Микроконтроллер BE-U1000 AppNote

Работа с Visual Studio Code

Содержание

1 Введение

В документе представлены рекомендации по применению редактора Visual Studio Code для разработки и отладки программного обеспечения Микроконтроллера BE-U1000 (далее – МК) под ОС Windows.

Редактор Visual Studio Code с рядом настроек и расширений, позволяет создавать программы для МК, структурировать код программ и данные компилировать и исполнять программы, просматривать значения переменных, взаимодействовать с МК и отлаживать программы, используя точки останова и режим пошагового исполнения.

Для ознакомления с общей информацией о МК см. документы:

- «BE-U1000. Краткое описание» и «BE-U1000. Руководство пользователя».
- «BE-U1000. Техническая спецификация» и «BE-U1000. Reference Manual».

Минимальные требования для разработки программ МК в Visual Studio Code:

- Частота процессора 1.6 ГГц и выше
- Объем оперативной памяти 4 Гб и выше
- Объем свободного дискового пространства 100 Мб
- Наличие свободного разъема USB 2.0 и выше
- ОС Windows 8 или выше
- Visual Studio Code v.1.79.2 или выше
- SDK MCU для Windows ver.2.3.0 или новее (далее - SDK)

Представленные материалы рассчитаны на взаимодействие с платой МК EVU-BA-2.5.

2 Условные обозначения, термины и сокращения

В документе используются условные обозначения, сокращения и термины, указанные в таблице ниже.

Таблица 1 – Название таблицы Условные обозначения, термины и сокращения

Обозначение	Расшифровка
DFU	Протокол обновления прошивки через USB-интерфейс микроконтроллера BE-U1000
eFlash	Встроенная флэш-память МК. Содержит две области – Main и NVR.
JTAG	Joint Test Action Group – стандартизированный аппаратный интерфейс (IEEE 1149.1), предназначенный для тестирования и отладки электронных устройств
Makefile	Сборочный скрипт с инструкциями по сборке и компиляции программы
SDK	Комплект средств разработки – Набор утилит, API, инструментов, библиотек, примеров кода и документации, для разработки программного обеспечения микроконтроллера BE-U1000
TCM	Tightly Coupled Memory (тесно связанная память) — память типа SRAM на МК BE-U1000, расположенная рядом с ядром для более быстрого доступа
Visual Studio Code	Бесплатный редактор и расширяемая среда программирования от компании Microsoft
Бутстрап Bootstrap	Конфигурационный вывод - Вывод МК, предназначенный для выбора режима работы при начальной загрузке
МК MCU	Микроконтроллер BE-U1000
Плата	Отладочная плата EVU-BA-2.5 с установленным на ней микроконтроллером BE-U1000
Рабочее пространство Workspace	Папка или совокупность нескольких папок, объединяющих код программ, утилиты, библиотеки и служебные файлы, с которыми взаимодействует разработчик
Расширение Extension	Плагин – Программный модуль или пакет кода, добавляющий новые возможности Visual Studio Code
Терминал	Командная строка (консоль) внутри редактора, позволяющая запускать файлы и команды и следить за их выполнением, не переключаясь в консоль ОС
Точка останова Breakpoint	Преднамеренное прерывание выполнения программы, устанавливаемое разработчиком на определенной строке кода. Позволяет изучить состояние программы, проверить значения переменных и пошагово выполнять код для поиска ошибок.

3 Установка SDK

3.1 Распаковка и настройка переменных среды

SDK представляет из себя комплект утилит, библиотек, модулей и примеров кода, необходимых для программирования МК.

SDK для МК Baikal BE-U1000 под Windows можно скачать бесплатно с сайта https://mcu.baikalelectronics.ru/, как это показано на рисунке 1, в формате одного архива. Рекомендуется распаковать SDK в корневой каталог рабочего диска (например, в папку `C:\SDK\`), чтобы символический путь к файлам пакета был короче, так как дерево папок SDK может содержать очень длинные имена файлов и пути, превышающие ограничения Windows.

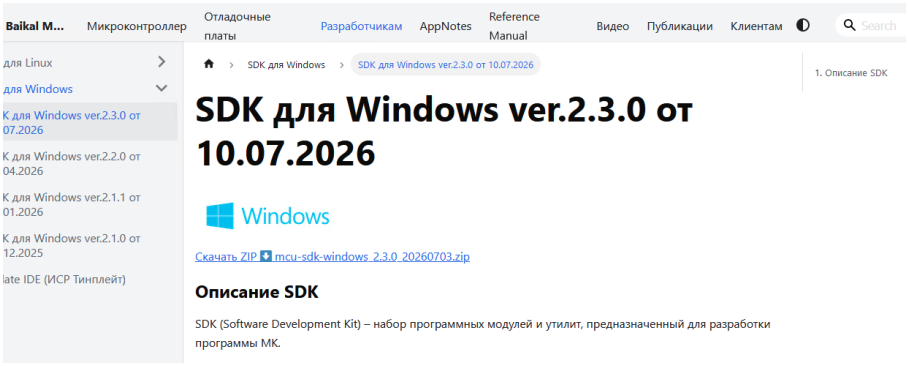

Рисунок 1 – Архив SDK на сайте «Байкал Электроникс»

Переход в терминал в Windows осуществляется через нажатие комбинации клавиш <Win>+<R> и вызов `cmd`.

Для корректного функционирования SDK и утилиты Makefile (мэйкфайл) в окне терминала перейти в папку `[SDK_Path]\Tools\toolchain\xpack-windows-build-tools` и запустить файл установки переменных среды add_to_path.bat.

В качестве альтернативы исполнения bat-файла можно добавить в системную переменную среды `Path` путь: `[SDK_Path]\Tools\toolchain\xpack-windows-build-tools\bin`.

3.2 Установка драйвера USB для взаимодействия через адаптер USB-JTAG

В этом руководстве описан способ взаимодействия с платой EVU-BA-2.5 через разъем JTAG, используя адаптер USB-JTAG. Для взаимодействия SDK с МК через JTAG установить драйвер. Эту операцию можно выполнить с помощью программы Zadig-2.9. Это программное обеспечение входит в состав SDK и находится в папке `[SDK_Path]\Tools`, как это показано на рисунке 2.

Установку программы выполнить самостоятельно или с помощью меню Visual Studio Code, как это описано в Разделе 5.

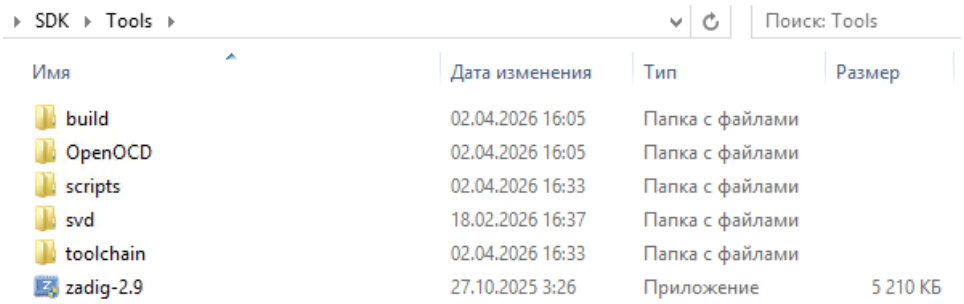

Рисунок 2 – Расположение файла программы Zadig-2.9 в каталоге SDK

Для настройки драйверов в системе с помощью программы Zadig микроконтроллер должен быть подключен к ПК через USB-разъем JTAG/UART0, как показано на рисунке 3, а перемычки на плате должны быть выставлены в положения: «JTAG» и «питание USB», как это показано на рисунке 4.

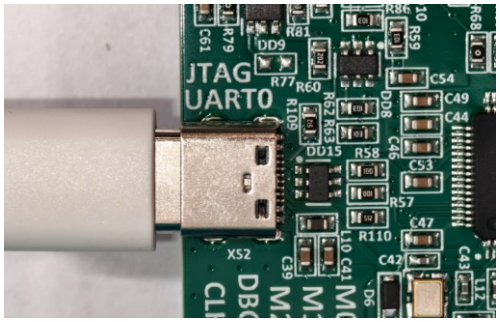

Рисунок 3 – Разъем для подключения в режиме JTAG/UART0

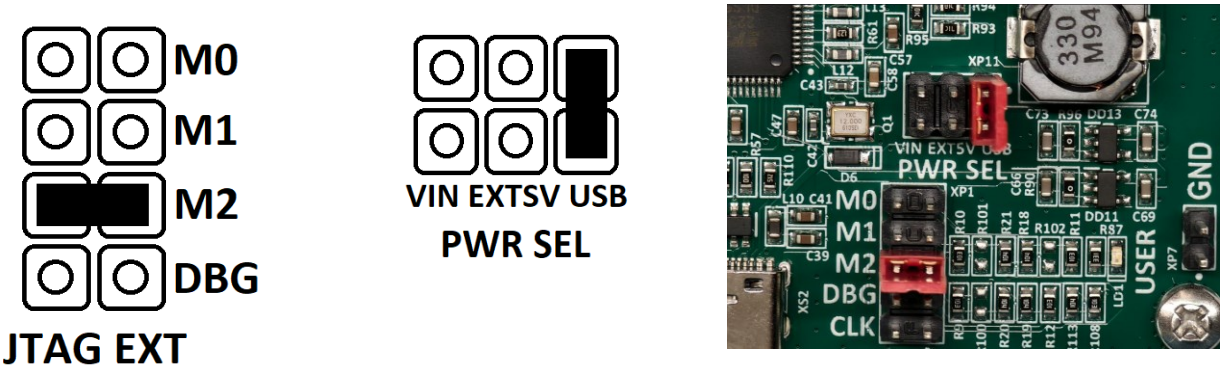

Рисунок 4 – Положение перемычек на плате

После запуска программы Zadig на экран выводится окно, показанное на рисунке 5.

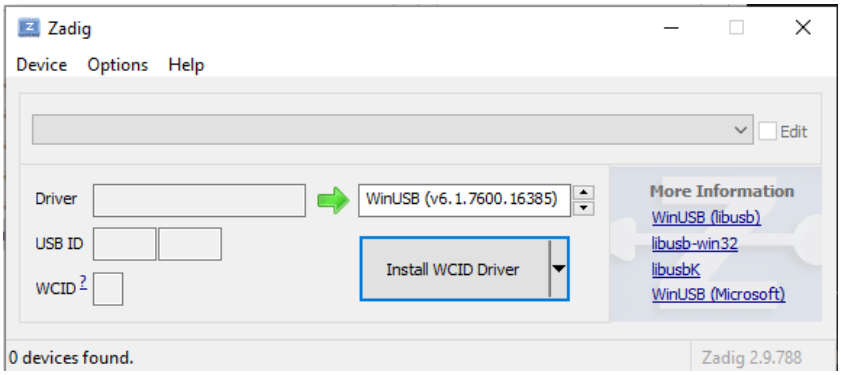

Рисунок 5 – Интерфейс программы Zadig непосредственно после запуска

В интерфейсе программы Zadig, в разделе Options (Опции), включить отображение всех доступных устройств, как показано на рисунке 6.

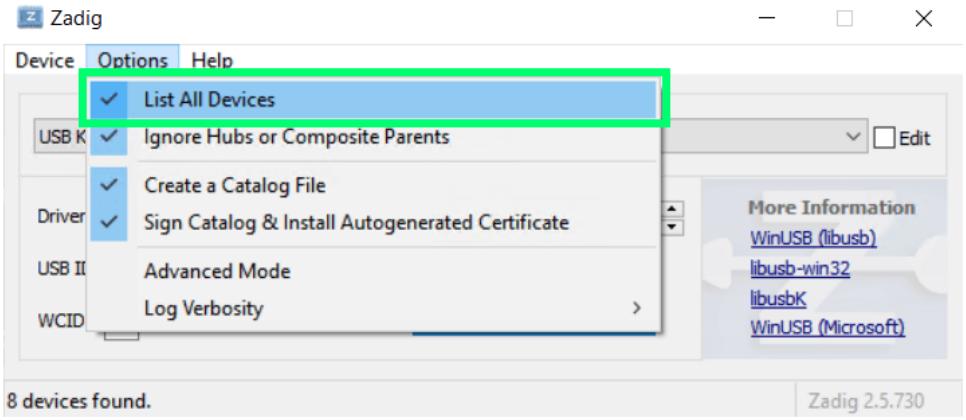

Рисунок 6 – Опция отображения всех устройств

Далее, в списке устройств (1) необходимо выбрать JTAG, как это показано на рисунке 7, так же обозначенный Interface 1, а UART соответствует Interface 0. В такой ситуации используется адаптер USB-to-JTAG, расположенный на плате и внешний по отношению к самому контроллеру.

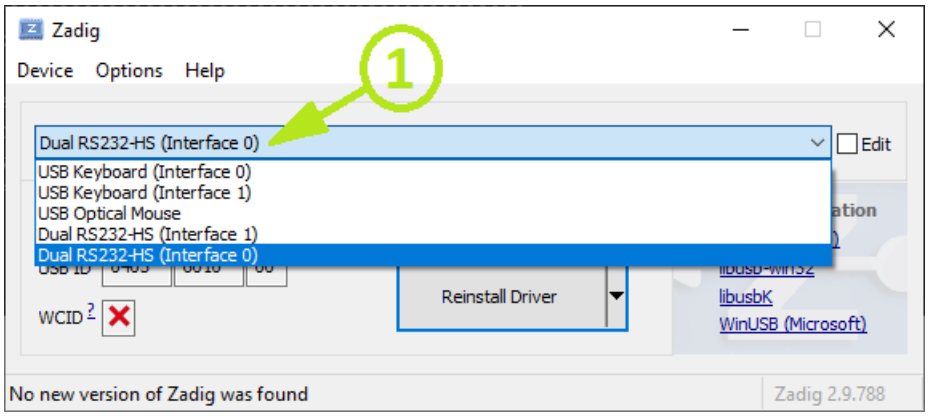

Рисунок 7 – Выбор устройств

После выбора интерфейса устройства в поле (2) ниже нужно выбрать драйвер WinUSB и нажать на расположенную ниже кнопку Install Driver (3), как показано на рисунке 8.

