Начало работы

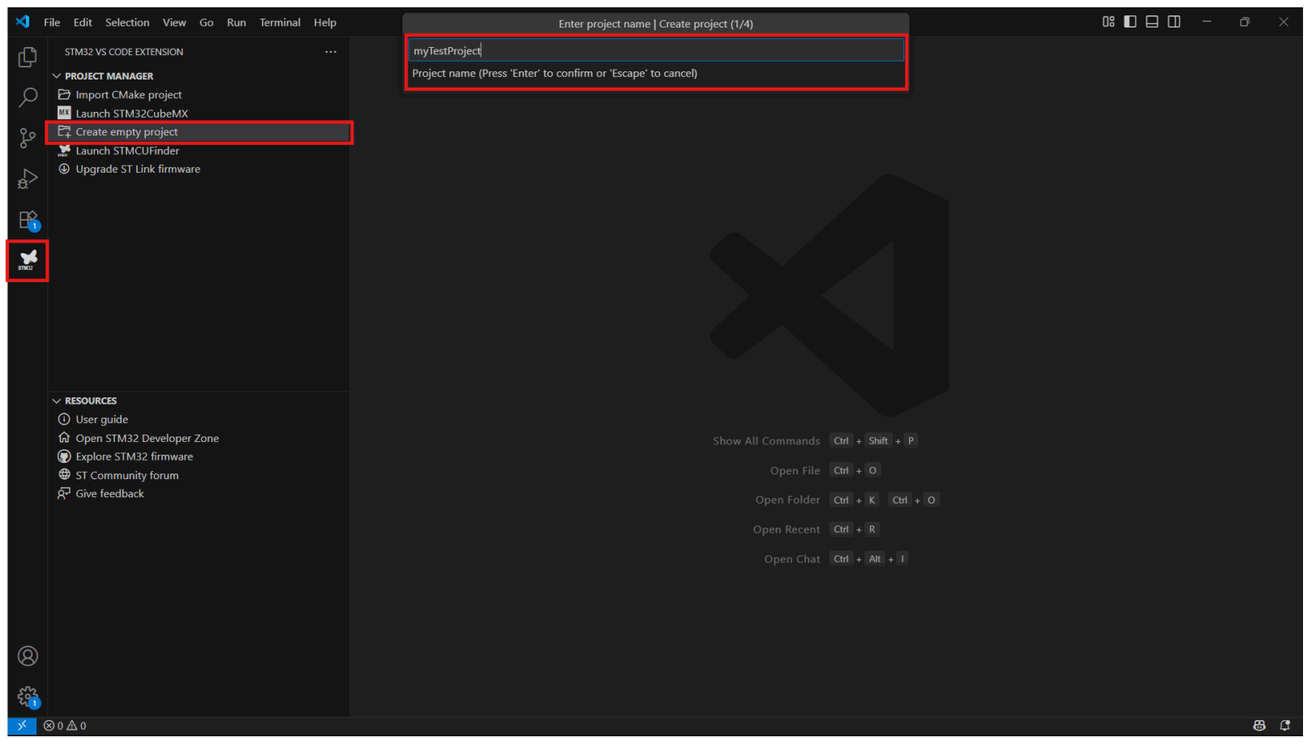
Установка ПО

Список ПО

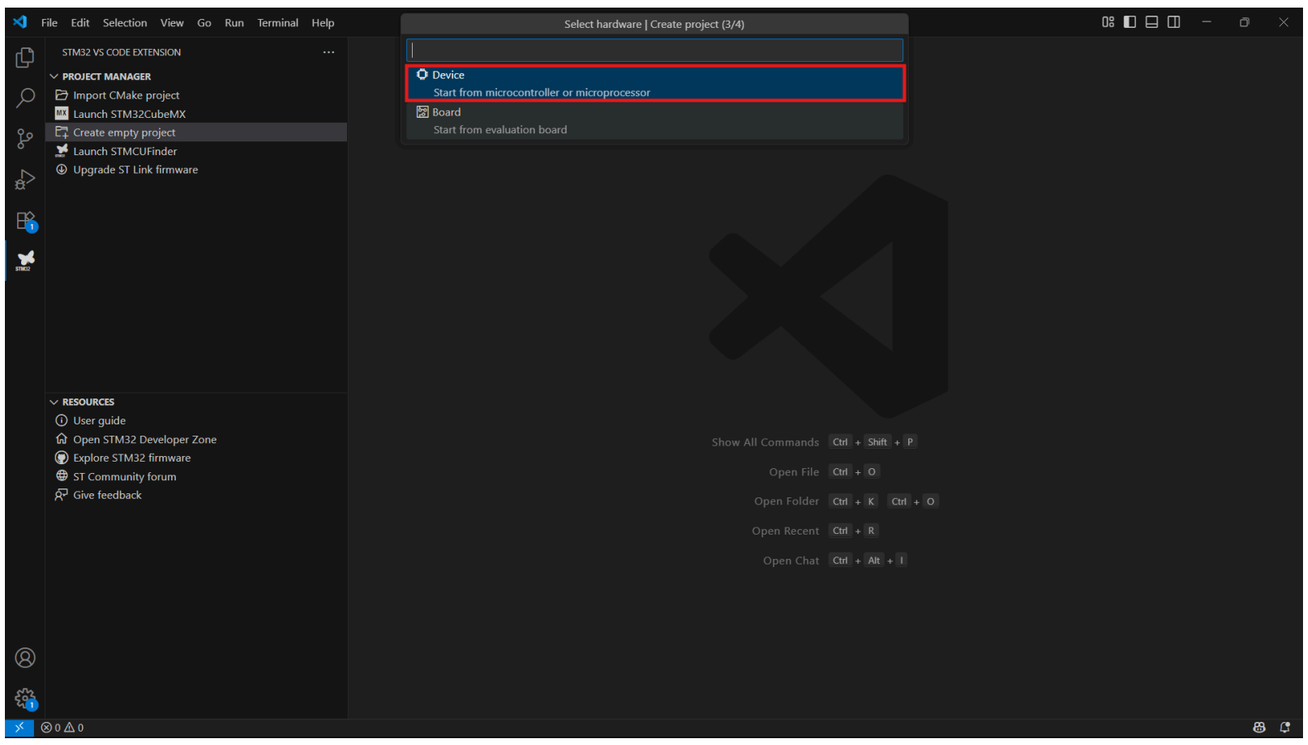
1. Среда разработки: Visual Studio Code.
2. Набор инструментов STM32: STM32CubeCLT.
3. Расширения VS Code:
   - STM32Cube for Visual Studio Code
   - Output Colorizer

