit_for_rec++; // увеличение количества уже полученных битов
}

void check_byte_receive_complete()
{
    if(count_bit_for_rec == DATA_SIZE)
    {
        Buffer_add_to_end(&Buf_For_receive, data_for_rec); // если получено 8
бит
        data_for_rec = 0; // тогда начать получать новые биты
        count_bit_for_rec = 0;
        check_key_for_blink_led(); // проверка, были ли первые 16 бит ключом
для мигания светодиода
    }
}

void check_receive_complete() {
    if(Buffer_is_full(&Buf_For_receive)) { // если буфер заполнен, то все данные
получены
        receive_state = RECEIVE_COMPLETE;
        Clear_buffer(&Buf_For_receive); // искусственно созданный buf
снова пуст
    }
}

void set_data_on_GPIO() {
    if(program_state == KEY_TRANSFER) {
        GPIOB->ODR &= (0xFE01); // очистка GPIO для установки правильного бита
для передачи данных
        // установка битов на PB1-PB8
        GPIOB->ODR |= ( ((KEY_FOR_BLINK_LED >> count_bit_for_tr) & 0x00FF ) <<
1);
        count_bit_for_tr += AMOUNT_RECEIVE_DATA;
        return;
    }
    // если это обычная передача, а не передача ключа
    data_for_tr = Buffer_get_from_front(&Buf_for_transfer); // получение элемента
из buf
    GPIOB->ODR &= (0xFE01); // очистка GPIO для установки правильного бита для
передачи данных
    GPIOB->ODR |= ( data_for_tr << 1 ); // установка битов на PB1-PB9
}

void Restart_work_after_transfer() {
    GPIOA->ODR &= ~GPIO_ODR_2; // сброс GPIO в исходное состояние
}

```

```

// Буфер для повторной передачи данных, заполненный искусственно
Buf_for_transfer.amount_data_in_buf = BUFFER_SIZE;
Turn_off_Led_PC8();
Turn_off_Led_PC9();
Button_reset_to_be_ready_for_work();
program_state = WAITING_FOR_ACTION;
}
#endif
*/
void Setup_flash_with_data() {
    InitBuffer(&Buf_for_transfer, &array_tr[0], BUFFER_SIZE);
    InitBuffer(&Buf_For_receive, &array_rec[0], BUFFER_SIZE);
    Write_data_to_flash(PAGE_FOR_TRANSFER, Buf_for_transfer.buf, BUFFER_SIZE); // запись пустого буфера во FLASH
    Write_data_to_flash(PAGE_FOR_RECEIVE, Buf_For_receive.buf, BUFFER_SIZE); // для того, чтобы проверить корректность
    Buf_for_transfer.amount_data_in_buf = BUFFER_SIZE;
    // данных, полученных с ПК или другого МК
}

void check_key_for_blink_led() {
    if(Buf_For_receive.amount_data_in_buf == 2) { // если получено 2 элемента
        (ключ состоит из 16 бит)
        if( (Buf_For_receive.buf[0] | (Buf_For_receive.buf[1] << 8)) != KEY_FOR_BLINK_LED )
            return;
        program_state = BLINK_MY_LED;
        Clear_buffer(&Buf_For_receive);
    }
}

void write_to_flash() {
    if(receive_state == RECEIVE_COMPLETE){ // если данные были получены,
записать их во FLASH
        Write_data_to_flash(PAGE_FOR_RECEIVE, Buf_For_receive.buf, BUFFER_SIZE);
        Button_reset_to_be_ready_for_work(); // кнопка сброса
        Turn_off_Led_PC8();
        receive_state = WAITING_FOR_ACTION; // готовность к приёму новых данных
        program_state = WAITING_FOR_ACTION;
    }
    else {
        Turn_off_Led_PC8(); // иначе, если данные не были получены
        Button_reset_to_be_ready_for_work(); // просто кнопка сброса
        program_state = WAITING_FOR_ACTION;
    }
}

void blink_led_3_times() {
    TIM14->CR1 |= TIM_CR1_CEN; // включение таймера на 0,5 секунды
    if(count_of_blink == 6) { // и ожидание, пока светодиод мигнёт 3 раза
        program_state = WAITING_FOR_ACTION;
        TIM14->CR1 &= ~TIM_CR1_CEN;
        count_of_blink = 0;
    }
}

// содержит функцию для инициализации всей периферии программы
void Tune_peripheries() {
    Button_tune();
    init_GPIO_for_transfer_and_receive_data(); /*
#endif SER_TRANSFER_AND_PAR_RECEIVE

```

```

        init_TIM15_for_clock_sig();
#endif */
    Setup_flash_with_data();
    setup_USART_DMA((uint32_t)(&Buf_for_transfer.buf[0]), PAGE_FOR_RECEIVE);
}

void process_cmd() { /*
#ifndef PAR_TRANSFER_AND_SERIAL_RECEIVE
    if(receive_state == RECEIVE_BIT) {
        receive_state = WAITING_FOR_ACTION; // изменение состояния приёма
        if((GPIOA->IDR & GPIO_IDR_1) == GPIO_IDR_1) // если сигнал d_send равен
        "1", значит, данные не передаются
            return;
        // а значит, return
        receive_bit(); // иначе, если сигнал на PA1 (d_send) "0" -> данные пе-
        редаются
        check_byte_receive_complete(); // а значит, получение этих данных
        check_receive_complete();
    }

    if(transfer_state == TRANSFER_8_BIT) { // изменение состояния передачи данных
        transfer_state = WAITING_FOR_ACTION;
        GPIOA->ODR &= ~GPIO_ODR_2; // установка 0 на PA2
        set_data_on_GPIO(); // установка 8 bit на GPIOx
        GPIOA->ODR |= GPIO_ODR_2; // установка 1 на PA2 ->
нарастающий фронт (включение сигнала) для PA2 другого МК
    }

    if(transfer_state == CATCH_SIG_READY) {
        transfer_state = WAITING_FOR_ACTION;
        if( Buffer_empty(&Buf_for_transfer) ) {
            Restart_work_after_transfer(); // если buf пуст -> все данные
передаются
            return; // нет необходимости
устанавливать бит SWI, поэтому перезапуск работы и return
        }
        if(program_state == KEY_TRANSFER && count_bit_for_tr == (DATA_SIZE * 2))
{
            count_bit_for_tr = 0; // если передан ключ для мигания
светодиода на другом МК ->
            Restart_work_after_transfer(); // нет необходимости устанавливать
бит SWI, поэтому перезапуск работы и return
            return;
        }
EXTI->SWIER |= EXTI_SWIER_SWI2; // если произошёл спад фронта, то выполнить
новую 8-битную передачу
    }
#endif */

#ifndef SER_TRANSFER_AND_PAR_RECEIVE
    if(receive_state == RECEIVE_8_BIT) {
        receive_state = WAITING_FOR_ACTION; // изменение состояния приёма
        GPIOA->ODR |= GPIO_ODR_3; // установка 1 на PA3 (сигнал готов)
        receive_8bit();
        check_byte_receive_complete();
        check_receive_complete();
        GPIOA->ODR &= ~GPIO_ODR_3; // установка 0 на PA3 (сигнал готов),
обеспечение спада для PA3 другого МК
    }
    if(transfer_state == TRANSFER_BIT) {

```

```

        transfer_state = WAITING_FOR_ACTION;      // изменение состояния передачи
данных
        if(program_state == KEY_TRANSFER) {
            blink_led_on_other_mk();
            return;
        }
        transfer_bit();
        check_byte_transfer_complete();
    }
#endif
}

// содержит переключатель, в котором с помощью состояния программы выбираются дей-
ствия
void manage_states() {
    switch(program_state) {
        case TRANSFER_DATA:
            program_state = PROGRAMMING;
            ReadFromFlash(PAGE_FOR_TRANSFER, Buf_for_transfer.buf,
BUFFER_SIZE); /*
#ifndef PAR_TRANSFER_AND_SERIAL_RECEIVE
EXTI->SWIER |= EXTI_SWIER_SWI2; // вызов прерывания на EXTI2 и
начало передачи данных
#endif */
#ifndef SER_TRANSFER_AND_PAR_RECEIVE
GPIOA->ODR &= ~GPIO_ODR_1; // начало передачи данных, поэтому
установка PA1(d_send) на нулевой уровень
for(int i = 0; i < 20; i++); // задержка для того, чтобы sig
d_send успел достичь другого МК
TIM15->CR1 |= TIM_CR1_CEN; // TIM15 включен для генерации
тактового сигнала
#endif
            break;
        case WRITE_DATA_INTO_FLASH:
            program_state = PROGRAMMING;
            write_to_flash();
            break;
        case BLINK_LED_ON_OTHER_MK:
            program_state = KEY_TRANSFER; /*
#ifndef PAR_TRANSFER_AND_SERIAL_RECEIVE
EXTI->SWIER |= EXTI_SWIER_SWI2;
#endif */
#ifndef SER_TRANSFER_AND_PAR_RECEIVE
GPIOA->ODR &= ~GPIO_ODR_1; // начало передачи данных, поэтому
установка PA1(d_send) на нулевой уровень
for(int i = 0; i < 20; i++); // задержка для того, чтобы sig
d_send успел достичь другого МК
TIM15->CR1 |= TIM_CR1_CEN; // TIM15 включен для генерации
тактового сигнала
#endif
            break;
        case BLINK_MY_LED:
            blink_led_3_times();
            break;
        case RECEIVE_DATA_FROM_PC:
            start_receive_data_from_PC();
            while(DMA1_Channel13->CCR & DMA_CCR_EN); // ожидание, пока будут
приняты данные
            Write_data_to_flash(PAGE_FOR_TRANSFER, Buf_for_transfer.buf,
BUFFER_SIZE); // запись переданных данных во FLASH
            Turn_off_Led_PC8();
    }
}

```

```

        Button_reset_to_be_ready_for_work(); // кнопка сброса
        program_state = WAITING_FOR_ACTION;
        break;
    case TRANSFER_DATA_TO_PC:
        start_transmit_data_to_PC();
        while(DMA1_Channel12->CCR & DMA_CCR_EN); // ожидание, пока
начнётся передача данных
        Turn_off_Led_PC9();
        Button_reset_to_be_ready_for_work(); // кнопка сброса
        program_state = WAITING_FOR_ACTION;
        break;
    default:
        break;
}
}

```

main.c