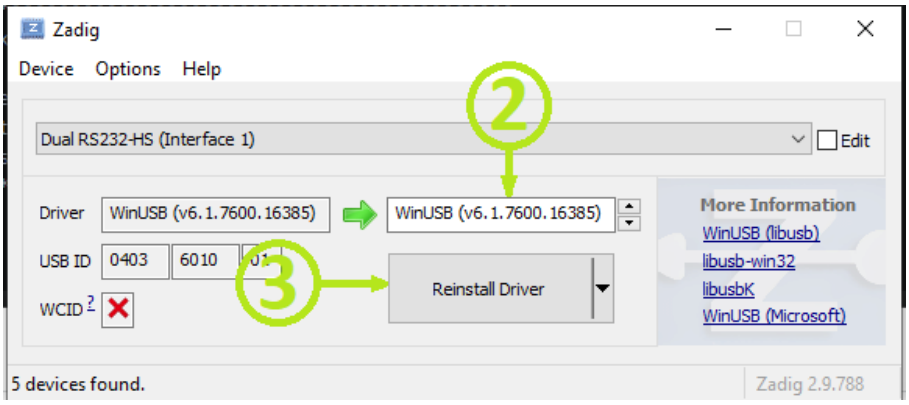

Рисунок 8 – Установка драйвера

При успешной установке драйвера будет выведено сообщение о завершении операции, показанное на рисунке 9. В поле названия драйвера будет указано WinUSB.

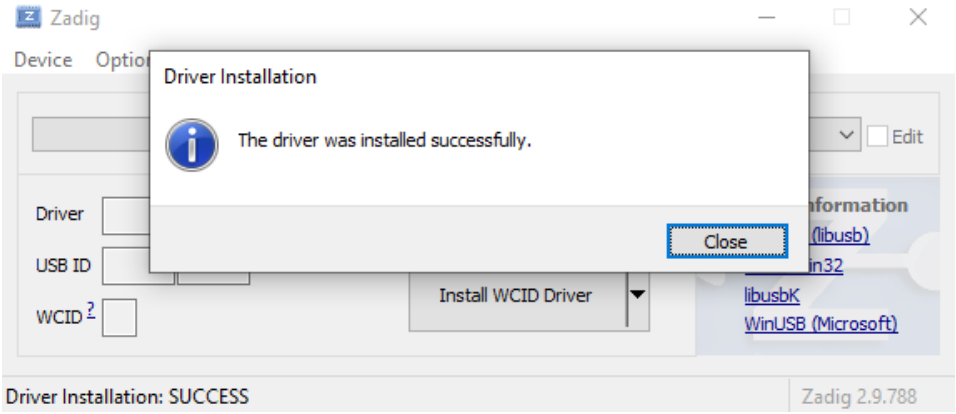

Рисунок 9 – Успешная замена драйвера

4 Установка Visual Studio Code

4.1 Установка редактора Visual Studio Code и его расширений

Редактор Visual Studio Code доступен для загрузки с сайта разработчика https://code.visualstudio.com/download. Установка проходит почти полностью в автоматическом режиме, без участия пользователя. В зависимости от типа дистрибутива, может потребоваться выбрать опции размещения иконки вызова и автоматического запуска редактирования для определенных типов файлов. Никаких настроек, влияющих на работу программы, на этом этапе производить не требуется.

4.2 Типичный вид графического пользовательского интерфейса Visual Studio Code

Общий вид интерфейса Visual Studio Code по умолчанию показан на рисунке 10.

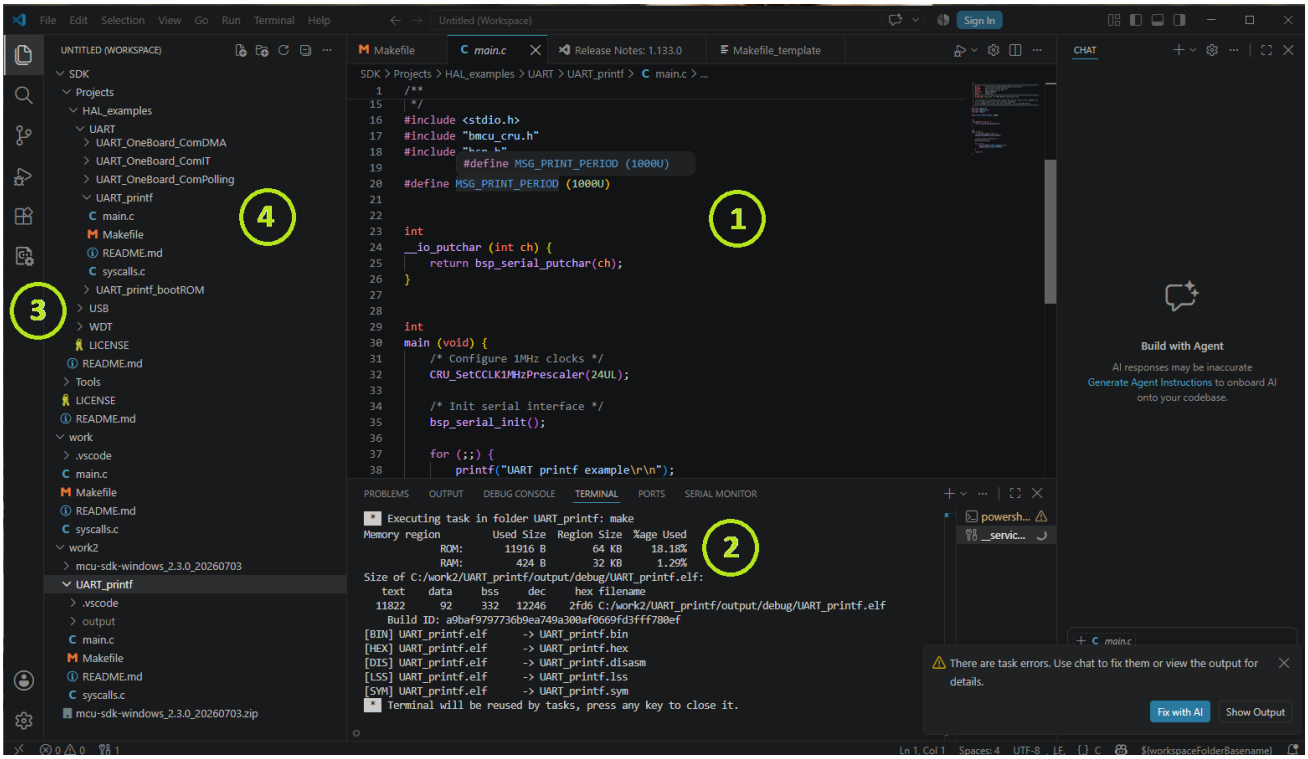

Рисунок 10 – Общий вид интерфейса Visual Studio Code по умолчанию

(1) – Редактор кода программы пользователя (Editor). Переключение между разными файлами в редакторе осуществляется в горизонтальной строке меню над текстом кода, с помощью закладок.

(2) – Нижняя сервисная панель (Panel) чаще всего используется как окно терминала Visual Studio Code (по умолчанию может отсутствовать, вызывается комбинацией клавиш <Ctrl>+<’>). Может использоваться в управлении процессами отладки, запуска программ на МК и процессов загрузки образов программ в память МК и т.п. Так же панель имеет вкладки:

- Проблемы (Problems) – в ней отображаются ошибки в коде и прочие ошибки;
- Вывод (Output) — в эту вкладку выводится информация по сборке и работе IDE;
- Консоль отладки (Debug console) – в этой вкладке можно управлять отладкой;
- Терминал (Terminal) – кроме функционала собственно терминала, в этой вкладке отображаются
- служебные фоновые процессы;

Также, в этой панели могут отображаться вкладки различных установленных расширений.

(3) – Панель действий с пиктограммами для выбора функций левого сервисного окна (Activity bar).

(4) – Левая сервисная панель, функции которой определяются выбранной пиктограммой в поле (3). Обычно в процессе работы эта панель используется для обзора рабочего пространства, в котором отображено дерево подключенных каталогов файлов, задействованных в проекте. В режиме отладки на этой панели отображаются текущие значения выбранных переменных, точки останова, значения ячеек памяти, стек и регистры, как это показано на рисунке 11.

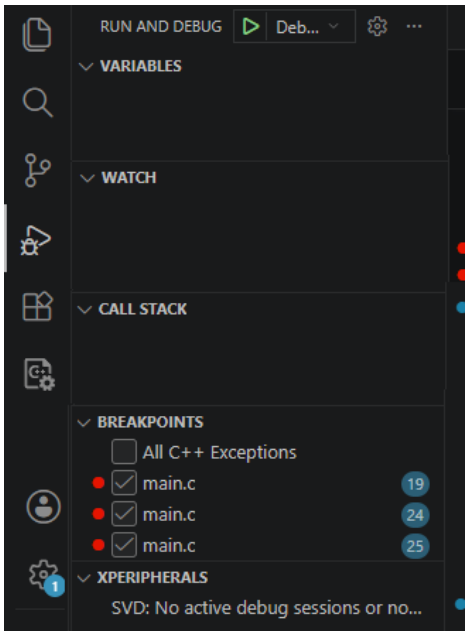

Рисунок 11 – Вид сервисного окна в режиме отладки

4.3 Расширения Visual Studio Code

Для расширения функций Visual Studio Code для определенного вида задач используются т.н. программные расширения. Просмотреть установленные программные расширения, найти программные расширения или добавить новые можно выбрав пиктограмму расширений в поле интерфейса (3), как это показано на рисунке 10. Расширения могут устанавливаться через Visual Studio Code из встроенного Магазина приложений (Extensions Market, <Ctrl>+<Shift>+<X>) или через VSIX-файл.

Рекомендуется установить программное расширение C/C++ от Microsoft (ms-vscode.cpptools). После установки этого расширения Visual Studio Code превратится в полноценную среду разработки на языках программирования C и C++. Опционально рекомендуется установить расширения Makefile Tools (ms-vscode.makefile-tools) и Peripheral Viewer (mcu-debug.peripheral-viewer). На рисунке 12 эти плагины показаны в библиотеке Visual Studio Code.

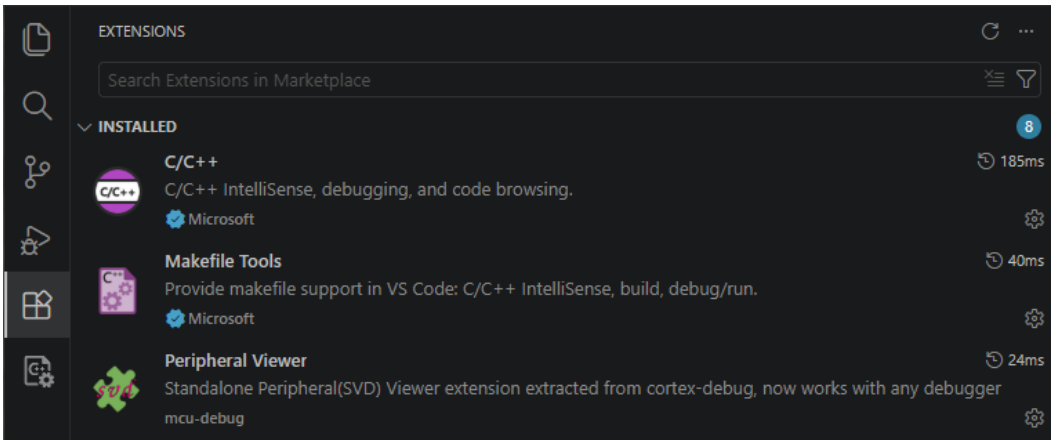

Рисунок 12 – Расширения для работы с C/C++

На ваше усмотрение, можно так же поставить расширение ms-ceintl.vscode-language-pack-ru от Microsoft, которое переводит язык интерфейса Visual Studio Code на русский язык. Далее, мы указываем названия элементов среды Visual Studio Code на английском языке, с русским переводом.

5 Конфигурация Visual Studio Code для работы с MCU SDK

В структуре файлов SDK находится папка `C:/SDK/Tools/IDE/.vscode`. В ней размещаются файлы .json с настройками среды Visual Studio Code, как это показано на рисунке 13. Далее, эту папку скопировать в свою рабочую папку проекта, например: `C:/WORK/.vscode`.

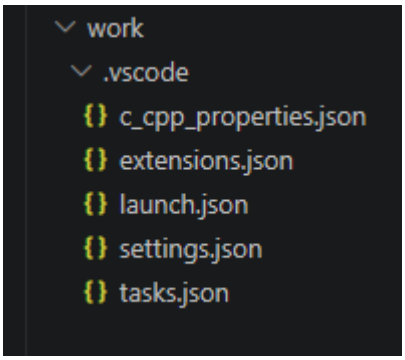

Рисунок 13 – Файлы папки vscode

Настройки, определенные в файлах .json в SDK по умолчанию, являются достаточными для создания программного обеспечения для МК с применением Visual Studio Code. Вносить в них изменения самостоятельно для решения базовых задач не обязательно.

Настройки среды сохраняются в файле settings.json. Их можно посмотреть из окна Настройки (Settings). Вызов этого окна осуществляется через меню File > Preferences > Settings или комбинацией клавиш <Ctrl>+<,>. Пример настроек приведен на рисунке 14: в этом файле настраиваются свойства агентов, чата, окон, сессий, инструментов, терминала и т.п.

```
{
  // Внешний вид
  "workbench.colorTheme": "Default Dark+",
  "editor.fontSize": 14,
  "editor.fontFamily": "Consolas, 'Courier New', monospace",
  "editor.fontLigatures": true,
  "editor.lineHeight": 24,
  "editor.letterSpacing": 0.5,

  // Отступы и пробелы
  "editor.tabSize": 2,
  "editor.insertSpaces": true,
  "editor.detectIndentation": true,
  "editor.trimAutoWhitespace": true,
```

Рисунок 14 – Пример настроек в файле settings.json

Файл tasks.json позволяет установить вызовы различных функций Visual Studio Code без использования окна терминала. Таким же образом в файле launch.json можно настроить функции горячих клавиш и меню среды Visual Studio Code, как это показано на рисунке 15.

```
[
    {
        "key": "f5",
        "command": "workbench.action.debug.start",
        "when": "inDebugMode"
    },
    {
        "key": "f6",
        "command": "workbench.action.debug.continue",
        "when": "inDebugMode"
    },
    {
        "key": "f7",
        "command": "workbench.action.debug.stop",
        "when": "inDebugMode"
```

Рисунок 15 – Пример настройки горячих клавиш

После установки файлов .json выполнить обновление настроек. Для этого вызвать выпадающее меню из верхней строки, расположенной над окнами работы с кодом, как это показано на рисунке 16.

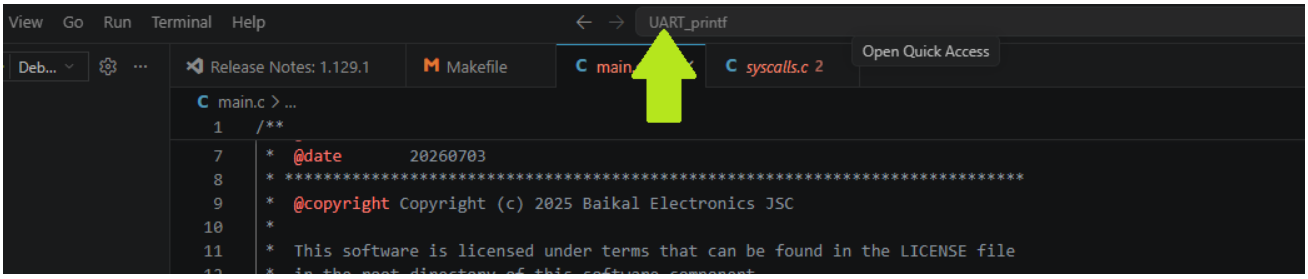

Рисунок 16 – Верхнее выпадающее меню

В этом меню вызвать пункт Запуск задачи (Run task), как показано на рисунке 17 и далее Обновить настройки (Update settings), как показано на рисунке 18.