Создание пустого проекта

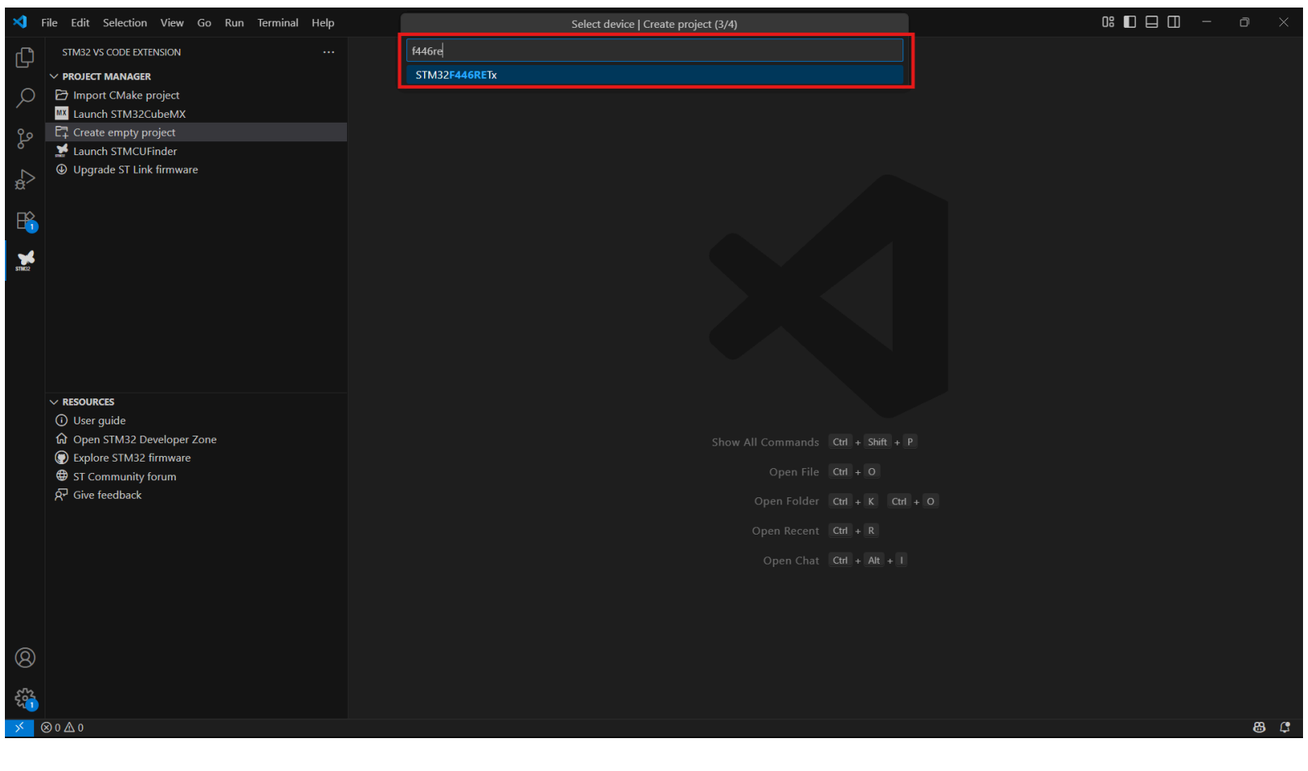
1. Откройте VS Code → вкладка **STM32 VS CODE EXTENSION → Project Manager → Create empty project**.



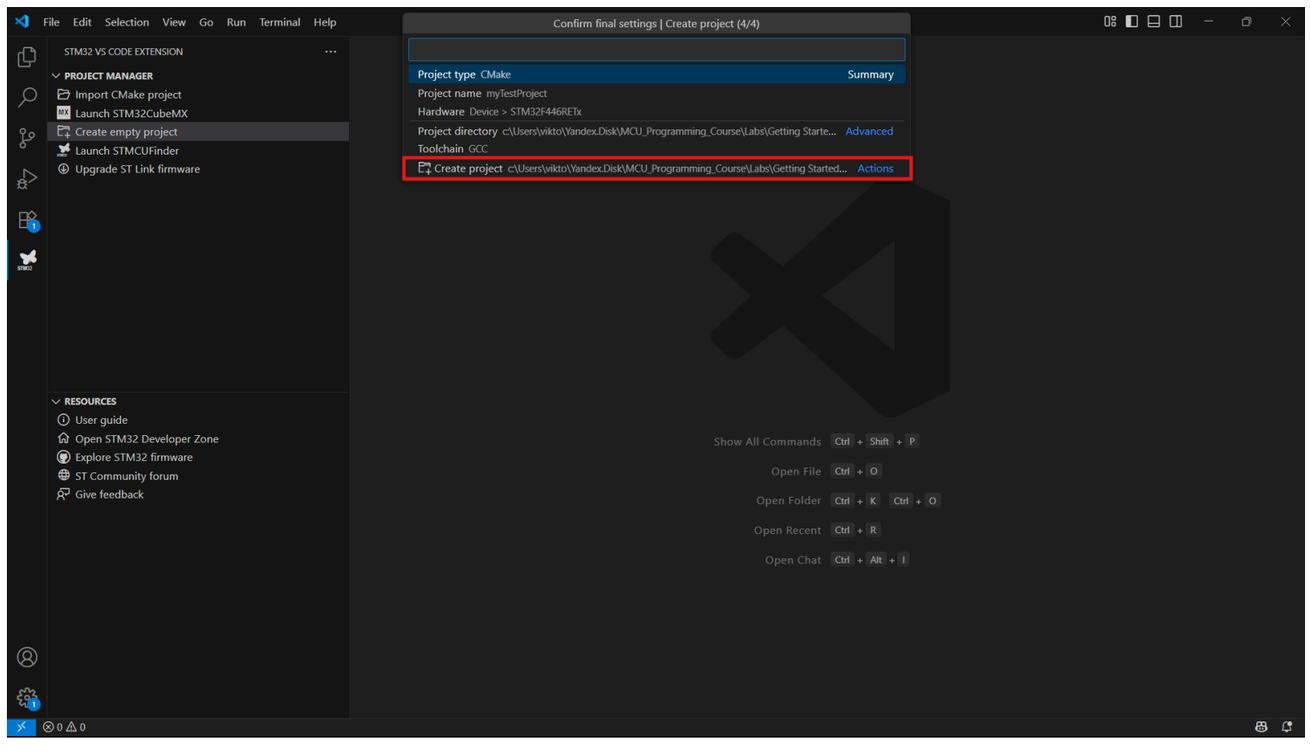
2. Выберите **папку** для проекта.
3. Выберите целевое устройство:
   - **By device** → нажмите **Device** и введите модель МК.
   - **By board** → если используете официальную плату NUCLEO/Discovery, выберите её здесь