```

#include "serial_receive_and_parallel_transfer.h"

int main(void)
{
    Tune_peripheries();
    while (1)
    {
        check_button_press();
        manage_states();
        process_cmd();
    }
}

```

3.4 Сборка и компиляция

После написания кода оставляем в открытых только main.c и нажимаем Project --> Build All (или ПКМ по названию вашего проекта в Project Explorer и Build Project).

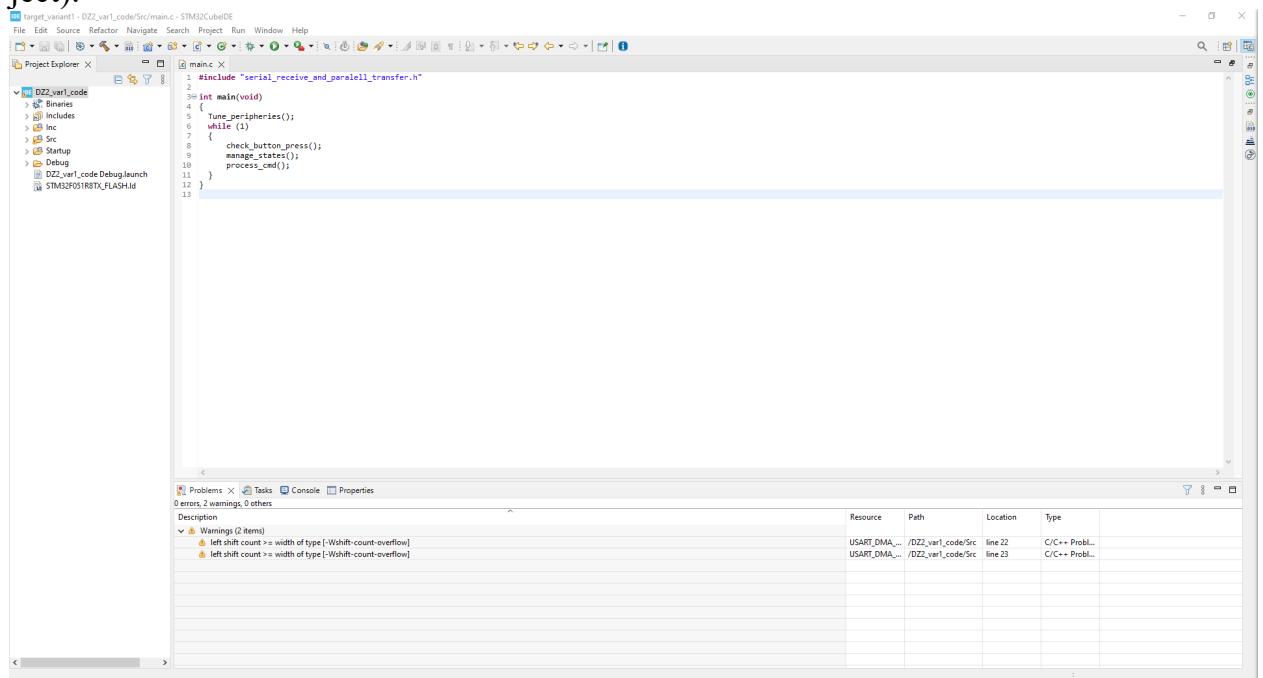


Рисунок 3.12 – Окно программы после сборки

Нажимаем Run -> Run Configurations... и дважды нажимаем ЛКМ по STM32 C/C++ Application. Должно появиться “*название вашего проекта* Debug”. Нажимаем на него. Во вкладке Main под C/C++ Application: прописываем Debug/*название вашего проекта*.elf. Напротив синей Build Configuration: ставим из выпадающего списка Use active. Далее внизу справа нажимаем Apply.

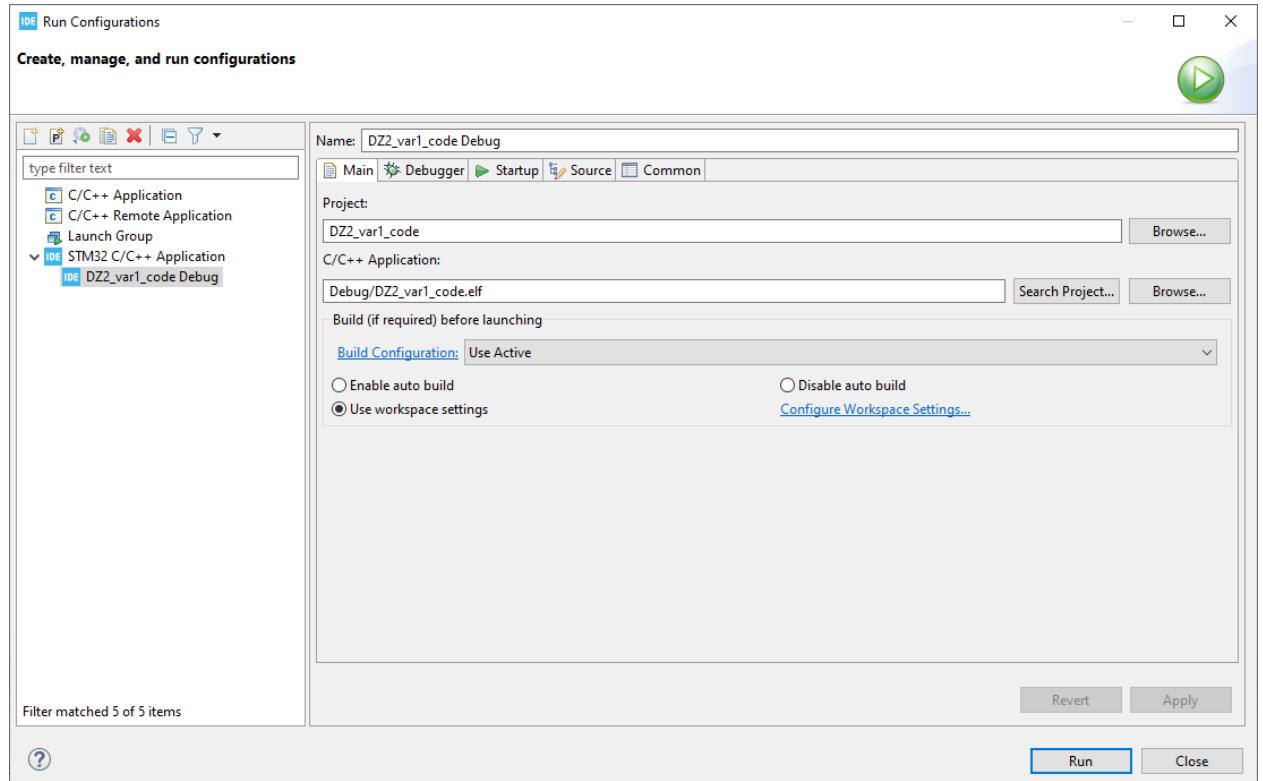


Рисунок 3.13 – Окно настройки конфигурации для компиляции – Main

Дальше переходим на вкладку Debugger. Тут ничего не трогаем кроме двух вещей. Первое – ставим галочку рядом с ST-LINK S/N, далее подключаем нашу плату к ПК и нажимаем Scan. После нажатия в поле рядом сразу должен появиться номер нашей платы. Второе – напротив Debug in low power modes: ставим Disable. Далее нажимаем Apply.

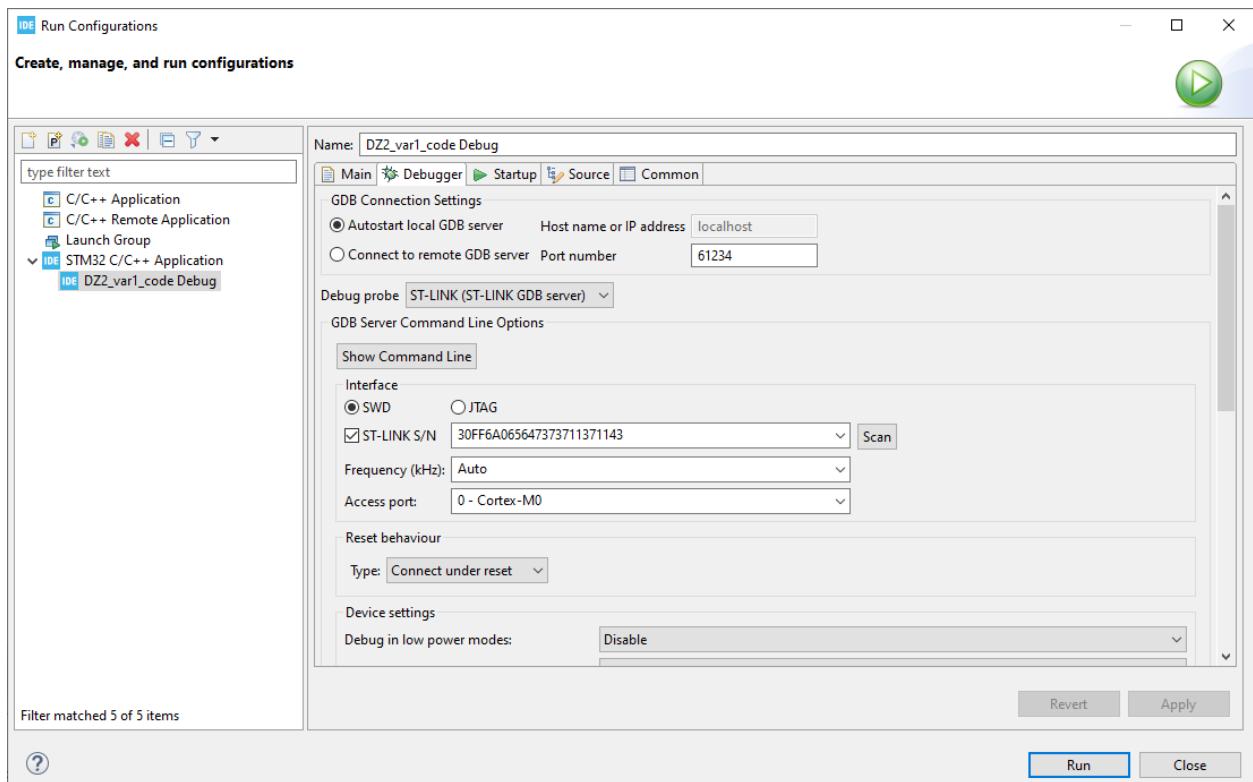


Рисунок 3.14 - Окно настройки конфигурации для компиляции – Debugger

Далее нажимаем Run.

При компиляции может появиться окно с сообщением: “Эта версия STM32CubeIDE предоставляет более новую версию встроенного ПО для подключенного ST-LINK. Продолжить с обновлением?”.

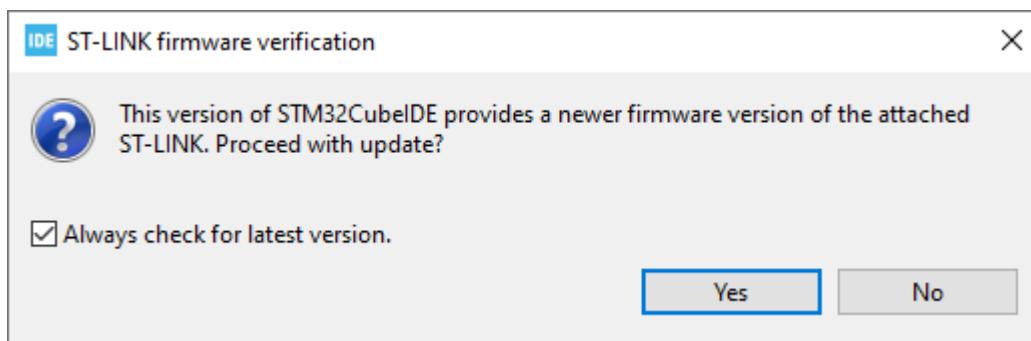


Рисунок 3.15 – Окно сообщения

