



User Guide

Getting Started with VS Code, GCC, and J-Link Using CMSIS Toolbox

Version 1.4

Table of Contents

| | |
|--|----|
| 1. Tools Installation..... | 3 |
| 2. VS Code and CMSIS Toolbox configuration..... | 8 |
| 3. Setting up your Alif Ensemble Development Kit..... | 11 |
| 4. Opening a CMSIS Toolbox project with Visual Studio Code | 12 |
| 5. Adding CMSIS Components to the application..... | 16 |
| Document History | 17 |

Introduction

The User Guide is designed to show how to set up a VS Code environment with the GNU tools including GCC and GDB plus CMSIS Toolbox to do design combined with Segger J-Link for debugging when targeting an Alif SoC on the Ensemble DevKit Gen 2.

If you have been doing work using VS Code + GCC for the Alif Beta Ensemble DevKit, you should create a separate instance of VS Code for the new DevKit since several parts of the configuration are different.

This user guide shows how to set up a VS Code environment with the GNU tools and J-Link debugger under Windows. The same overall process can be used to create a Linux environment by installing Linux versions of the different tools listed and updating the paths appropriately in the different settings files.

1. Tools Installation

- 1.1. Begin by downloading Arm GNU Toolchain from <https://developer.arm.com/downloads/-/arm-gnu-toolchain-downloads>. Select the executable from Windows -> AArch32 bare-metal target category. Using the latest stable version is always recommended, however, this document has been written and tested against version 12.2.Rel1.

What's new in 12.2.Rel1

This release is based on GCC 12.2

In this release

Windows (mingw-w64-i686) hosted cross toolchains

AArch32 bare-metal target (arm-none-eabi)

- o [arm-gnu-toolchain-12.2.rel1-mingw-w64-i686-arm-none-eabi.zip](#)
- o [arm-gnu-toolchain-12.2.rel1-mingw-w64-i686-arm-none-eabi.zip.asc](#)
- o [arm-gnu-toolchain-12.2.rel1-mingw-w64-i686-arm-none-eabi.zip.sha256asc](#)
- o [arm-gnu-toolchain-12.2.rel1-mingw-w64-i686-arm-none-eabi.exe](#)
- o [arm-gnu-toolchain-12.2.rel1-mingw-w64-i686-arm-none-eabi.exe.asc](#)
- o [arm-gnu-toolchain-12.2.rel1-mingw-w64-i686-arm-none-eabi.exe.sha256asc](#)

- 1.2. Run the installer executable and **make note of the location the compiler is installed into**. After the installation is completed, leave “add path to environment variable” **unchecked**.
- 1.3. Download the CMake version 3.26.0 (or newer) from <https://cmake.org/download/>. Select the executable from Platform -> Windows x64 Installer. For convenience, the installer has also been bundled with these instructions.

Latest Release (3.26.0)

The release was packaged with CPack which is included as part of the release. The .sh files are self extracting gzipped tar files. To install a .sh file, run it with /bin/sh and follow the directions. The OS-machine.tar.gz files are gzipped tar files of the install tree. The OS-machine.tar.Z files are compressed tar files of the install tree. The tar file distributions can be untared in any directory. They are prefixed by the version of CMake. For example, the linux-x86_64 tar file is all under the directory cmake-linux-x86_64. This prefix can be removed as long as the share, bin, man and doc directories are moved relative to each other. To build the source distributions, unpack them with zip or tar and follow the instructions in README.rst at the top of the source tree. See also the CMake 3.26 Release Notes.

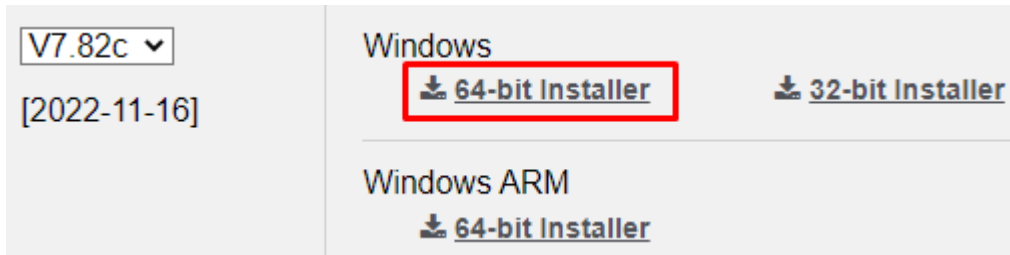
Source distributions:

| Platform | Files |
|---------------------------------------|-------------------------------------|
| Unix/Linux Source (has \n line feeds) | cmake-3.26.0.tar.gz |
| Windows Source (has \r\n line feeds) | cmake-3.26.0.zip |

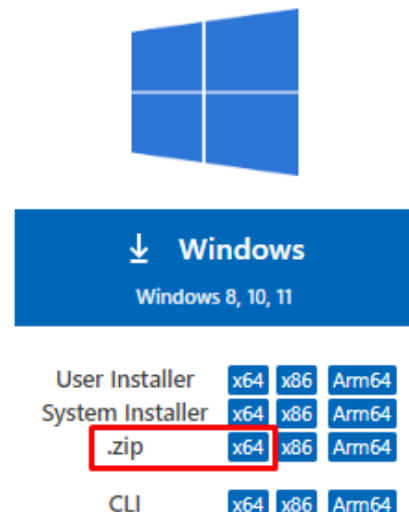
Binary distributions:

| Platform | Files |
|-------------------------|---|
| Windows x64 Installer: | cmake-3.26.0-windows-x86_64.msi |
| Windows x64 ZIP | cmake-3.26.0-windows-x86_64.zip |
| Windows i386 Installer: | cmake-3.26.0-windows-i386.msi |
| Windows i386 ZIP | cmake-3.26.0-windows-i386.zip |