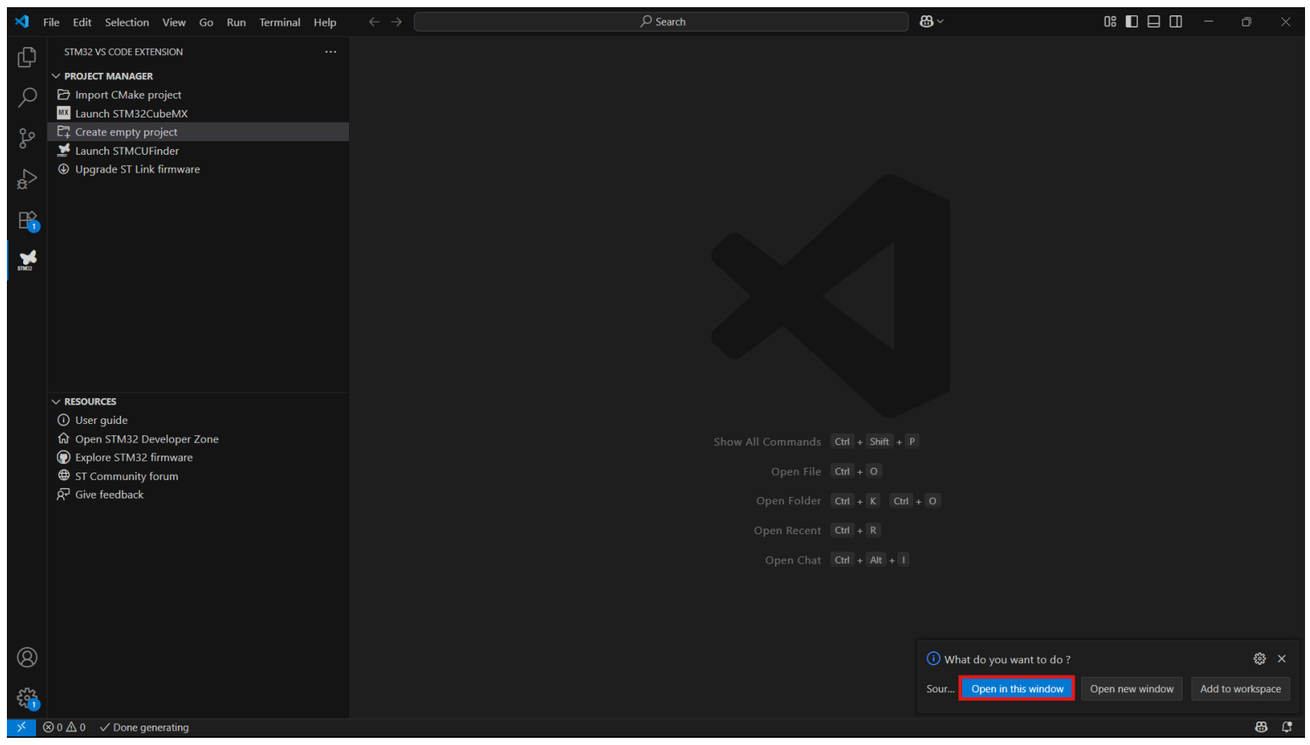
В поле *Device* введите модель микроконтроллера — ***STM32F446ret***.