Если мы нажмём Yes, то высветится окно STLinkUpgrade 3.11.3, где будет выпадающий список с подключенными платами, кнопка Open in update mode, ID нашей платы и кнопка Upgrade, на которую нам никак не получится нажать.

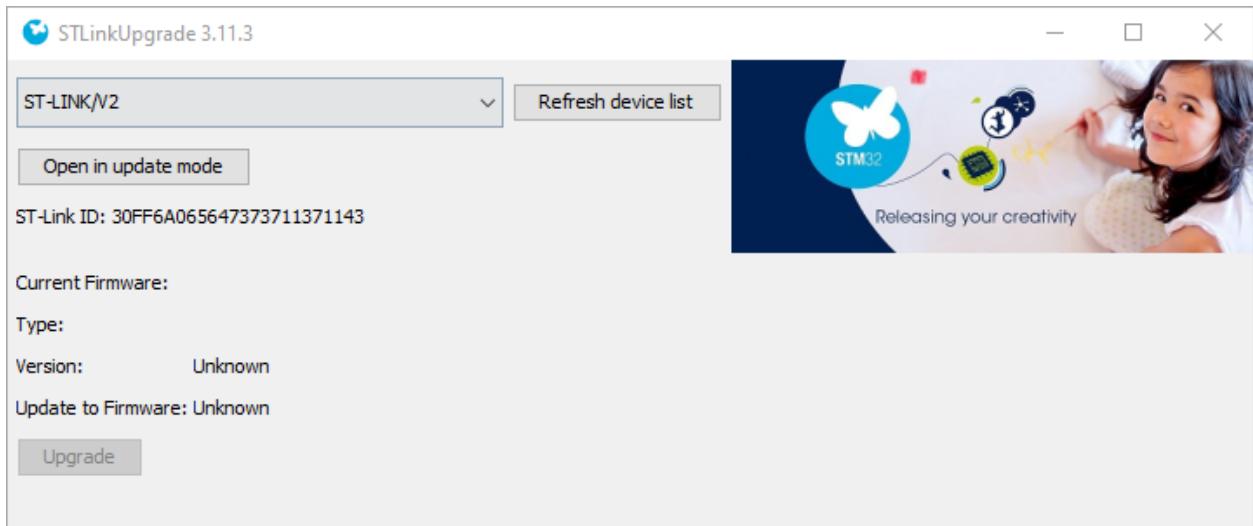


Рисунок 3.16 – Окно после окна с сообщением

Если мы нажмём на кнопку Open in update mode 1 или несколько раз, то будет высвечиваться одно и то же сообщение: “ST-LINK не в DFU моде. Пожалуйста перезапустите его.”

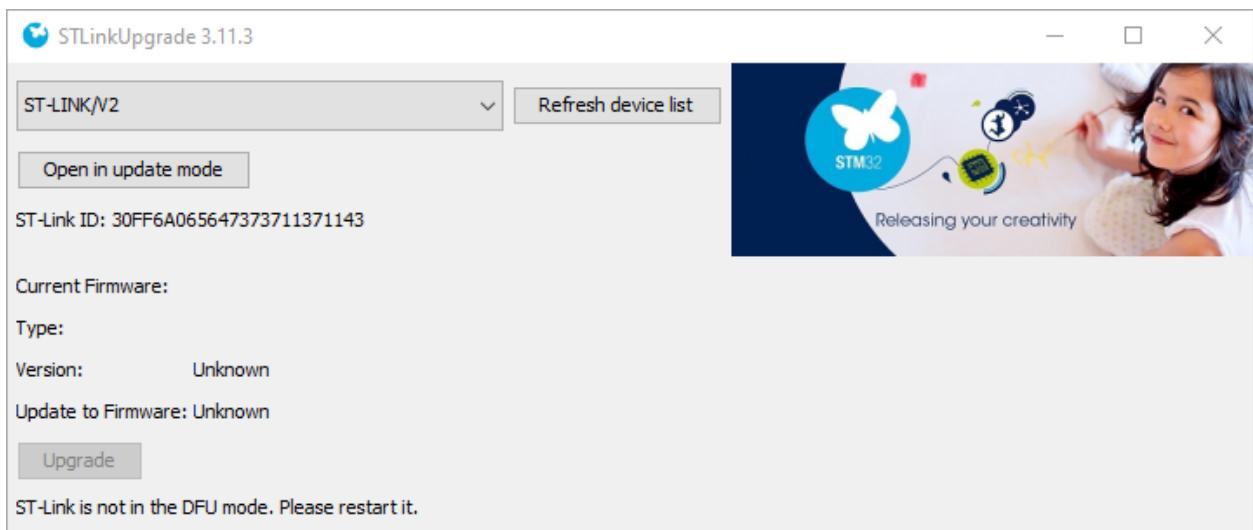


Рисунок 3.17 – Тупик

Скажу наперёд, что ввести свою плату в DFU мод я пытался (с помощью программ DfuFileMgr и DfuSeDemo), но для отладки это не понадобилось (об этом ниже).

Если мы нажмём No, то проект просто скомпилируется и после завершения процесса компиляции по указанному ранее пути появится файл с расширением elf. Он-то нам и понадобится для дальнейшей отладки.

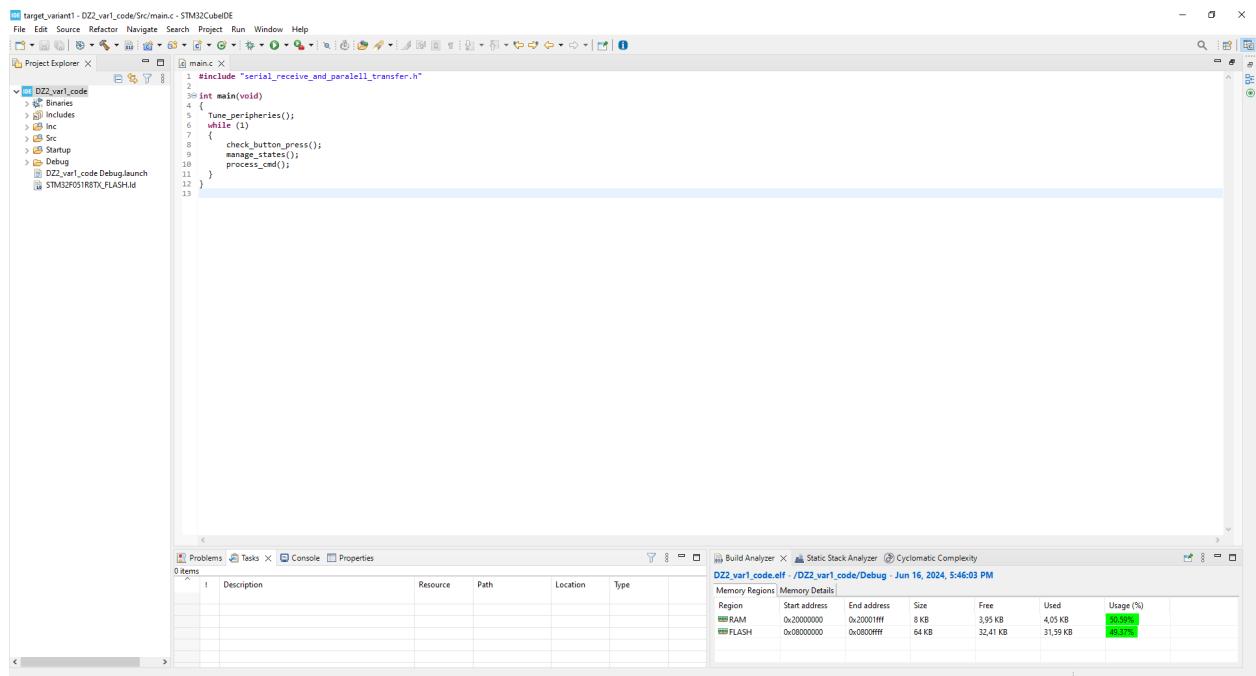


Рисунок 3.18 – Окно программы после компиляции

Теперь программу STM32CubeIDE можно закрыть. В следующий раз она нам понадобится либо подправить код, либо создать уже новый проект со вторым вариантом.

3.5 Обновление драйверов платы

Для дальнейшей работы понадобится обновить драйвера нашей платы. Для этого запускаем STM32 ST-LINK Utility.

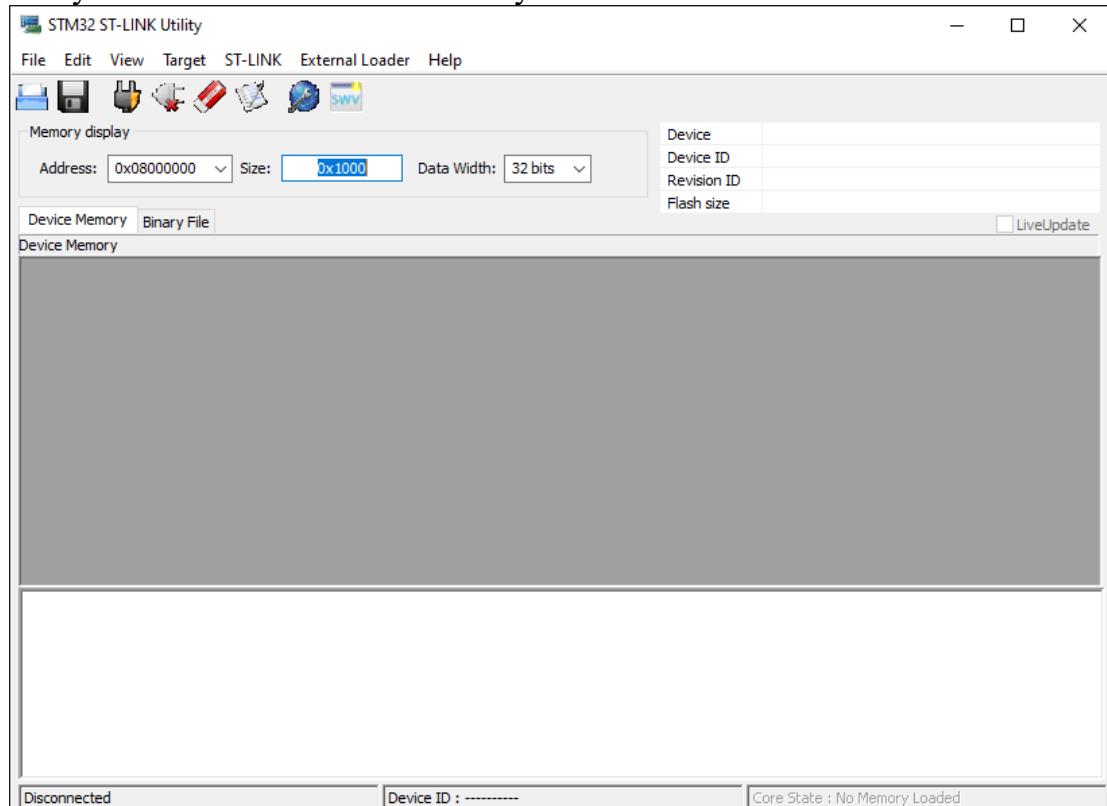


Рисунок 3.19 – Окно программы STM32 ST-LINK Utility

Отсоединяем провод, соединяющий плату и ПК, от ПК, ждём 5 секунд, затем снова подсоединяем.

Нажимаем ST-LINK --> Firmware update. Высветится окно со знакомой картинкой.

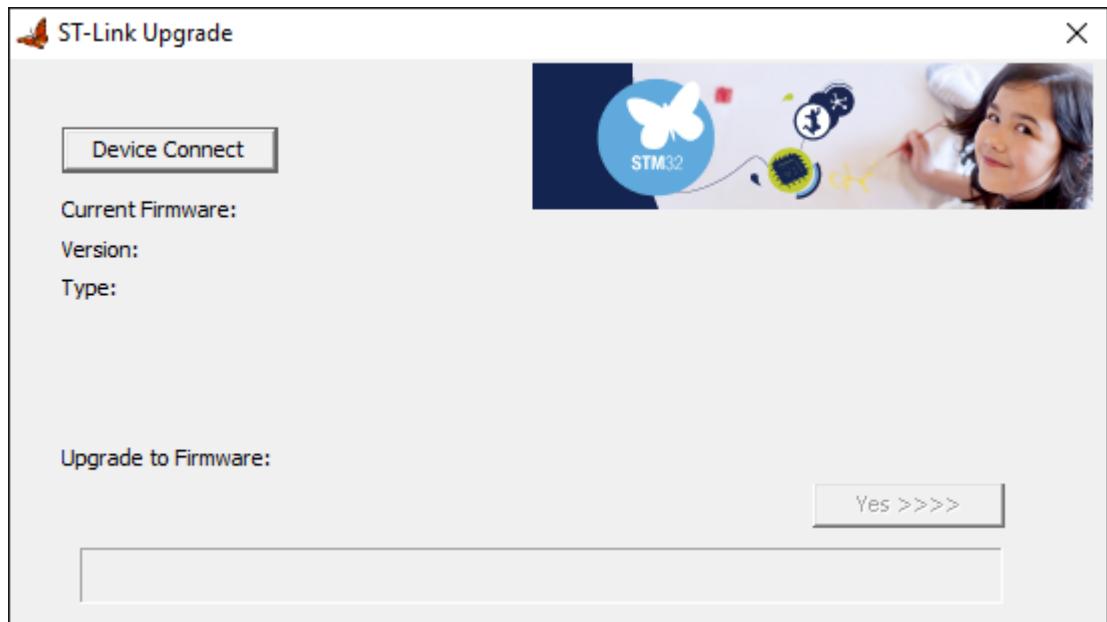


Рисунок 3.20 – Окно для обновления

Нажимаем Device connect.

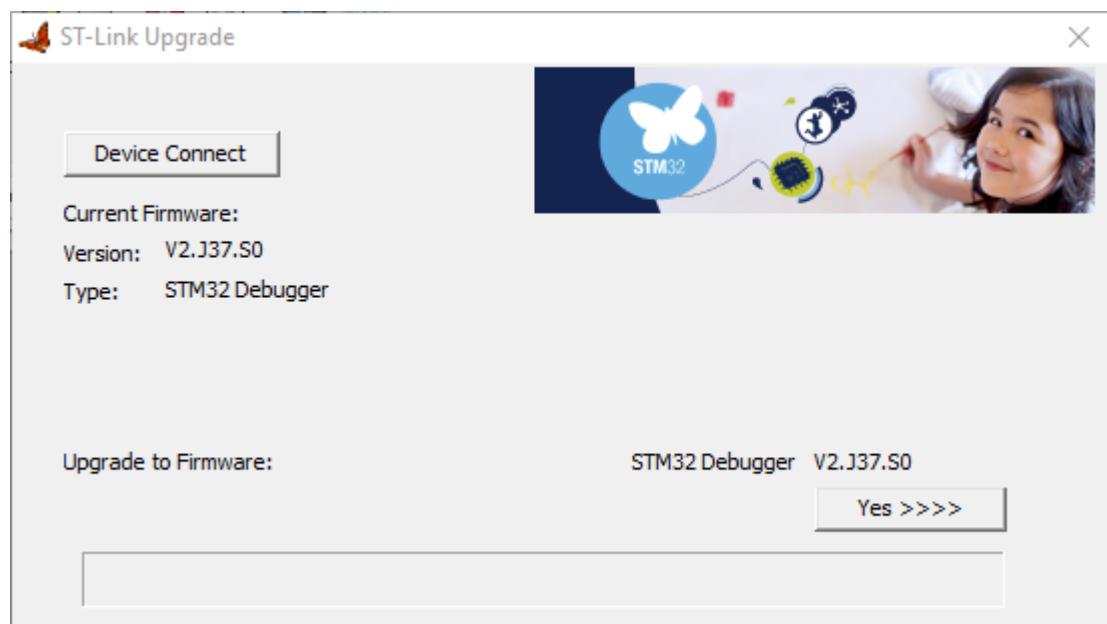


Рисунок 3.21 – Окно после нажатия на кнопку