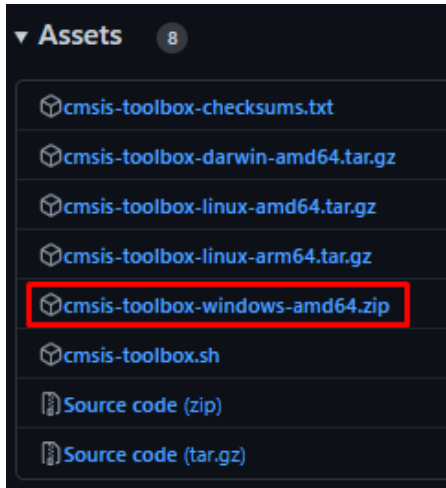
- 1.4. Run the installer executable and **inside the installation wizard, select “Add CMake to system PATH for all users”**. This step is essential to make CMake visible from VS Code.
- 1.5. Download latest stable version of the J-Link Software (64-bit installer) from <https://www.segger.com/downloads/jlink/>. This document has been tested with J-Link version 7.82c.



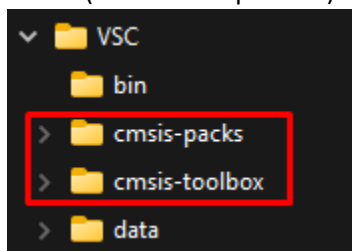
- 1.6. Run the J-Link installer and once again **make note of the installation directory** specified.
- 1.7. Go to the Visual Studio Code website (<https://code.visualstudio.com/download>) and download .zip for x64 Windows environment. Using .zip package will prevent VS Code from keeping its settings on per-user or per-system basis, allowing multiple installations to coexist without causing clashes in configuration and extensions:
- 1.8. Unpack the VS Code .zip file to the installation directory of your choice. It is recommended to keep the path short for convenience.



- 1.9. Inside the VS Code directory (Code.exe is present) create a folder named “data”. If present, VS Code will use this folder to store all the configuration and extensions data, keeping your VS Code installation fully self-contained and portable. This will allow you to have multiple installations present on your system, if needed, without any clashes in the configuration.
- 1.10. Obtain version 1.7.0 of CMSIS Toolbox from Open CMSIS Pack Github: <https://github.com/Open-CMSIS-Pack/cmsis-toolbox/releases/tag/1.7.0>. This document has been tested with version 1.7.0 and later versions can produce errors during the build process. Also, be sure to download the “amd64” version and not “arm64”.

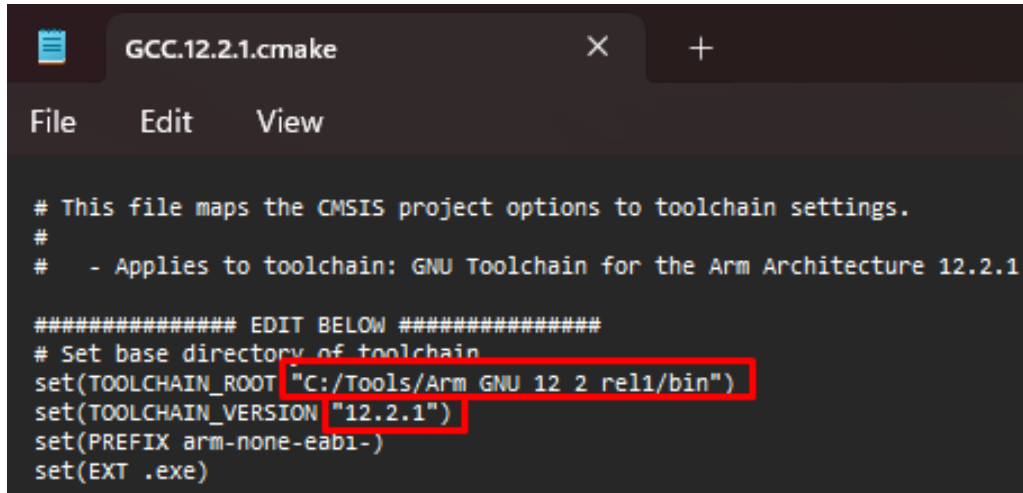


- 1.11. It is recommended that the CMSIS tools, packs and configurations are kept together with the VS Code installation to simplify configuration and improve portability. Inside the VS Code folder (Code.exe is present) create folders “cmsis-toolbox” and “cmsis-packs”:



- 1.12. Extract CMSIS Toolbox archive downloaded in step 1.8 into the “cmsis-toolbox” directory. Folders “bin”, “doc” and “etc” should be present immediately inside “cmsis-toolbox”.
- 1.13. Navigate inside the cmsis-toolbox/etc directory. This folder contains CMake templates for building with various compilers. Duplicate GCC.10.3.1.cmake file and rename it to reflect the version of the GCC compiler used (e.g.: GCC.12.2.1.cmake).