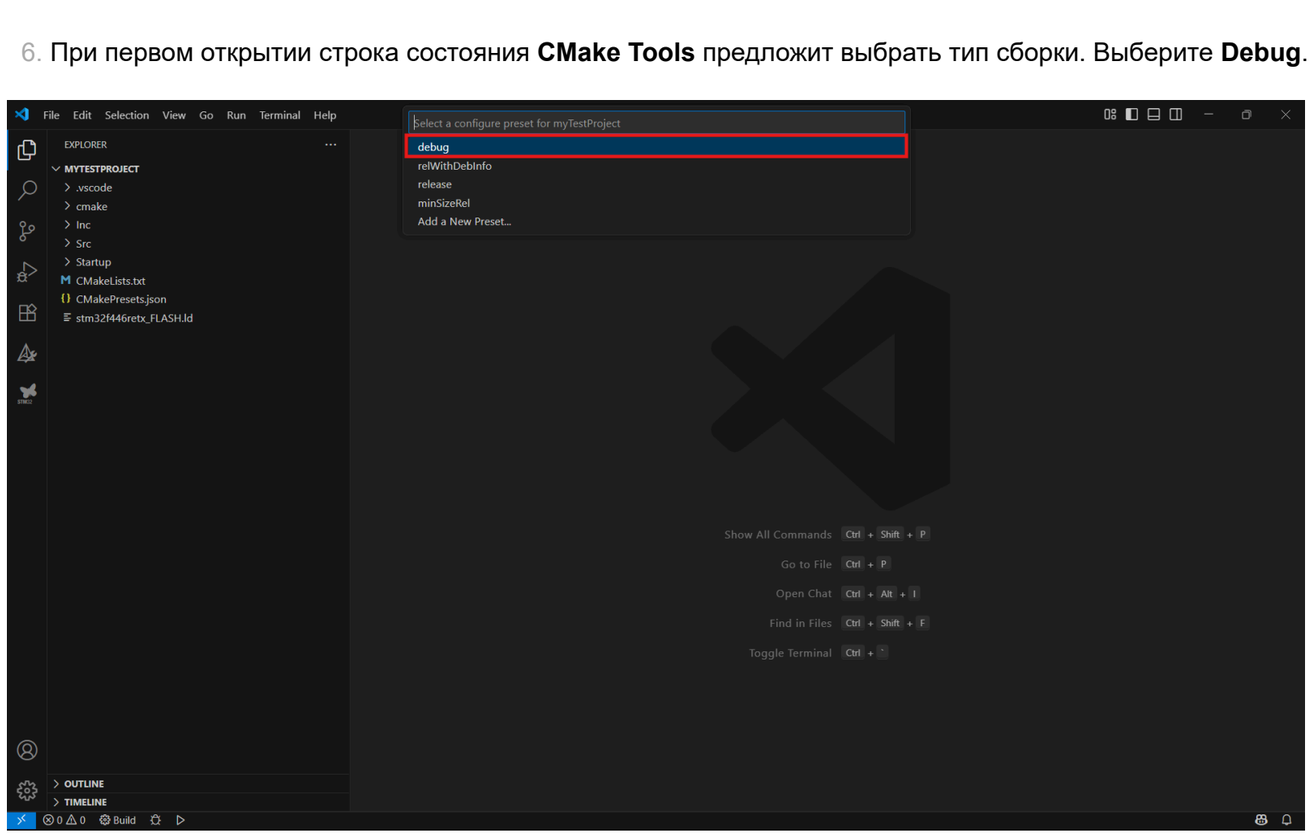
4. Появится окно **Project Settings** (сводка настроек проекта). Проверьте параметры и нажмите на **последнюю строку для подтверждения**.



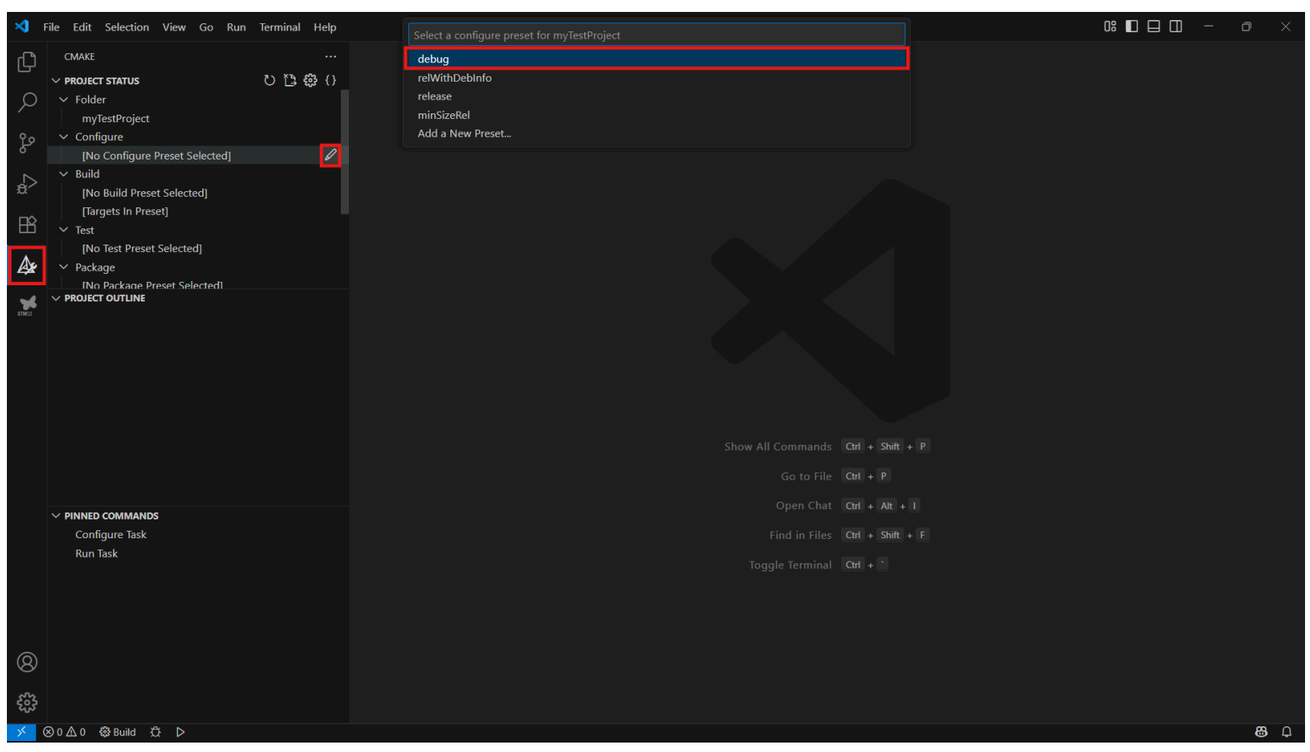
5. VS Code спросит, как открыть проект. Выберите **Open** (в текущем или новом окне)