Нажимаем на Yes >>> и ждём. После процедуры высветится уведомление об успешном завершении обновления и светодиоды на плате начнут мигать по-другому.

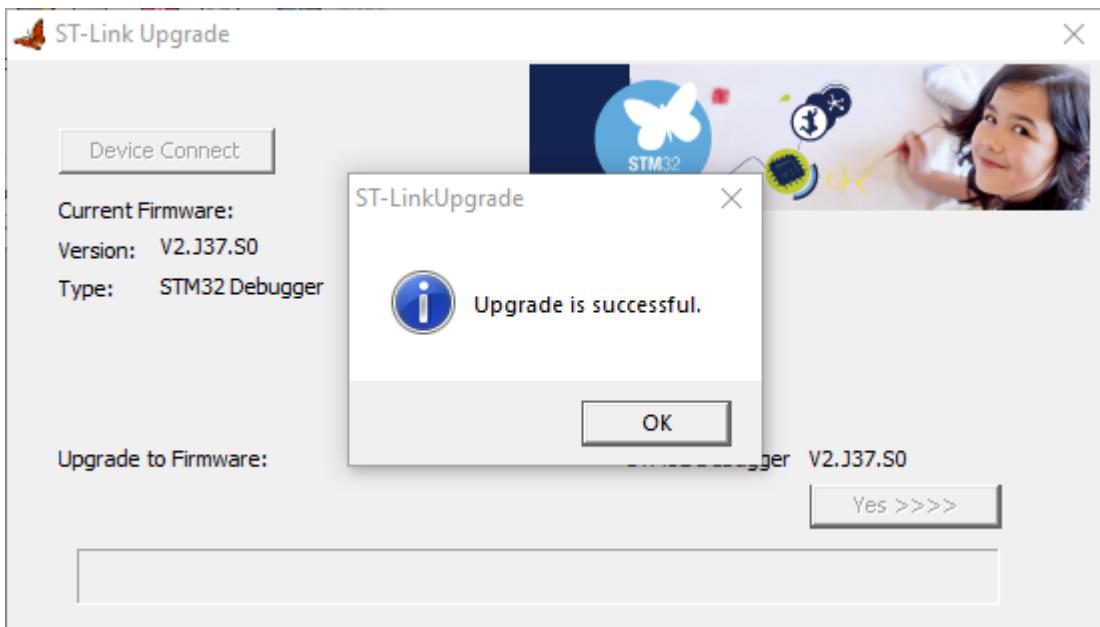


Рисунок 3.22 – Окно после завершения обновления

Нажимаем OK и выходим из программы.

3.6 Выполнение 2-го варианта перед отладкой

Создаём проект DZ2_var2_code в STM32CubeIDE для второго варианта. Копируем туда уже написанный код. Некоторые строчки убираем из комментариев, а некоторые наоборот – комментируем – в следующих скриптах: “Periphery_for_transfer_and_receive_init.h”, “Periphery_for_transfer_and_receive_init.c”, “serial_receive_and_parallel_transfer.h”, “serial_receive_and_parallel_transfer.c”.

Собираем и компилируем код. Обновляем драйвера второй платы с помощью вышеупомянутого STM32 ST-LINK Utility.

Теперь соединяем с помощью 13-ти проводов мама-мама одноимённые пины на обеих платах:

- PA1 – сигнал «d_send»;
- PA2 – сигнал «en»;
- PA3 – сигнал «ready»;
- PA4 – clock (генератор тактового сигнала);
- PA5 – последовательные данные;
- PB1-PB8 – параллельные данные.

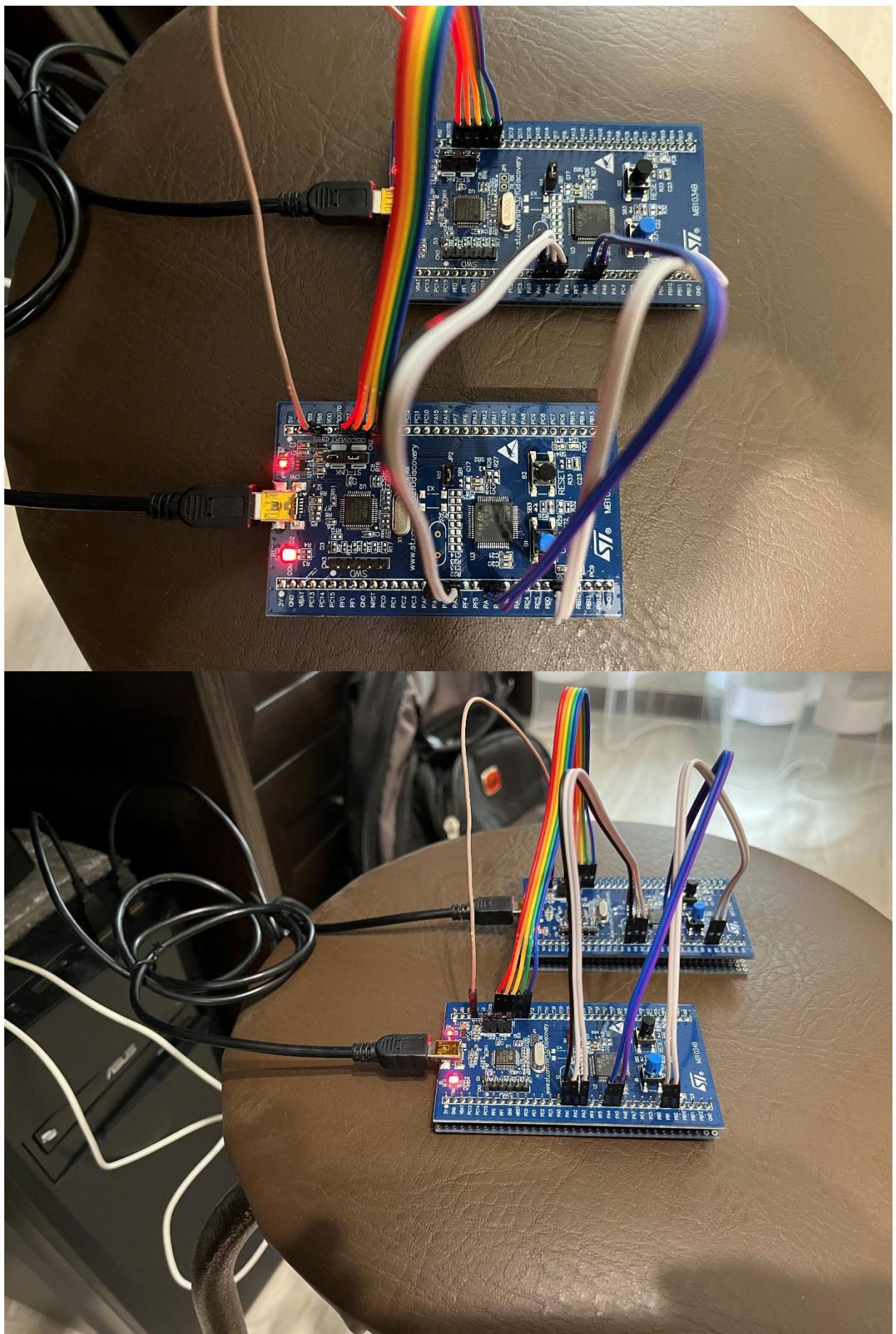


Рисунок 3.23 – Платы, соединённые проводами и подключенные к ПК

3.7 Отладка

STM32CubeProgrammer

Для отладки запускаем STM32CubeProgrammer. Если всплывает окно Connection Error, то нажимаем Cancel.

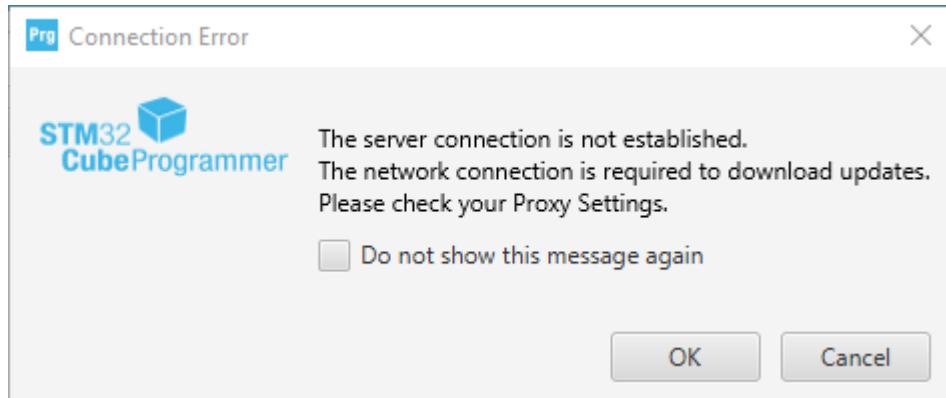


Рисунок 3.24 – Окно Connection Error

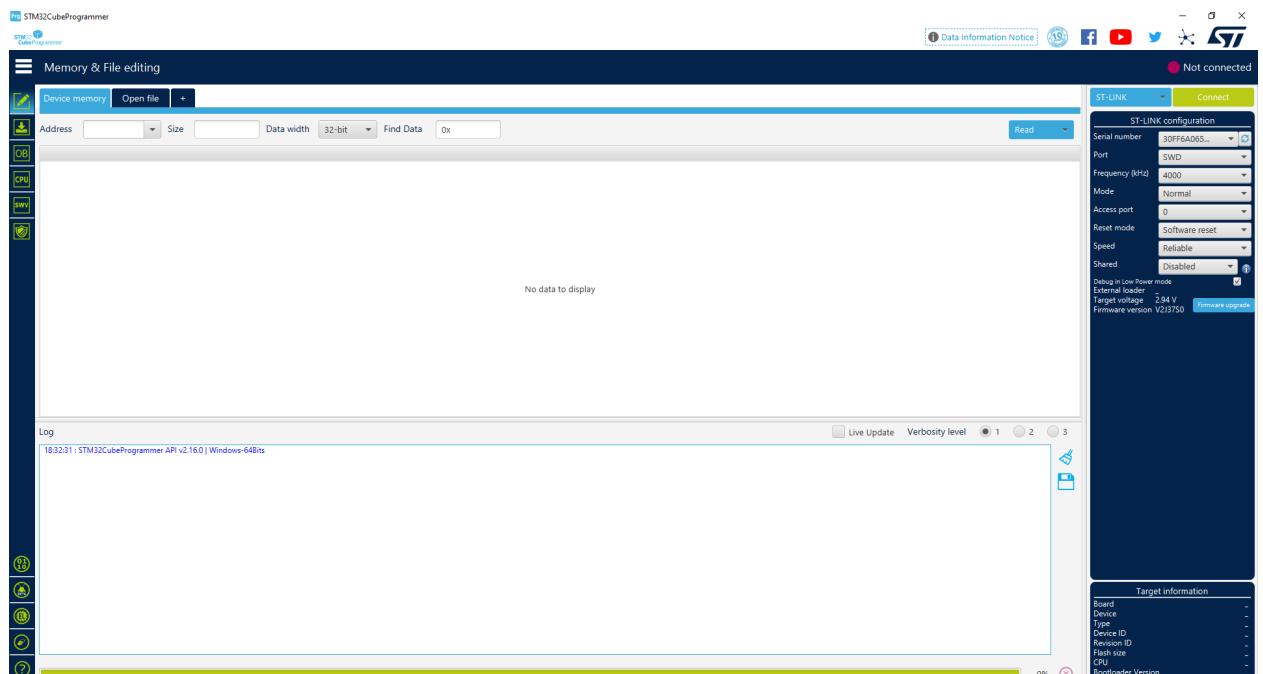


Рисунок 3.25 – Окно программы STM32CubeProgrammer

Программа в реальном времени отслеживает, подключены ли платы к ПК, а также их серийные номера.