- 1.14. Open the renamed duplicate in any text editor and modify path on line 7 (after TOOLCHAIN_ROOT) with the path specified in step 1.2, followed by “/bin”. **Only forward slashes are supported in the path.** Immediately below, change TOOLCHAIN_VERSION to the number matching the GCC version installed earlier. For example:

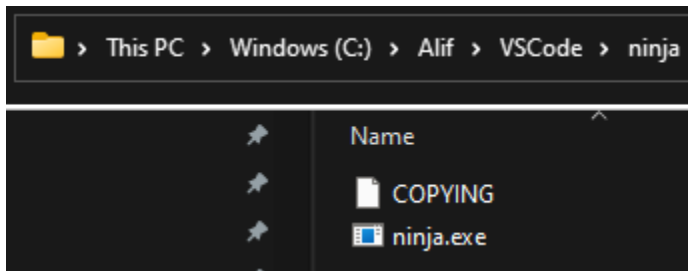


```
GCC.12.2.1.cmake
File Edit View

# This file maps the CMSIS project options to toolchain settings.
#
# - Applies to toolchain: GNU Toolchain for the Arm Architecture 12.2.1

##### EDIT BELOW #####
# Set base directory of toolchain
set(TOOLCHAIN_ROOT "C:/Tools/Arm GNU 12 2 re11/bin")
set(TOOLCHAIN_VERSION "12.2.1")
set(PREFIX arm-none-eabi-)
set(EXT .exe)
```

- 1.15. Save and close the file.
- 1.16. Download the Ninja executable from <https://github.com/ninja-build/ninja/releases>. Create folder named “ninja: inside the VS Code installation directory and extract the Ninja executable into it:



- 1.17. Next, you will need to download **version 1.0.0 or later of the Alif Security Toolkit**. This can be found on the Software & Tools page of the Alif website under Support [[link here](#)]. You will have to log into the website with your Alif credentials to access this archive.
- 1.18. Inside your VS Code installation directory create an “alif-tools” folder and extract the Security Toolkit into it.

1.19. Download Git for Windows from <https://git-scm.com/download/win>:



Download for Windows

[Click here to download](#) the latest (**2.40.0**) **64-bit** version of **Git for Windows**. This is the most recent **maintained build**. It was released **22 days ago**, on 2023-03-14.

Other Git for Windows downloads

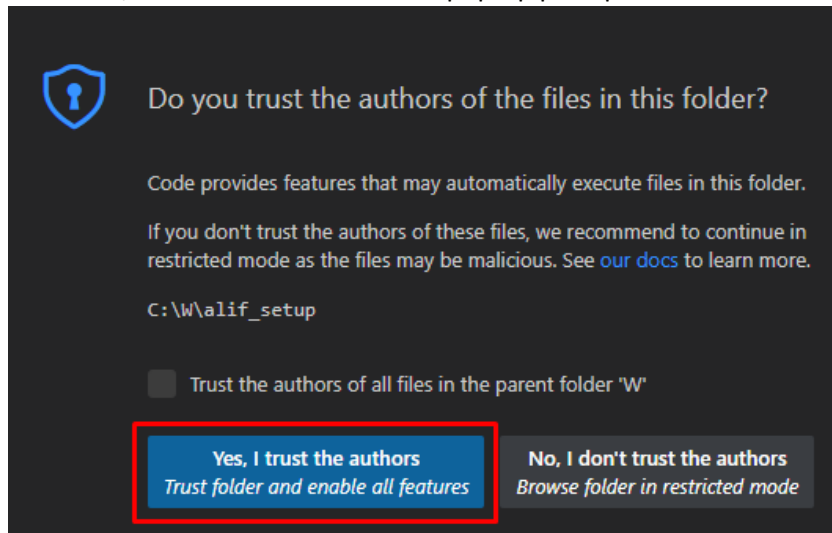
Standalone Installer

[32-bit Git for Windows Setup.](#)

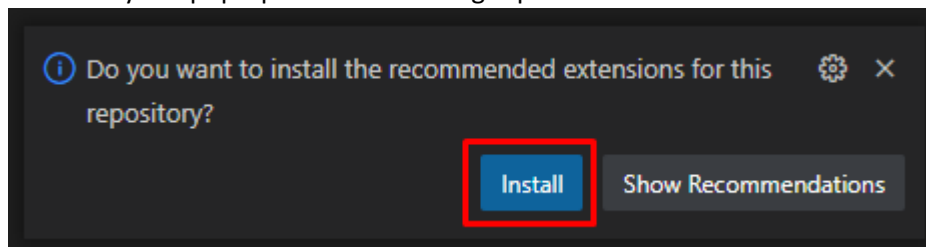
[64-bit Git for Windows Setup.](#)