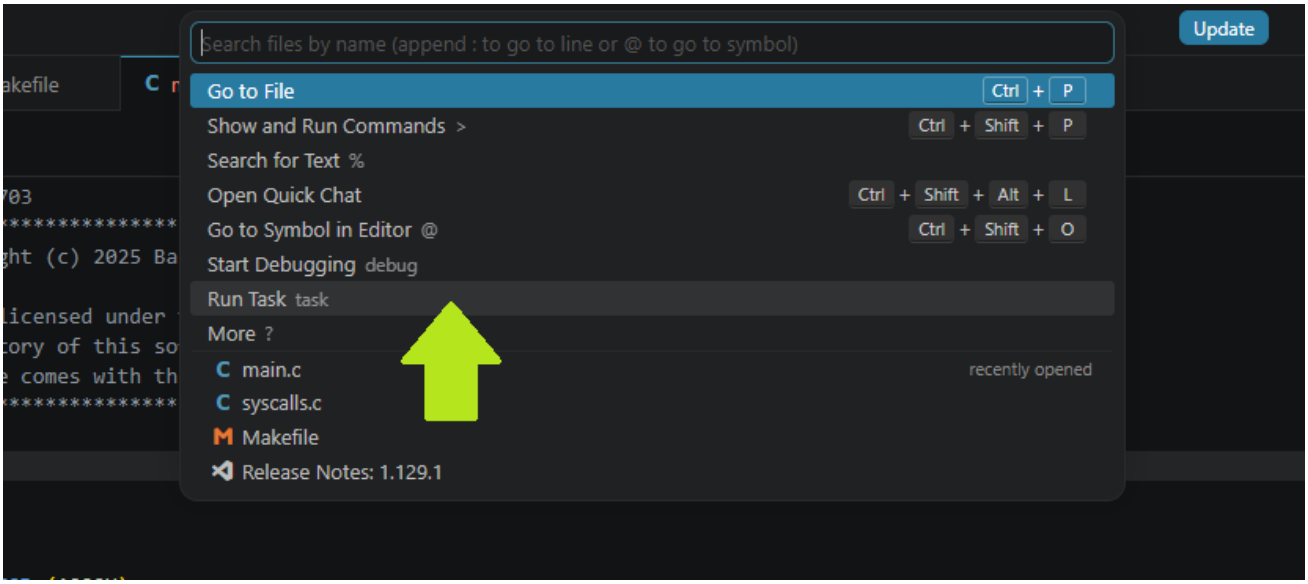

Рисунок 17 – Пункт Run Task

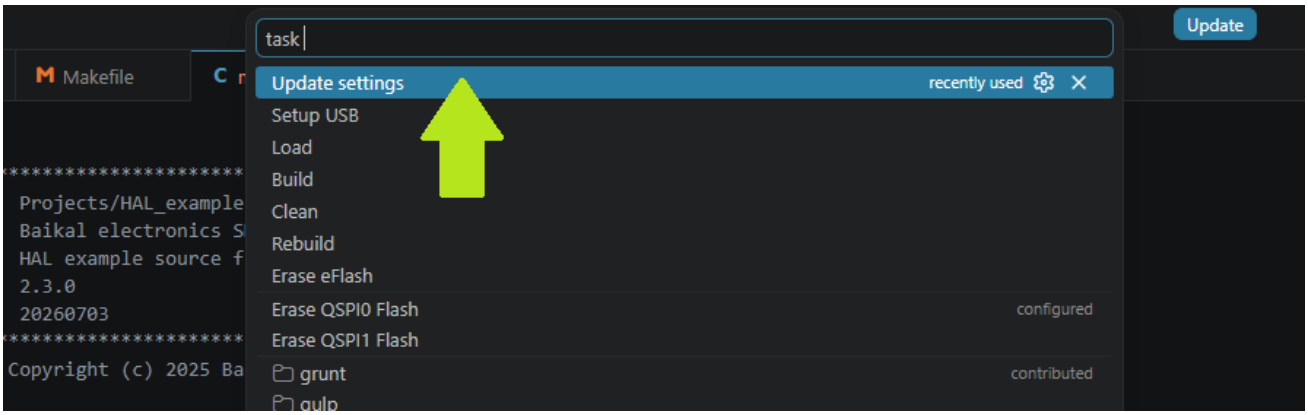

Рисунок 18 – Обновление настроек

Эту операцию желательно производить при каждом изменении настроек Visual Studio Code.

6 Создание проекта МК

6.1 Создание рабочего пространства

Для создания свой проекта, можно в качестве основы использовать один из готовых примеров проектов из состава SDK. В разделе `[SDK_Path]/Projects/HAL_examples/` приведены примеры проектов для демонстрации различных возможностей МК, как это показано на рисунке 19. Нужный пример рекомендуется скопировать в свою папку (например: `C:/WORK/`) пространство и далее изменять под свои нужды.

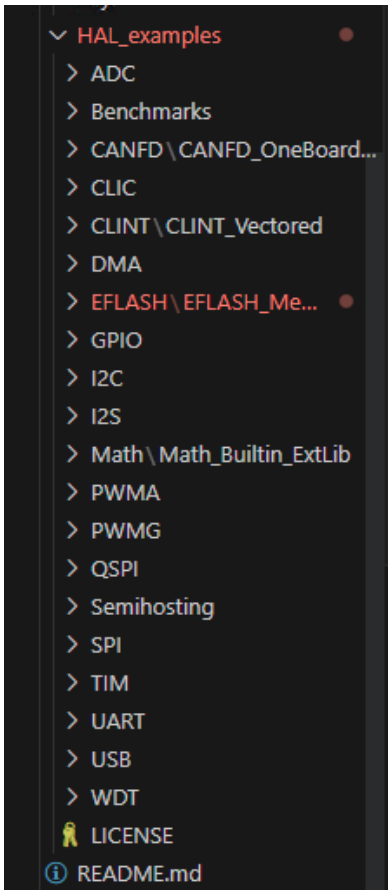

Рисунок 19 – Примеры проектов

Папки, с которыми вы работаете, и к которым должен иметь доступ Visual Studio Code, должны быть добавлены в ваше рабочее пространство. Для этого нажать Добавить папку в рабочее пространство (Add – Folder or workspace), как это показано на рисунке 20.

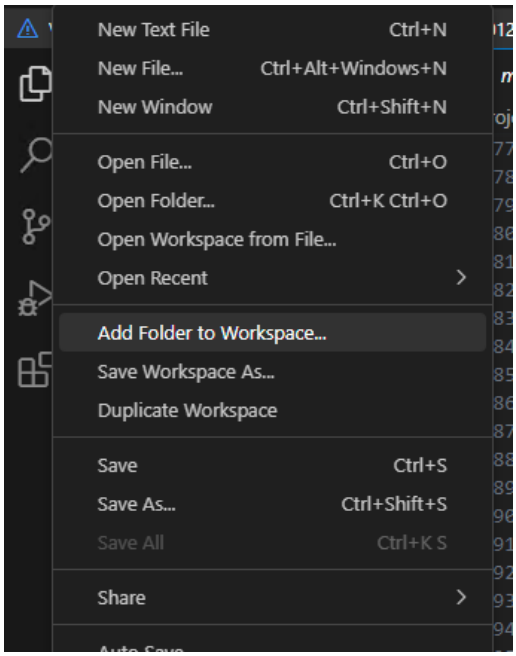

Рисунок 20 – Добавление папки в рабочее пространство

Для того, чтобы отобразить всю структуру папок, нажать Select folder, как это показано на рисунке 21. Кроме папки work добавить в рабочее пространство и папку SDK.

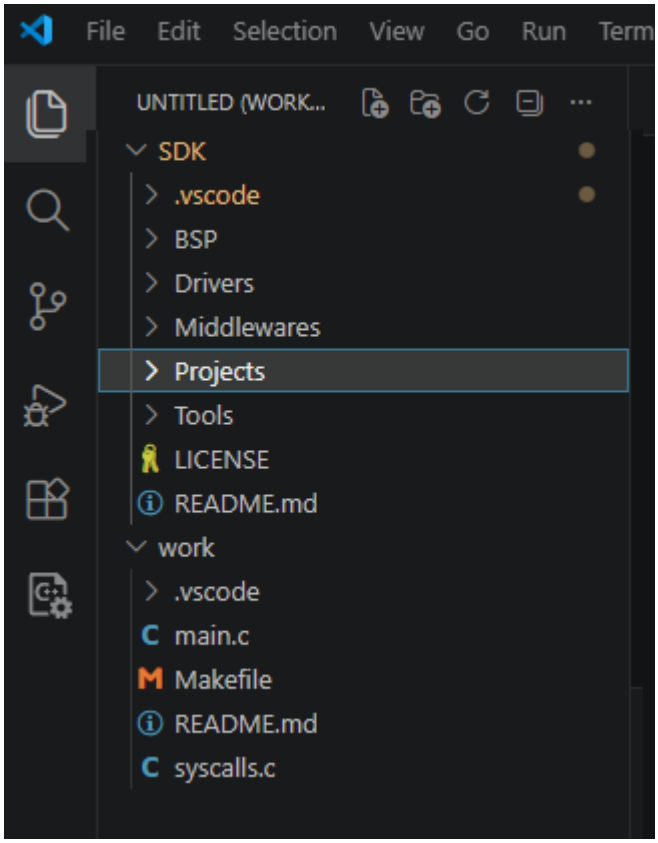

Рисунок 21 – Структура папок в рабочем пространстве

В документе рассматривается пример проекта `UART_printf`. Этот пример необходимо скопировать из папки `[SDK_Path]/Projects/HAL_examples/UART`, как показано на рисунке 22, в рабочую папку (например, ранее созданную `C:/work`).

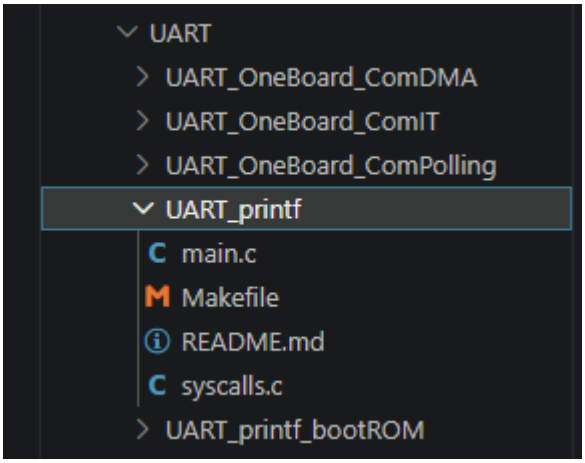

Рисунок 22 – Папка с примером UART_printf

6.2 Настройка Makefile

Проект должен содержать файлы Makefile и файл с исходным кодом программы (например, main.c). В Makefile указать пути к файлам SDK в строке 2, как это показано на рисунке 23.

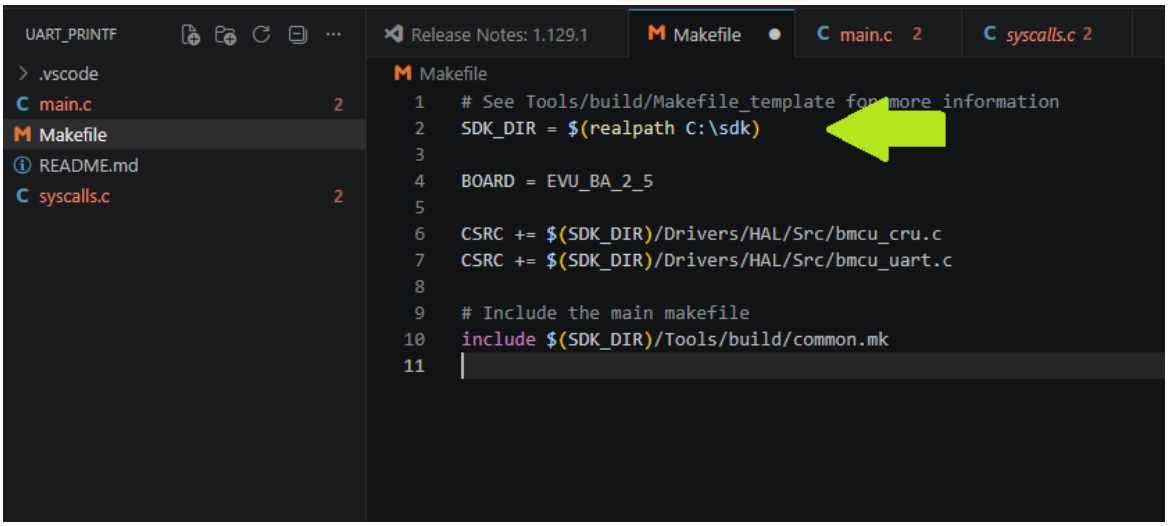

Рисунок 23 – Указание путей в Makefile

В путях файлов ОС Windows и ОС Linux применяются разные знаки «/» («прямая косая черта») и «\» («обратная косая черта»). Желательно придерживаться порядка, показанного на рисунке 23 и применять в пути Makefile знаки «\», например: `C:\SDK`.

При сборке программы компилятор сам «находит» все файлы с исходным кодом во вложенных папках проекта по умолчанию.

После внесения изменений в конфигурацию Visual Studio Code, файлы .json и Makefile, желательно обновить конфигурацию, как это показано в Разделе 5, на рисунках 17 и 18.

Настройки, Правила сборки (makifile rules), используемые в SDK, добавляют пути к стандартным библиотекам C/C++ автоматически, но пути к библиотекам `../HAL` необходимо добавить вручную, в зависимости от используемых модулей.

7 Сборка образа проекта в Visual Studio Code

7.1 Установка параметров компиляции и сборки

Для сборки образа настроить ряд параметров компилятора в Makefile.

Детальное описание параметров приводится в файле `[SDK_Path]\Tools\build\Makefile_template`, входящем в состав SDK. Для большинства параметров подходят настройки по умолчанию.

Пример описания настроек параметров показан на рисунке 24.