В нашем случае платы подключены к ПК. Выбираем серийный номер одной из них и нажимаем Connect в правом верхнем углу. Т.к. мы обновили драйвера платы, соединение прошло успешно. Сразу после соединения нам высветилась память платы, а один из светодиодов на плате (LD2) начал мигать красно-зелёным светом.

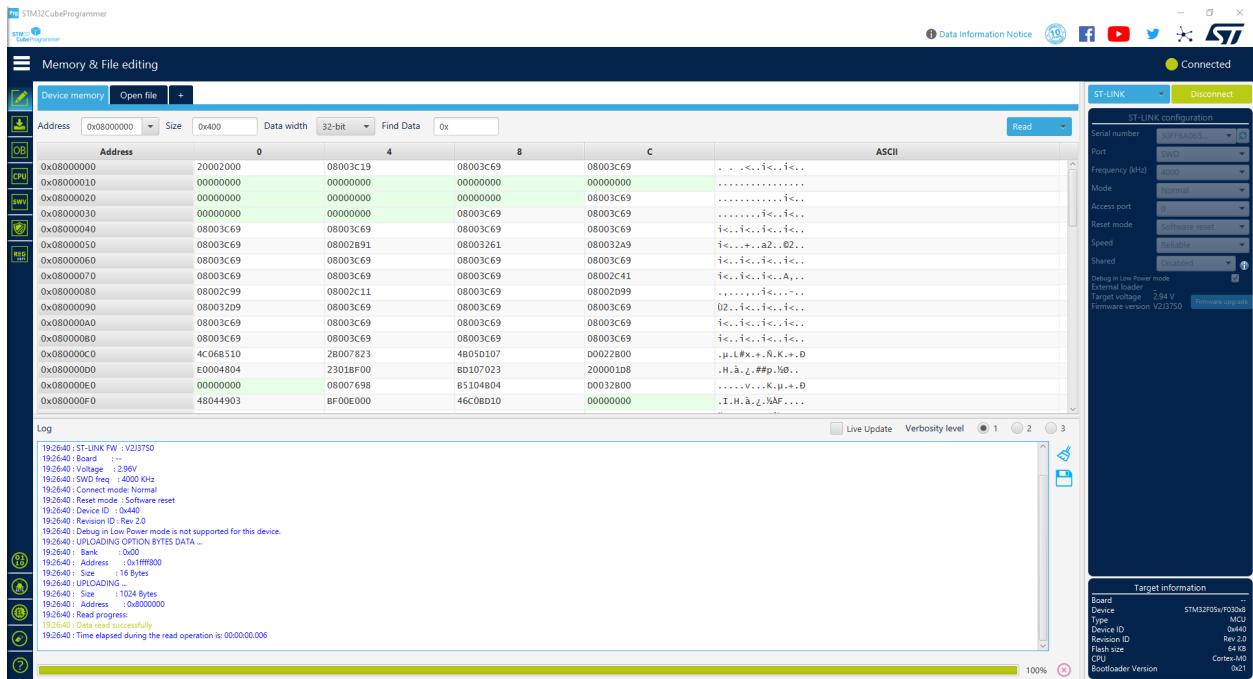


Рисунок 3.26 – Окно программы после установки соединения

Нажимаем на кнопку Erasing & Programming (вторая зелёная кнопка сверху) и в части Download указываем путь к файлу с расширением elf. Ставим галочки напротив Verify programming и Run after programming. В части Automatic Mode ставим галочку напротив Download file.

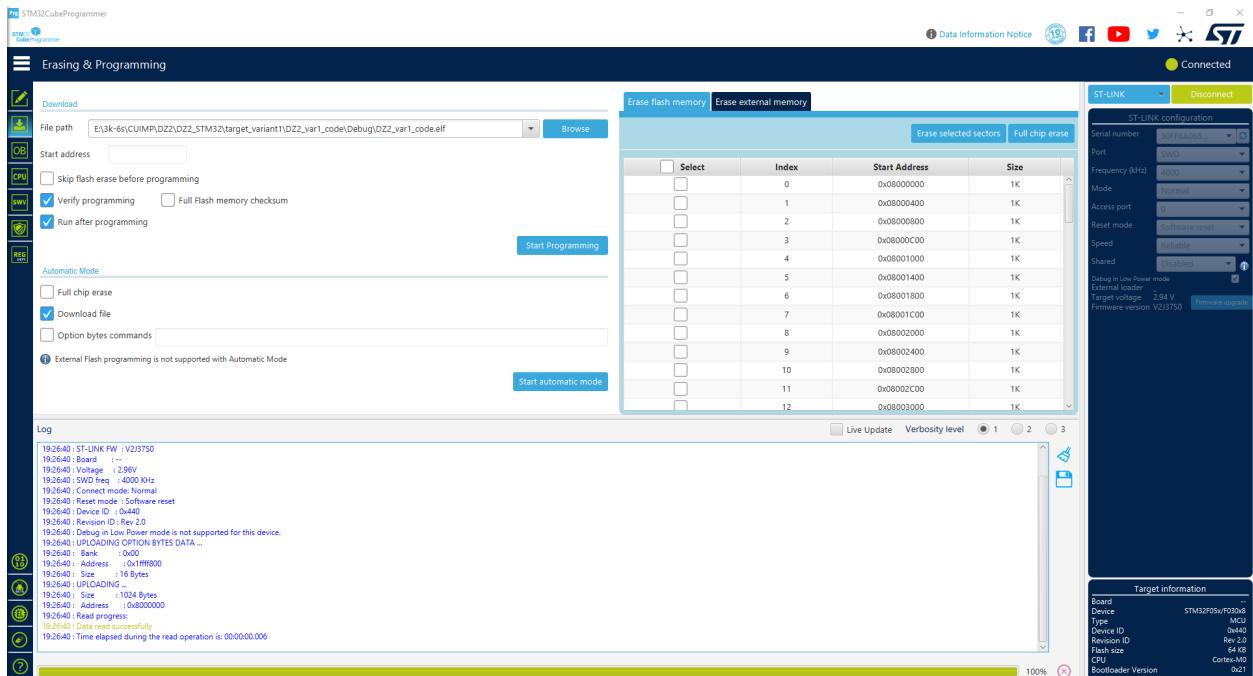


Рисунок 3.27 – Указание пути и проставление галочек

Теперь нажимаем на кнопку Start Programming. Ждём, пока произойдёт программирование платы. По окончании процесса должны высветиться 3 сообщения об успешном окончании процесса программирования: “Start operation achieved successfully”, “Download verified successfully”, “File download complete”.

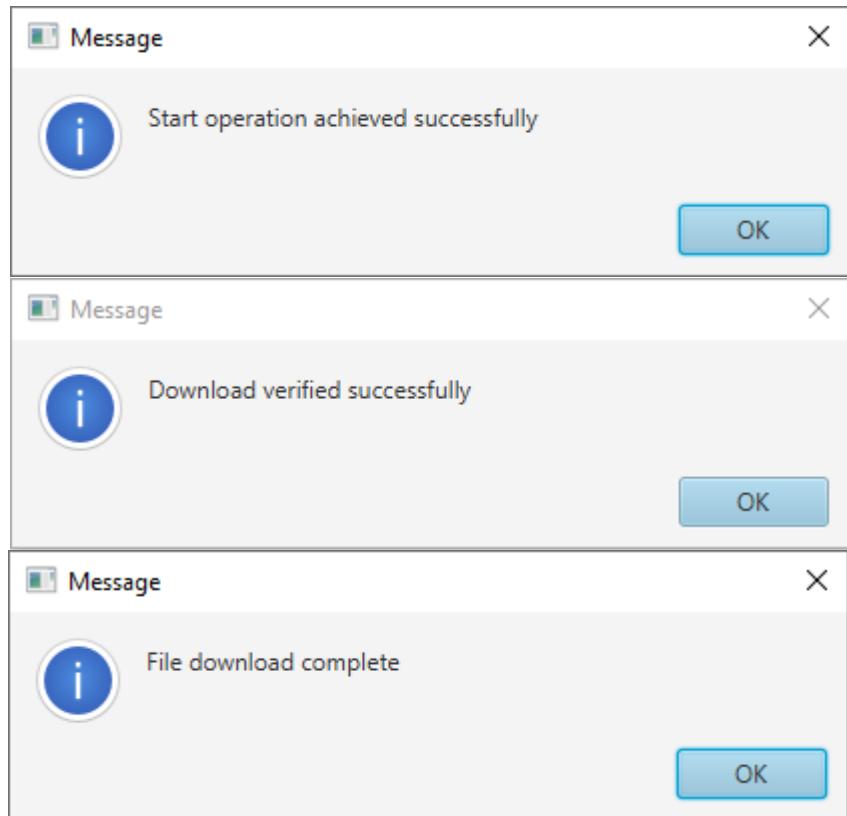


Рисунок 3.28 – Сообщения

Теперь в нашу плату зашит код для варианта 1, который мы писали, собирали и компилировали с помощью STM32CubeIDE.

Нажмём Disconnect в правом верхнем углу, выберем серийный номер второй платы, снова нажмём Connect. Т.к. драйвера платы обновлялись, с соединением не должно возникнуть проблем. Далее переходим на вкладку Erasing & Programming, меняем путь к файлу с расширением elf (выбираем файл для варианта 2), ставим нужные галочки и снова нажимаем Start Programming. Теперь в обе наши платы зашит код под каждый вариант.

Можно в режиме реального времени проверить работу плат в соответствии с кодом, просматривая память какой-нибудь одной подключенной к программе платы во вкладке Memory & File editing.

STM32CubeMonitor