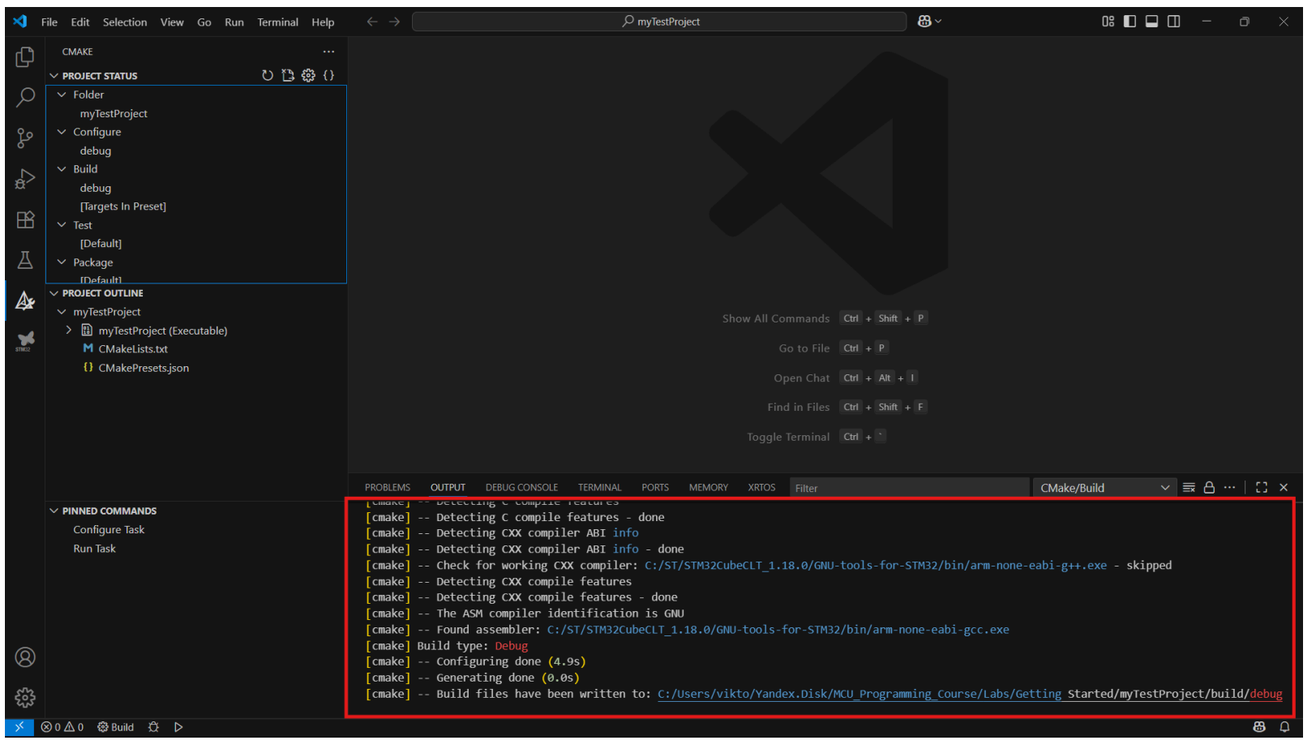
6. При первом открытии строка состояния **CMake Tools** предложит выбрать тип сборки. Выберите **Debug**.



Если меню не появилось или было случайно закрыто, откройте вкладку **CMake**, нажмите на карандаш рядом с **Configure** и выберите **Debug**.

