|  |  |
| --- | --- |
|  | **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 Семеренко Д.А. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(должность) (фамилия и.о.) (подпись, дата)*

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

2024 г.

Оглавление

[1 Оборудование. 4](#_Toc169549792)

[1.1 stm32f051 4](#_Toc169549793)

[1.2 Дополнительное оборудование 5](#_Toc169549794)

[2 Задание по вариантам и условие ЛР№2. 6](#_Toc169549795)

[2.1 Вариант 1 – Цель 6](#_Toc169549796)

[2.2 Вариант 1 – Условия 6](#_Toc169549797)

[Параллельный интерфейс (чтение) 6](#_Toc169549798)

[Последовательный интерфейс (запись) 6](#_Toc169549799)

[2.3 Вариант 2 – Цель 6](#_Toc169549800)

[2.4 Вариант 2 – Условия 6](#_Toc169549801)

[Параллельный интерфейс (запись) 6](#_Toc169549802)

[Последовательный интерфейс (чтение) 6](#_Toc169549803)

[2.5 Общие замечания 7](#_Toc169549804)

[2.6 Условие лабораторной работы №2 8](#_Toc169549805)

[3 Выполнение. 9](#_Toc169549806)

[3.1 Предисловие 9](#_Toc169549807)

[3.2 Создание проекта 9](#_Toc169549808)

[3.3 Код 17](#_Toc169549809)

[stm32f0xx.h 17](#_Toc169549810)

[buffer.h 21](#_Toc169549811)

[buffer.c 22](#_Toc169549812)

[FLASH.h 23](#_Toc169549813)

[FLASH.c 23](#_Toc169549814)

[Variables.h 25](#_Toc169549815)

[Variables.c 25](#_Toc169549816)

[Button\_init.h 26](#_Toc169549817)

[Button\_init.c 26](#_Toc169549818)

[Button\_cmd.h 27](#_Toc169549819)

[Button\_cmd.c 28](#_Toc169549820)

[Periphery\_for\_transfer\_and\_receive\_init.h 31](#_Toc169549821)

[Periphery\_for\_transfer\_and\_receive\_init.c 31](#_Toc169549822)

[USART\_DMA\_setup.h 33](#_Toc169549823)

[USART\_DMA\_setup.c 33](#_Toc169549824)

[Serial\_receive\_and\_parallel\_transfer.h 35](#_Toc169549825)

[Serial\_receive\_and\_parallel\_transfer.c 36](#_Toc169549826)

[main.c 42](#_Toc169549827)

[3.4 Сборка и компиляция 42](#_Toc169549828)

[3.5 Обновление драйверов платы 46](#_Toc169549829)

[3.6 Выполнение 2-го варианта перед отладкой 48](#_Toc169549830)

[3.7 Отладка 50](#_Toc169549831)

[STM32CubeProgrammer 50](#_Toc169549832)

[STM32CubeMonitor 52](#_Toc169549833)

[Какая из программ оказалась лучше для отладки 59](#_Toc169549834)