А можно запустить STM32CubeMonitor – программу, рисующую графики. Правда, одновременно и отслеживать память платы, и рисовать для неё графики не получится – если плата подключена в STM32CubeProgrammer, то STM32CubeMonitor её не видит, и наоборот.

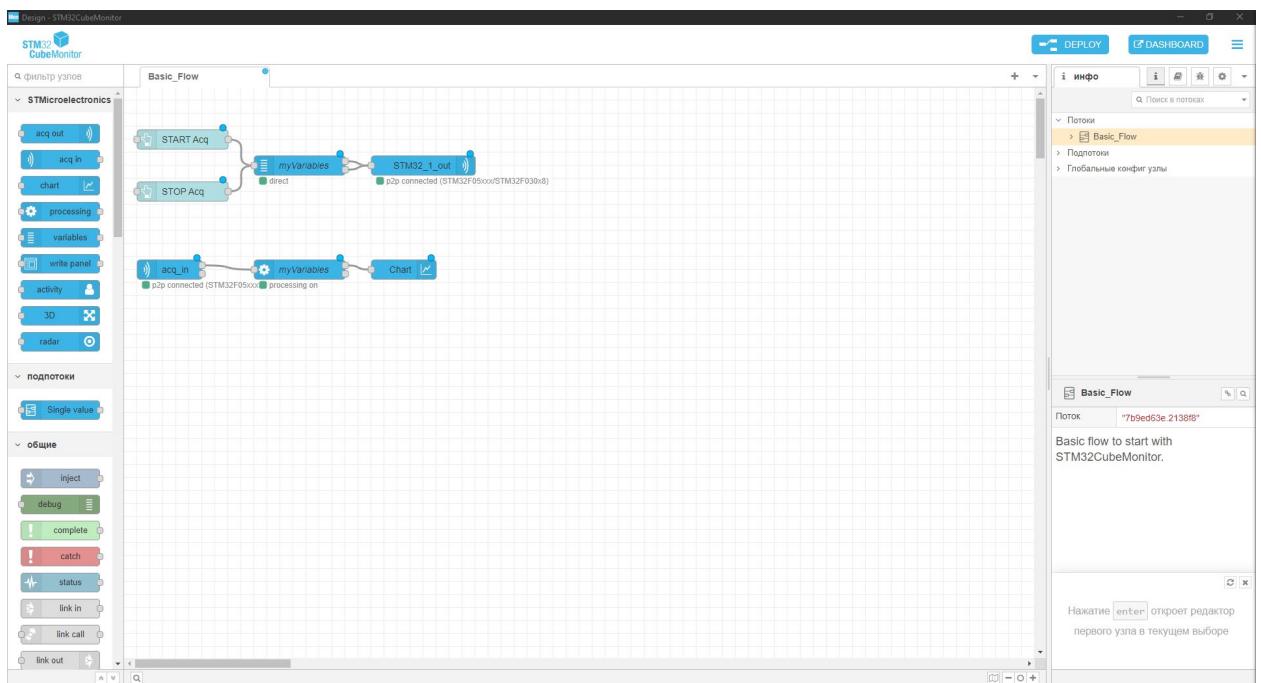
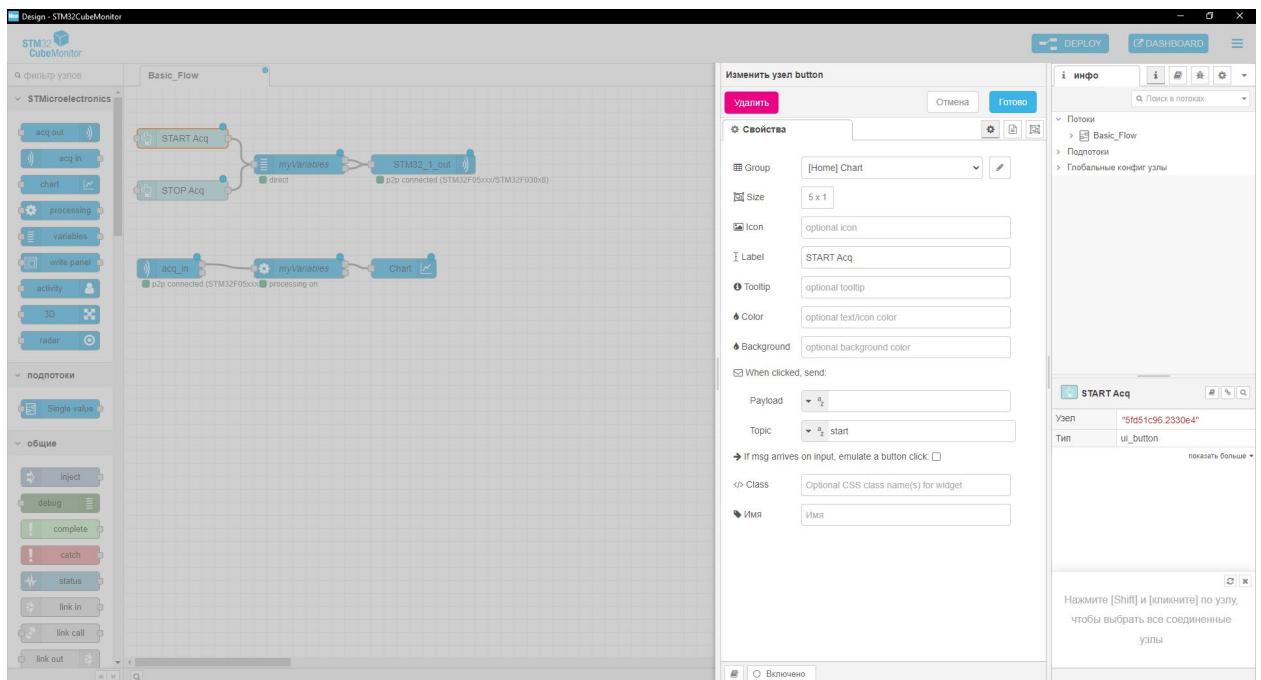


Рисунок 3.29 – Окно программы STM32CubeMonitor с собранной ранее схемой для вывода графиков

Чтобы рисовать графики, необходимо собрать схему, приведённую на рисунке 3.29.

Добавление элементов (цветных прямоугольников) осуществляется перетаскиванием таковых из списка в левой части окна программы в рабочую область (ко-торая имеет сетку тетрадного листа в клетку).

START Acq и STOP Acq – кнопки.



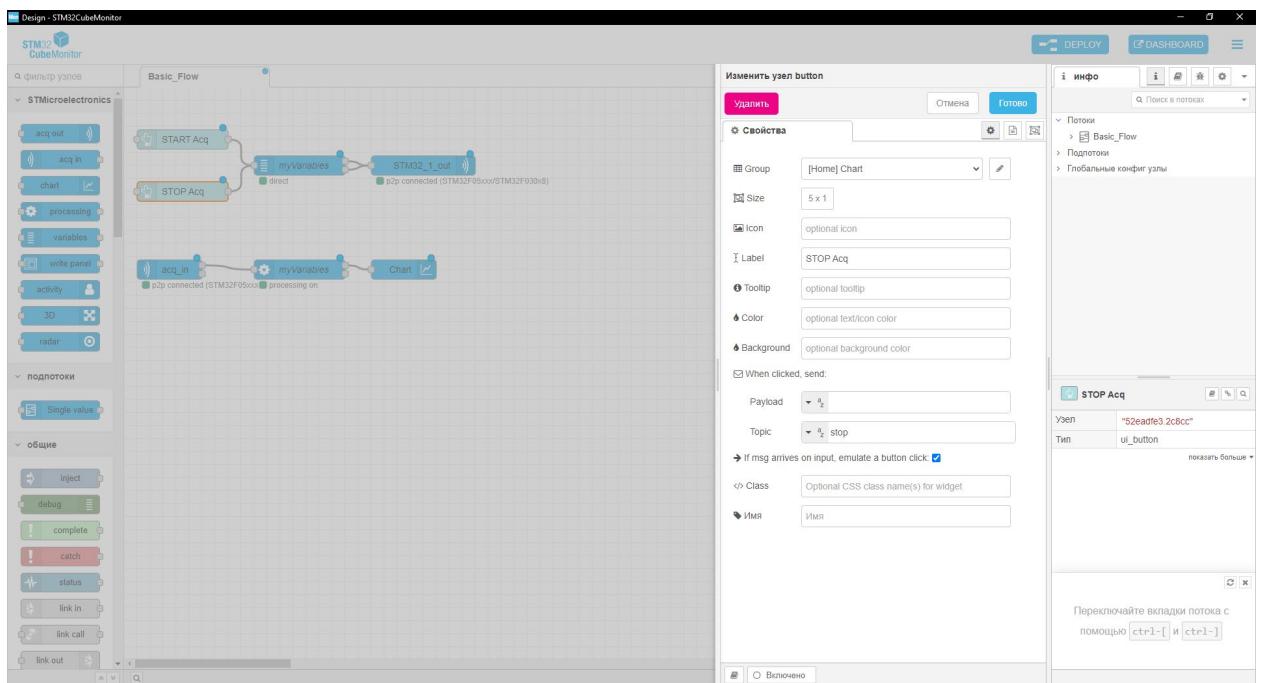


Рисунок 3.30 – Свойства кнопок START Acq и STOP Acq

Эти кнопки подсоединяются к элементу myVariables – элементу, в котором мы задаём переменные, вывод на график которых нас интересует. Для его настройки нужно дважды нажать ЛКМ по нему, далее нажать на значок карандаша напротив Executable, а далее – указать имя узла exe-config, путь, в котором располагается нужный нам файл с расширением elf и сам файл с этим расширением. Ниже, в списке Variable List, после указания файла с расширением elf высветятся все возможные переменные. Ставим напротив нужных для построения графика галочки. Для вывода нас интересуют следующие переменные: program_state, receive_state, transfer_state, count_of_blink, button_state, rattle_check, press_check, count_short_press.

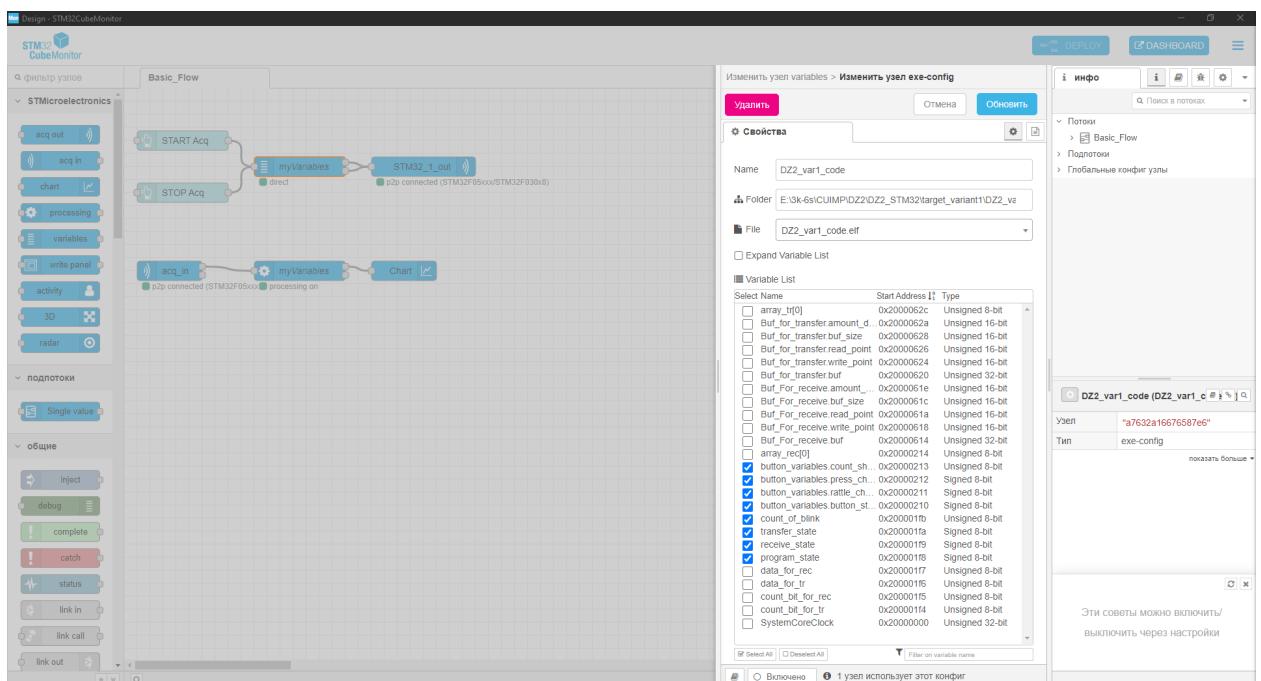


Рисунок 3.31 – Окно с выбором переменных для вывода на график

Далее нажимаем сверху кнопки Обновить --> Готово.