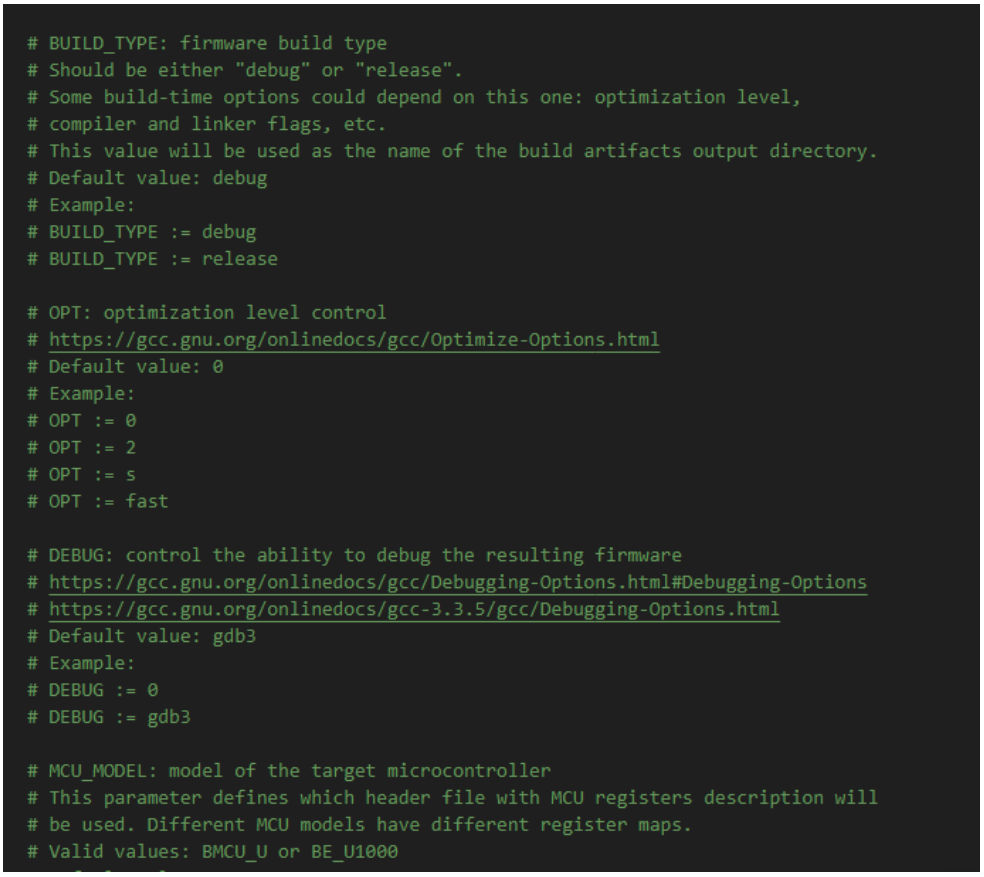

Рисунок 24 – Пример настроек и опций компилятора

Наиболее полезными могут быть следующие настройки:

- расположение директории (папки) SDK: `SDK_DIR`. В этом поле нужно указать путь к папке SDK, например, `C:/SDK/`;
- расположение директорий (папок) файлов исходного кода проекта и заголовочных: `SRC_DIR`, `SRC_DIR_EXCLUDE` и `INC_DIR`. В этом поле нужно указать путь к рабочей папке проекта, например, `C:/WORK/`;
- тип сборки программы - для исполнения или отладки: `BUILD_TYPE`. Результаты и порядок компиляции могут различаться в зависимости от того, предполагается ли отладка программы. Если предполагается отладка программы, необходимо

присвоить этой опции значение `debug`, а если предполагается компиляция окончательной версии, то `release`.

- уровень оптимизации компиляции: `OPT`. Доступны следующие степени оптимизации: `0`, `1`, `2`, `3`, `s` и `fast`;
- тип платы микроконтроллера: `BOARD`. По умолчанию установлена версия платы `EVU_BA_2_5`. Если вы пользуетесь другой платой, необходимо указать ее;
- используемый тип памяти МК для размещения кода программы: `MEM_REG_ROM`. Для отладки программы предпочтительно использовать память `TCMA` или `TCMB`. Если программа отлажена, ее можно поместить в долговременную память `EFLASH`;
- используемый тип памяти МК для размещения данных программы: `MEM_REG_RAM`. Для ускорения работы программы, при использовании `TCMA` или `TCMB` памяти, желательно размещать код и данные в разных блоках TCM, например, если код размещен в `TCMA`, то данные целесообразно разместить в `TCMB`.

7.2 Настройка взаимодействия МК с Visual Studio Code

Если вы не настроили взаимодействие с МК через разъем JTAG/UART ранее, то это можно сделать из меню управления Visual Studio Code. Выбрать пункт Run task, как это показано на рисунке 17, и далее Настроить USB (Setup USB), как это показано на рисунке 25.

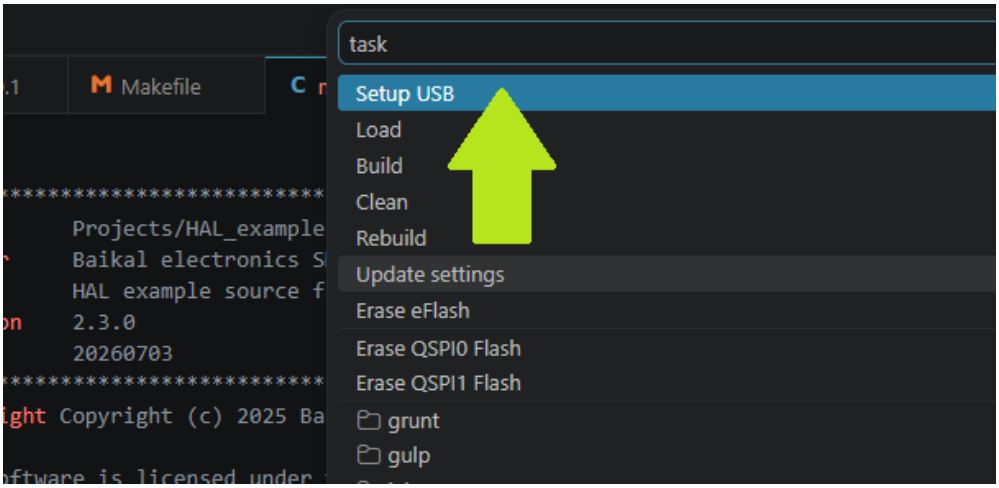

Рисунок 25 – Настройка соединения с МК по USB

После выбора этого пункта меню Visual Studio Code автоматически настроит соединение и вызовет установку программы Zadig, как это описано в Раздел 3, рисунки 2 – 9. Если вы уже настроили соединение по USB ранее, то этот пункт можно пропустить.

При многократном подключении/отключении МК, ОС может утратить настройки драйвера USB и вернуться к своим штатным настройкам, что вызовет исчезновение связи между вашим ПК и МК. В этой ситуации настройку необходимо повторить, как это описано в этом подразделе.

Перед настройкой драйверов USB убедитесь, что перемычки на плате МК выставлены правильно, согласно указаниям Раздел 5, рисунки 3, 4. Также необходимо проверить, что контактные площадки в перемычках присутствуют на своих местах.

7.3 Компиляция и сборка образа программы

Если предполагается отладка программы, ее сначала удобнее собрать для исполнения в ТСМ. В этом случае образ программы загружается в оперативную память МК, где ее можно исполнять многократно, не переключая перемычки конфигурационных выводов. После успешной отладки программы ее можно собрать для размещения во флеш-памяти.

Для сборки образа прямо из оболочки Visual Studio Code выбрать в выпадающем меню над окном редактирования кода пункт Run task, как это показано на рисунке 26.

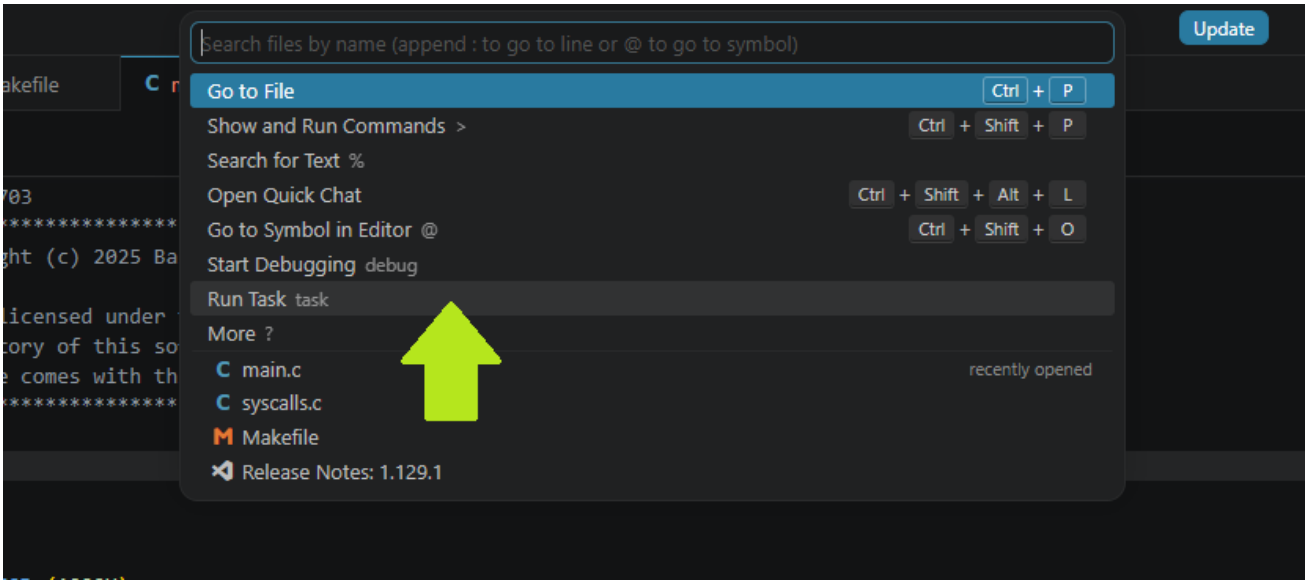

Рисунок 26 – Пункт Run Task

Далее выбрать в подменю пункты Компиляция и сборка (Build) или Компиляция и сборка с предварительным удалением результатов предыдущих компиляций (Rebuild), как это показано на рисунке 27.

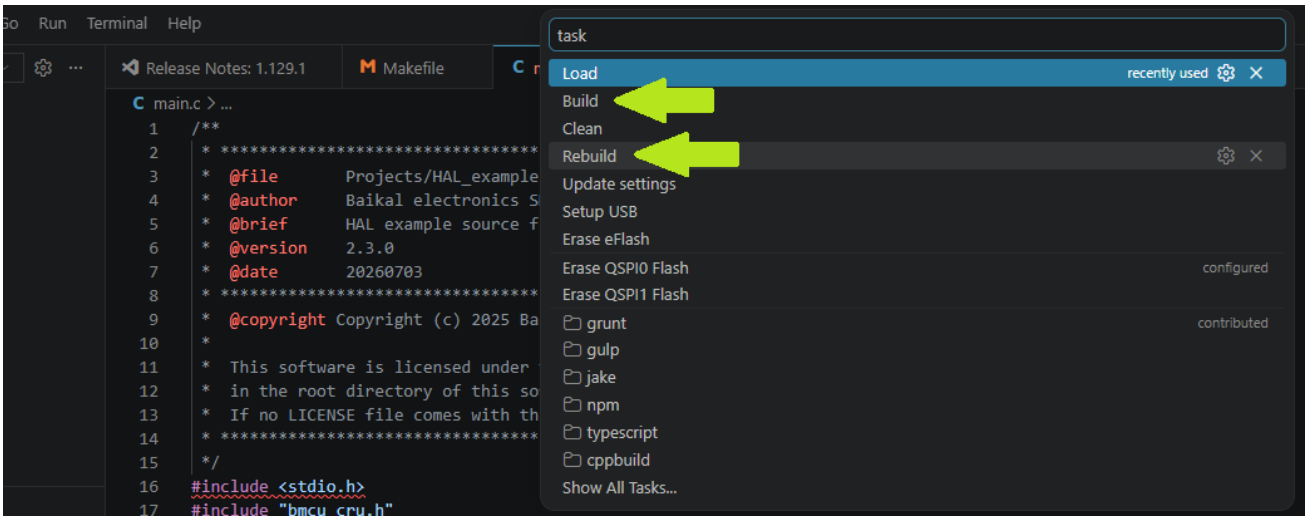

Рисунок 27 – Выбор сборки из Visual Studio Code

Если сборка прошла успешно, сообщение об этом будет выведено в окне терминала, как это показано на рисунке 28. По результатам сборки будут сформированы файлы UART_printf.elf, UART_printf.bin и ряд других. По умолчанию результаты компиляции будут размещаться в сформированной при компиляции в директории `output/debug`, как это показано на рисунках 29 и 30.

```
16  #include <stdio.h>
17  #include "bmcu_cru.h"
18  #include "bsp.h"
19
20  #define MSG_PRINT_PERIOD (1000U)
21

PROBLEMS 2  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SERIAL MONITOR

 *  Executing task: make

[CC ] main.c                -> main.o
[LD ] generated.ld          : UART_printf.elf
Memory region         Used Size  Region Size  %age Used
            ROM:       11916 B       256 KB      4.55%
            RAM:         424 B        32 KB      1.29%
Size of C:/work2/UART_printf/output/debug/UART_printf.elf:
   text    data     bss     dec     hex filename
  11822      92     332   12246    2fd6 C:/work2/UART_printf/output/debug/UART_printf.elf
    Build ID: 8dd74a25cadafa45c2fbb2d7757af99351325c6f
[BIN] UART_printf.elf       -> UART_printf.bin
[HEX] UART_printf.elf       -> UART_printf.hex
[DIS] UART_printf.elf       -> UART_printf.disasm
[LSS] UART_printf.elf       -> UART_printf.lss
[SYM] UART_printf.elf       -> UART_printf.sym
 *  Terminal will be reused by tasks, press any key to close it.
```