2. VS Code and CMSIS Toolbox configuration

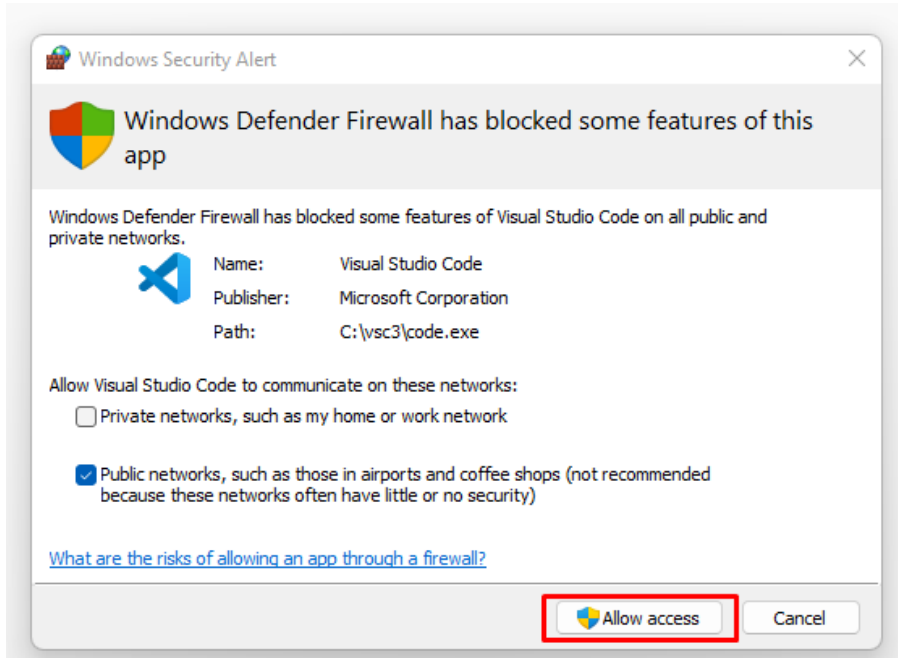
- 2.1. Clone the Alif VS Code template for Gen 2 silicon (Github repo name “[alif_vscode-template](#)”). This can be found on the Alif Semiconductor Github page [\[link here\]](#). Clone this template to a project folder of your choosing on your hard drive.
- 2.2. After cloning the template file to your project folder, open up a terminal window in that template folder and execute the command “`git submodule update --init`”. This will load the board library sub-module.
- 2.3. Run VS Code for the first time by launching Code.exe. Since this installation uses its own local configuration, the first-time setup wizard will be shown even if there are other VS Code instances on your machine. Adjust the personalization settings and dismiss the wizard window.
- 2.4. Click “Open Folder” on the Get Started page and select the template folder extracted in step 2.1 (ensure that you can see several files directly inside of it).
- 2.5. Click “Yes, I trust the authors” in the pop-up prompt.



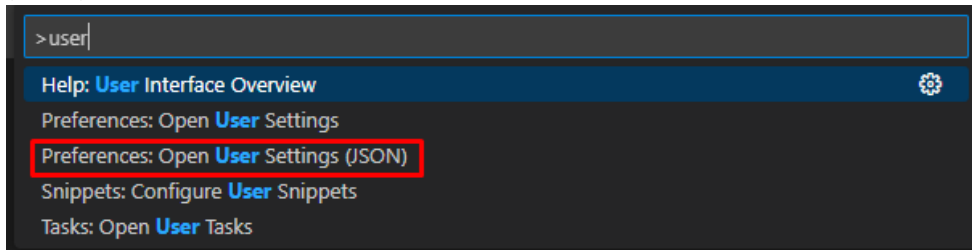
- 2.6. Project includes file with a list of recommended (required for our purposes) extensions, as notified by the pop-up in the bottom right part of the VS Code window. Click “Install”:



- 2.7. While extensions are being installed, Windows Defender Firewall might raise warnings about VS Code. Press “Allow access” as network access is required for the connection to the GDB server:



- 2.8. Before the project content can be built or debugged, additional configuration is required. Press F1 to bring focus to the search bar and type “user”. Select “Preferences: Open User Settings (JSON)”:



- 2.9. This file controls settings for the VS Code installation, regardless of which project or folder is selected. As such, these settings only need to be configured once. Add a comma behind the last entry and add the following lines before the closing brace:

```
{
  "terminal.integrated.env.windows": {
    "CMSIS_COMPILER_ROOT": "${execPath}/../cmsis-toolbox/etc",
    "CMSIS_PACK_ROOT": "${execPath}/../cmsis-packs",
    "PATH": "${execPath}/../cmsis-toolbox/bin;${execPath}/../ninja;${env:PATH}",
    "SETTOOLS_ROOT": "${execPath}/../alif-tools"
  },
  "cortex-debug.armToolchainPath": "C:/Tools/Arm_GNU_12_2_rel1/bin",
  "cortex-debug.JLinkGDBServerPath": "C:/Tools/JLink_V7_82c/JLinkGDBServerCL.exe"
}
```