# 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

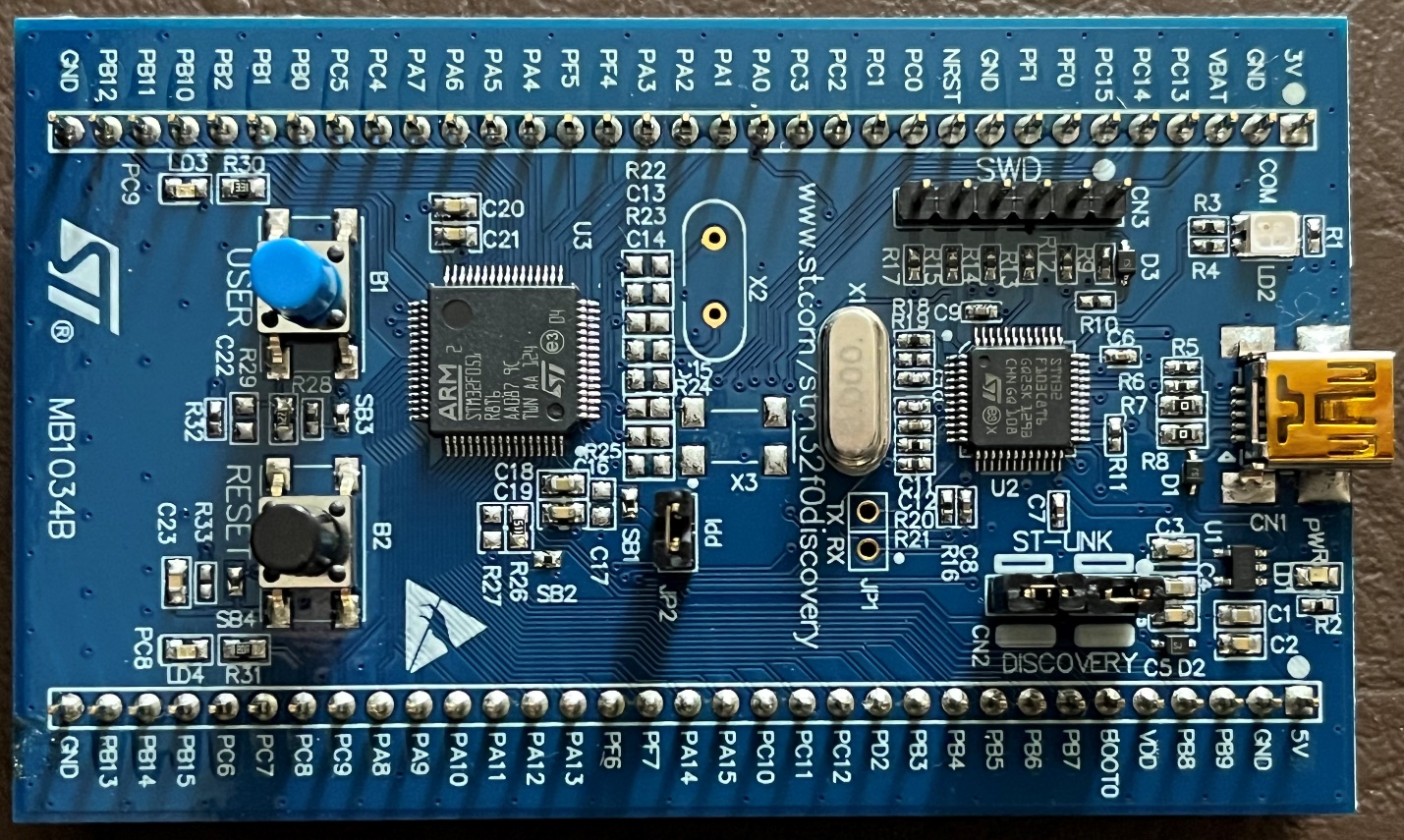


Рисунок 1.1 – stm32f051

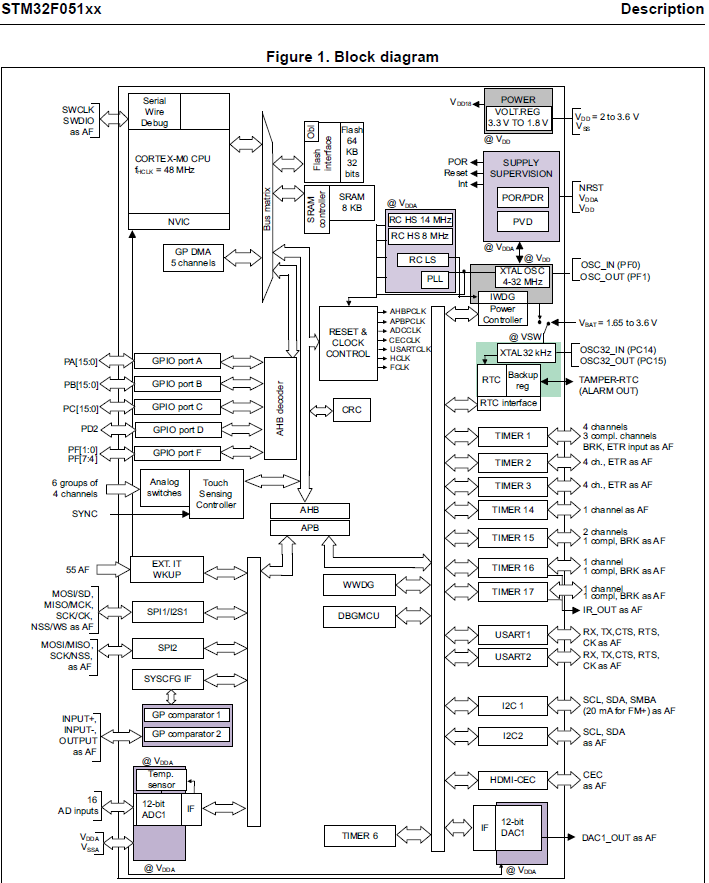


Рисунок 1.2 – Схема чипа

## 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.

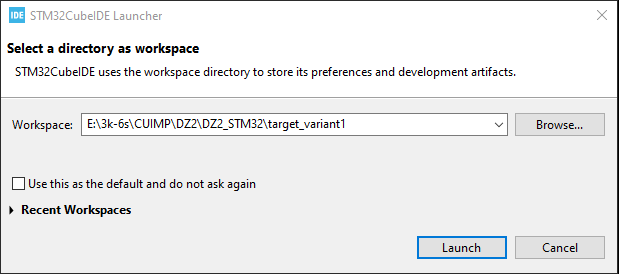


Рисунок 3.1 – Выбор директории для работы

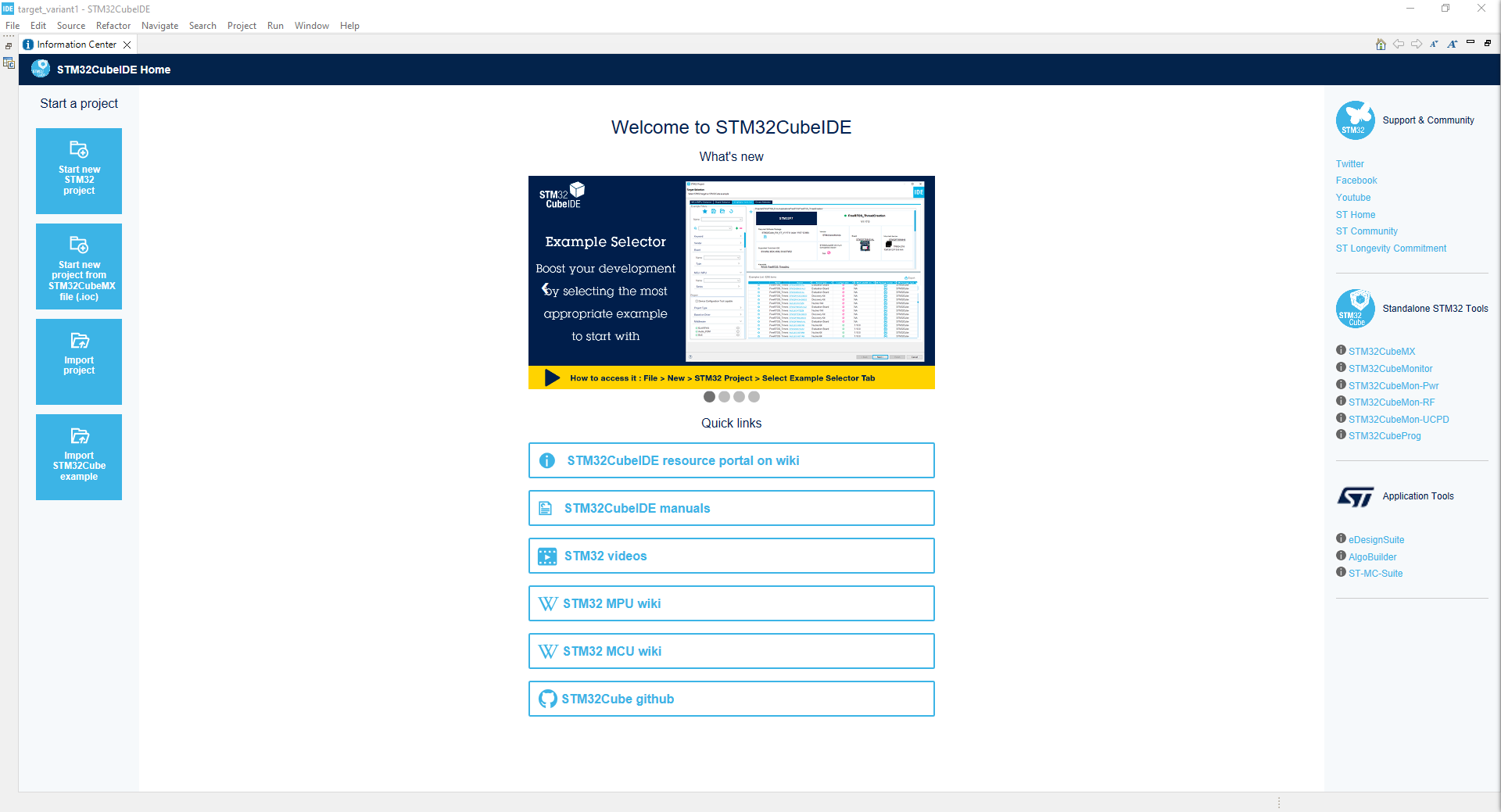


Рисунок 3.2 – Окно после выбора директории запуска STM32CubeIDE

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

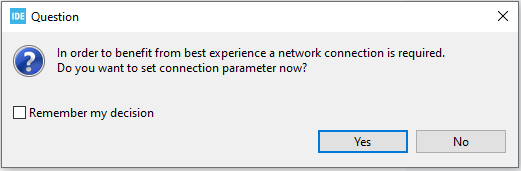


Рисунок 3.3 – Иконка с вопросом

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

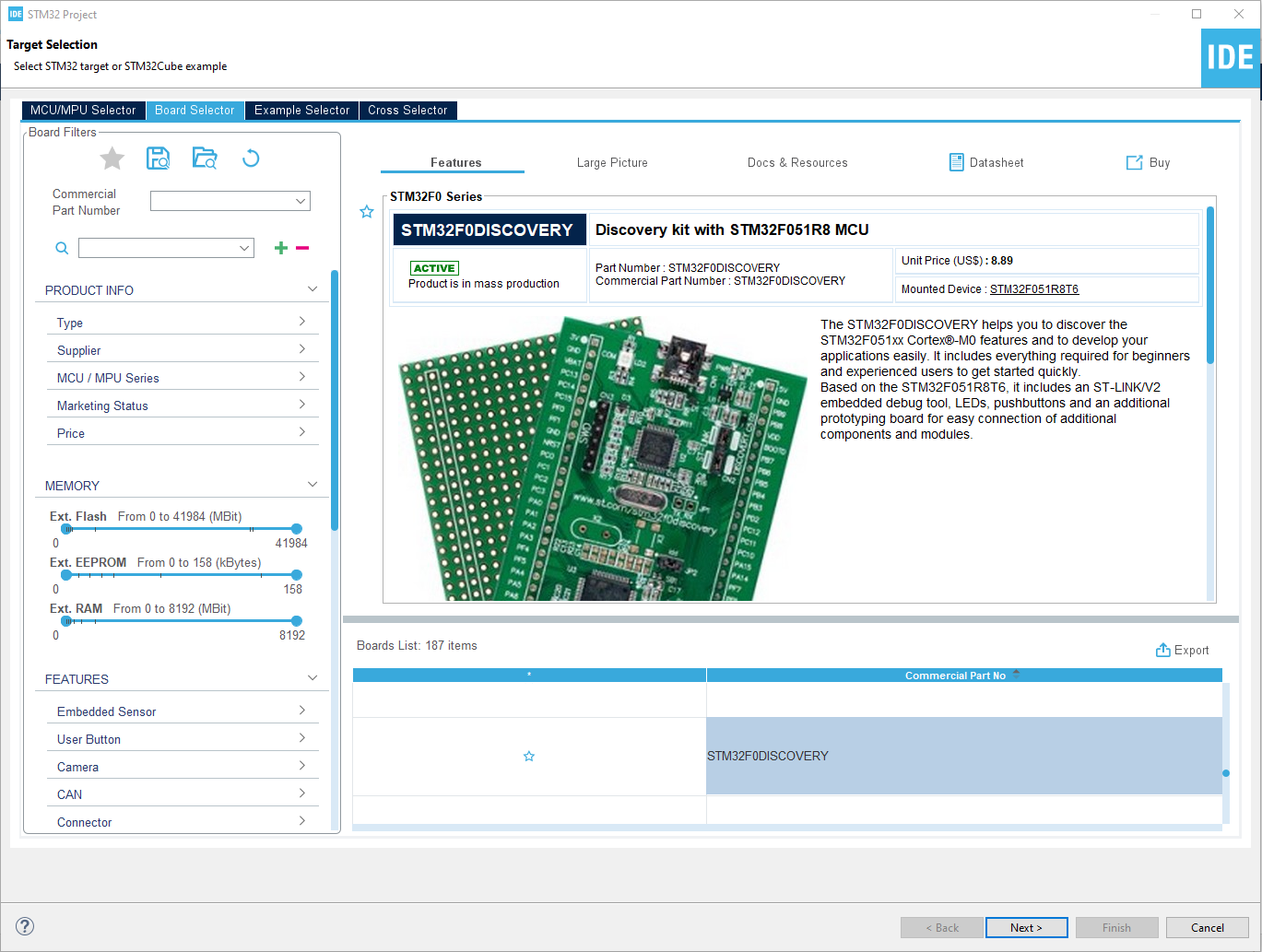


Рисунок 3.4 – Выбор платы во всплывшем окне

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

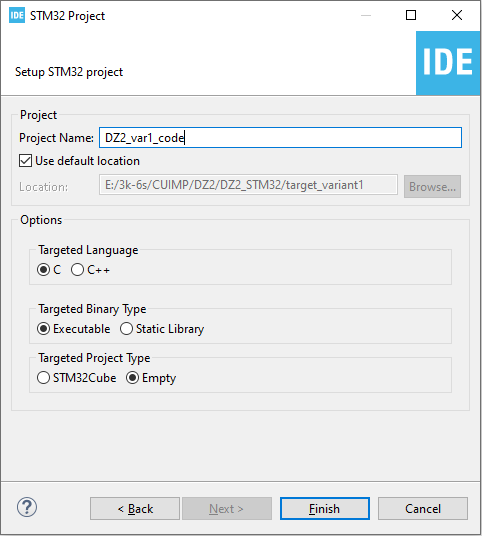


Рисунок 3.5 – Называние проекта и смена галочки в новом всплывшем окне

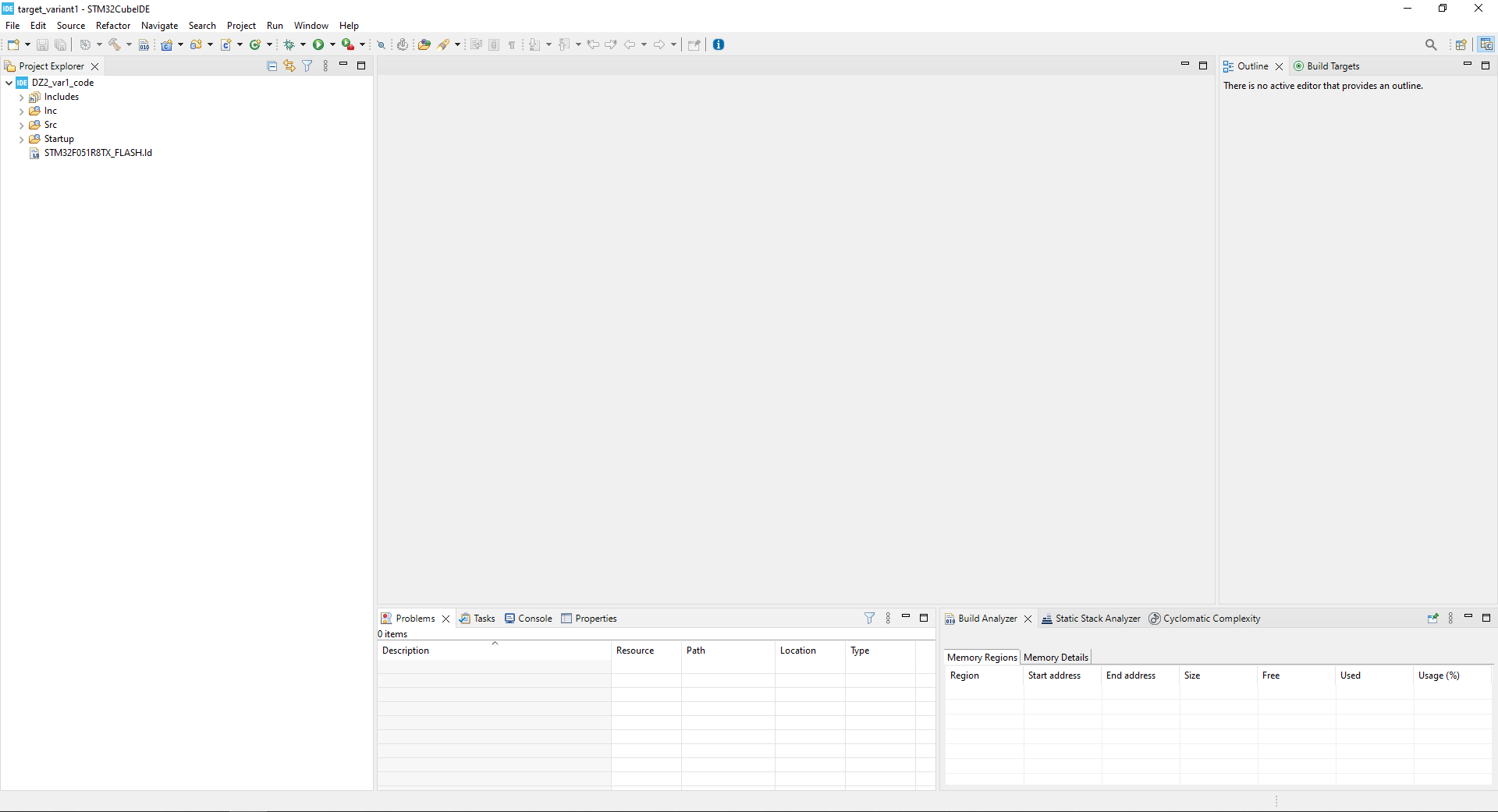


Рисунок 3.6 – Окно после создания проекта

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

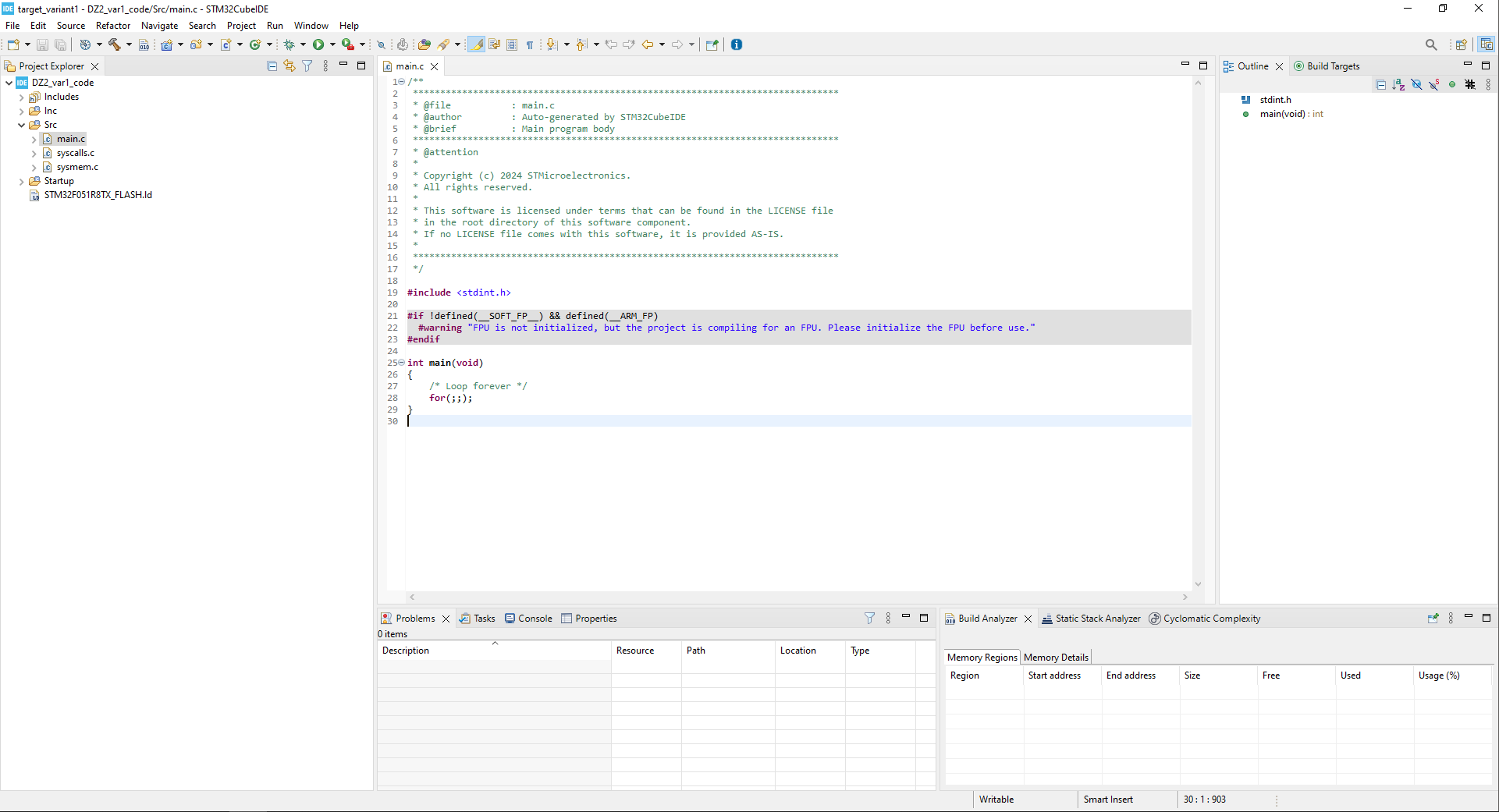


Рисунок 3.7 – Открытие файла main.c

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.

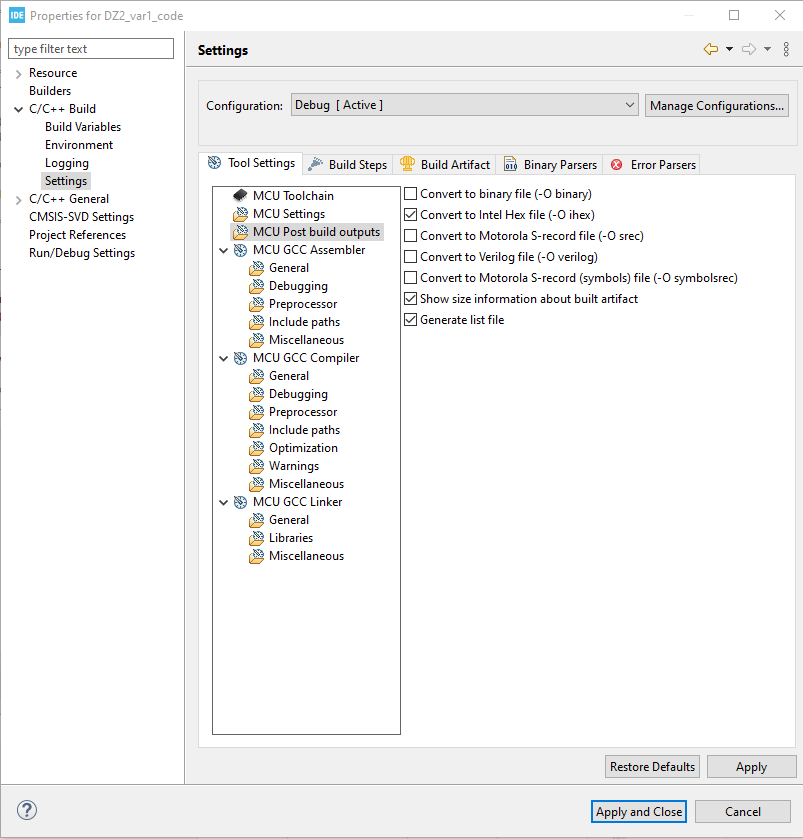


Рисунок 3.8 – Ставим галочку там, где нужно

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

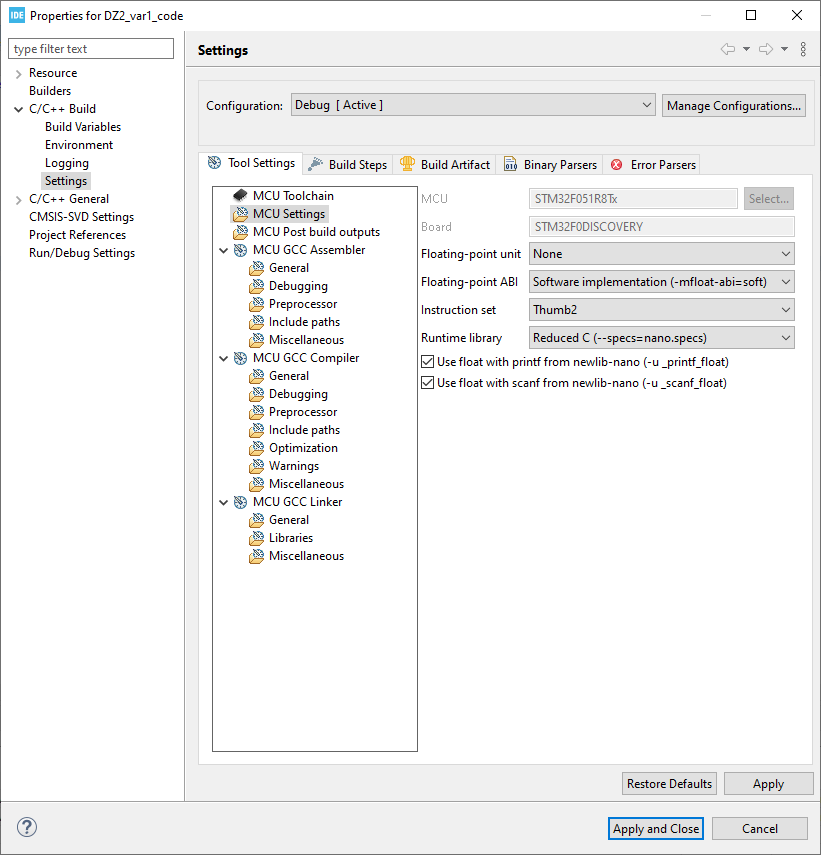


Рисунок 3.9 – Ставим две галочки там, где нужно

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.