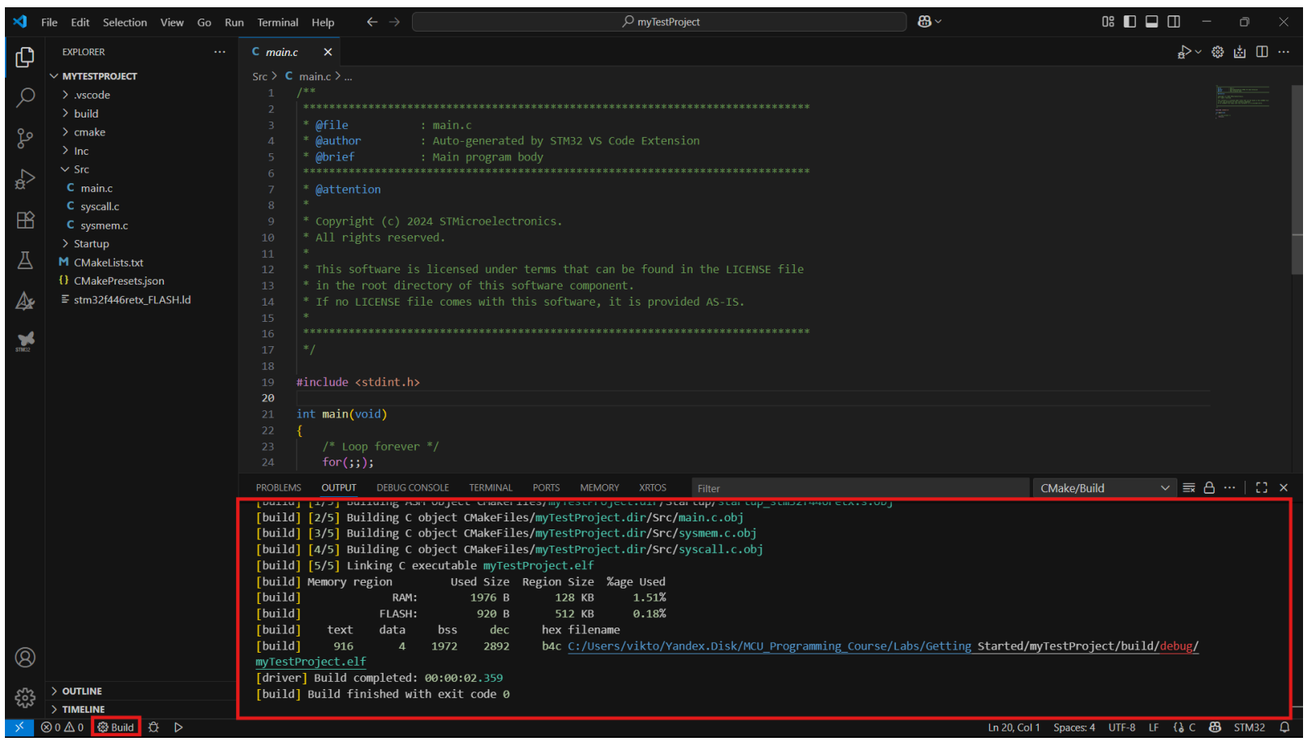


7. CMake выполнит этап **configure** (сгенерирует файлы сборки). После завершения без ошибок проект готов к сборке.



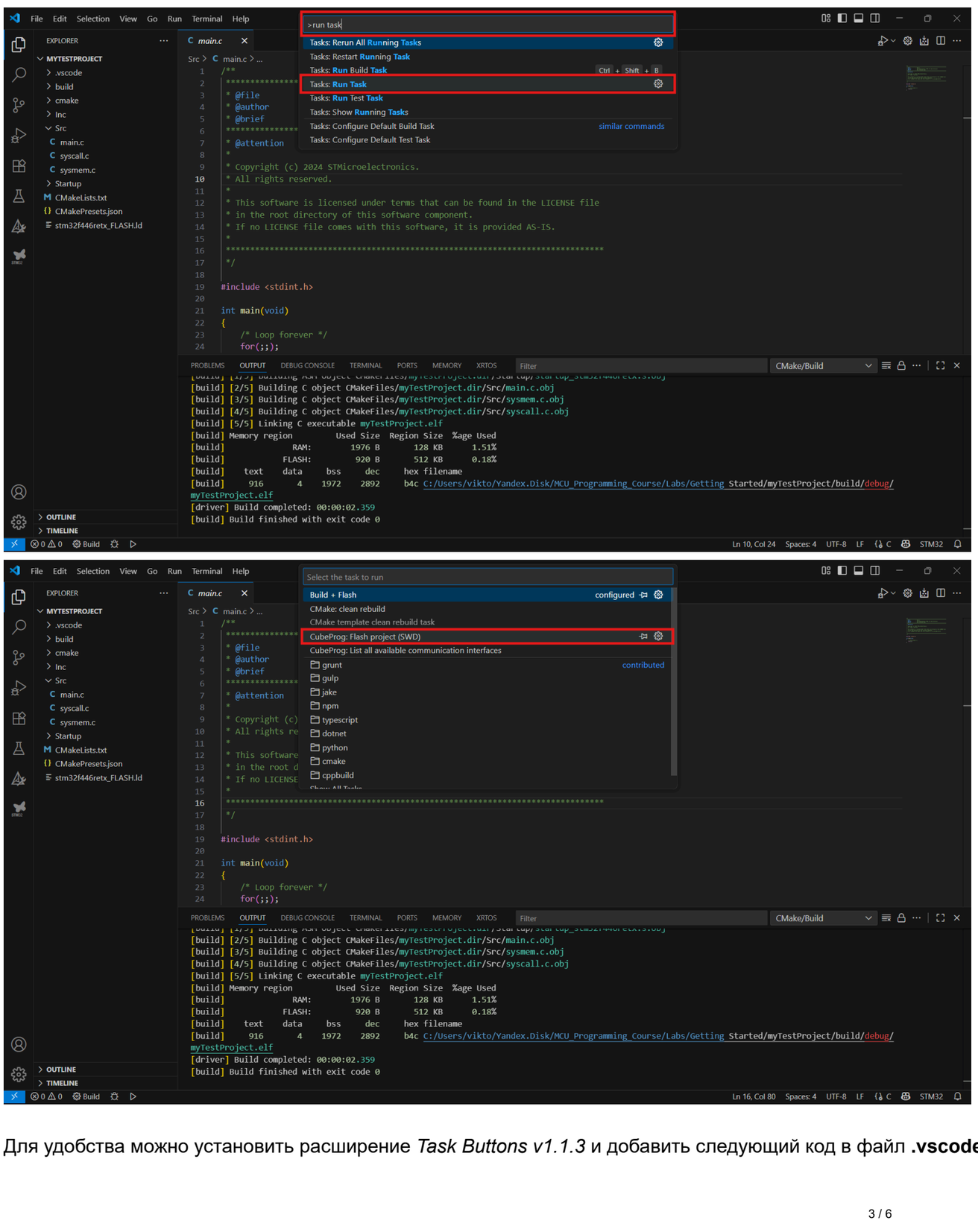
Компиляция проекта и прошивка контроллера

После подготовки **CMake** можно открыть файл *main.c* и собрать проект. Для этого нажмите кнопку *Build* (значок шестерёнки в нижней области окна).



Журнал сборки и результат появятся во вкладке *Output*.

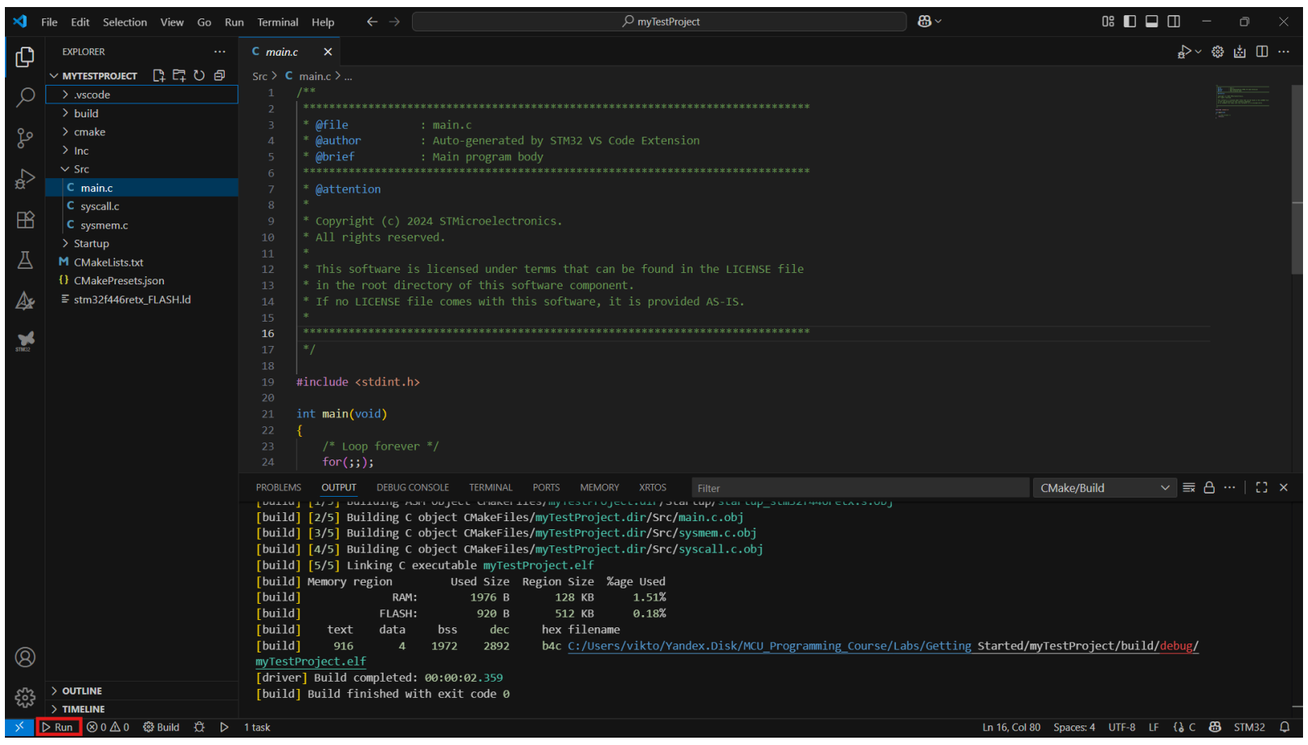
Для прошивки контроллера *STM32 Extension* автоматически сгенерировал задачу. Все сгенерированные задачи находятся в файле **.vscode/tasks.json**. Чтобы залить прошивку в контроллер, запустите задачу *CubeProg: Flash project (SWD)*. Для этого откройте командную палитру VS Code (*Ctrl* + *Shift* + *P*), введите **Run task**, нажмите *Enter* и выберите нужную задачу из списка.