Рисунок 28 – Сообщение об удачной сборке

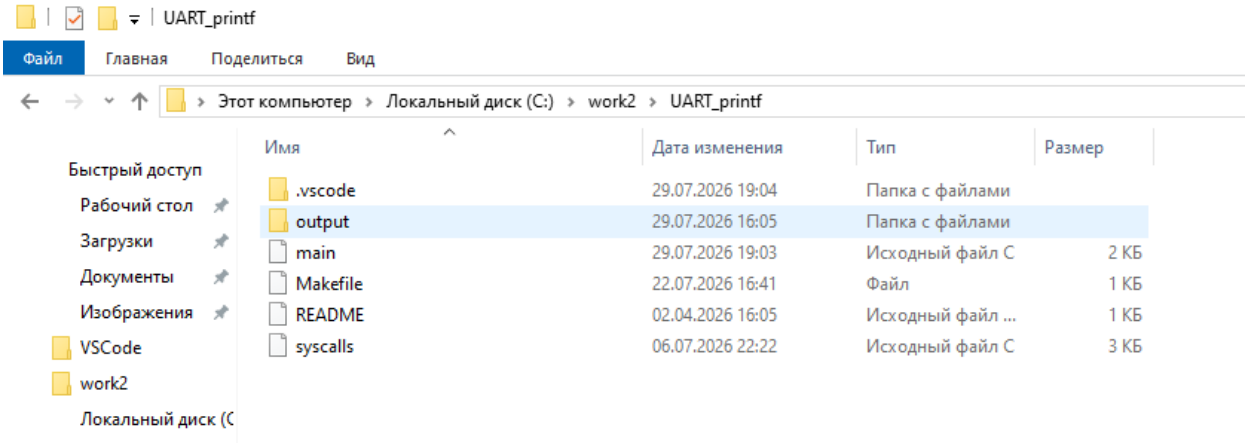

Рисунок 29 – Папка output, сформированная при компиляции

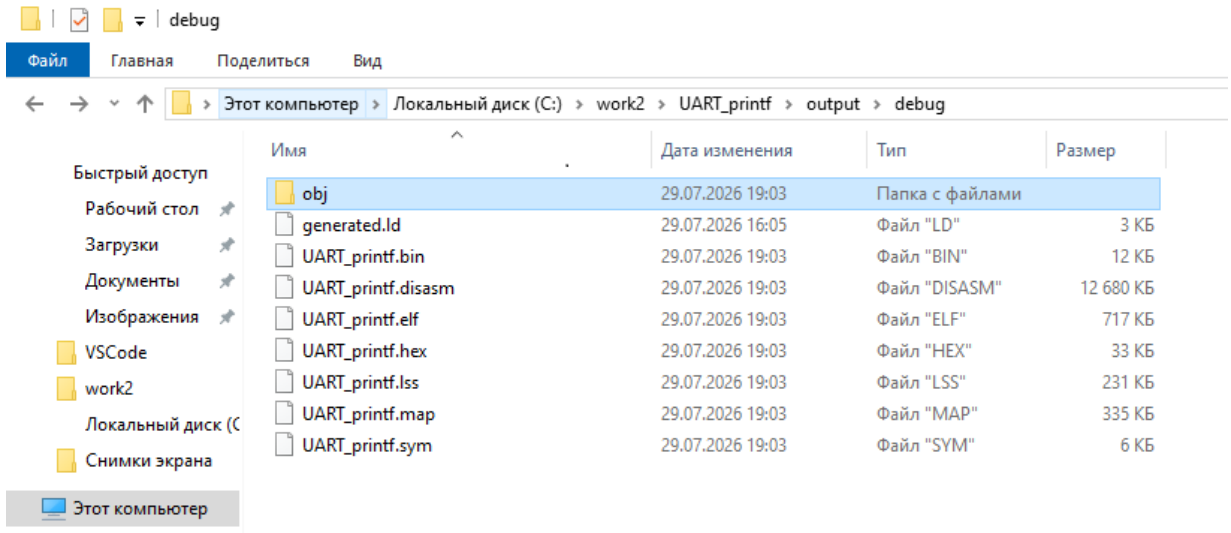

Рисунок 30 – Файлы, сформированные при компиляции программы

При обнаружении ошибок компилятором, сборка не будет завершена. В этой ситуации в окне терминала появится сообщение об ошибке, а файлы .elf и .bin сгенерированы не будут, как показано на рисунке 31. В ряде случаев Visual Studio Code может подсвечивать ошибку в коде программы.

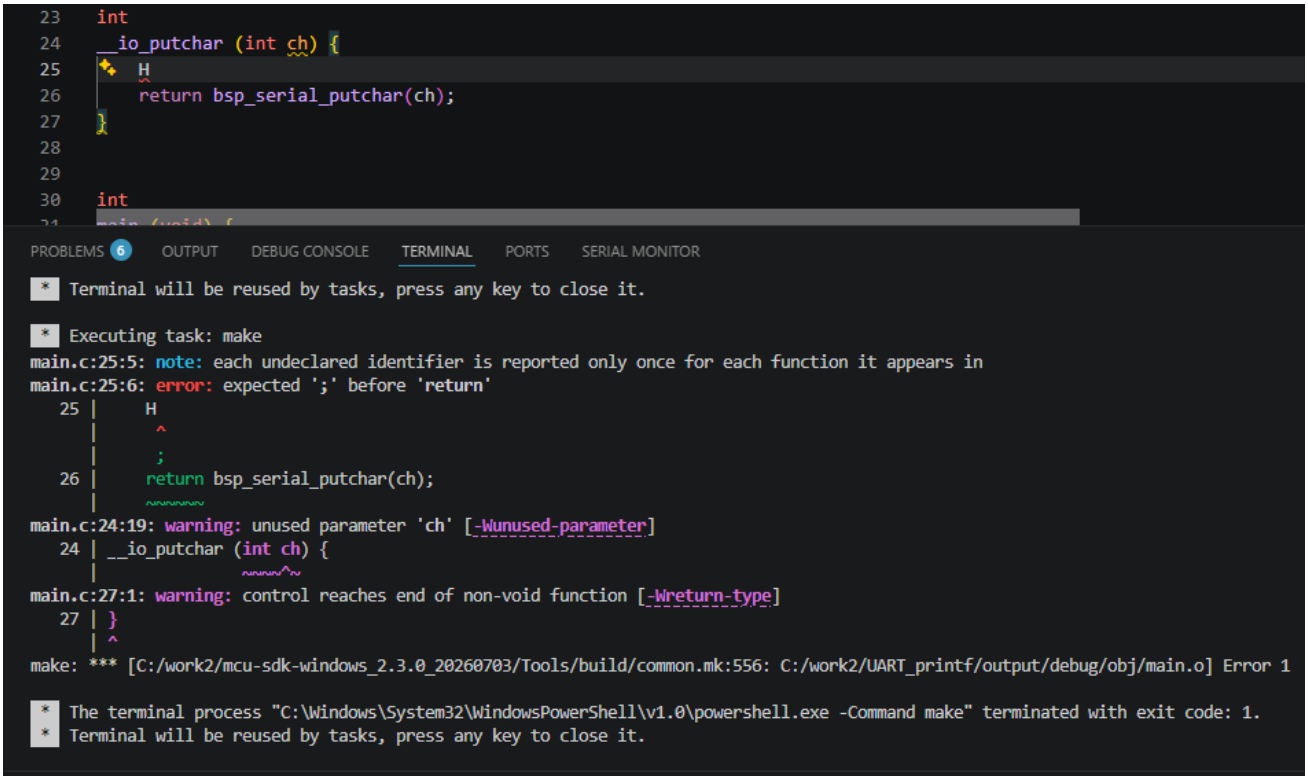

Рисунок 31 – Сообщение об ошибке при неудачной сборке

Сборку можно произвести также из терминала, вызвав из командной строки: `make`.

При вызове из командной строки, можно добавлять в качестве параметров опции компилятора, например, тип памяти: `make MEM_REG_ROM=EFLASH`.

8 Загрузка образа в МК, запуск исполнения

Загрузка образа программы в МК может осуществляться через:

- JTAG/UART0 через USB-UART-адаптер , как описано в разделе Глава 3, рисунки 3 и 4;
- DFU.

Для загрузки образа через DFU на плате EVU-BA-2.5 воспользоваться разъемом MCU и выставить перемычки на плате в состояние USB. Процедура загрузки образа программы через DFU описана в документе «Микроконтроллер Baikal-U (BE-U1000) Руководство пользователя».

Загружаемый образ программы может размещаться в разных видах памяти: TCM или eFlash. При работе с платой EVU-BA-2.5 выбор памяти осуществляется в Makefile при помощи опции `MEM_REG_ROM=EFLASH`, которой могут присваиваться значения для TCM – `TCMA` или `TCMB`, а для eFlash - `EFLASH`, как это показано на рисунке 32.

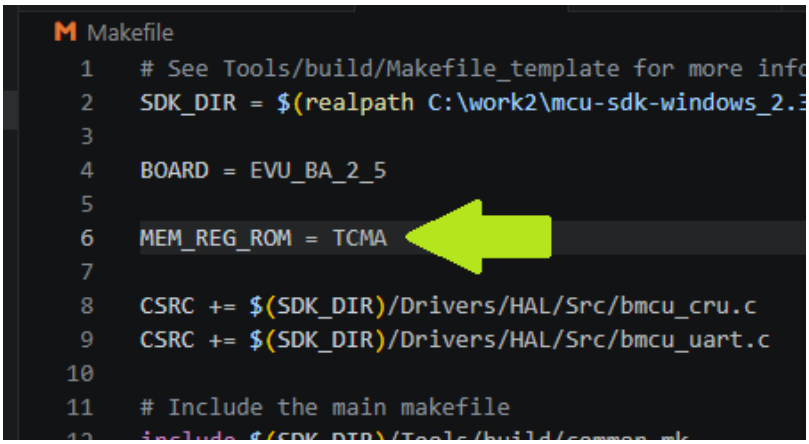

Рисунок 32 – Выбор памяти, в которую будет загружен образ программы

Для загрузки уже собранного образа программы в память МК убедиться, что плата МК подключена к ПК, а перемычки на плате выставлены в правильное положение, как это показано в разделе 3 на рисунках 3, 4. Далее перейти в подменю выбора операций, как это показано на рисунке 33.

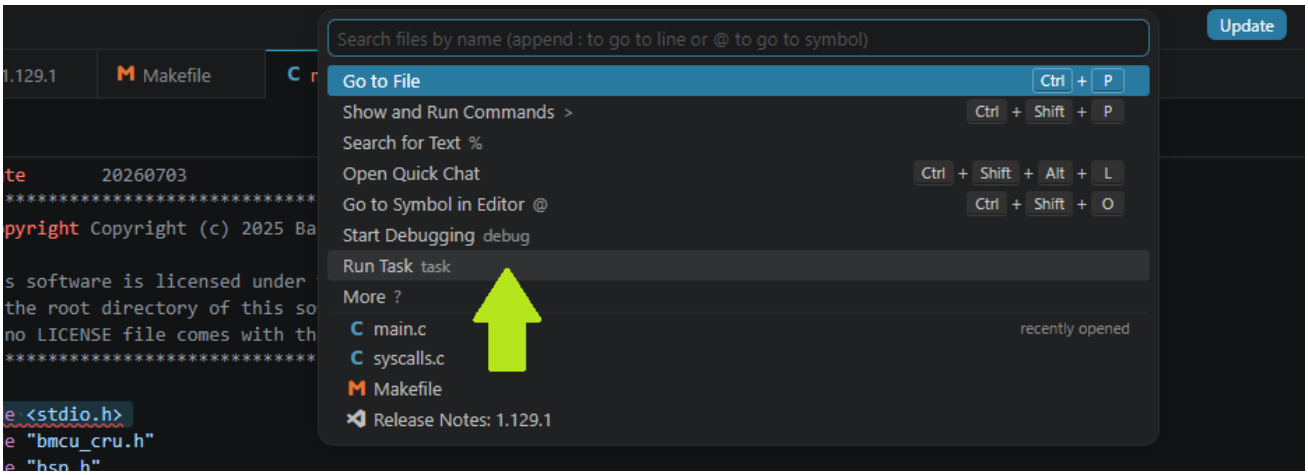

Рисунок 33 – Переход в подменю выбора операций

Перед загрузкой готового образа программы, память МК желательно очистить при помощи команды Стереть eFlash (Erase eFlash), как это показано на рисунке 34.

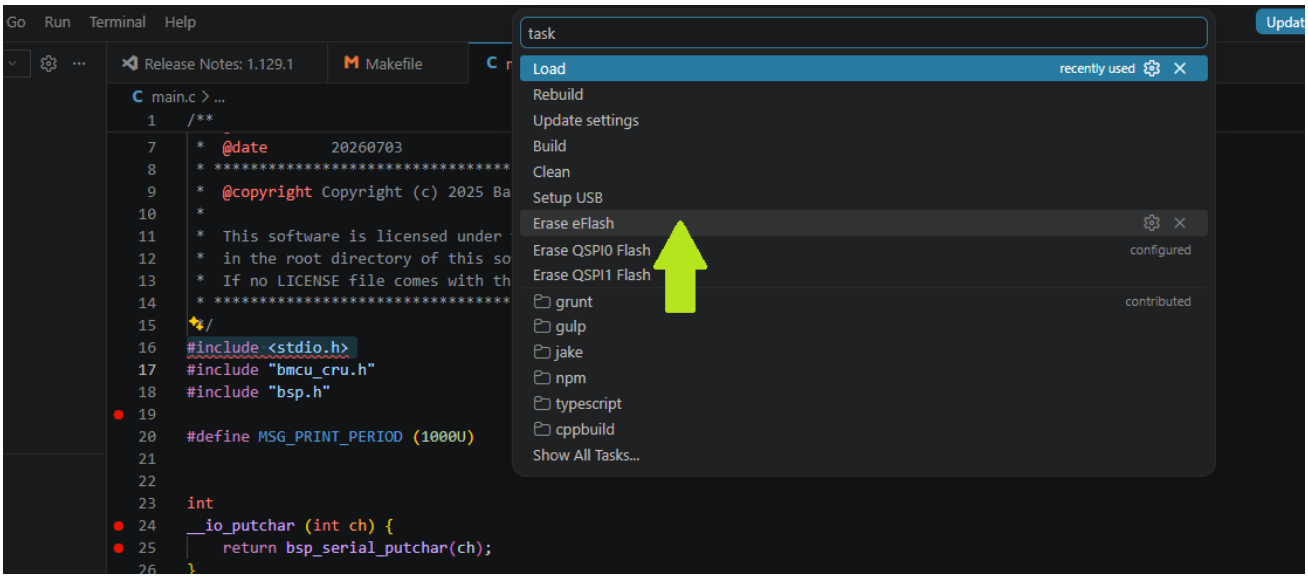

Рисунок 34 – Очистка памяти eFlash

Далее выбрать пункт Загрузить (Load), как это показано на рисунке 35.