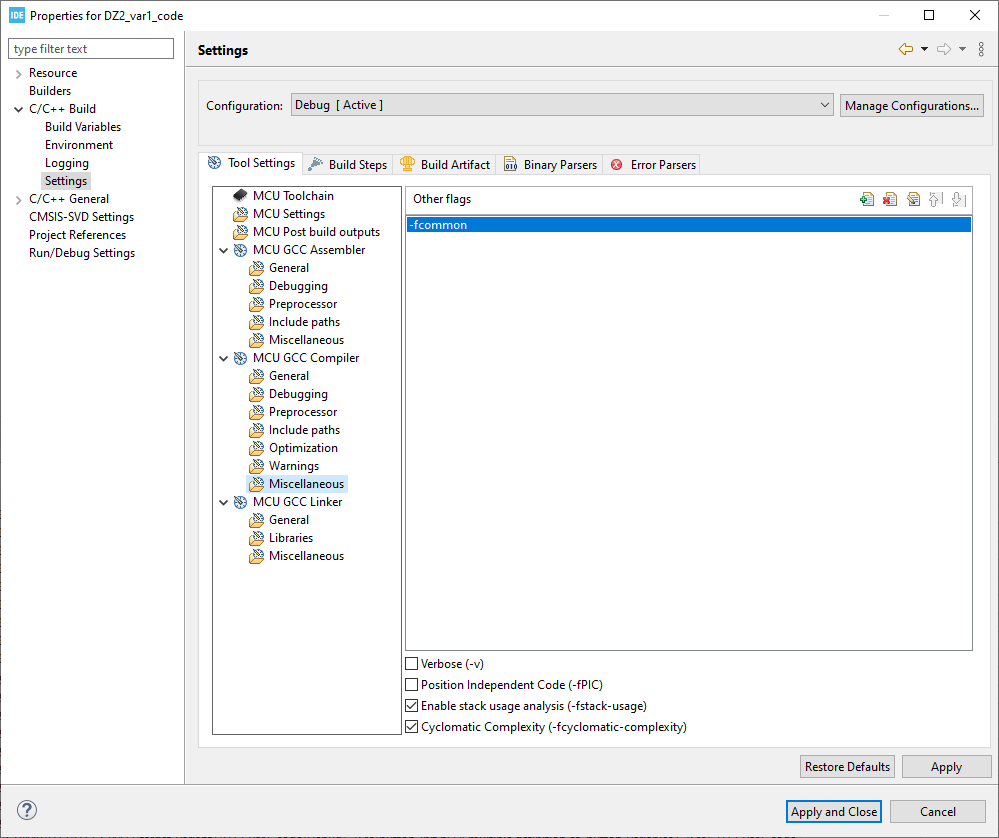


Рисунок 3.10 – Введение очень важной команды куда надо

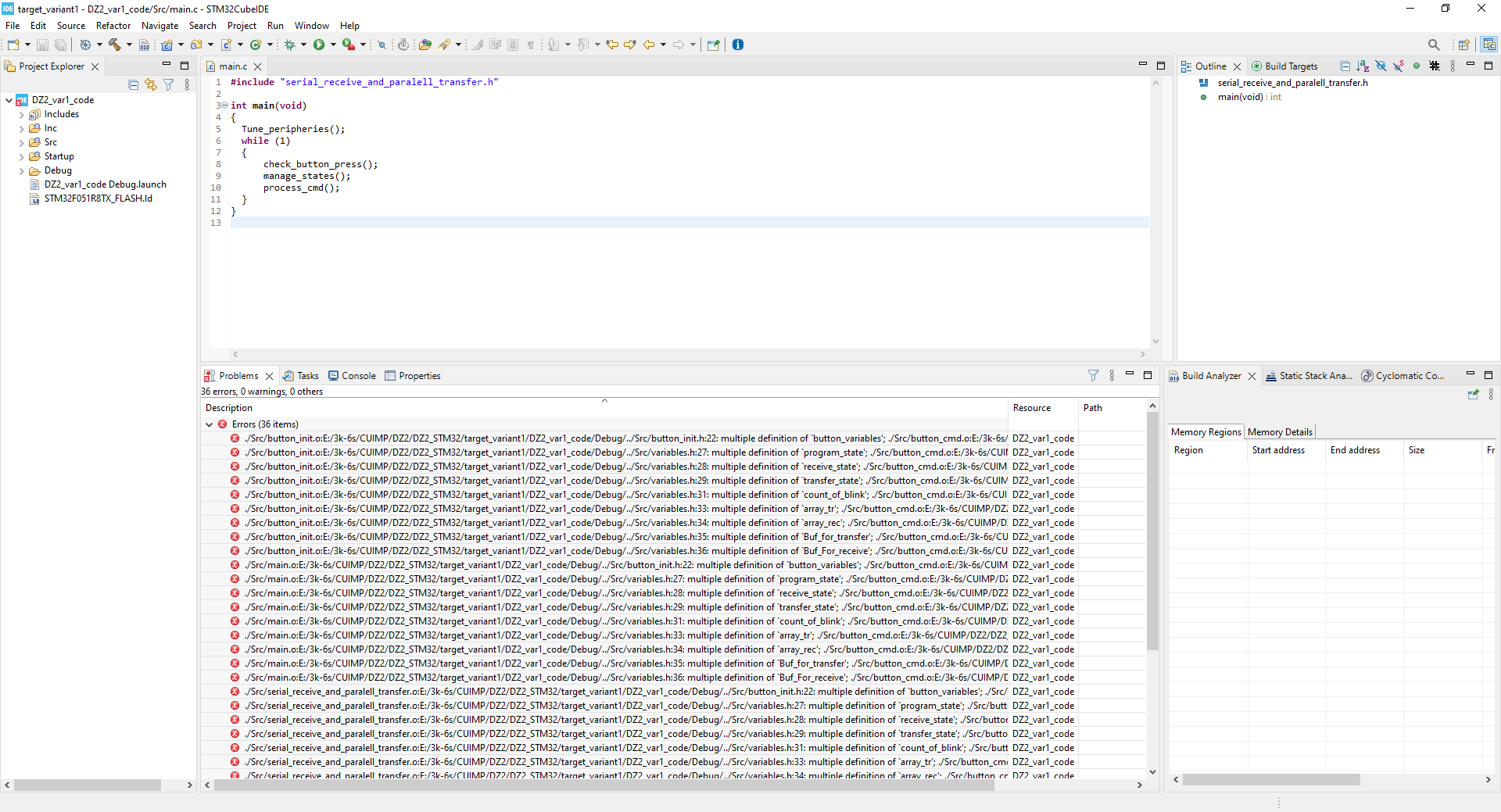


Рисунок 3.11 – Вот что будет, если команду не ввести – 36 ошибок после сборки

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 microcontrollers where the Flash memory ranges between 64 and 128 Kbytes) \*/

/\* #define STM32F072xB \*/ /\*!< STM32F072x8, STM32F072xB Devices (STM32F072xx microcontrollers 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 microcontrollers 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.

\*/

**#if** !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 \*/

**#ifdef** \_\_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 буфера

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 буфере = 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 объёма 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, заполнен 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); // чтение 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\_MODER0; // 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 понижающего 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 нажатия

**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\_variables.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\_UNPRESSED*) {

**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 более 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 коротких 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

/\*

#ifdef 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

\*/

**#ifdef** 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"

/\*

#ifdef 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 MK

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\_IRQn);

NVIC\_SetPriority(EXTI2\_3\_IRQn, 9);

NVIC\_EnableIRQ(EXTI4\_15\_IRQn);

NVIC\_SetPriority(EXTI4\_15\_IRQn, 1);

}

#endif

\*/

**#ifdef** 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\_Channel2->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\_PARALELL\_TRANSFER\_H\_

**#define** SERIAL\_RECEIVE\_AND\_PARALELL\_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**

/\*

#ifdef 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\_paralell\_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; /\*

#ifdef PAR\_TRANSFER\_AND\_SERIAL\_RECEIVE

transfer\_state = TRANSFER\_8\_BIT;

#endif \*/

**#ifdef** 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; /\*

#ifdef 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; /\*

#ifdef PAR\_TRANSFER\_AND\_SERIAL\_RECEIVE

receive\_state = RECEIVE\_BIT;

#endif \*/

**#ifdef** SER\_TRANSFER\_AND\_PAR\_RECEIVE

transfer\_state = *TRANSFER\_BIT*;

**#endif**

}

}

**#ifdef** 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 достичь другого MK

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), поскольку 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 завершена

**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 переменные 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 мигания 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**

/\*

#ifdef 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(); /\*

#ifdef 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**() { /\*

#ifdef 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 передаются

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 MK

}

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 \*/

**#ifdef** 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 действия

**void** **manage\_states**() {

**switch**(program\_state) {

**case** *TRANSFER\_DATA*:

program\_state = *PROGRAMMING*;

ReadFromFlash(PAGE\_FOR\_TRANSFER, Buf\_for\_transfer.buf, BUFFER\_SIZE); /\*

#ifdef PAR\_TRANSFER\_AND\_SERIAL\_RECEIVE

EXTI->SWIER |= EXTI\_SWIER\_SWI2; // вызов прерывания на EXTI2 и начало передачи данных

#endif \*/

**#ifdef** 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 MK

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*; /\*

#ifdef PAR\_TRANSFER\_AND\_SERIAL\_RECEIVE

EXTI->SWIER |= EXTI\_SWIER\_SWI2;

#endif \*/

**#ifdef** 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 MK

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\_Channel3->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\_Channel2->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\_paralell\_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).

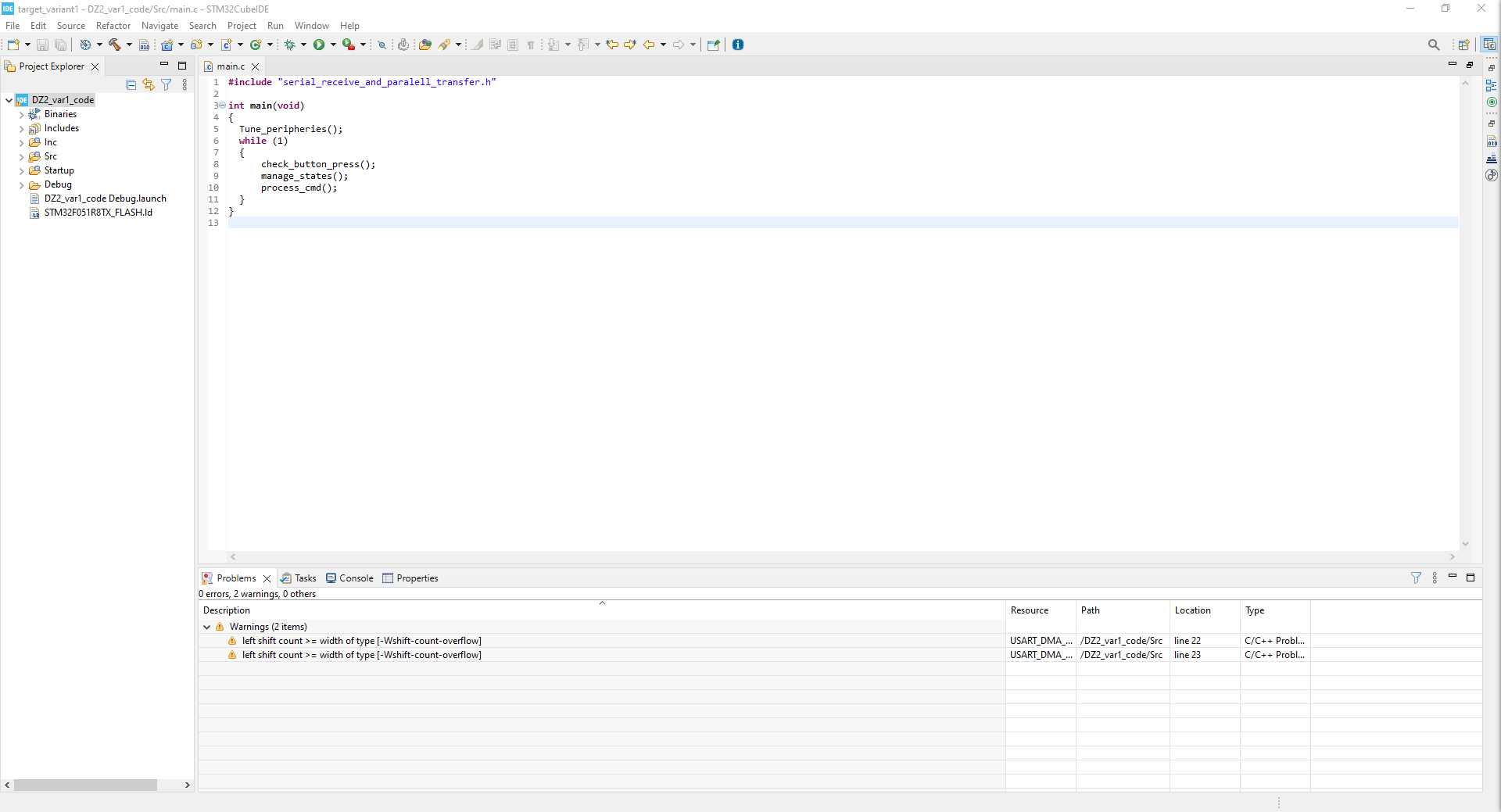


Рисунок 3.12 – Окно программы после сборки

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. Дальше внизу справа нажимаем Apply.