Для удобства можно установить расширение *Task Buttons v1.1.3* и добавить следующий код в файл **.vscode/settings.json** (если файла нет — создайте его):

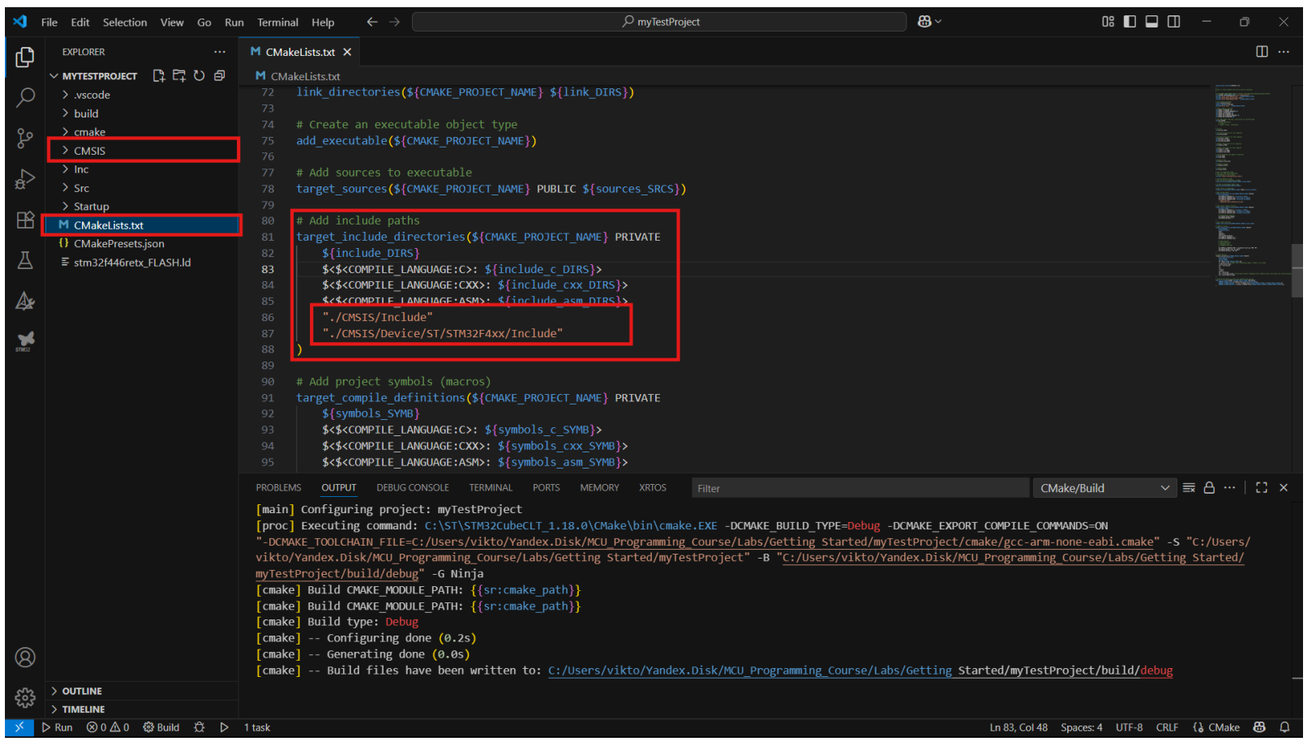
```
{
    "VsCodeTaskButtons.tasks": [
    {
        "label": "$(run) Run",
        "task": "CubeProg: Flash project (SWD)",
        "tooltip": "Build and run"
    },
    ]
}
```

После этого в нижней части экрана появится кнопка **Run** — при нажатии проект будет собран и загружен в МК. Аналогично можно добавить и другие задачи.



Подключение заголовков CMSIS

Чтобы подключить дополнительные заголовочные файлы, например **CMSIS**, сначала скопируйте их в папку проекта, затем укажите путь к ним в файле **CMakeLists.txt**.
После копирования папки **CMSIS** в проект откройте **CMakeLists.txt** и найдите раздел `# Add include paths`.