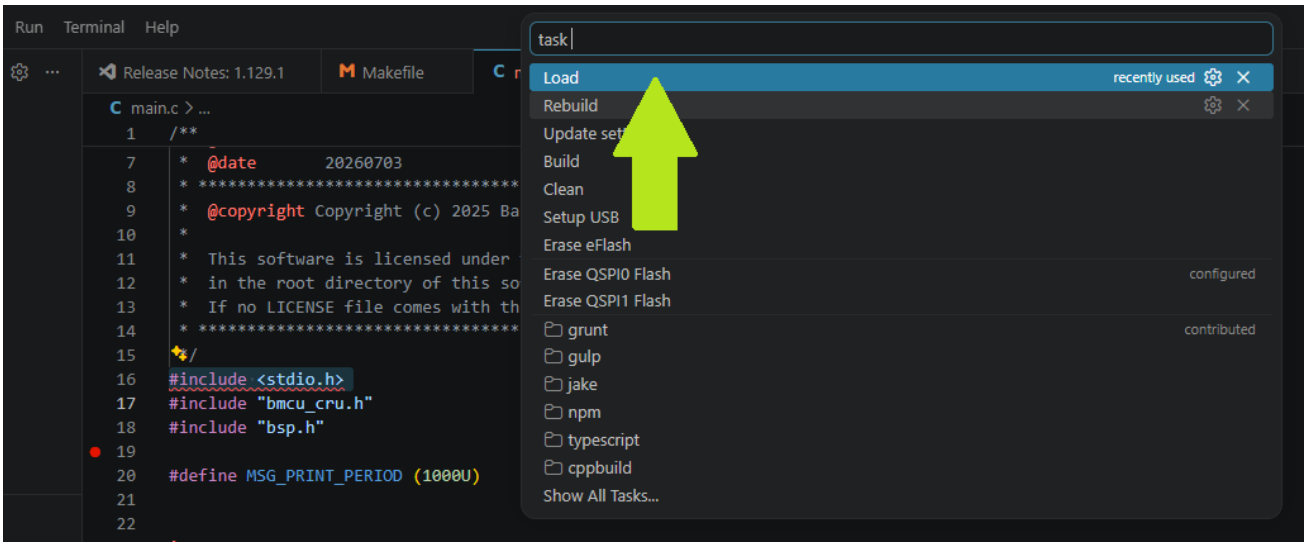

Рисунок 35 – Загрузка образа программы

При удачной загрузке образа программы в память в окне терминала будет выведено сообщение, аналогичное показанному на рисунке 36.

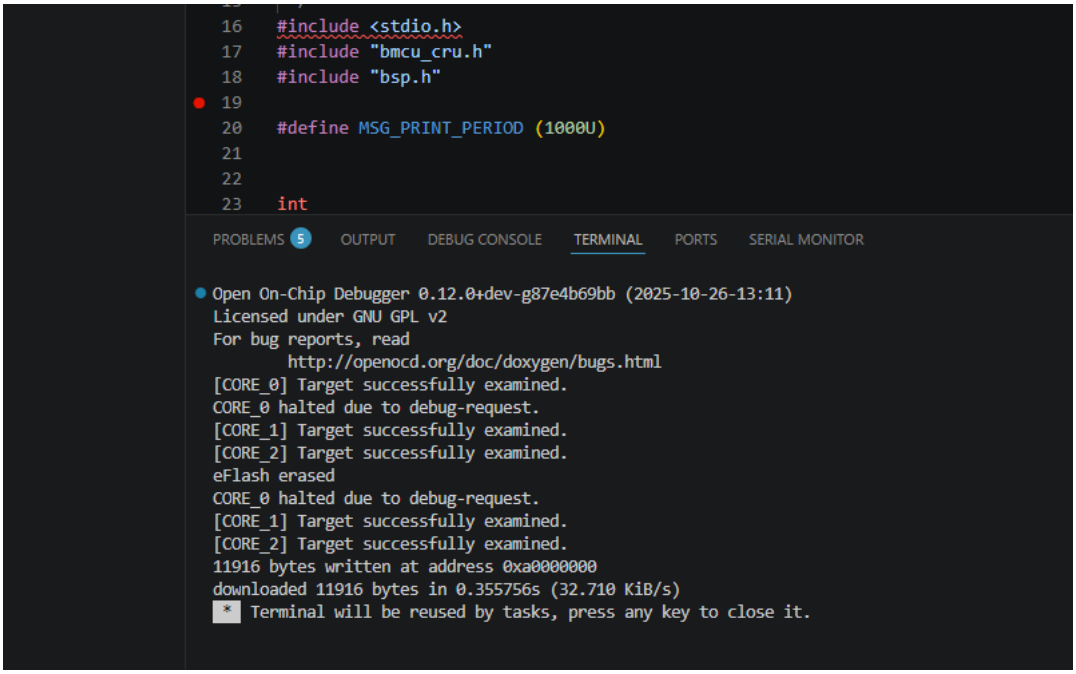

Рисунок 36 – Сообщение об удачной загрузке образа программы в память МК

Если загрузка образа в память не удалась, в окно терминала будут выведены сообщения об ошибках, как это показано на рисунке 37. В поле Проблемы (Problems) будет показано общее число ошибок.

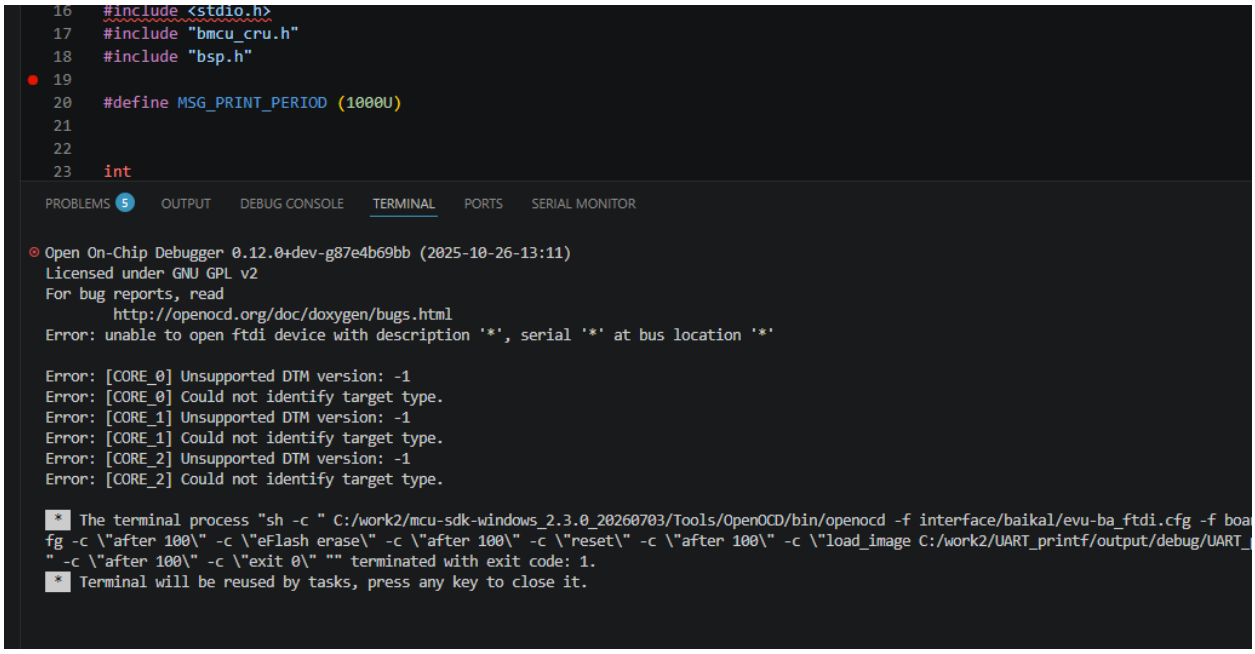

Рисунок 37 – Сообщения об ошибках при неудачной загрузке образа в память МК

Ошибки могут быть вызваны:

- плохим контактом в USB-кабеле, соединяющим МК и ПК или в перемычках на плате МК;
- неправильной установкой перемычек на плате МК;
- неправильным выбором разъема на плате МК;
- проблемами с драйвером USB в ОС ПК (требует переустановки).

9 Отладка проекта при помощи JTAG

Возможны два метода отладки приложений для МК:

- через GUI интерфейс Visual Studio Code;
- через терминал Visual Studio Code. Этот способы аналогичен описанному в документе «BE-U1000. Отладка по интерфейсу JTAG».

Встроенный в МК JTAG-отладчик исполняется на ядре Core 2 и предназначен для отладки программ, запускаемых на ядрах Core 0 и Core 1 с использованием интерфейса USB.

При отладке используется .elf-файл (образ с отладочными символами), который при удачной сборке создается в папке `output/debug` внутри рабочей папки вашего проекта.

9.1 Подготовка и загрузка образа программы

Для отладки программы необходимо присвоить параметру `BUILD_TYPE` в Makefile значение `debug`.

Загрузку образа программы в память МК нужно выполнить через JTAG, как это описано в Главе 8.

9.2 Установка точек останова

Точки останова (Breakpoints) можно добавить из оболочки Visual Studio Code нажав на нужную строку программы в левой части экрана проекта, левее номера строки, как это показано на рисунке 38. Напротив строки появится маркер точки останова, по умолчанию он выглядит как красная точка.

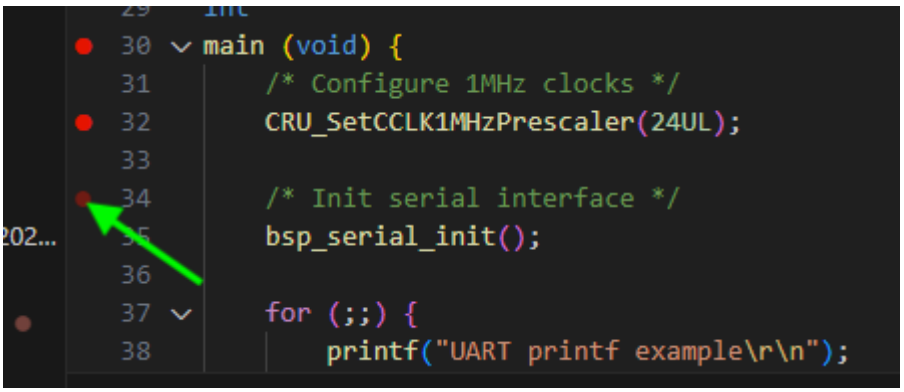

Рисунок 38 – Точки останова в интерфейсе Visual Studio Code

Точки останова отображаются в нижней части левого сервисного окна в режиме отладки, как это показано на рисунке 39.

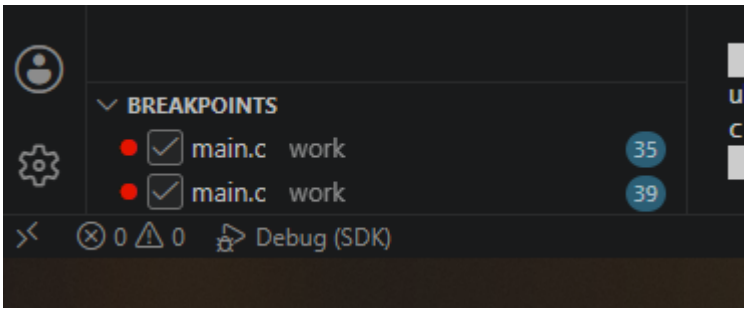

Рисунок 39 – Точки останова в сервисном окне

При использовании Visual Studio Code для отладки программ количество точек останова не ограничено, но при отладке через терминал, для программы размещенной во флеш-памяти, доступны только три аппаратных точки останова.

9.3 Запуск программы для отладки

Запуск ПО для отладки осуществляется из оболочки Visual Studio Code нажатием клавиши F5 или пиктограммы запуска и отладки справа над окном редактирования программы, как это показано на рисунке 40.

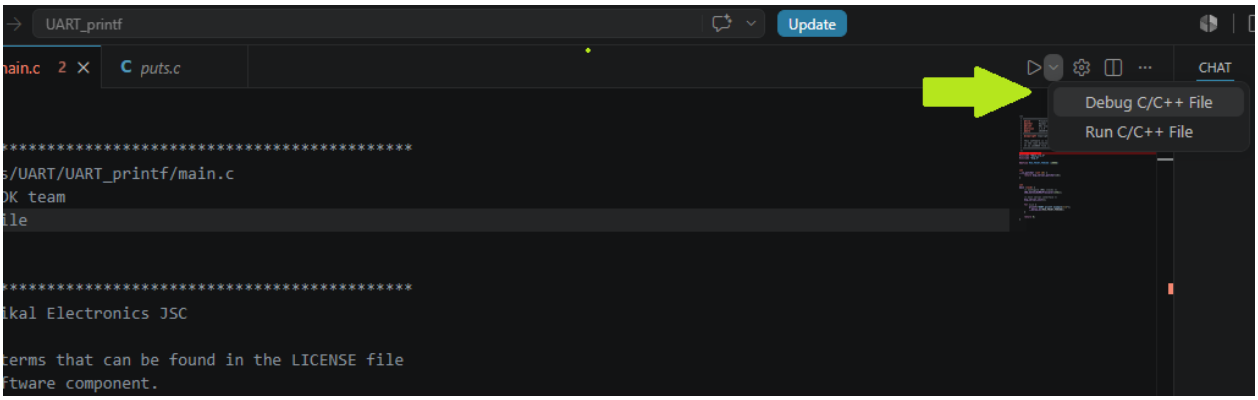

Рисунок 40 – Вызов отладки

После этого в выпадающем меню по центру предлагаются несколько вариантов отладки и запуска программы, как это показано на рисунке 41. В самом простом случае желательно выбрать отладку на ядре Core 0.

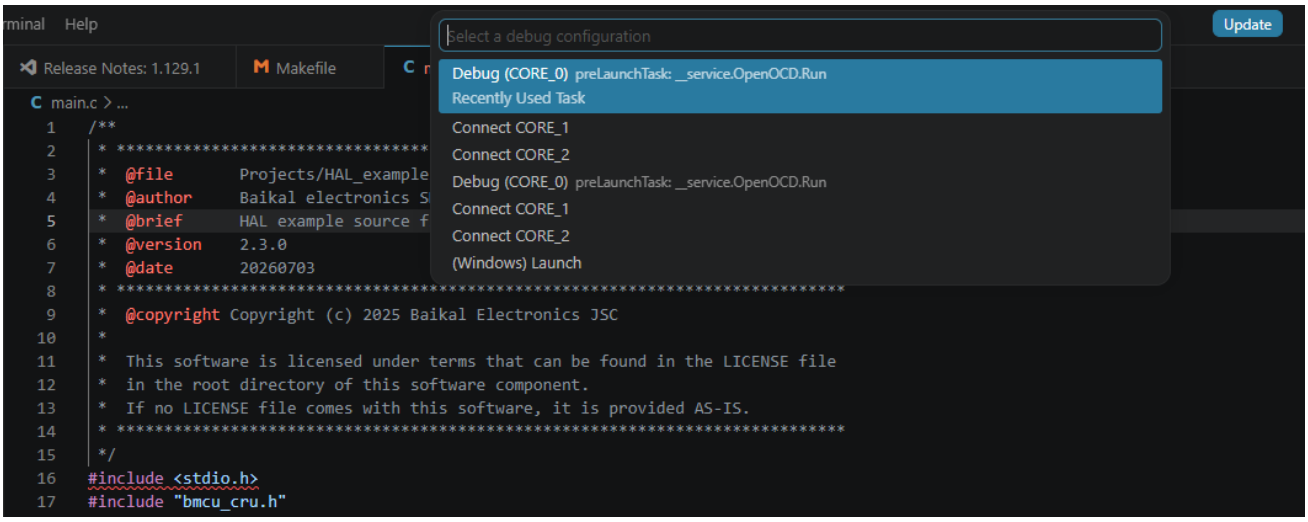

Рисунок 41 – Запуск процесса отладки

Другим способом вызова отладки является нажатие соответствующей пиктограммы в окне отладки слева от окна редактирования программ, как это показано на рисунке 42.