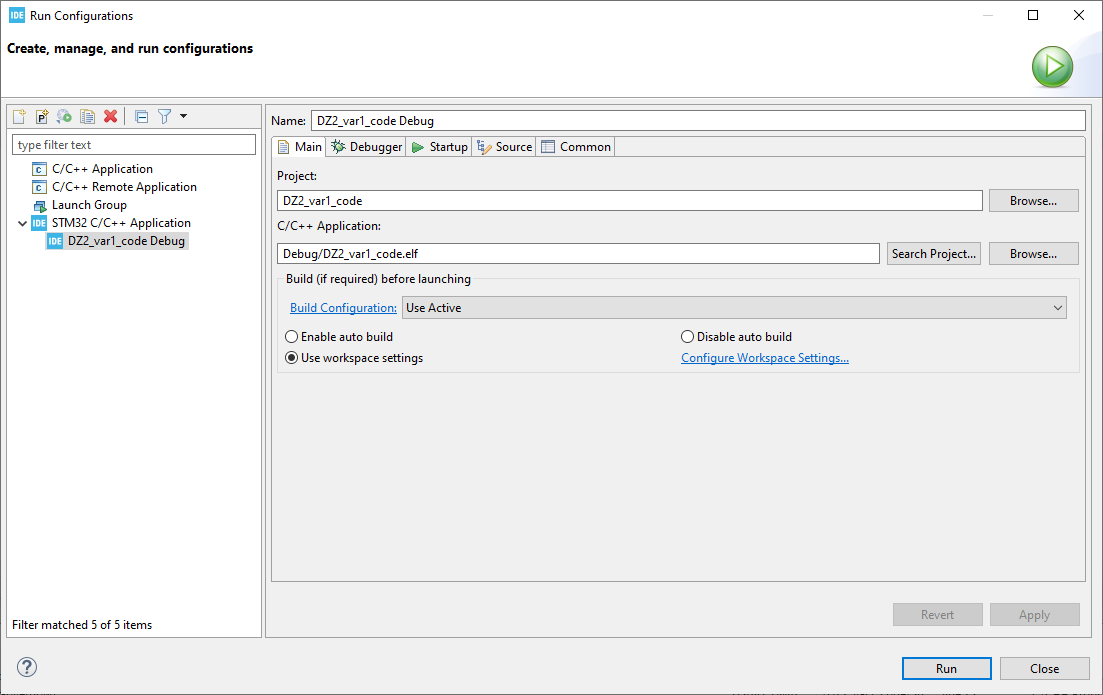


Рисунок 3.13 – Окно настройки конфигурации для компиляции – Main

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.

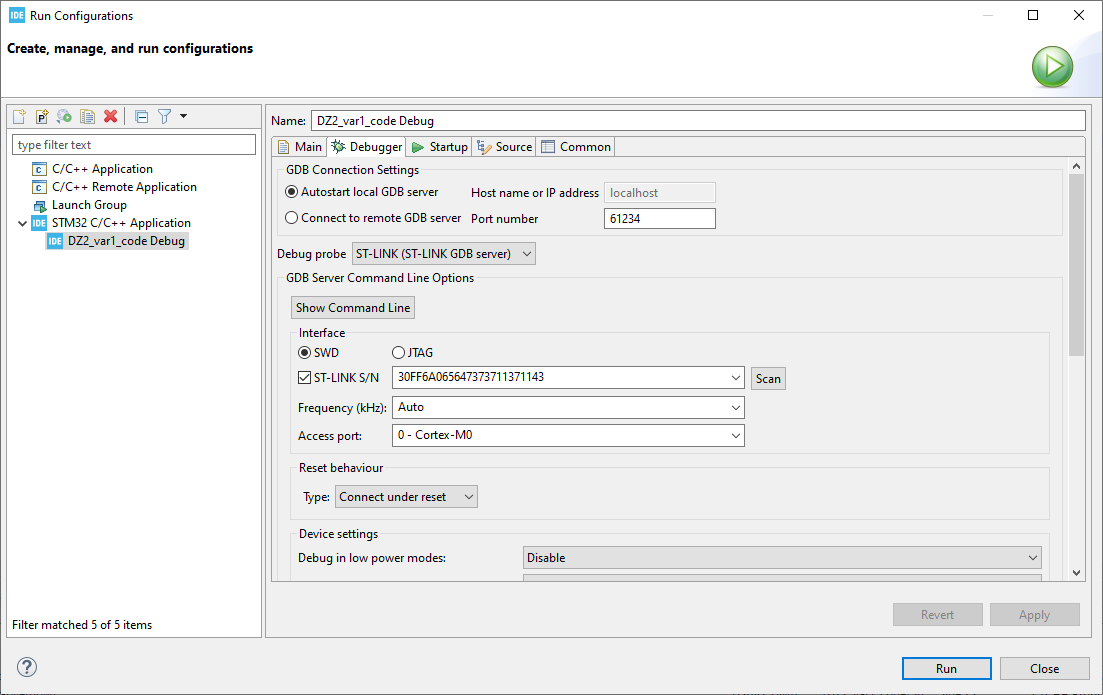


Рисунок 3.14 - Окно настройки конфигурации для компиляции – Debugger

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?”.

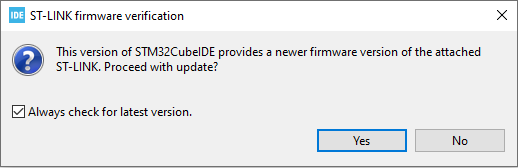


Рисунок 3.15 – Окно сообщения

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.

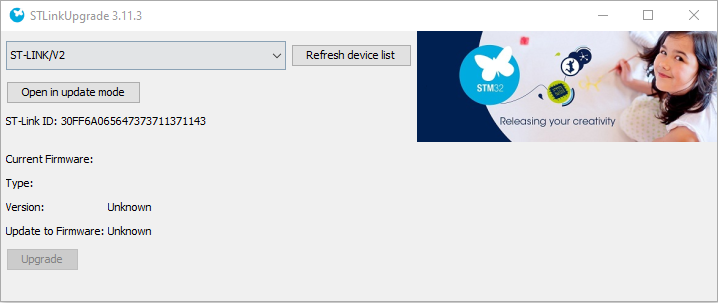


Рисунок 3.16 – Окно после окна с сообщением

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.”

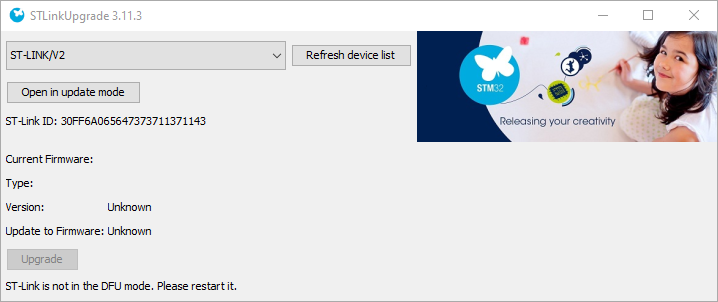


Рисунок 3.17 – Тупик

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.

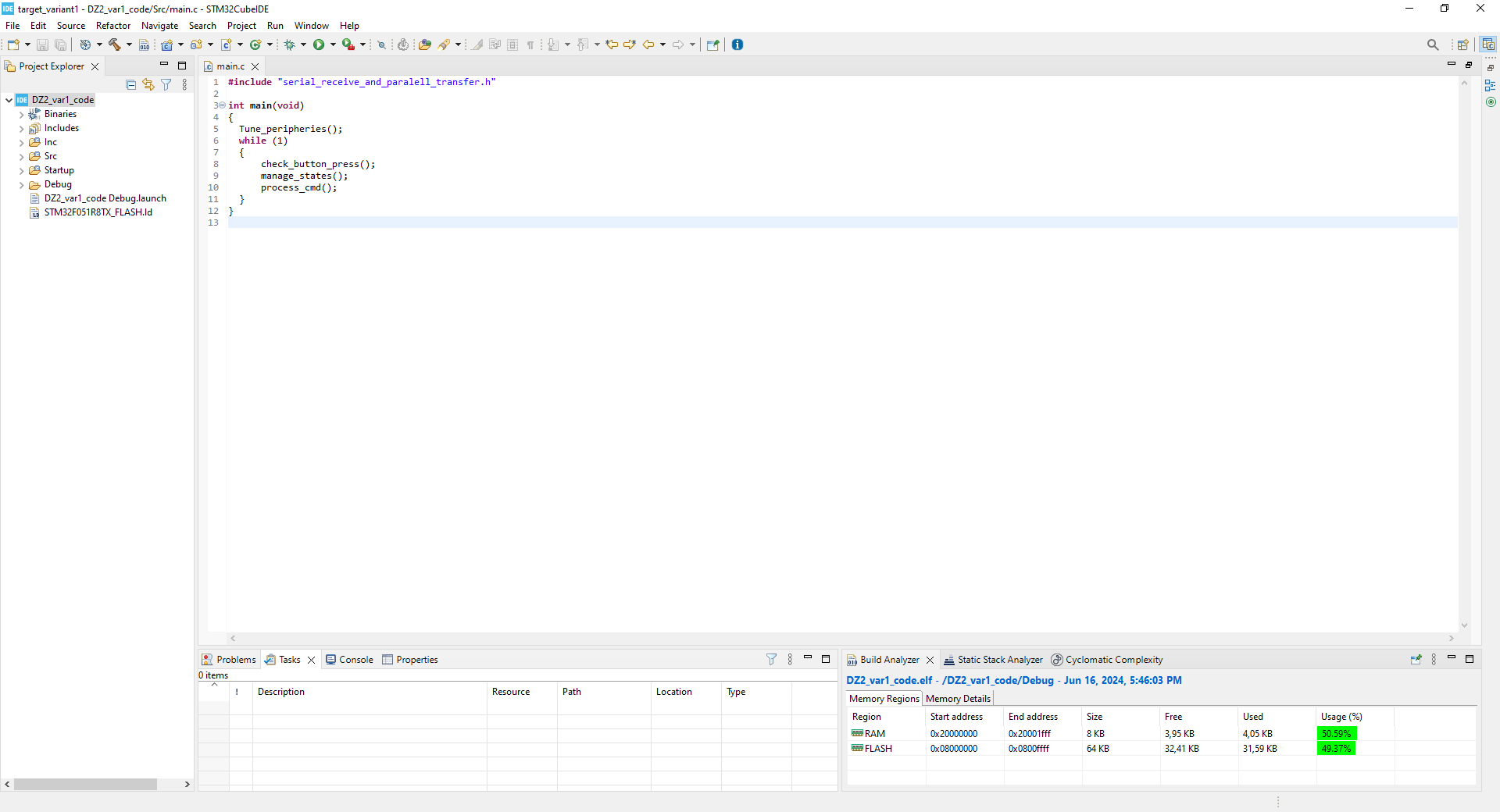


Рисунок 3.18 – Окно программы после компиляции

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.

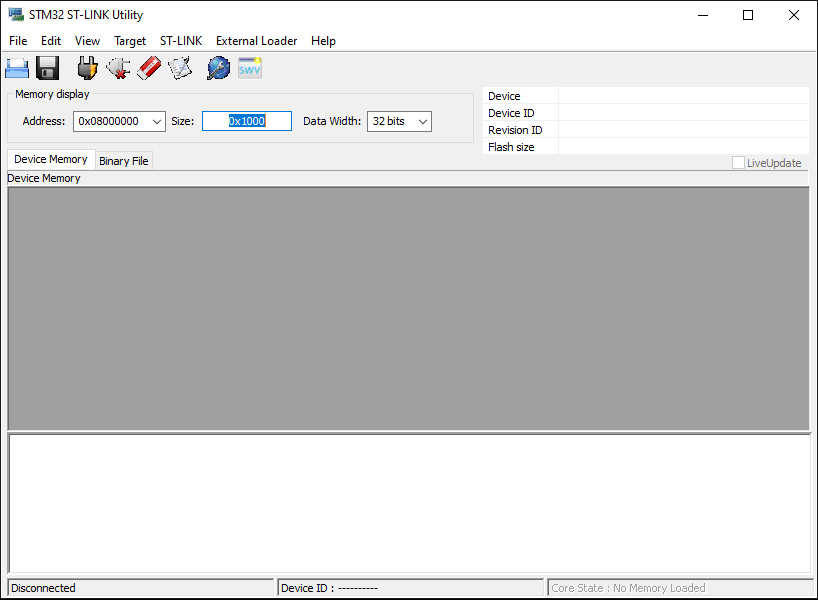


Рисунок 3.19 – Окно программы STM32 ST-LINK Utility

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

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

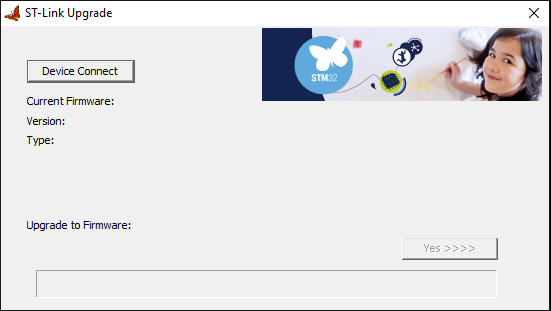


Рисунок 3.20 – Окно для обновления

Нажимаем Device connect.

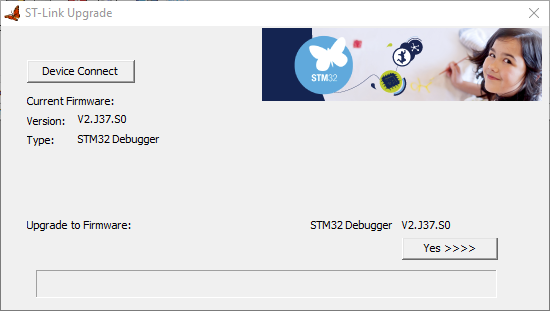


Рисунок 3.21 – Окно после нажатия на кнопку

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

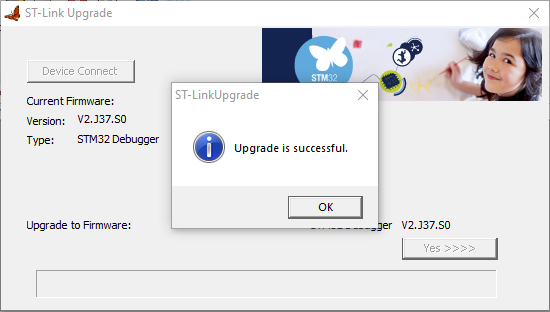


Рисунок 3.22 – Окно после завершения обновления

Нажимаем ОК 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\_paralell\_transfer.h”, “serial\_receive\_and\_paralell\_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»;