Добавьте пути к заголовкам в этот раздел:

```
"./CMSIS/Include"
"./CMSIS/Device/ST/STM32F4xx/Include"
```

В результате раздел должен выглядеть так:

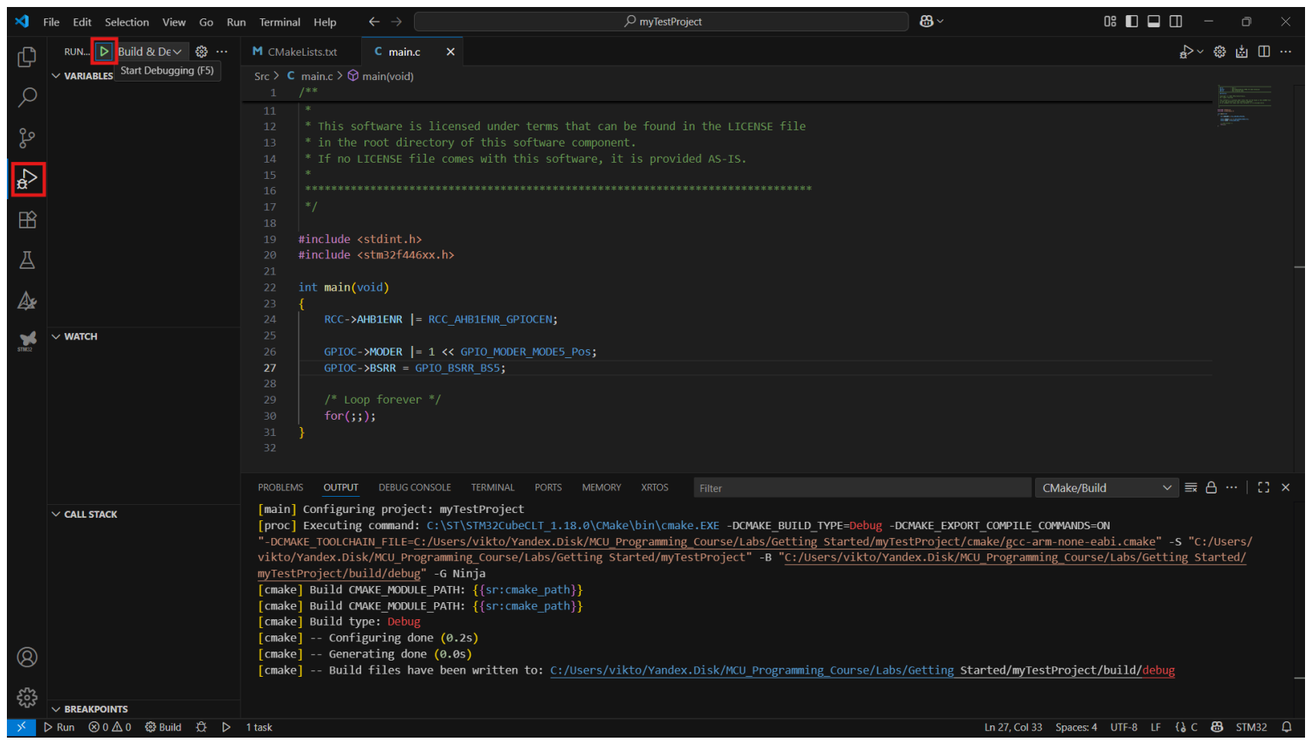
```
# Add include paths
target_include_directories(${CMAKE_PROJECT_NAME} PRIVATE
    ${include_DIRS}
    $<$<COMPILE_LANGUAGE:C>: ${include_c_DIRS}>
    $<$<COMPILE_LANGUAGE:CXX>: ${include_cxx_DIRS}>
    $<$<COMPILE_LANGUAGE:ASM>: ${include_asm_DIRS}>
    "./CMSIS/Include"
    "./CMSIS/Device/ST/STM32F4xx/Include"
)
```

После этого в файле **main.c** можно подключить заголовок выбранного МК: `#include "stm32f446xx.h"`.

Project Debugging

Для отладки *STM32 VS Code Extension* автоматически сгенерировал конфигурацию запуска отладчика и подключения VS Code к МК: файл **.vscode/launch.json**.

Чтобы начать отладку, нажмите **Run and Debug** на левой панели, затем **Start Debugging**.



Работа с регистрами микроконтроллеров

При работе с аппаратными регистрами микроконтроллеров часто требуется **установить, сбросить или изменить отдельные биты**, не задевая остальные.
Обычно для этого в C используют **побитовые операторы**.

Установка одного бита

Чтобы установить конкретный бит в `1`, используйте оператор **сдвига (`<<`)** для перемещения `1` в нужную позицию, затем примените **побитовое ИЛИ (`|`)**.
Это изменит только целевой бит, а остальные останутся без изменений.

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24	Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

```
*((uint32_t *) REGISTER_ADDRESS) = *((uint32_t *) REGISTER_ADDRESS) | (1 << 9);          // Установить бит 9
```

или проще

```
*((uint32_t *) REGISTER_ADDRESS) |= 1 << 9;
```

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24	Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0

Установка нескольких бит (битового поля)

Если нужно обновить поле (несколько последовательных бит), сдвиньте желаемое значение в нужную позицию:

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24	Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0

```
*((uint32_t *) REGISTER_ADDRESS) |= 5 << 6;          // Записать 0b101 = 5 в биты [8:6]
```

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24	Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	0	1	1	0	1	0	0	0	0	0	0

Сброс (очистка) одного бита

Чтобы сбросить бит в `0`, используйте **побитовое И (`&`)** с **инвертированной маской** (`~`).
Так целевой бит принудительно станет `0`, а остальные не изменятся.

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24	Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	0	1	1	0	1	0	0	0	0	0	0

```
*((uint32_t *) REGISTER_ADDRESS) &= ~(1 << 6)          // Сбросить бит 6
```

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24	Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0

Очистка битового поля

Та же логика применима к полям: создайте маску, покрывающую все биты поля, инвертируйте её и примените `&=`.

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24	Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0

```
*((uint32_t *) REGISTER_ADDRESS) &= ~(3 << 8)          // 3 = 0b11
```

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24	Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Инвертирование (переключение) бита

Чтобы **инвертировать** бит (`0→1` или `1→0`), используйте **побитовое XOR (`^`)**.

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24	Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

```
*((uint32_t *) REGISTER_ADDRESS) ^= (1 << 3);          // Переключить бит 3
```

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24	Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

```
*((uint32_t *) REGISTER_ADDRESS) ^= (1 << 3);          // Снова переключить бит 3
```

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24	Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Проверка (чтение) бита

Чтобы проверить, установлен ли бит:

```
if (*((uint32_t *) REGISTER_ADDRESS) & (1 << 12))
{
    // Бит 12 установлен
}
```