- 2.10. The lines above are responsible for setting up the environment variables for the command line which will be used to run CMSIS Toolbox utilities as well as setting up GDB server and client paths. CMSIS_COMPILER_ROOT, CMSIS_PACK_ROOT, PATH, and SETOOLS_PATH should all be specified as above unless different locations were used in step 1.11. Lines starting with "cortex-debug" should point to the JLinkGDBServerCL.exe file inside the J-Link installation folder (step 1.6) and to the bin folder inside the directory with the Arm GNU Compiler (step 1.2).
- 2.11. Press F1 and select "Run Task". From the list of tasks, select "First-time pack installation". VS Code will exercise cpackget to obtain the necessary Alif and Arm pack files. Once completed, you should see "Pack installation has been completed" message in the console.
- 2.12. Your VS Code installation and CMSIS Toolbox are now fully configured to build and debug GCC-based projects for Alif devices. tasks.json file provided inside the .vscode folder in the project defines few commands to automate clean & build process (accessible under F1-> Tasks: Run Task). Applications are configured using csolution.yaml and cproject.yaml files and are generated and built with Ctrl + Shift + B combination.
- 2.13. Press Ctrl + Shift + B combination to generate the content and build the project. At the end of the successful build, the console window will show the following:

```

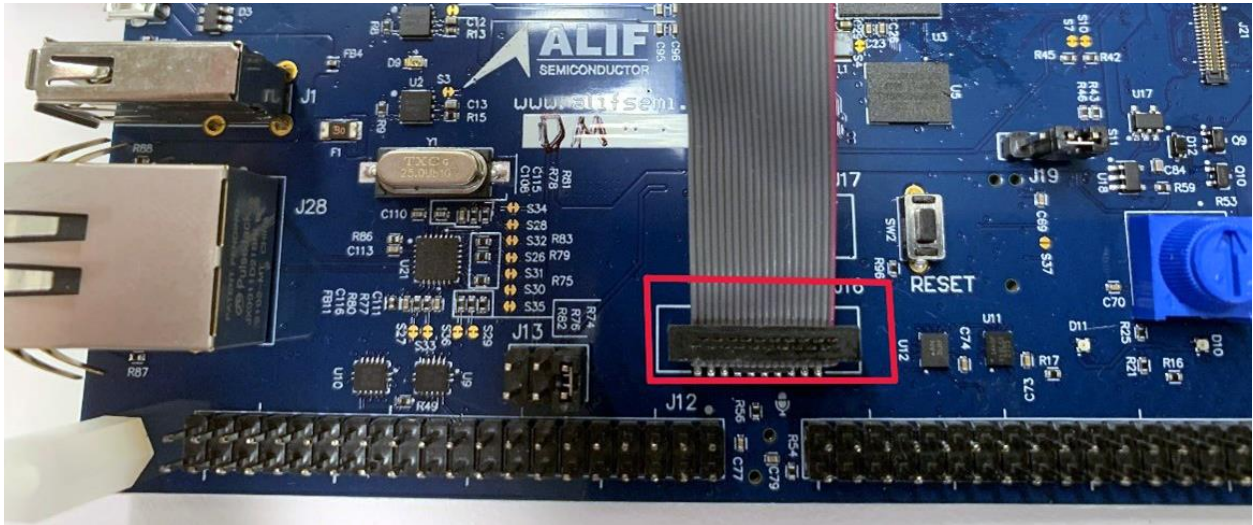
Memory region      Used Size  Region Size  %age Used
ITCM:              6648 B    256 KB      2.54%
DTCM:             256 KB    256 KB     100.00%
SRAM0:              0 GB      4 MB        0.00%
SRAM1:              0 GB     2560 KB     0.00%
MRAM:             10604 B   5628 KB     0.18%
info cbuild: build finished successfully!
* Terminal will be reused by tasks, press any key to close it.

```

If your project builds correctly and with no errors, you have fully completed your installation and you're ready to develop for Alif Ensemble.

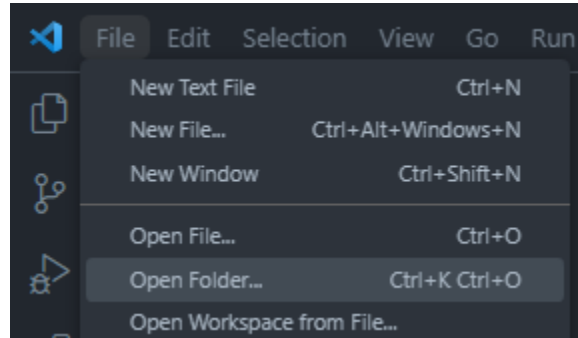
3. Setting up your Alif Ensemble Development Kit Gen 2

1. To power your DevKit, connect a USB cable from your computer to the “PRG USB” micro-USB socket on the board.
2. Determine which COM port is connected to the SEUART by following the steps at the beginning of the Ensemble DevKit User Guide. This can be found on the Customer Files page of the Alif website [\[link here\]](#). You will have to log into the website with your Alif credentials to access this archive as well as other Rev-B silicon and software documentation. Make of note of which COM port is connected to the SEUART.
3. Connect J-Link debugger to the 19-pin connector in the top center of the CPU board, as shown below. JTAG debugger connections are available in the middle of the board near the lower edge. The picture below shows a Segger J-Link with 19-pin Cortex-M adapter connected to J16 on the DevKit which is the 19-pin JTAG-0 connector. Ensure your J-Link is connected to the USB port on your computer.

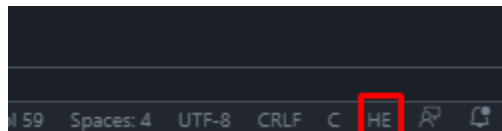


4. Opening a CMSIS Toolbox project with Visual Studio Code

1. If you closed VS Code after the previous section, open VS Code and navigate to File -> Open Folder and select the alif_vscode-template-DEV folder that you downloaded in section 2.1 (you should see several directories and files inside of it).