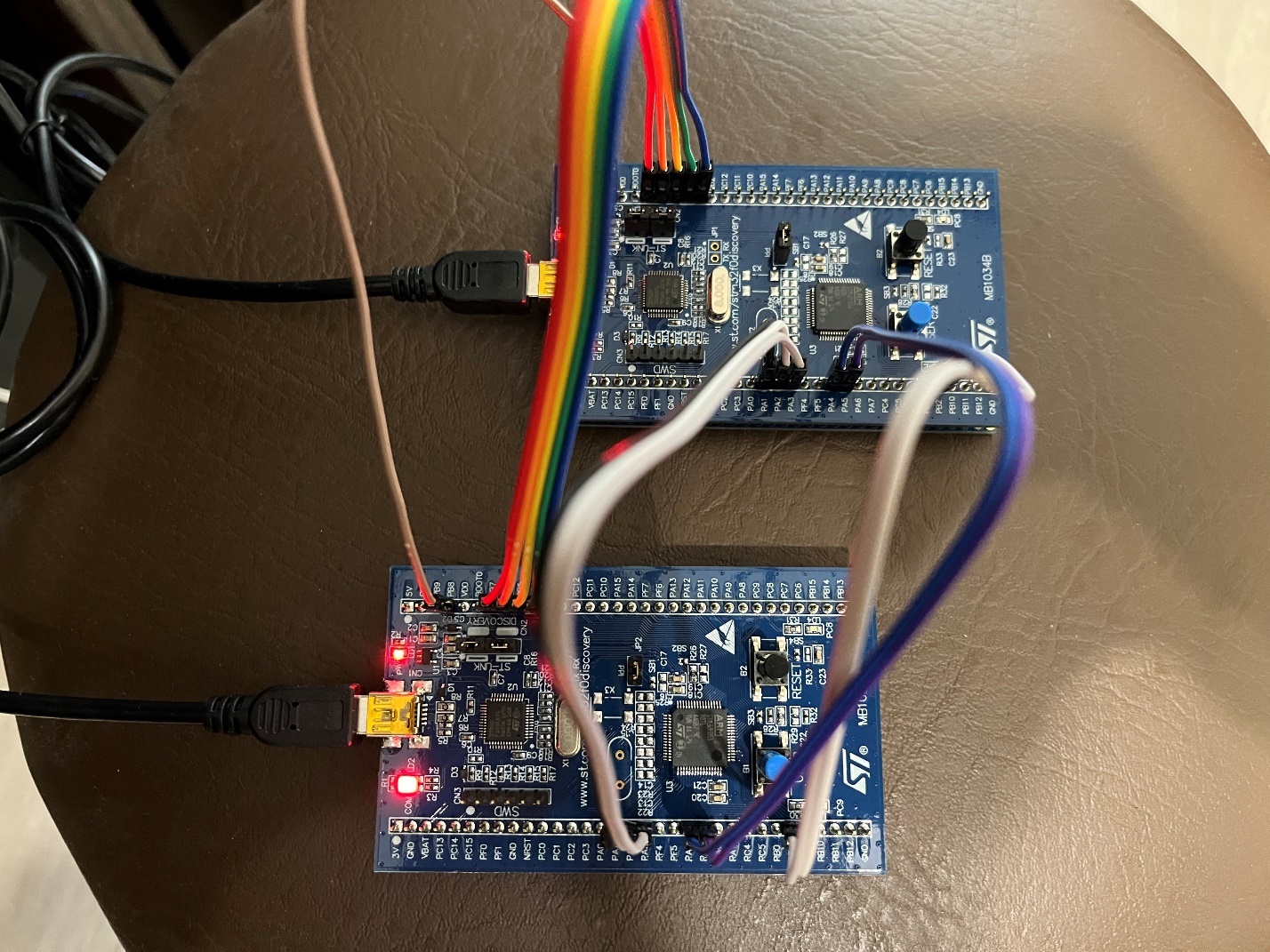
- PA2 – сигнал «en»;

- PA3 – сигнал «ready»;

- PA4 – clock (генератор тактового сигнала);

- PA5 – последовательные данные;

- PB1-PB8 – параллельные данные.



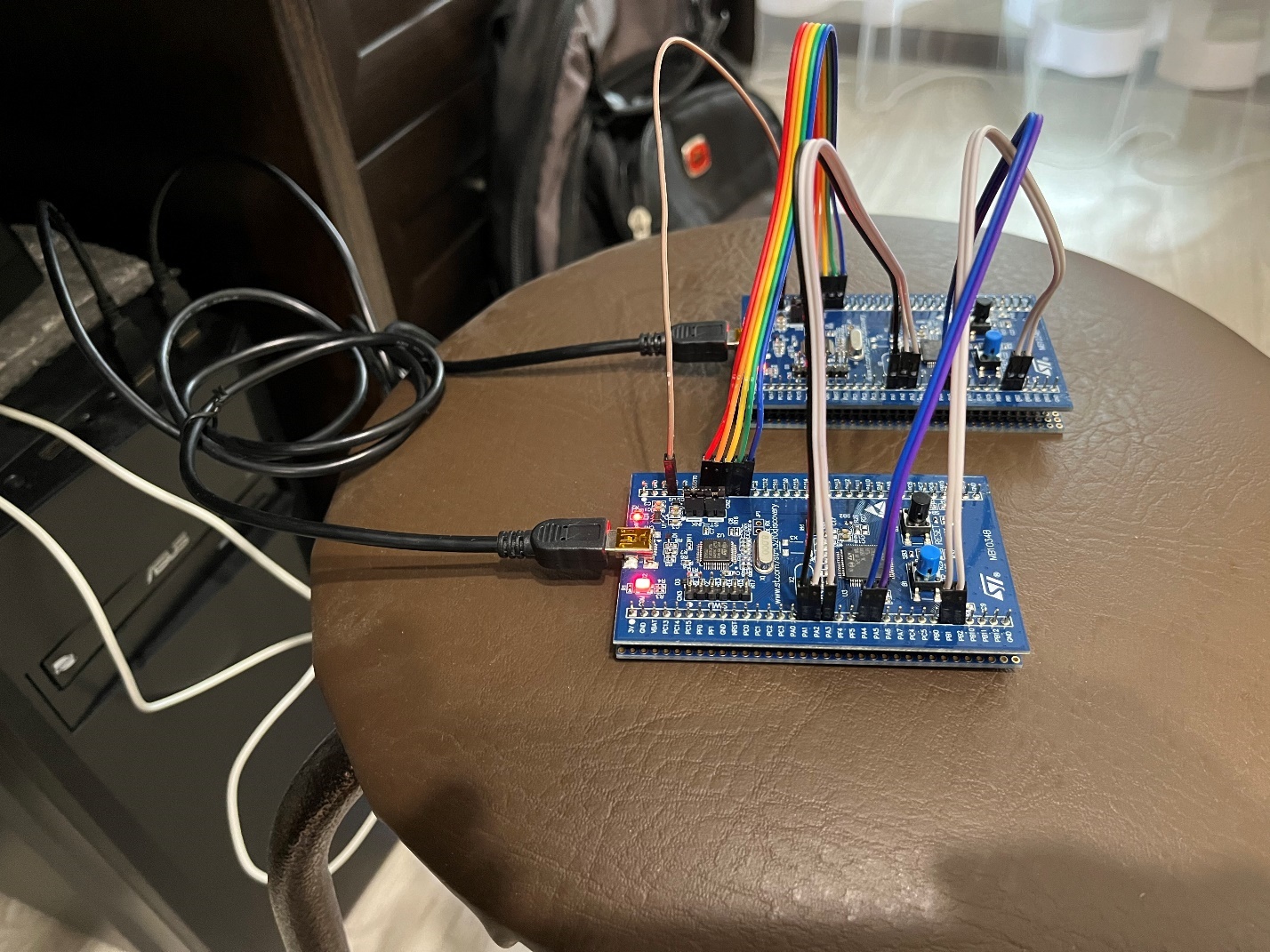


Рисунок 3.23 – Платы, соединённые проводами и подключенные к ПК

## 3.7 Отладка

### STM32CubeProgrammer

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

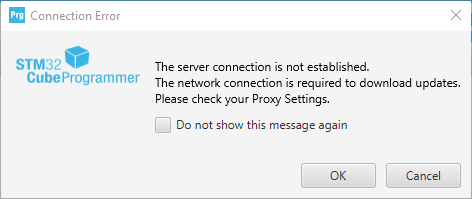


Рисунок 3.24 – Окно Connection Error

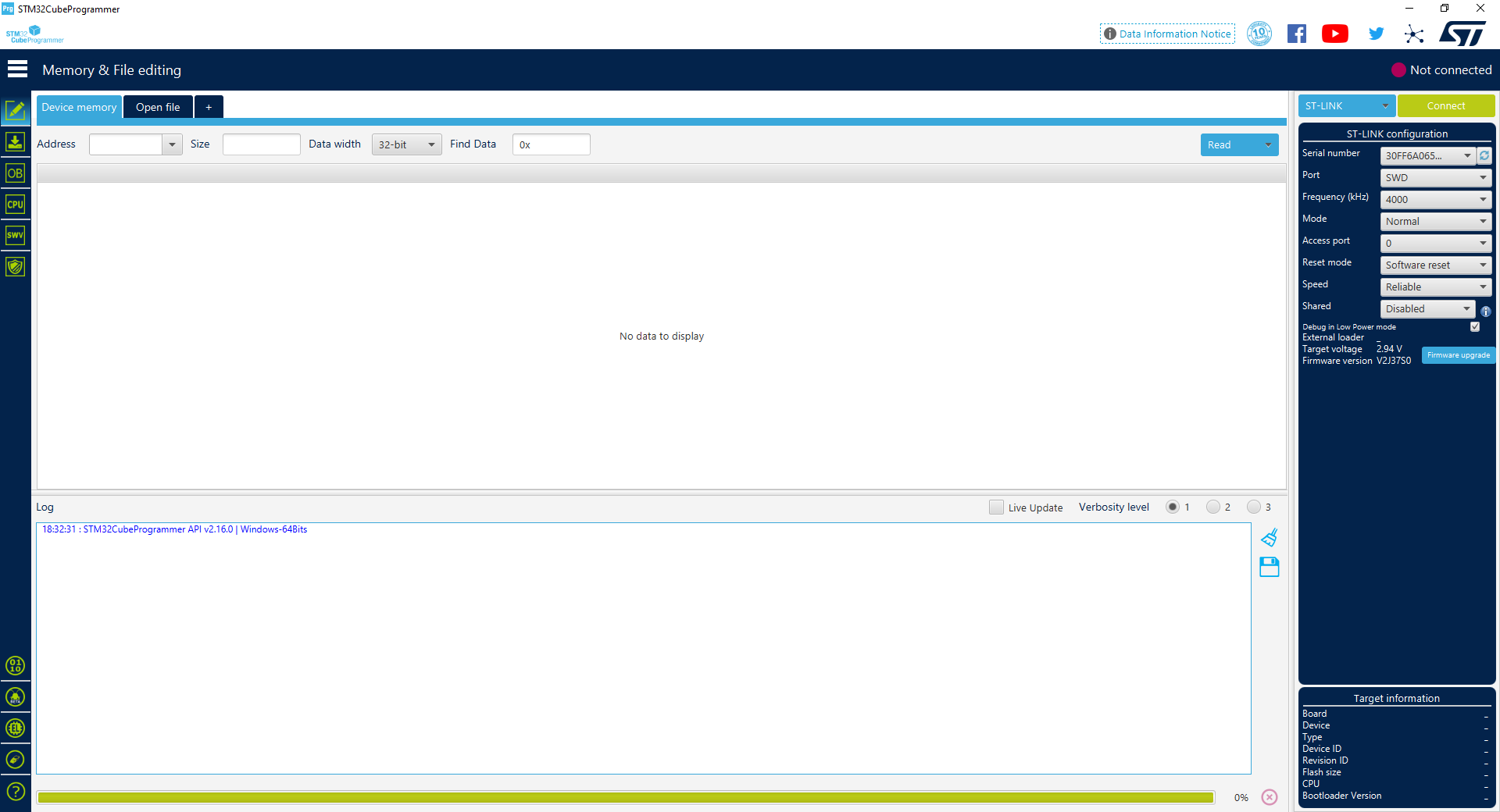


Рисунок 3.25 – Окно программы STM32CubeProgrammer

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.

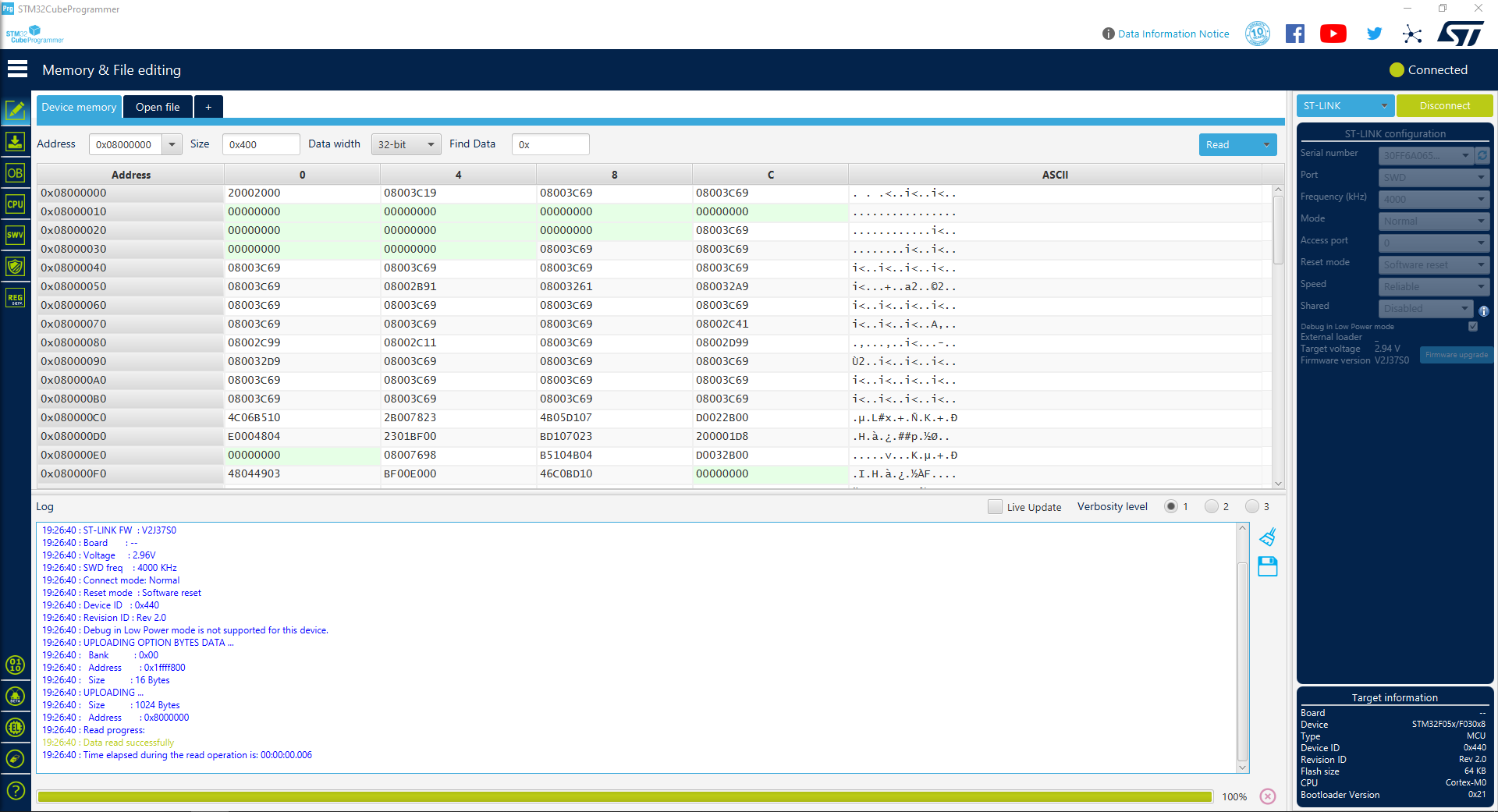


Рисунок 3.26 – Окно программы после установки соединения

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.