Элемент myVariables подсоединяется к элементу acq out. При нажатии на него 2 раза ЛКМ открывается окно с настройкой его свойств. Здесь нажимаем на иконку карандаша напротив Probe Config – откроется окно. В нём, напротив Probe выбираем серийный номер одной из плат, далее меняем название узла Name. Protocol и Frequency не трогаем.

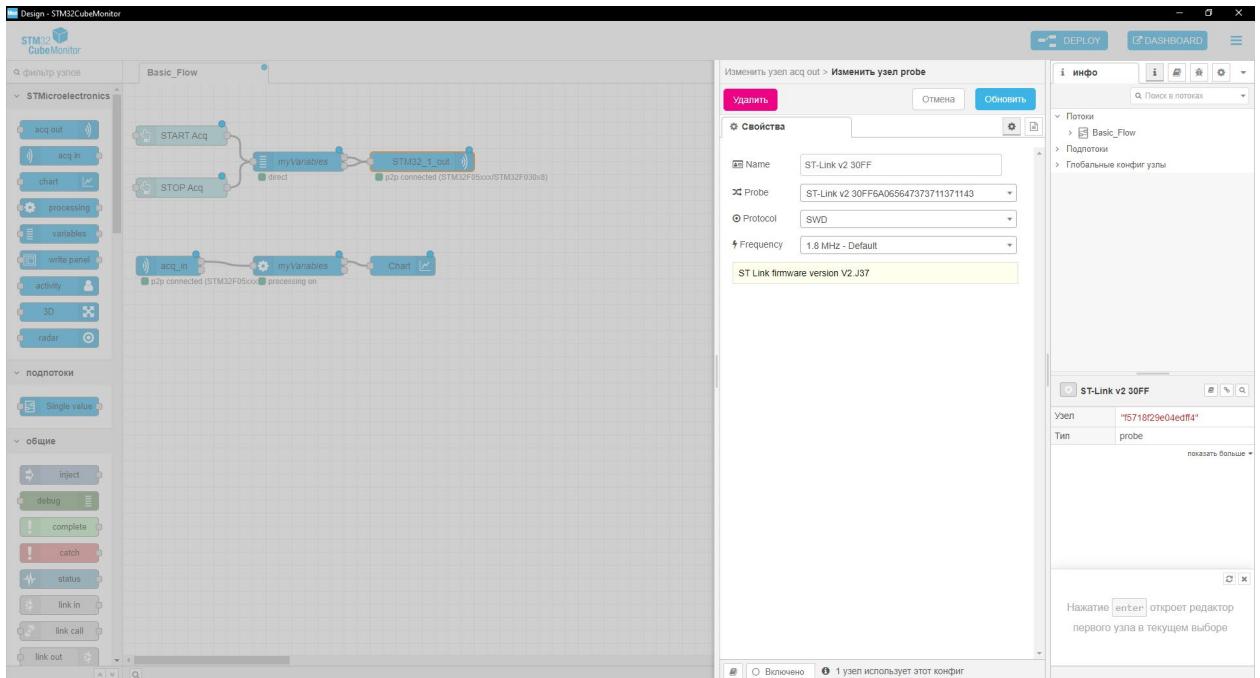


Рисунок 3.32 – Настройка свойств узла probe

Далее нажимаем Обновить. Теперь напротив Probe Config в выпадающем списке выбираем наш узел с заданным названием Name. Меняем название acq out на то, которое нам удобно. Если нажать на иконку лампочки напротив Probe Config, то один из светодиодов на плате (LD2) начнёт мигать красно-зелёным светом.

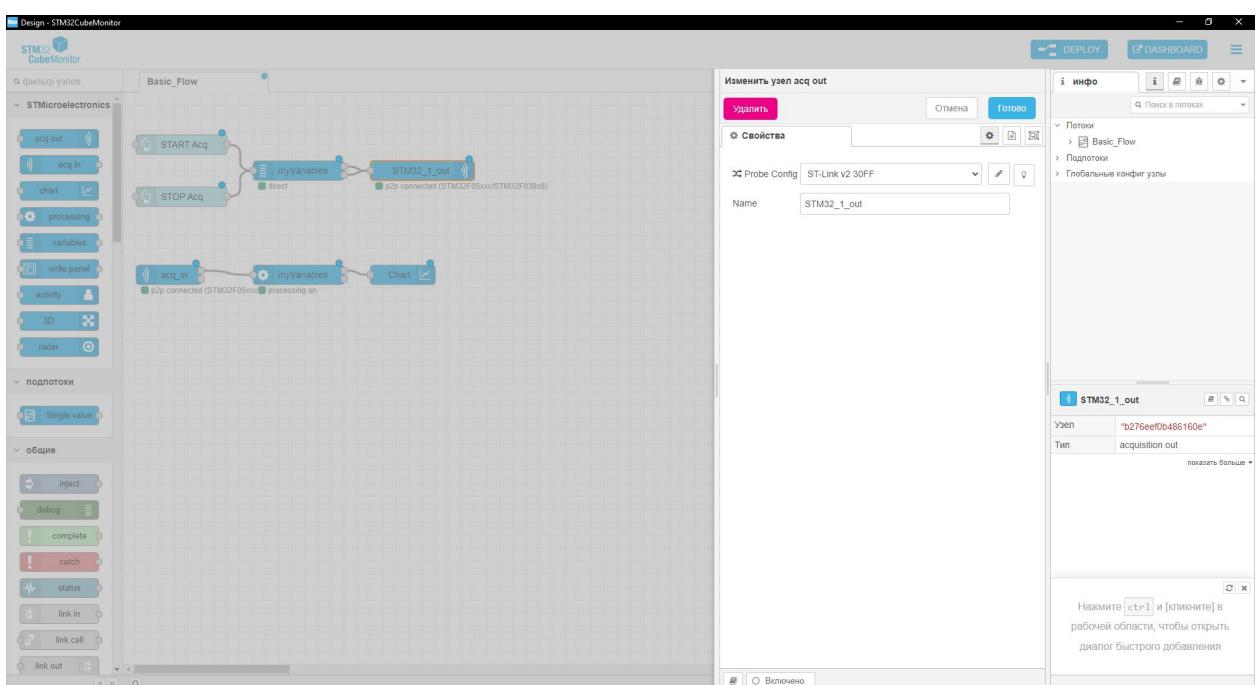
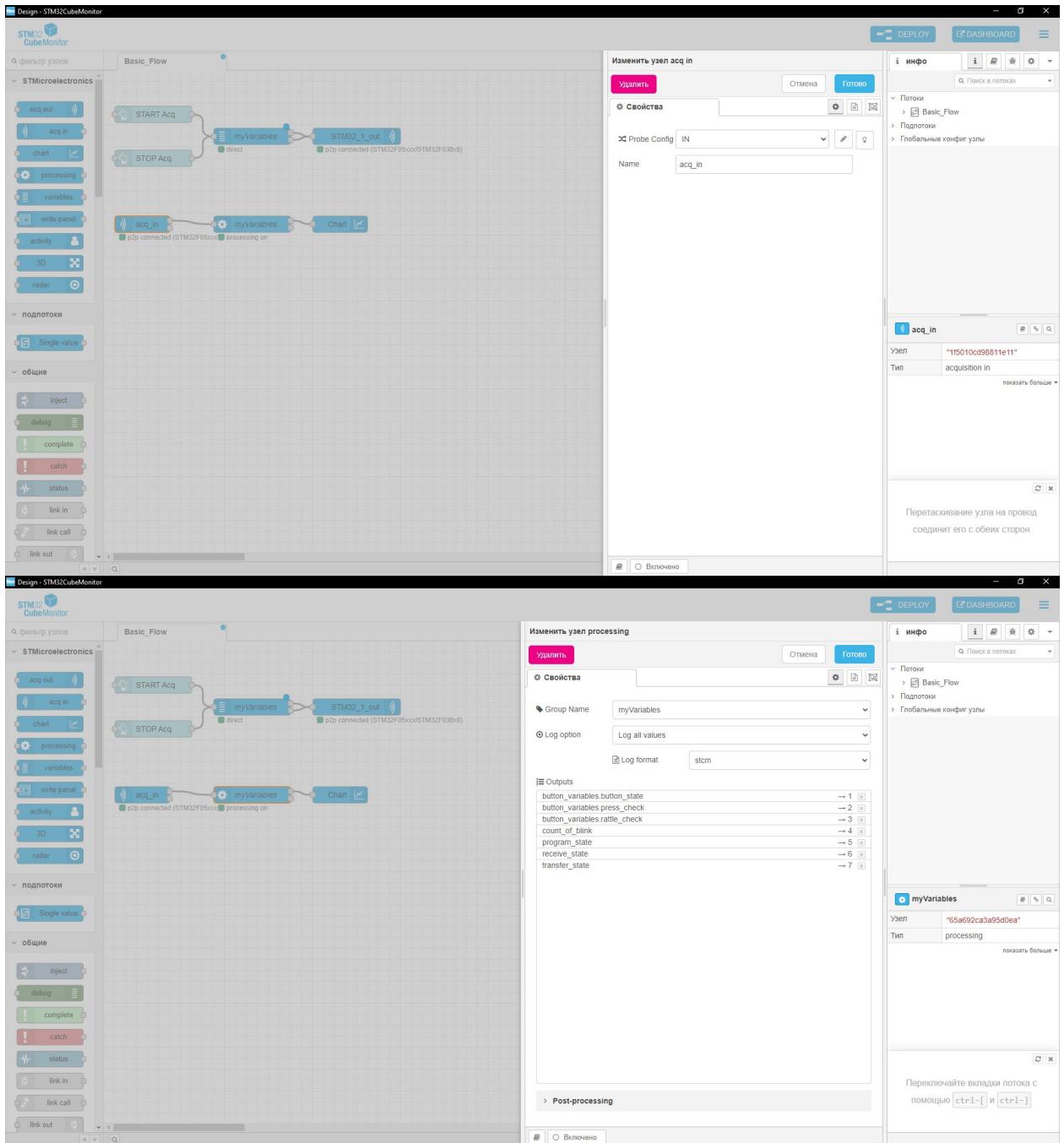


Рисунок 3.33 – Настройка узла acq out

Нажимаем Готово.

Теперь ниже выстраиваем систему из трёх элементов: acq in --> processing --> chart. Свойства приведены ниже.



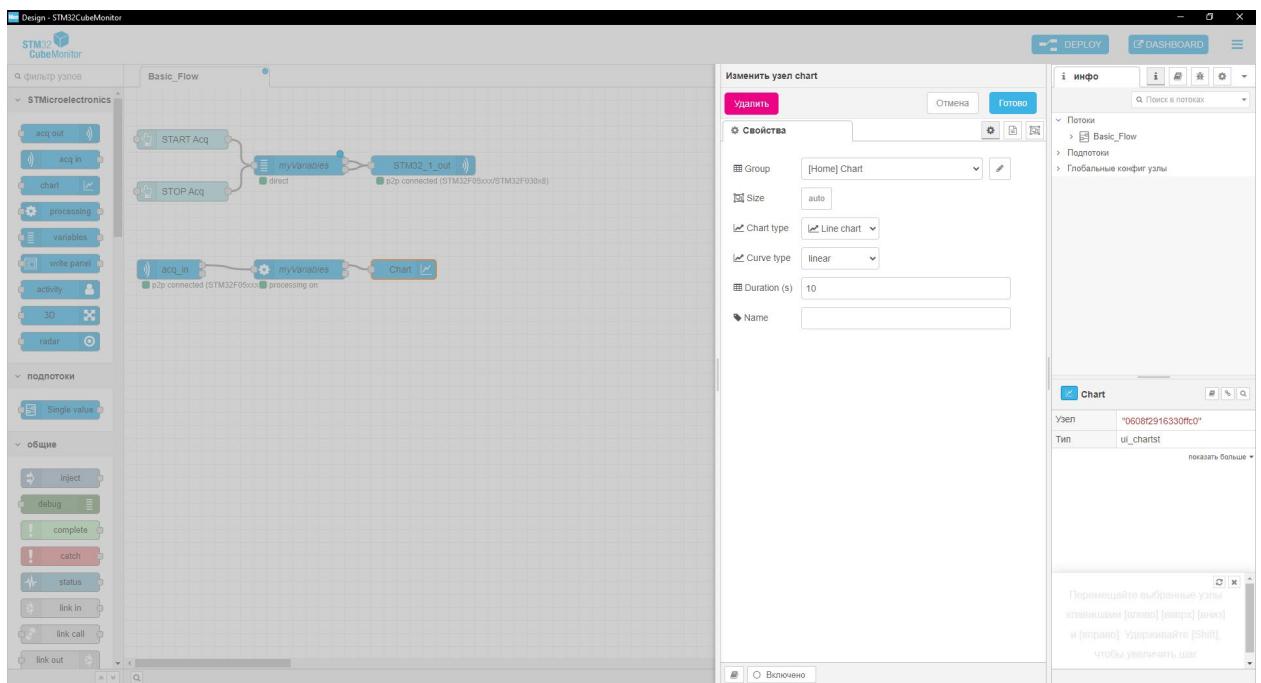


Рисунок 3.34 – Свойства 3-х соединённых узлов

Теперь справа сверху нажимаем Deploy. Сверху появится сообщение: “Успешно развернуто”. Теперь нажимаем Dashboard. Появится окно с двумя большими кнопками и системой координат. Развернём его.

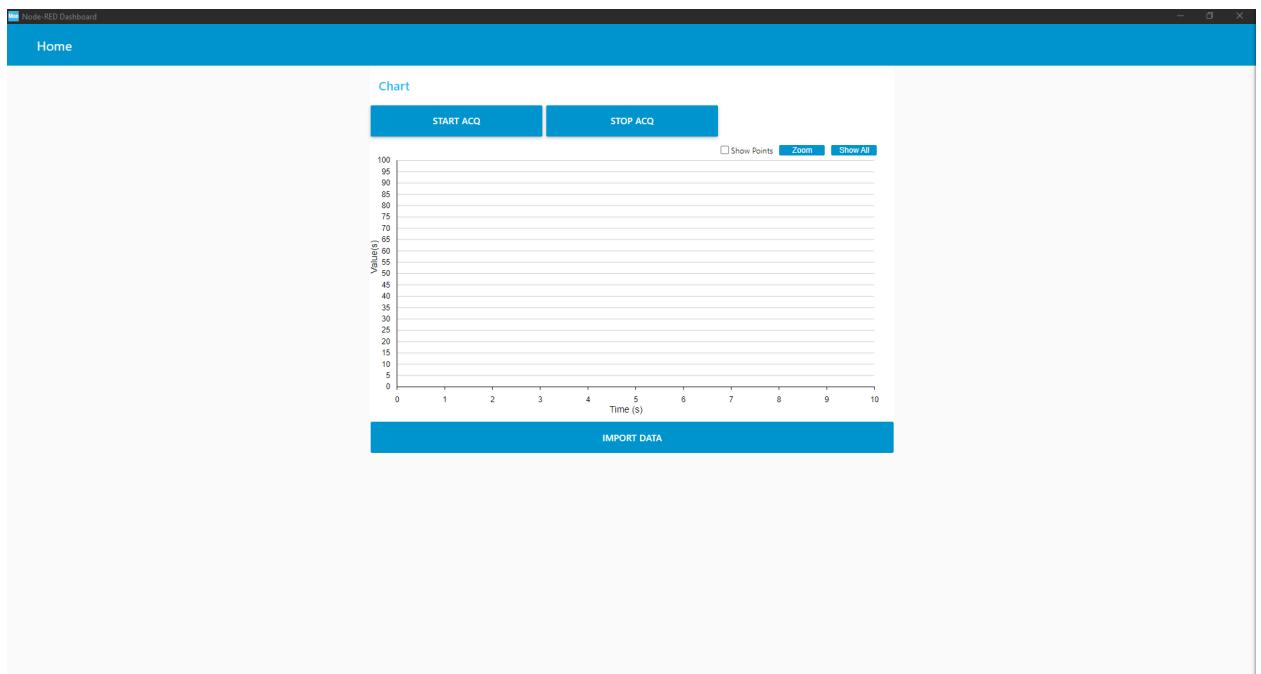


Рисунок 3.35 – Появившееся окно

Теперь нажимаем на кнопку START ACQ, и программа начнёт в режиме реального времени отслеживать изменение переменных, используемых кодом, который зашит в плату, и выводить их в удобном виде на график.

Напомню принцип работы нашей платы в соответствии с кодом:

Факт нажатия на кнопку --> после 25 мс отключается проверка на дребезг контактов.