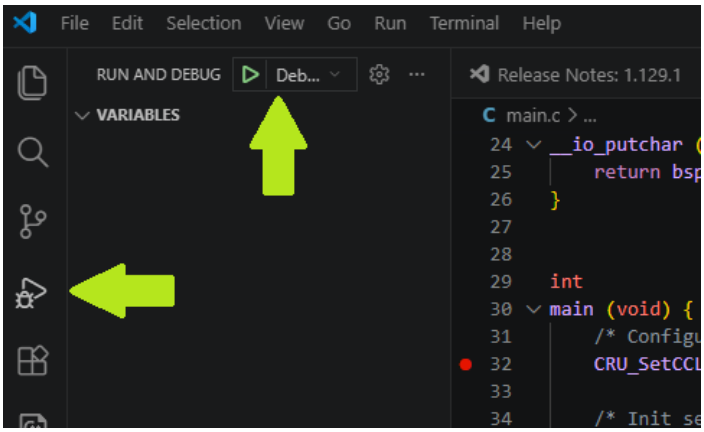

Рисунок 42 – Вызов из окна отладки

При нажатии на пиктограмму запуска будет так же предложено несколько вариантов запуска отладки, как это показано на рисунке 43.

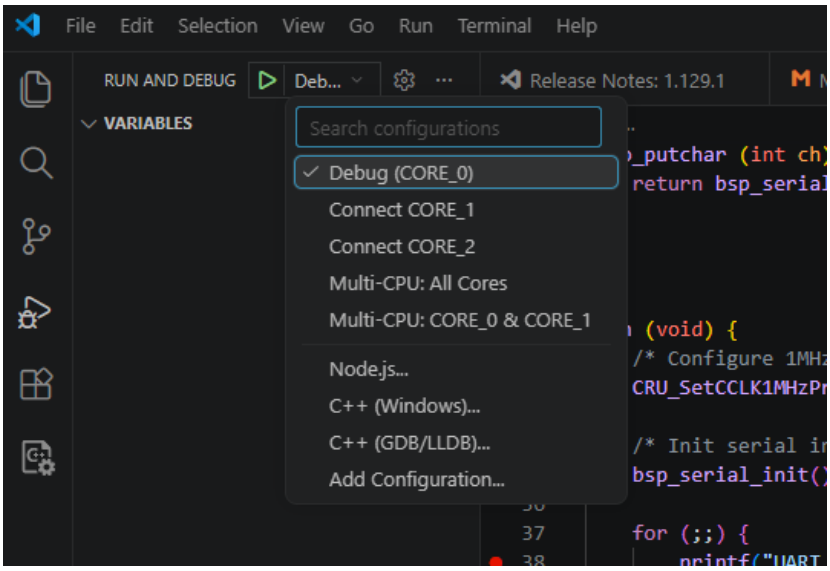

Рисунок 43 – Выбор вариантов отладки

После входа в режим отладки над полем редактирования программы появится панель управления, как это показано на рисунке 44, а текущая исполняемая операция программы будет подсвечена, как это показано на рисунке 45.

Рисунок 44 – Панель управления отладкой

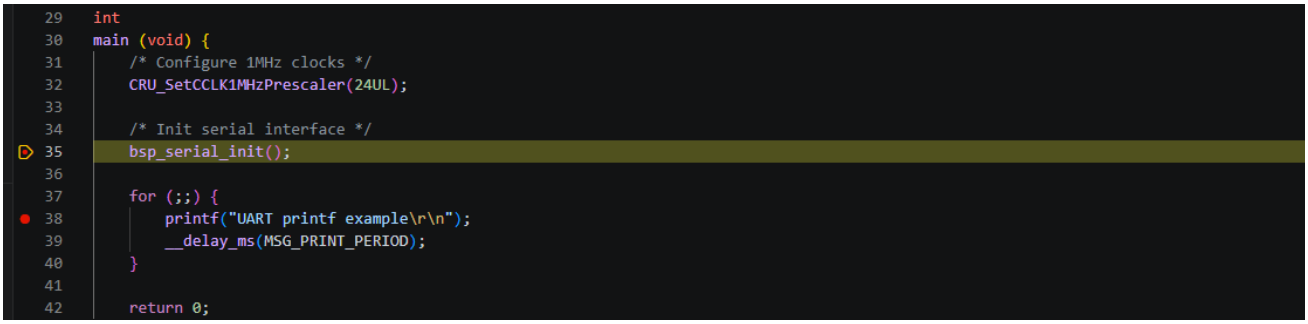

Рисунок 45 – Текущая выполняемая операция

9.4 Панель управления отладкой

Панель управления процессом отладки, показанная на рисунке 44, имеет следующий набор функций:

 Пауза / продолжение процесса исполнения программы

 Пошаговое исполнение программы без захода в функцию

 Пошаговое исполнение программы с заходом в функцию

Исполнение программы до возврата из текущей функции

Перезапуск процесса отладки с начала программы

Завершение процесса отладки программы

9.5 Просмотр значений переменных

В этой версии SDK просмотр значений переменных возможен только через память. Для этого поменять настройку в файле launch.json: зайти в файл launch.json и поменять `DDR - Set unaccessible by default` с `ON` на `OFF` для всех трех регионов памяти.

После этого установить адрес переменной в памяти после сборки. Этот адрес можно узнать по имени переменной в файле MAPDEBUG, расположенном в папке `../output`.

Далее адрес переменной добавить в поле `XPERIPHERALS` в окне Отладка (Debug), слева от текста программы, как это показано на рисунке 46. Таким образом, появится возможность наблюдать изменение значений переменных в процессе отладки программ.

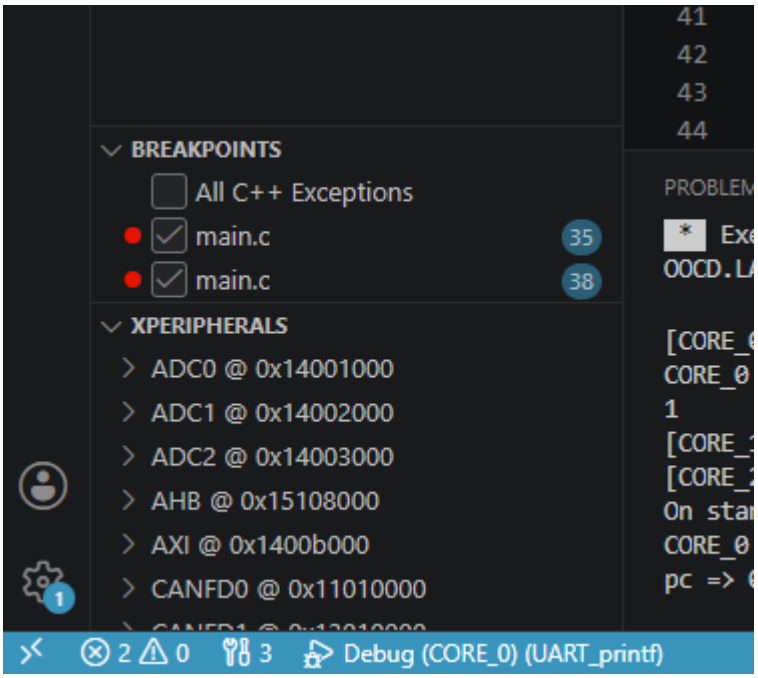

Рисунок 46 – Текущие значения переменных при отладке

10 Навигация по исходному коду больших проектов

10.1 Поиск в файлах проекта

Visual Studio Code представляет несколько инструментов для навигации по коду проекта.

В левом сервисном окне, показанном на рисунке 47, отображаются все файлы и папки, входящие в текущий проект.

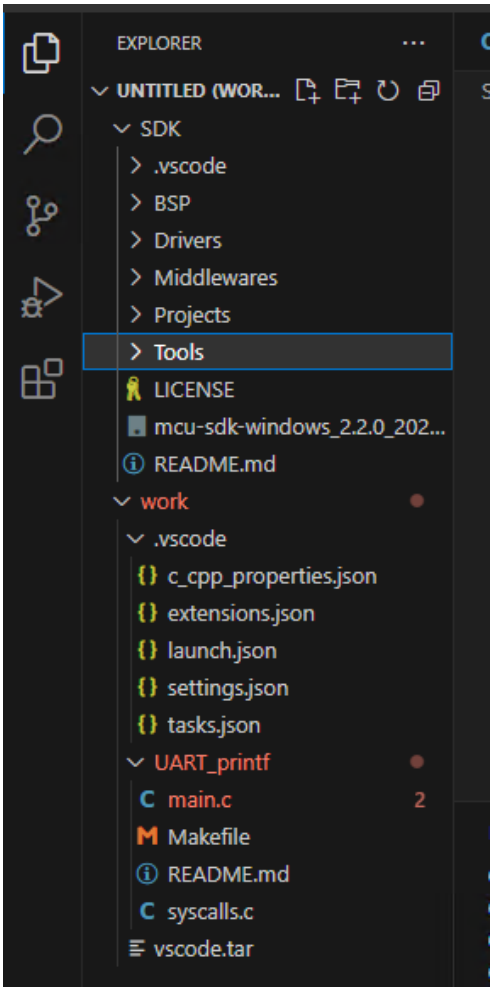

Рисунок 47 – Структура файлов и папок проекта

Пользователь может перемещаться по ним и проводить поиск. Поиск осуществляется в том числе и во вложенных папках всех уровней. Для поиска всех упоминаний той или иной функции, переменной или другого элемента программы навести на него курсор и нажать правую копку мыши, а далее выбрать пункт Найти все упоминания (Find all references) или воспользоваться горячими клавишами, как это показано на рисунке 48.

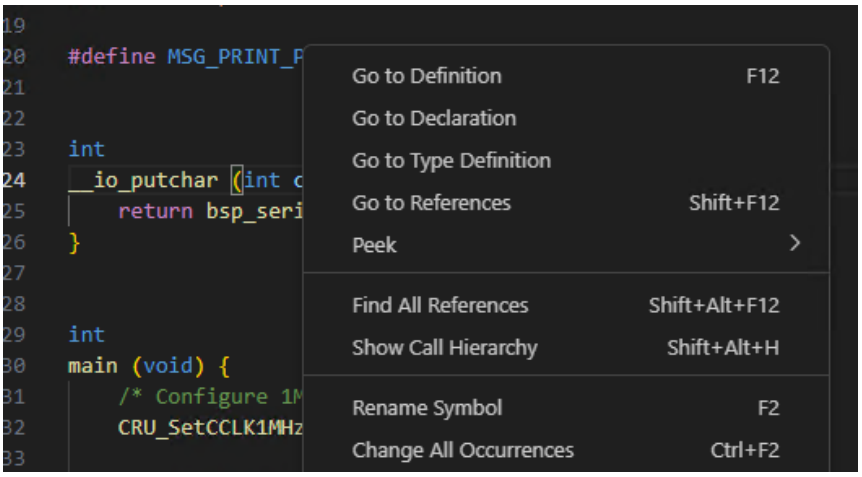

Рисунок 48 – Поиск всех включений функции или переменной

Откроем для примера SDK. Исполняемый код проекта размещается в файле `main.c`.

Найдем, где определен макрос `#define MSG_PRINT_PERIOD`. Для этого нажать <Ctrl>, навести на него курсор и кликнуть мышью. Если указанный макрос задействован в других файлах или компонентах проекта, то они будут выведены на экран в окне редактирования, как это показано на рисунке 49.

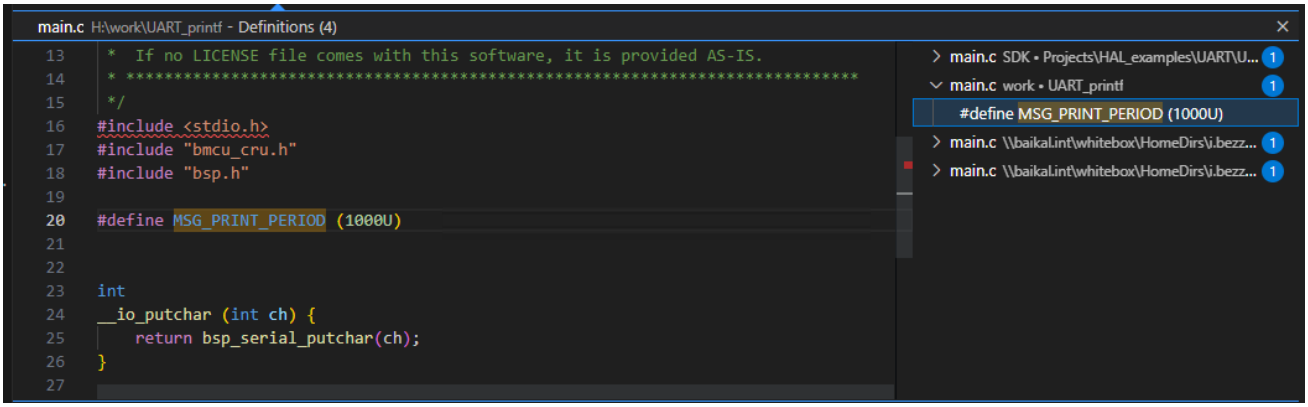

Рисунок 49 – Все включения макроса в окне редактирования

В правом углу окна показаны все включения макроса в проекте. Текст программы выбранного файла, содержащего включение, показан в окне редактирования.

Поиск по всем файлам проекта можно произвести с помощью комбинации клавиш <Ctrl>+<Shift>+<F>. На экран будет выведено окно со строкой поиска, как это показано на рисунке 50.