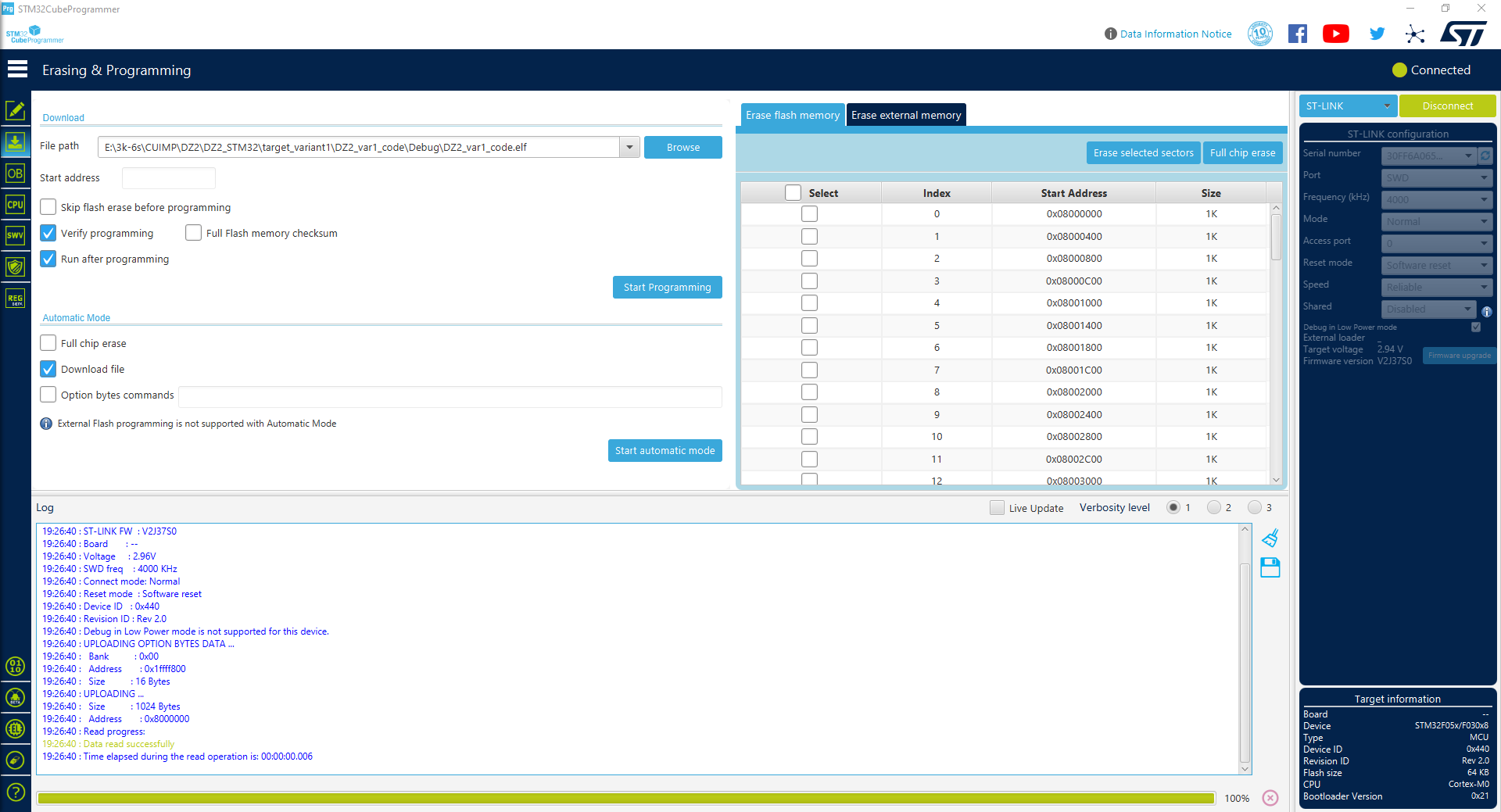
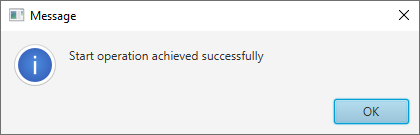
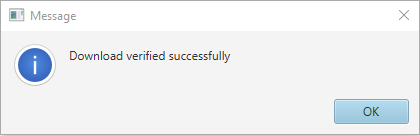


Рисунок 3.27 – Указание пути и проставление галочек

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





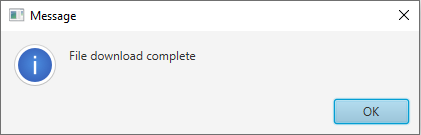


Рисунок 3.28 – Сообщения

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.

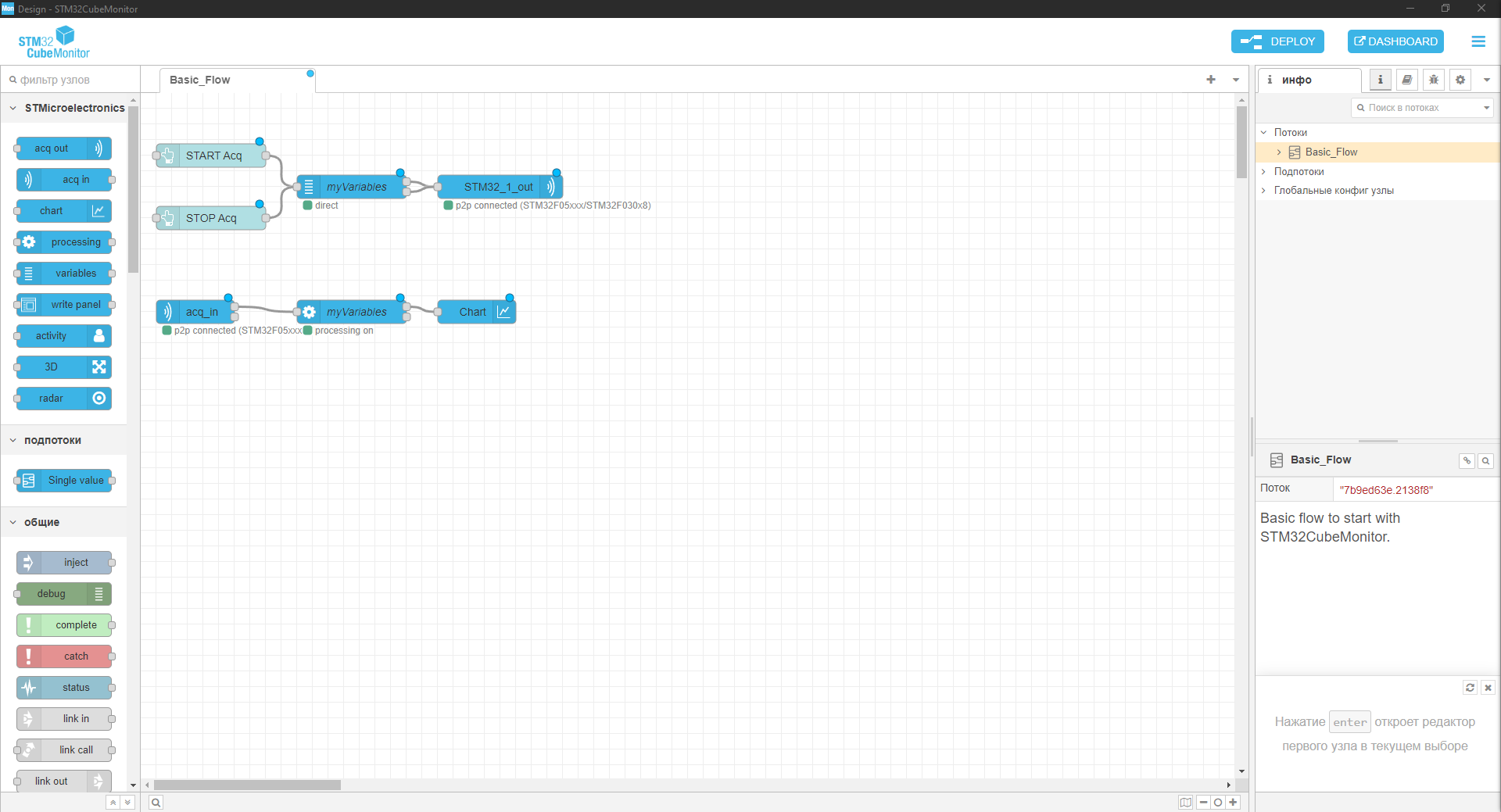
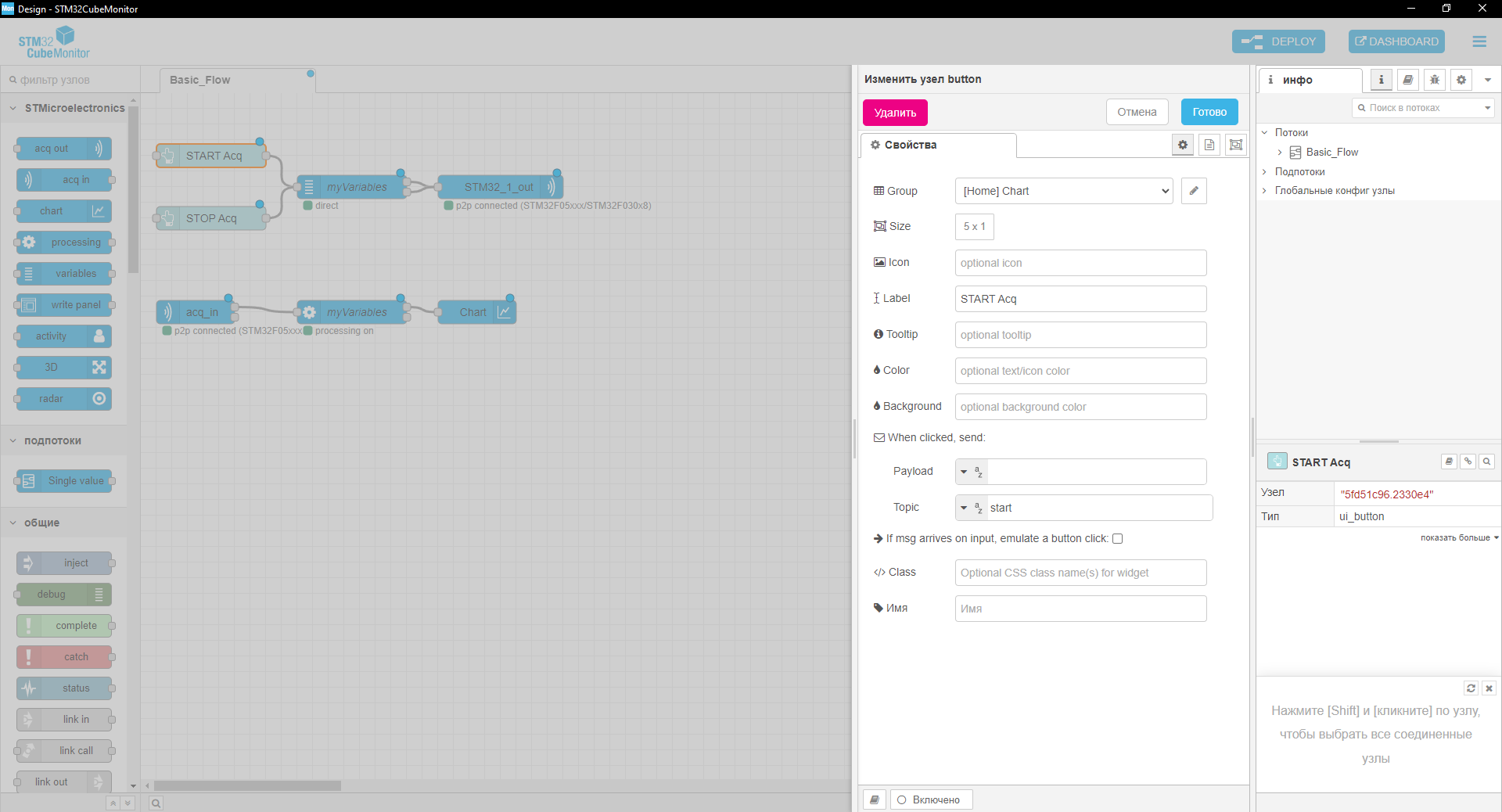


Рисунок 3.29 – Окно программы STM32CubeMonitor с собранной ранее схемой для вывода графиков

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 (которая имеет сетку тетрадного листа в клетку).

START Acq и STOP Acq – кнопки.



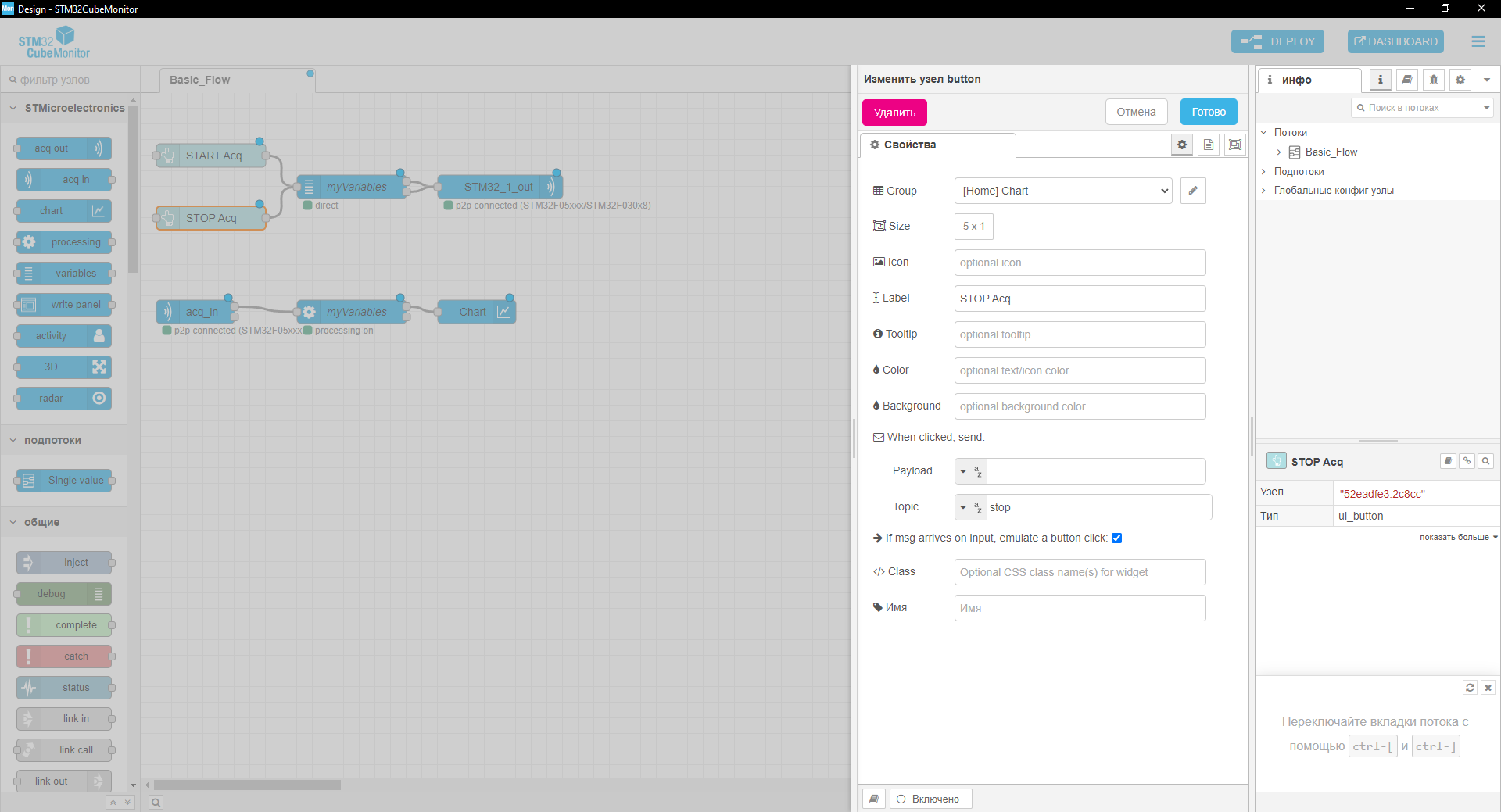


Рисунок 3.30 – Свойства кнопок START Acq и STOP Acq

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.