2. This is a basic “starting point” project for developing on Alif Ensemble using CMSIS Toolbox. You should see following files and directories:
 - csolution.yaml – describes the whole “solution” (consisting of one or more projects) and specifies toolchain settings such as compiler and linker properties and build targets.
 - cproject.yaml – describes a project – specifies the content to include in the build, e.g. user sources, libraries, include paths and components from installed CMSIS packs.
 - app/main.c – a placeholder for user sources. Additional directories and files can be specified inside cproject file.
 - .vscode folder – provides settings specific to Visual Studio Code for improved integration. It specifies debug properties, CMSIS Toolbox commands for setup, generating and building content, additional scripting for Alif Security Toolbox and a list of recommended extensions to make sure your VS Code installation has everything needed to develop for Alif Ensemble.
 - .alif folder – configuration files used by Alif Security Toolbox.
3. Open main.c inside “app” folder. On the bottom right you will see the active C/C++ Configuration. This correlates to the Cortex M55 core the project is targeting on the Alif Ensemble device. To build and debug the project on High Efficiency core, click “HP” shown on the bottom right, then from the drop-down at the top select “HE”. This setting is only visible when a C file is open.



4. Since the project comes with a set of predefined tasks (defined in .vscode/tasks.json file) to help you with operating CMSIS Toolbox, you can build the project by executing the build task in one of the 3 possible ways:
 - Press Ctrl+Shift+B key combination.
 - Pressing F1 and then selecting “Tasks: Run Build Task”.
 - Pressing F1 and then selecting “Tasks: Run Task” followed by “Generate and build with csolution + cbuild”.

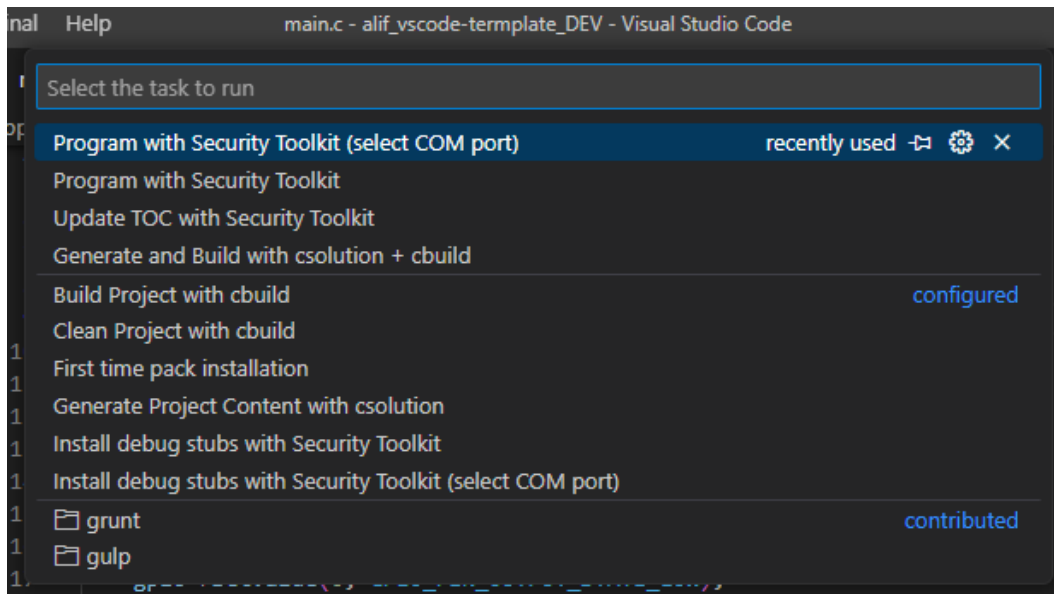
- VS Code will use CMSIS Toolbox to configure and build the project using definitions provided in the aforementioned .yaml files. At the end of successful build, you should see the following message in the in the Terminal window:

```

Memory region      Used Size  Region Size  %age Used
ITCM:              6648 B     256 KB      2.54%
DTCM:             256 KB     256 KB     100.00%
SRAM0:             0 GB       4 MB        0.00%
SRAM1:             0 GB     2560 KB     0.00%
MRAM:            10604 B    5628 KB     0.18%
info cbuild: build finished successfully!
* Terminal will be reused by tasks, press any key to close it.