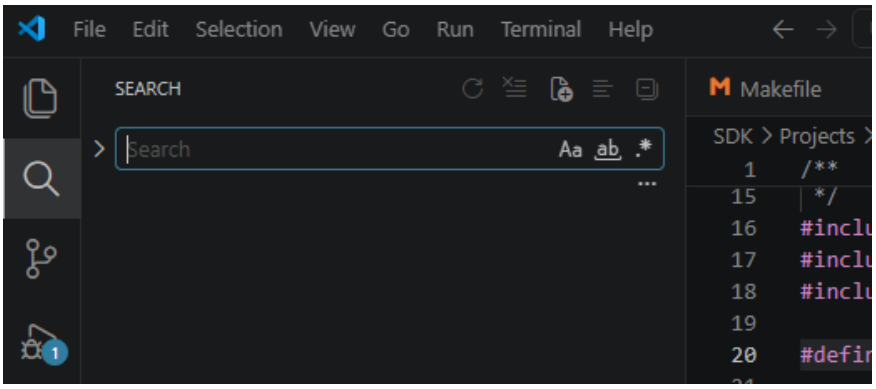

Рисунок 50 – Окно поиска

Visual Studio Code найдет все упоминания текста, введенного в окне поиска, во всех файлах проекта (в папке рабочего пространства и включенных в нее папках), как это показано на рисунке 51. Возможны варианты поиска по части или полному фрагменту слова, как это показано на рисунке 52.

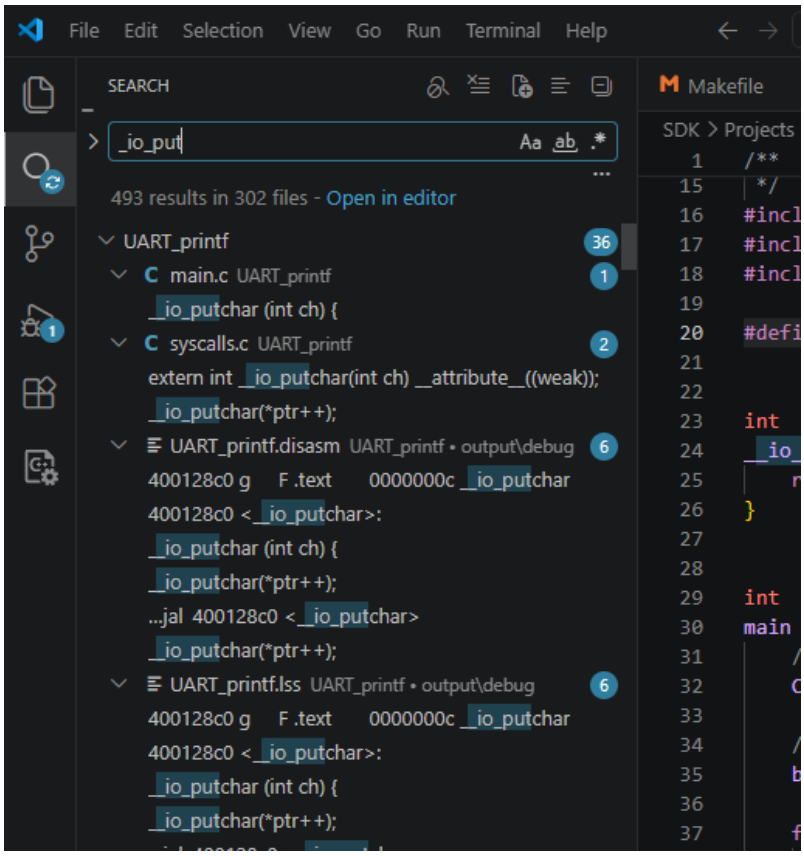

Рисунок 51 – Поиск всех включений функции

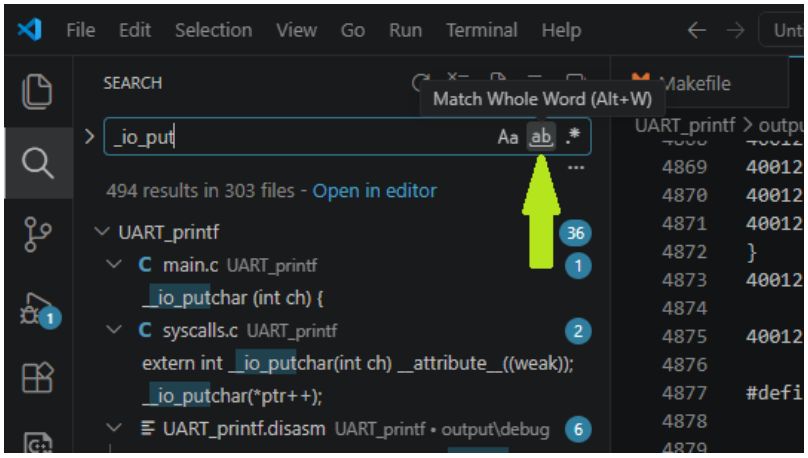

Рисунок 52 – Поиск по части или по полному слову

Кликнув мышью в упоминание текста, можно перейти в окне редактирования к файлу, где он упоминается, как это показано на рисунке 53.

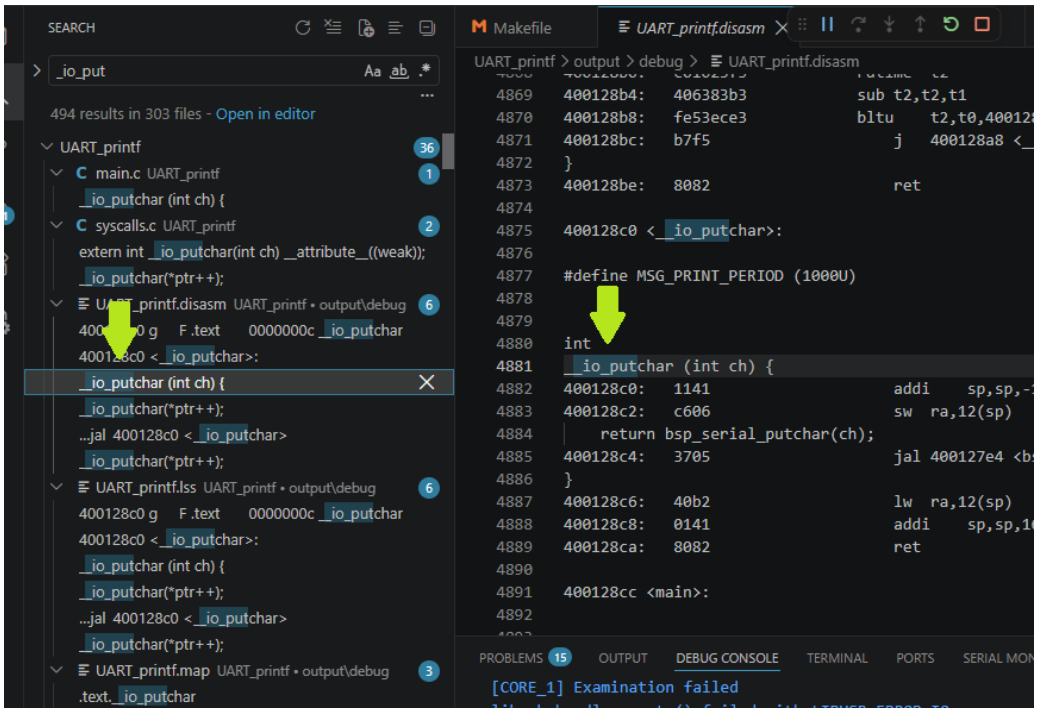

Рисунок 53 – Переход в файл, в котором упомянута искомая функция или переменная

Переключение между шаблонами поиска и перемещение по истории поиска осуществляется нажатиями клавиш «вверх» и «вниз», когда курсор помещен в поле поиска.

10.2 Переход в заголовочные файлы

При работе с библиотеками в Visual Studio Code необходимо следовать синтаксису языка программирования C/C++. Стандартные библиотеки, такие же как в C/C++, указываются в угловых скобочках, а библиотеки SDK и библиотеки, созданные пользователем, указываются в двойных кавычках, как это показано на рисунке 54.

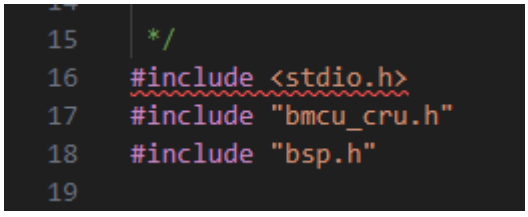

Рисунок 54 – Заголовочные файлы в программе

Если библиотека находится вне файла проекта, то необходимо указать полный путь к файлу .h.

Нажав <Ctrl> и наведя курсор на название библиотеки, можно перейти в нее.

Для любой функции, определенной пользователем ранее или в другом файле, как и для любой функции SDK через нажатие <Ctrl> и клик мышью на название, можно найти все упоминания функции в проекте и обратиться к любому из них, аналогично тому, как это было показано ранее, в подразделе 10.1.

Поиск только по текущему файлу возможен комбинацией клавиш <Ctrl>+<F>.

10.3 Переход к определению символических имен

Под символическим именем директивой `#define` может быть определен фрагмент программного кода, который необходимо повторять в различных частях проекта. По присвоенному такому коду имени его можно использовать многократно, не копируя из других мест вручную. Пример определения показан на рисунке 55.

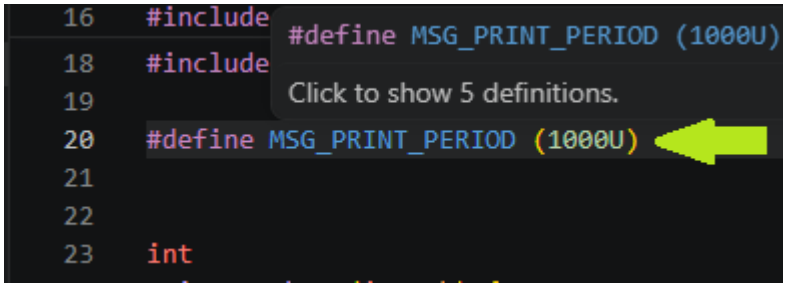

Рисунок 55 – Определение символического имени

Программный код может быть определен не в том же файле проекта, где осуществляется его вызов – Visual Studio Code найдет его самостоятельно среди файлов проекта. Наведя на символическое имя курсор мыши и нажав <Ctrl> можно найти все его упоминания в файлах рабочего пространства.

10.4 Переход к декларациям переменных и функций

Если функция упоминается в одном из файлов проекта, то по клику на ее название правой кнопкой мыши можно найти все ее упоминания в проекте и открыть их в отдельных окнах редактора или перейти к ее декларации нажав Перейти к декларации (Go to declaration) во всплывающем меню, как это показано на рисунке 56.

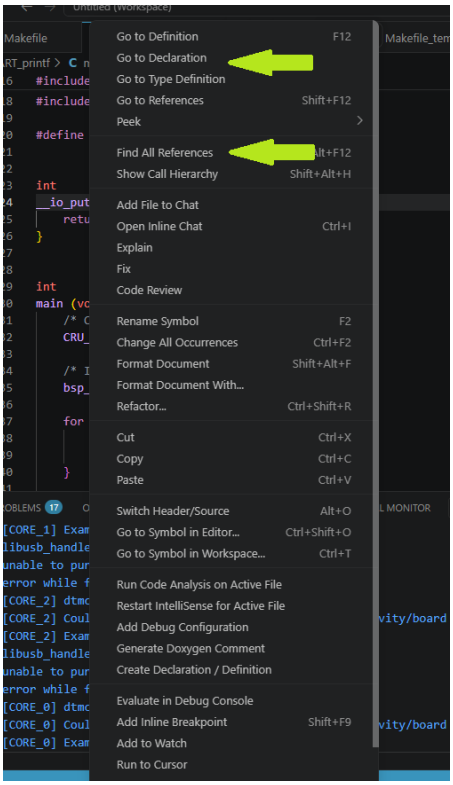

Рисунок 56 – Переход к декларации функции

10.5 Переход к определению функций

Для перехода к определению функции в одном из файлов проекта надо нажать на ее название правой кнопкой мыши и выбрать Перейти к определению (Go to definition) во всплывающем меню, как это показано на рисунке 57.

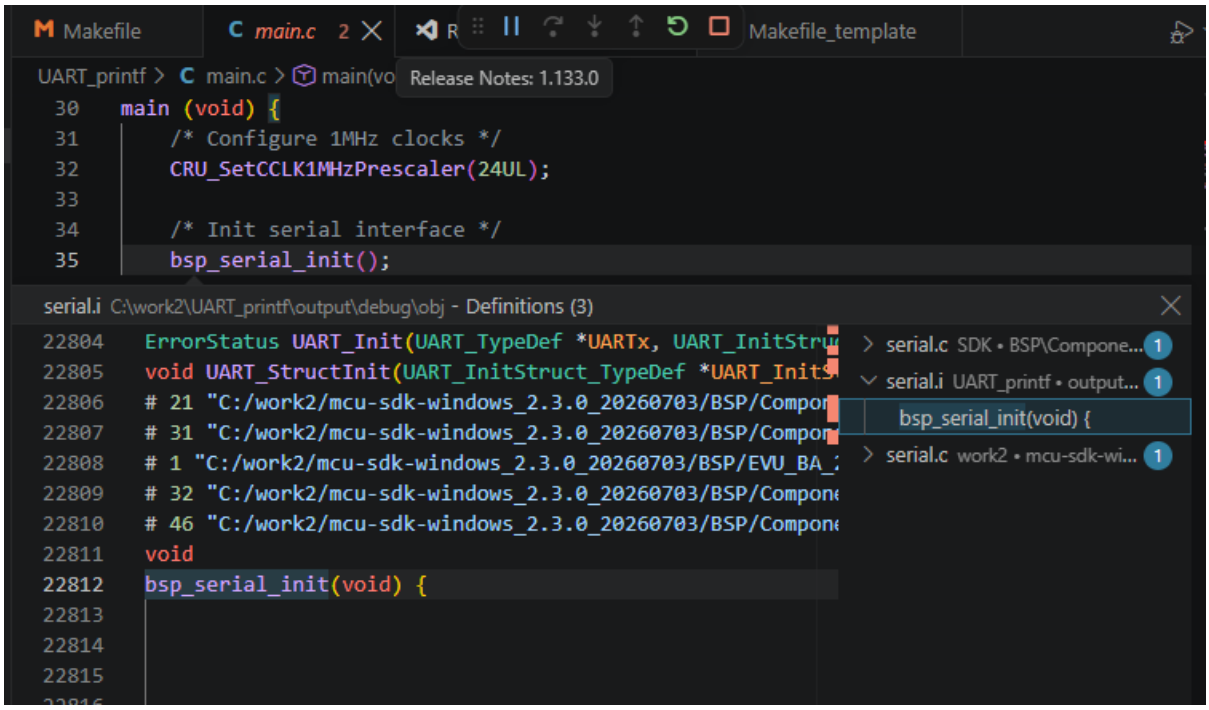

Рисунок 57 – Переход в окно, где функция была реализована

История изменений

Версия	Дата	Описание
1	03.09.2026	Начальная версия

Контактные данные

Официальный сайт
www.baikalelectronics.ru

Главный офис
www.baikalelectronics.ru/contacts

Электронная почта
info@baikalelectronics.ru

Телефон
+7 495 221-39-47