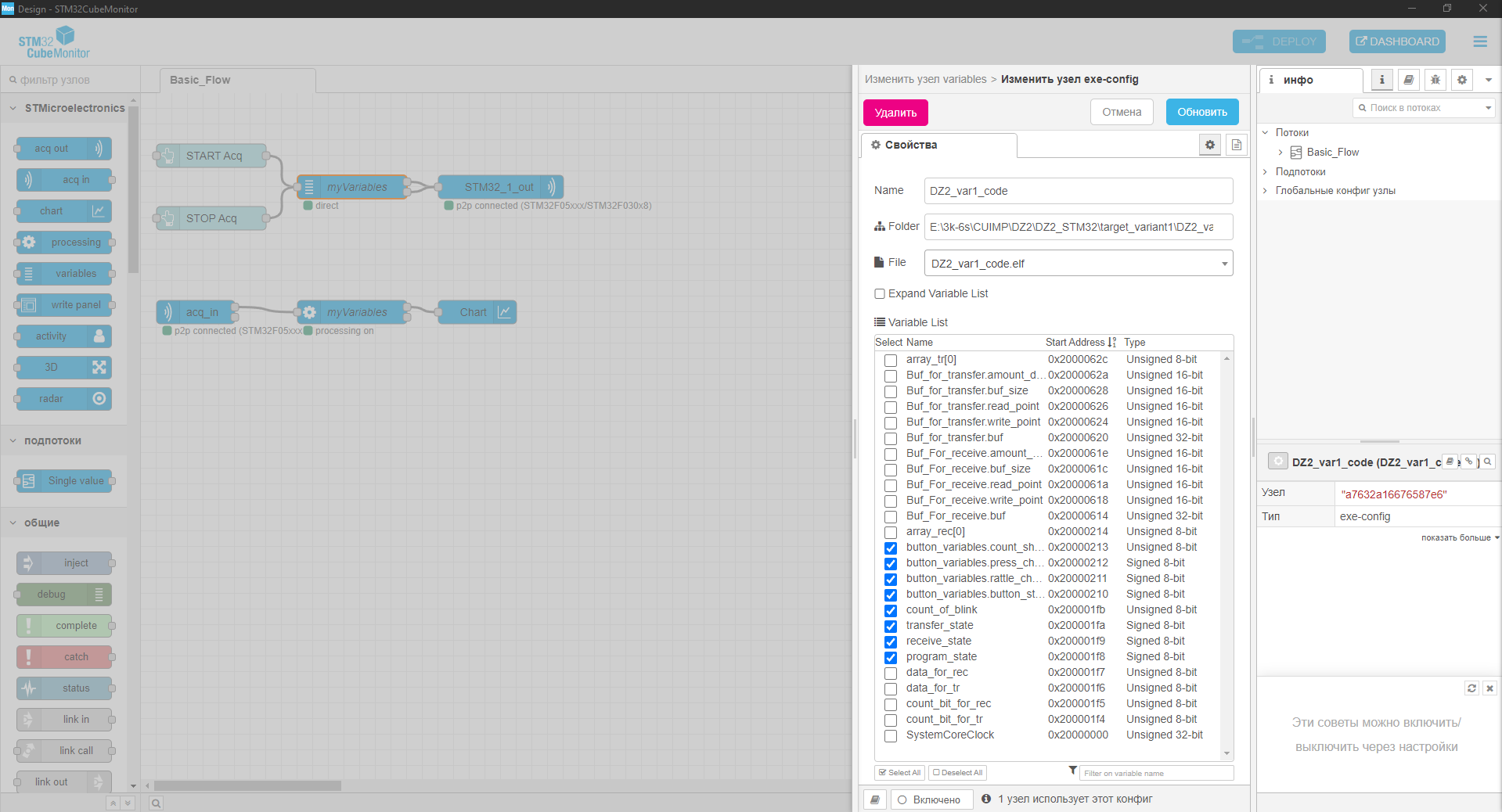


Рисунок 3.31 – Окно с выбором переменных для вывода на график

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.

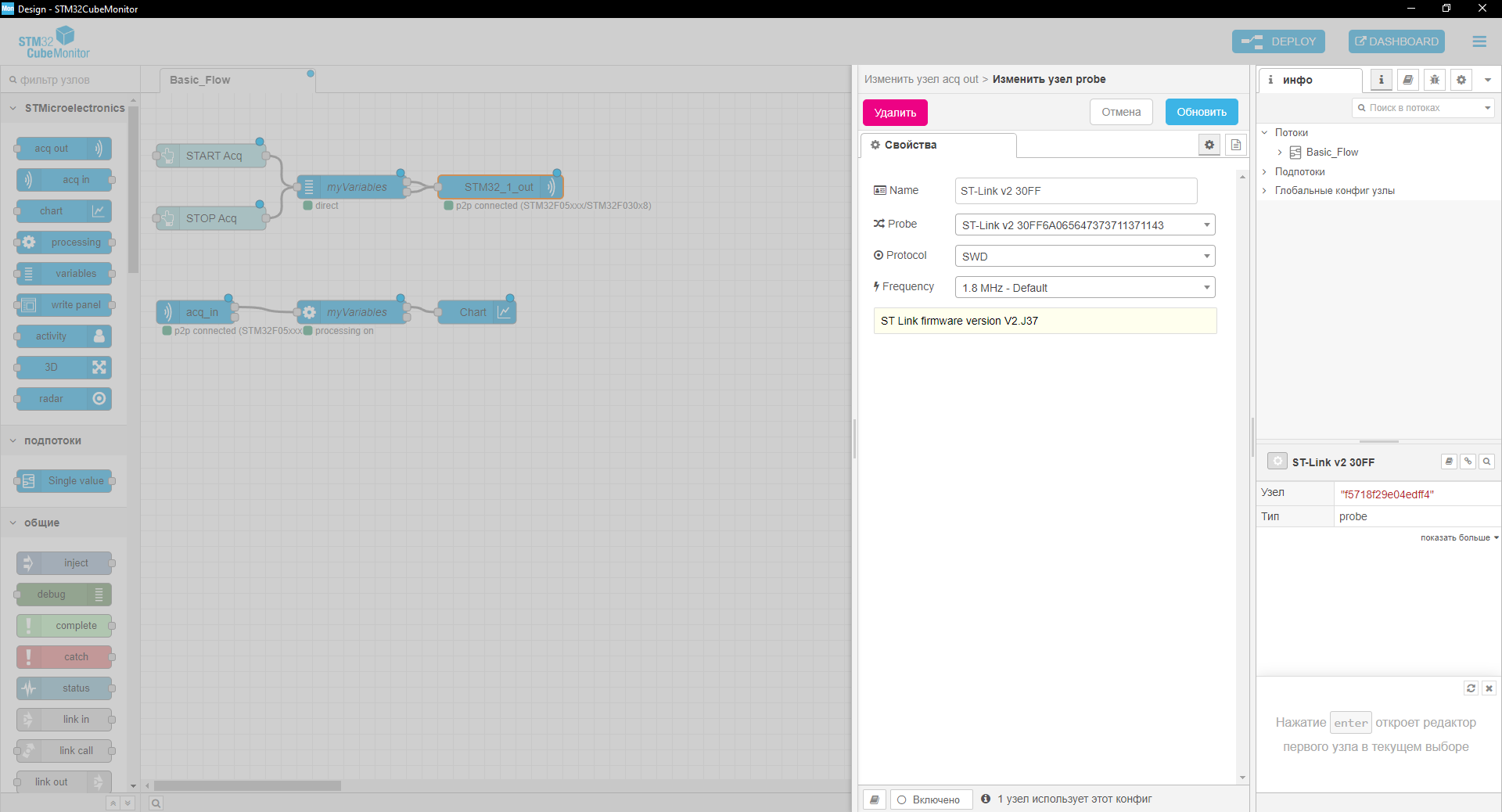


Рисунок 3.32 – Настройка свойств узла probe

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.

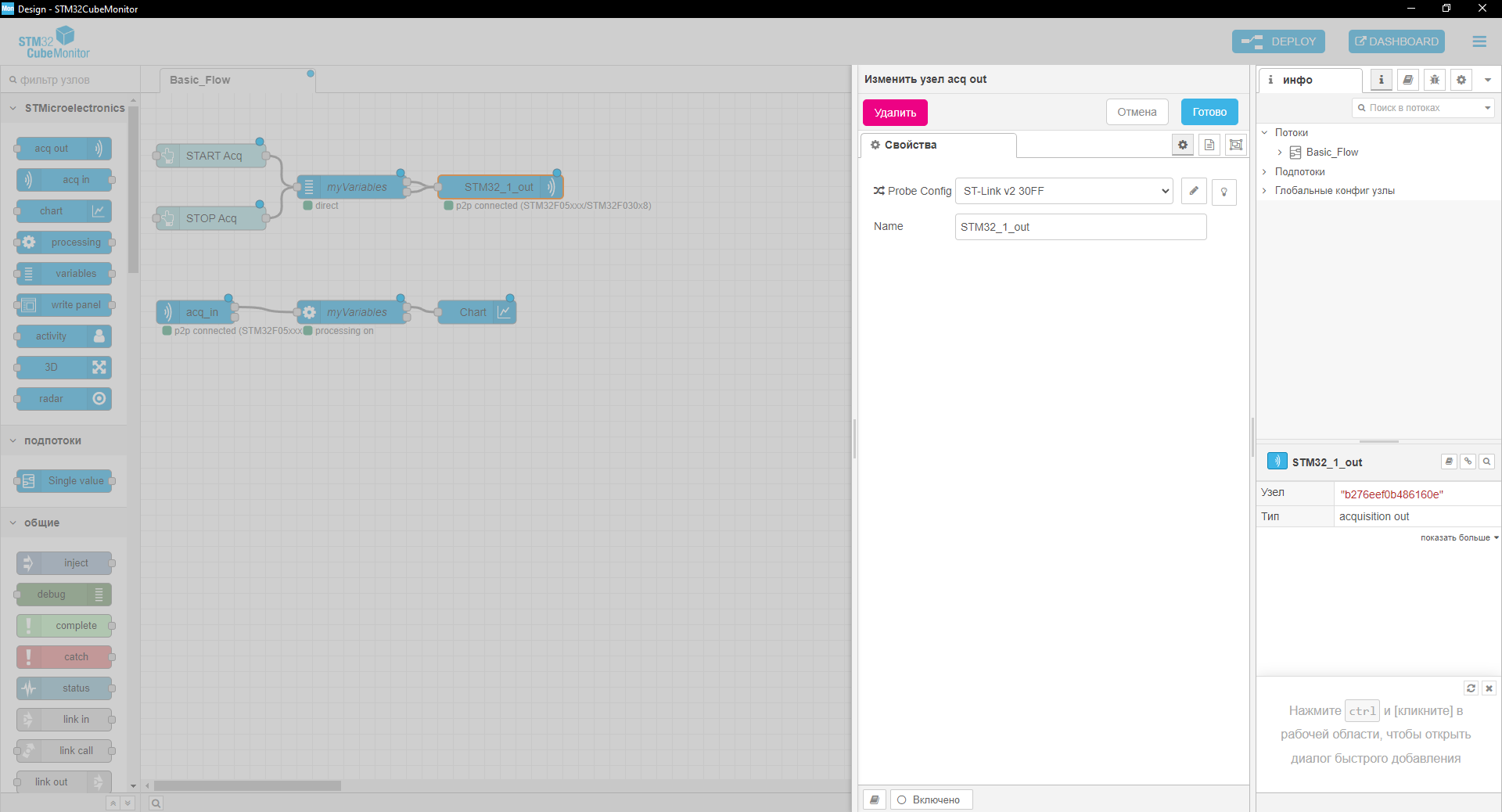
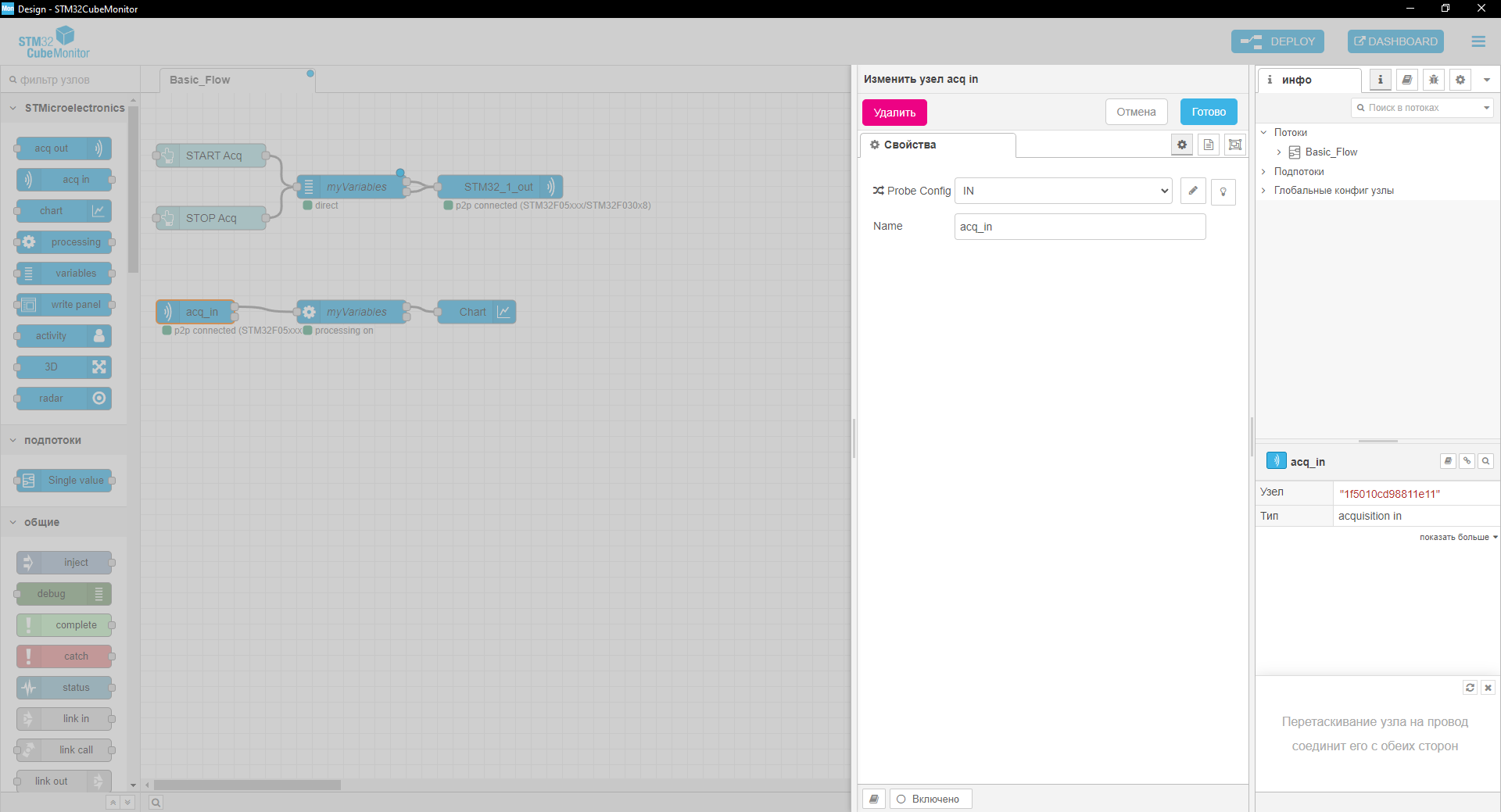
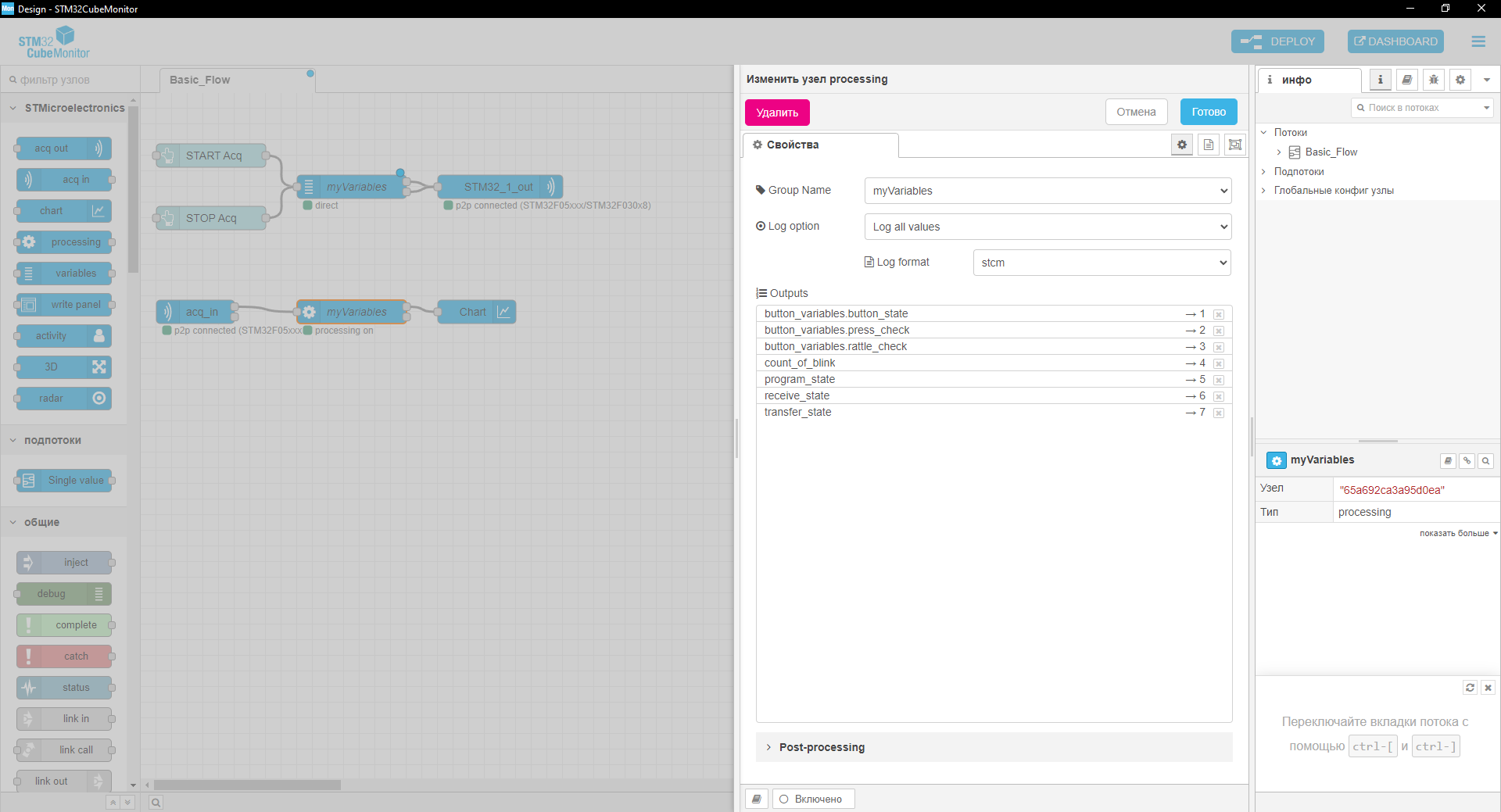