```

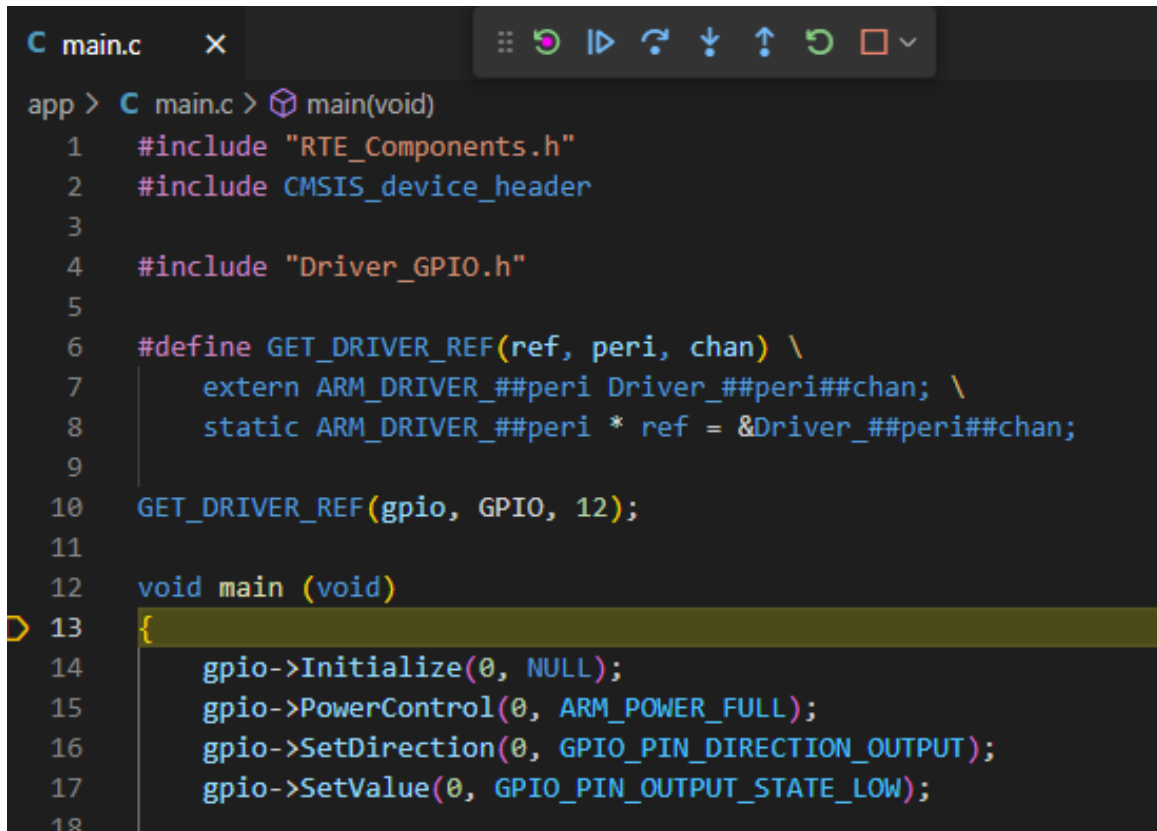
- Once the project is built successfully, you are ready to debug it. If this is the first time you have built this design for the selected core, you need to program the design using the Alif Security Toolkit. Press F1 and then select “Tasks: Run Task”, then select “Program with Security Toolkit (select COM port)”. If you have previously used a Security Toolkit task with this board, you do not need to select the COM port and can use “Program with Security Toolkit”.



- You will be prompted to select the COM port for your SEUART adapter, and the script will build the Alif ATOC image and program it into MRAM. At that point, the program is running, and you can now debug using the F5 debug function. For subsequent code changes and new builds using the same core, debugging can be done without performing this step. After the programming finishes, press the RESET button to execute the program.

If you change which core you are targeting as described in step 3, then you must execute “Program with Security Toolkit” at least once and then you can debug further changes using only the F5 debug function.

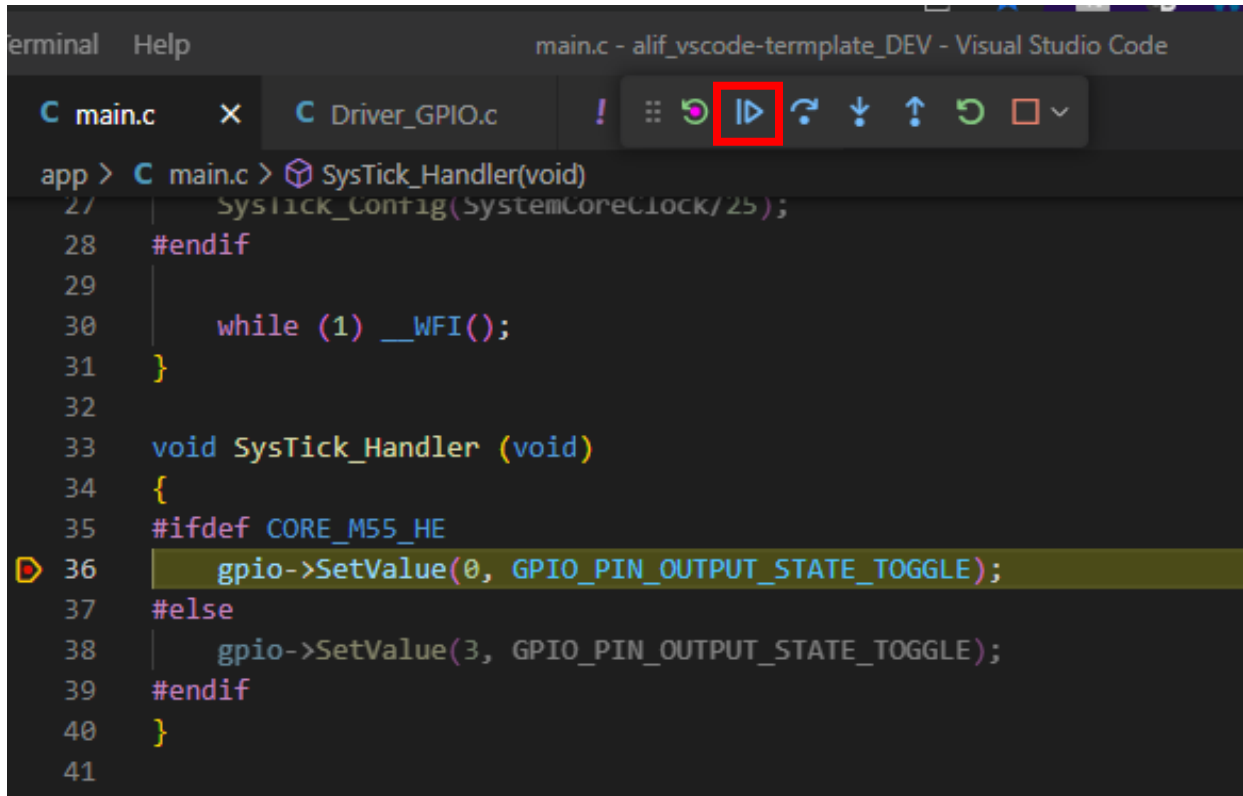
- Now press F5 to start the debug session. J-Link will download the application image into the Alif Ensemble SoC and Visual Studio Code will pause the hardware at the first line of main() function. If you're having trouble connecting, try power cycling your development board.