Если удерживаем кнопку 3 секунды = 1 длинное нажатие --> установка режима настройки для платы, сброс счётчика количества нажатий до нуля, включение таймера для мигания светодиода. Если 1 короткое нажатие – запись данных во FLASH. Если 2 коротких нажатия – запуск мигания светодиода на 2-м МК (3 раза). Если >2 нажатий – перезапуск кнопки.

Если не удерживаем кнопку: если 1 короткое нажатие – передача данных; если 2 коротких нажатия – получение данных с ПК через USART; если 3 коротких нажатия – передача данных на ПК через USART; если >3 нажатий – перезапуск кнопки.

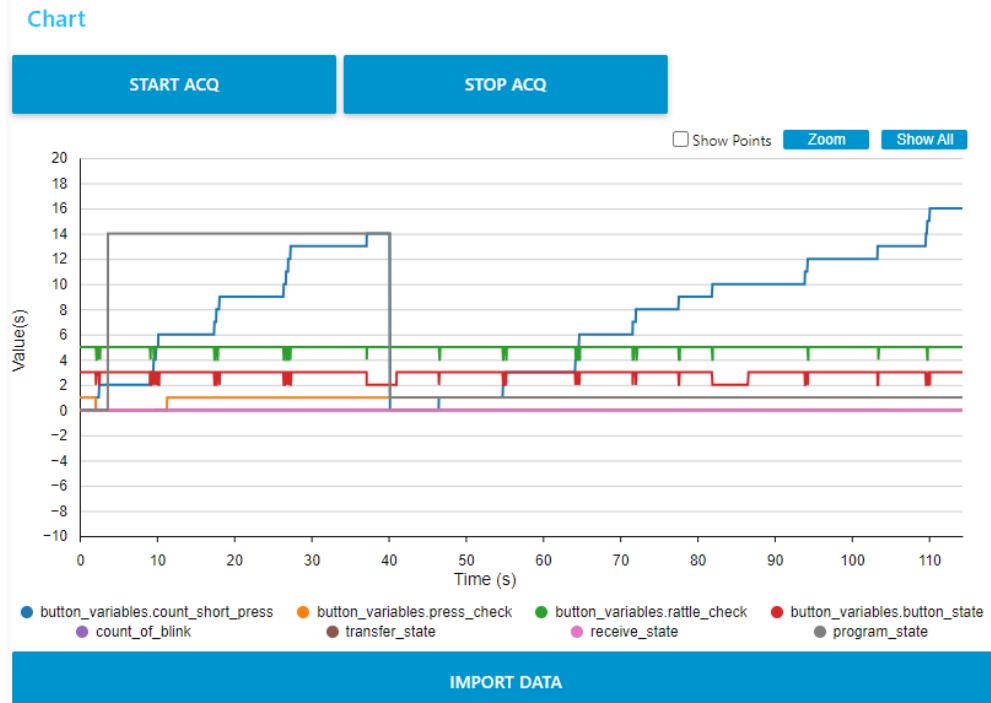


Рисунок 3.36 – Графики для переменных, используемых кодом, зашитым в плату – общий вид



Рисунок 3.37 – Графики для переменных, используемых кодом, зашитым в плату – приближенно

Какая из программ оказалась лучше для отладки

Проведя ряд опытов с платами и программами, я пришёл к выводу, что STM32CubeMonitor годится только для построения графиков на основе переменных, которые изменяются при взаимодействии непосредственно с платой (т.е. нажатие кнопки). Получение платой данных через эту программу невозможно, а значит, невозможно и отображение всего спектра функций, которые должна выполнять плата из-за вшитого в неё кода. В то же время плата, которая работала, будучи подключенной к STM32CubeProgrammer: исправно получала данные с ПК через USART (при этом горел голубой светодиод); исправно передавала данные на ПК через USART (при этом горел зелёный светодиод); исправно передавала данные на другую плату (оба светодиода загорелись на короткий промежуток времени); исправно уходила в режим настройки после зажатия кнопки (оба светодиода начинали мигать), а уже в этом режиме записывала данные во FLASH и запускала мигания светодиодов на 2-й плате 3 раза.