Рисунок 3.33 – Настройка узла acq out

Нажимаем Готово.

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





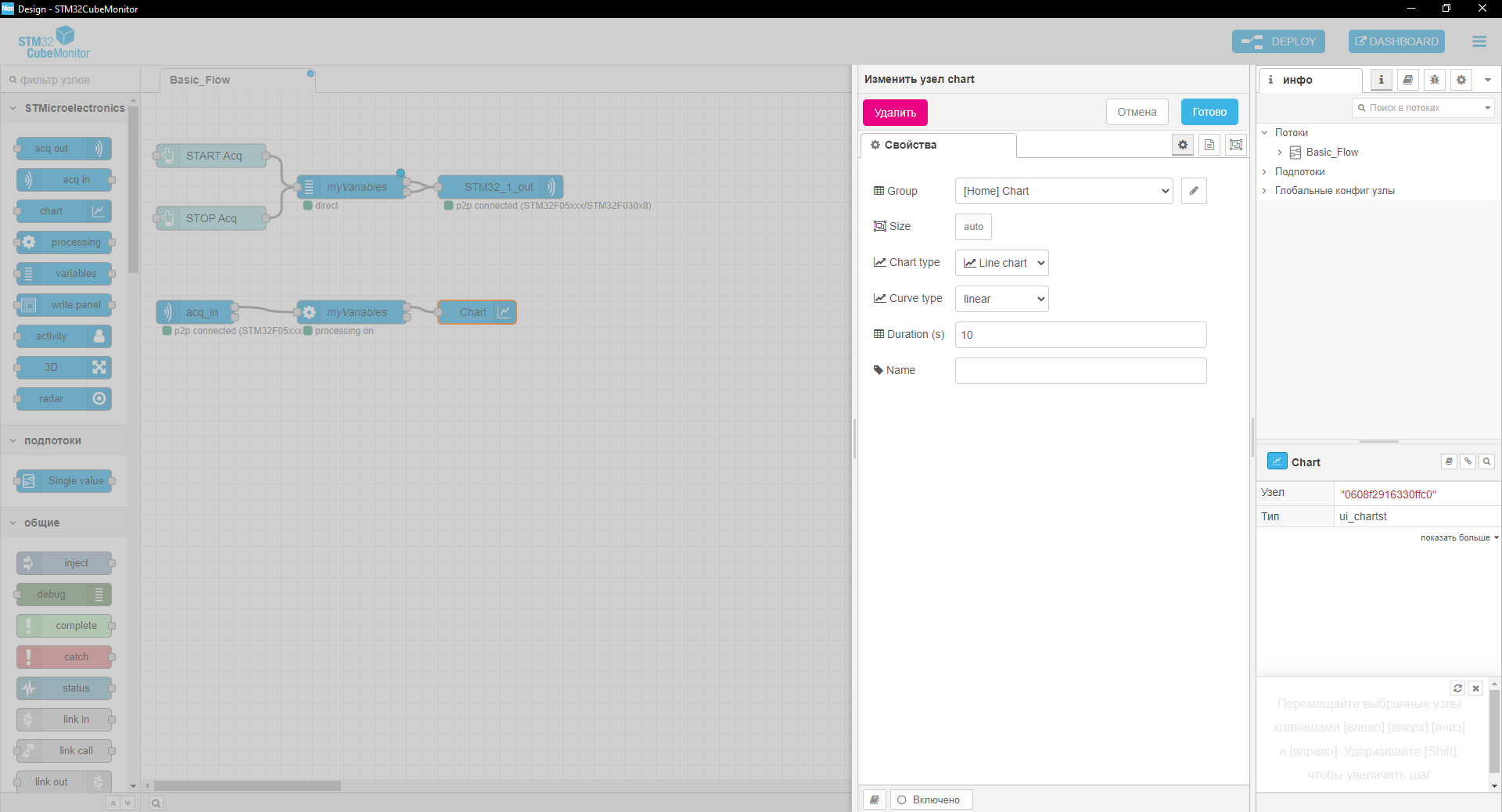


Рисунок 3.34 – Свойства 3-х соединённых узлов

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

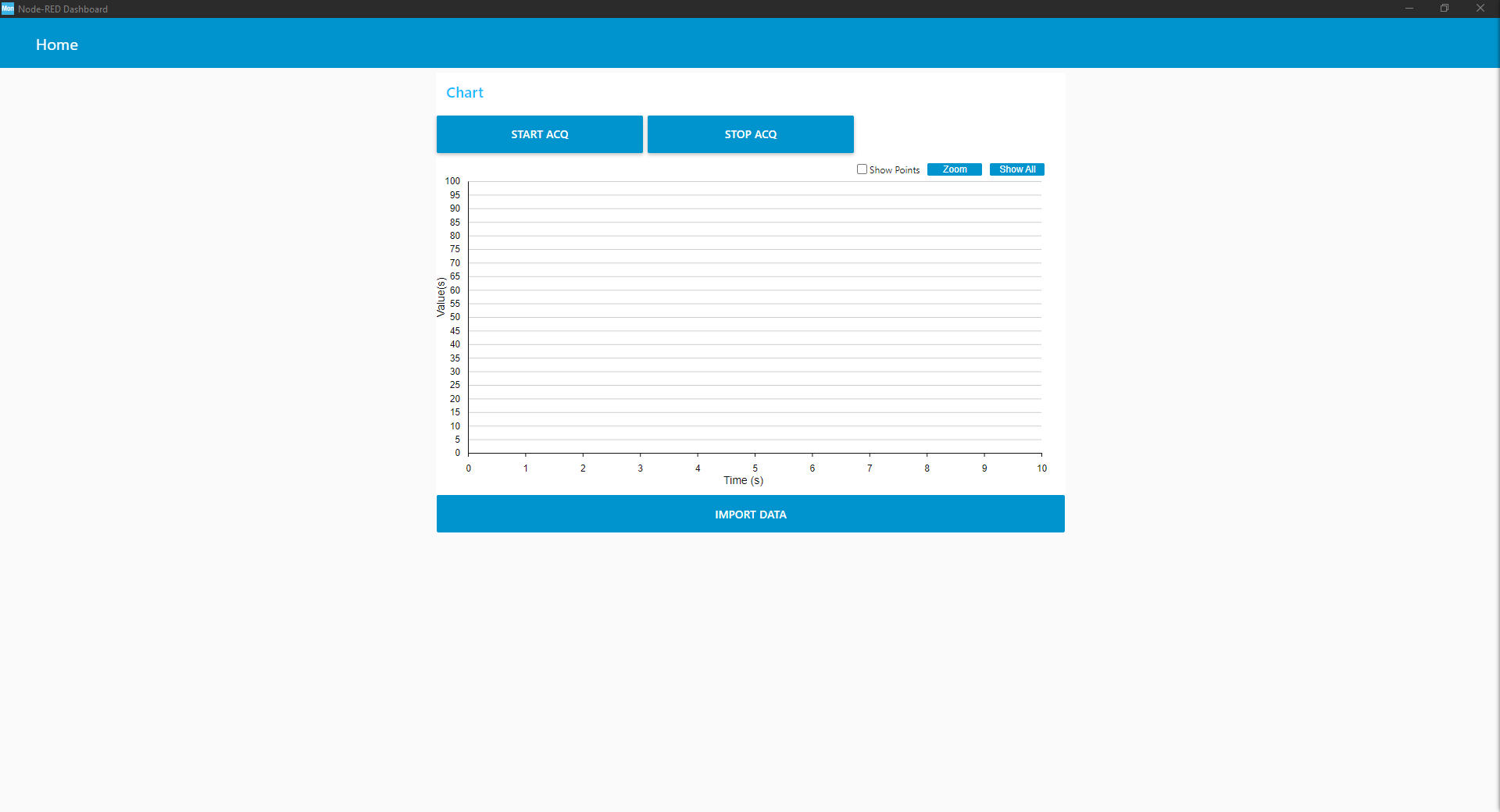


Рисунок 3.35 – Появившееся окно

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

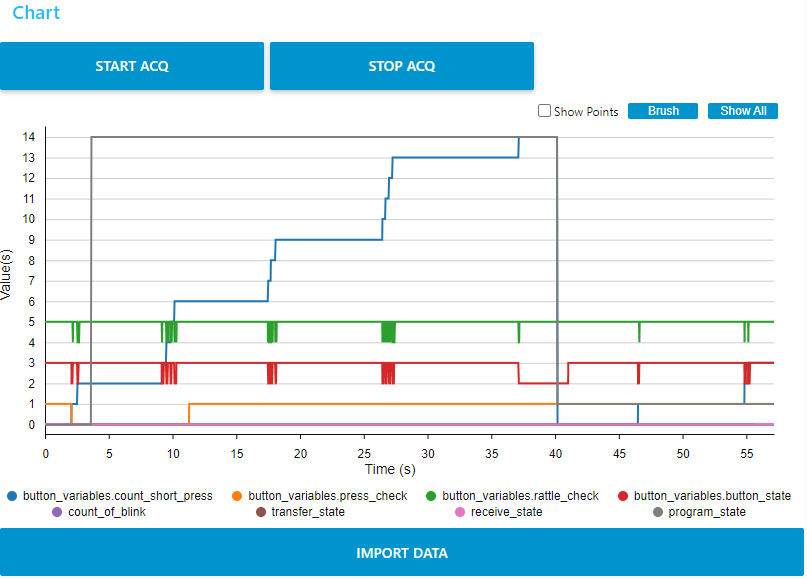


Рисунок 3.37 – Графики для переменных, используемых кодом, зашитым в плату – приближенно

### 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.