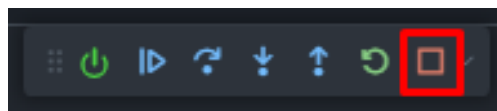
```
C main.c x
app > C main.c > main(void)
1  #include "RTE_Components.h"
2  #include CMSIS_device_header
3
4  #include "Driver_GPIO.h"
5
6  #define GET_DRIVER_REF(ref, peri, chan) \
7      extern ARM_DRIVER_##peri Driver_##peri##chan; \
8      static ARM_DRIVER_##peri * ref = &Driver_##peri##chan;
9
10 GET_DRIVER_REF(gpio, GPIO, 12);
11
12 void main (void)
13 {
14     gpio->Initialize(0, NULL);
15     gpio->PowerControl(0, ARM_POWER_FULL);
16     gpio->SetDirection(0, GPIO_PIN_DIRECTION_OUTPUT);
17     gpio->SetValue(0, GPIO_PIN_OUTPUT_STATE_LOW);
18
```

- This project is very basic and toggles the blue LED on and off on each interrupt from the SysTick timer.

10. If you set a breakpoint at line36, the LED will toggle each time you press the continue icon in the debugger control bar.



11. When you are done debugging, you can stop the debugger and disconnect by pressing the red square STOP icon on the debugger control bar.



5. Adding CMSIS Components to the application

1. When developing your application, you'll get access to the powerful collection of various CMSIS components provided by Alif, Arm, Microsoft, and other third parties. Basic set of necessary components has already been installed in the setup section. Open the `project.yaml` file:

```
! cproject.yaml
1  project:
2    groups:
3      - group: App
4        files:
5          - file: app/main.c
6            - file: app/gcc_M55_HE.ld
7              for-context: +HE
8            - file: app/gcc_M55_HP.ld
9              for-context: +HP
10   components:
11     # needed for Alif Ensemble support
12     - component: AlifSemiconductor::Device:Startup&C Startup
13     - component: ARM::CMSIS:CORE@5.6.0
14
15     # peripheral drivers & middleware, uncomment as needed
16     # - component: AlifSemiconductor::Device:SOC Peripherals:ADC
17     # - component: AlifSemiconductor::Device:SOC Peripherals:Analog Config
18     # - component: AlifSemiconductor::Device:SOC Peripherals:CAMERA Controller
19     # - component: AlifSemiconductor::Device:SOC Peripherals:CDC200
20     # - component: AlifSemiconductor::Device:SOC Peripherals:CRC
21     # - component: AlifSemiconductor::Device:SOC Peripherals:Comparator
22     # - component: AlifSemiconductor::Device:SOC Peripherals:DAC
23     # - component: AlifSemiconductor::Device:SOC Peripherals:DMA
24     # - component: AlifSemiconductor::Device:SOC Peripherals:Ethernet MAC
25     - component: AlifSemiconductor::Device:SOC Peripherals:GPIO
26     # - component: AlifSemiconductor::Device:SOC Peripherals:HWSEM
27     # - component: AlifSemiconductor::Device:SOC Peripherals:I2S
28     # - component: AlifSemiconductor::Device:SOC Peripherals:I3C
29     # - component: AlifSemiconductor::Device:SOC Peripherals:LPTIMER
30     # - component: AlifSemiconductor::Device:SOC Peripherals:MIPI CSI2
31     # - component: AlifSemiconductor::Device:SOC Peripherals:MIPI DSI
32     # - component: AlifSemiconductor::Device:SOC Peripherals:MRAM Flash
33     # - component: AlifSemiconductor::Device:SOC Peripherals:OSPI Controller
34     - component: AlifSemiconductor::Device:SOC Peripherals:PINCONF
35     # - component: AlifSemiconductor::Device:SOC Peripherals:RTC
36     # - component: AlifSemiconductor::Device:SOC Peripherals:SPI
37     # - component: AlifSemiconductor::Device:SOC Peripherals:USART
38     - component: AlifSemiconductor::Device:SOC Peripherals:UTIMER
39     # - component: AlifSemiconductor::Device:SOC Peripherals:WDT
40
```

2.

3. Currently the project is configured to only use CMSIS Core and Alif Startup components, but other common modules have been included and commented out. For example, you could add the UTIMER driver by placing cursor on line 38 and pressing DELETE to remove the "#".
4. Pressing `Ctrl+Shift+B` would then generate the project content and build the project.

Congratulations! You have completed the VS Code/GCC setup process for the Alif Ensemble DevKit.

Document History

| Version | Change Log |
|----------------|--|
| 1.0 | Initial release for Ensemble DevKit with Gen 2 silicon |
| 1.1 | Minor edits and formatting |
| 1.2 | Changed the link to Alif Security Toolkit for new web structure and link to the Alif VS Code template to point to the public repo on Github. |
| 1.3 | Added instructions on how to download the board library sub-module to the template |