AN1322 – Opening, Building, and Debugging a Project in VS Code

August 2024 Version 0.1

VA416xx

Abstract

This document is intended to provide instructions on how to get started with using Visual Studio Code (abbreviated VS Code) to develop, compile, download, and debug embedded firmware projects for the VORAGO VA416xx ARM® Cortex®-M4 processor. This powerful and extremely configurable IDE supports many languages, target devices, and compilers, but this document will focus on VA416xx embedded projects using CMake and the GCC compiler. This document assumes that the steps in the PEB1 User's Manual section 3.2.3 have already been followed to install the toolchain.

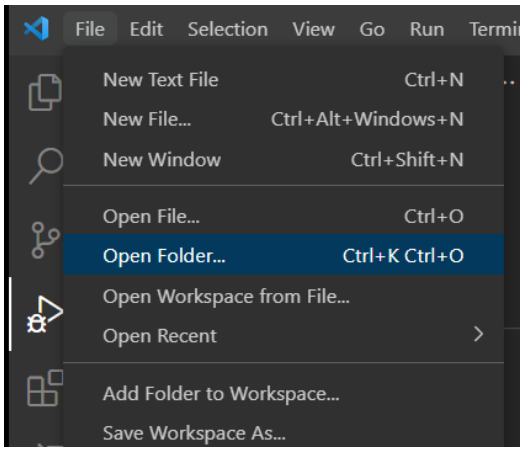
Table of Contents

1 Opening and Building a Project 1
1.1 Opening a workspace 2
1.2 Navigating through the projects and their files 3
1.3 Opening and editing the PEB1 EVK 'demo' project 4
1.4 Building the PEB1 EVK demo project 6
2 Downloading and debugging the demo project 8
2.1 J-link setup 8
2.2 How the VA416xx boots and runs code 9
2.3 Debug methods using VS Code 9
2.4 Entering debug mode 10
2.5 Running, stopping, and stepping through code 11
2.6 Exiting debug mode 12
2.7 The debug console 12
2.8 Setting and removing breakpoints 13
2.9 Viewing / watching variables 14
2.10 Call stack and breakpoints tabs 15
2.11 Viewing and editing MCU peripheral registers 15
2.12 Viewing and editing MCU core registers 16
2.13 Viewing memory 17
3 Conclusion 18
4 Other Resources 18

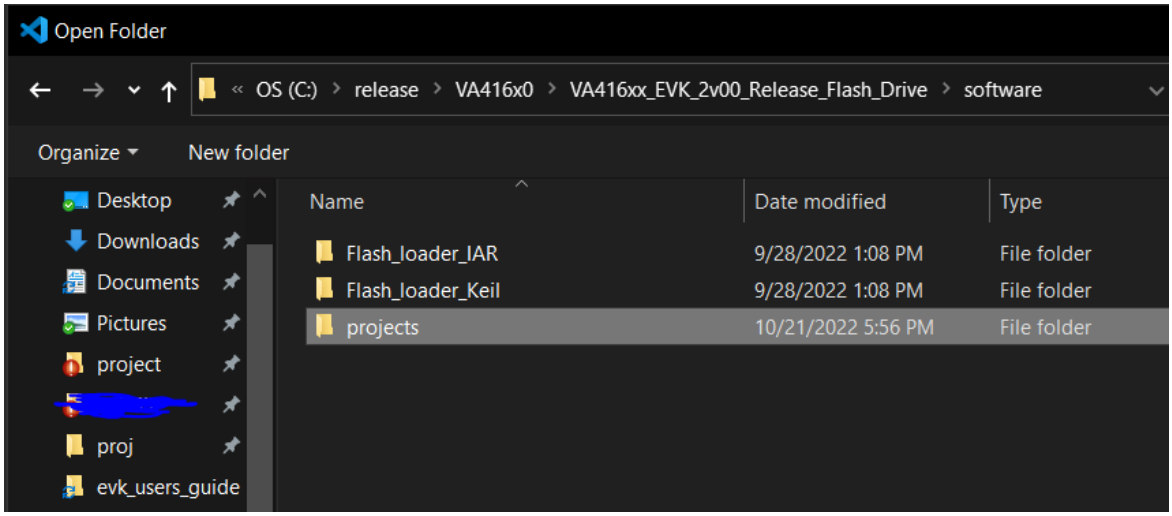
1 Opening and Building a Project

1.1 Opening a workspace

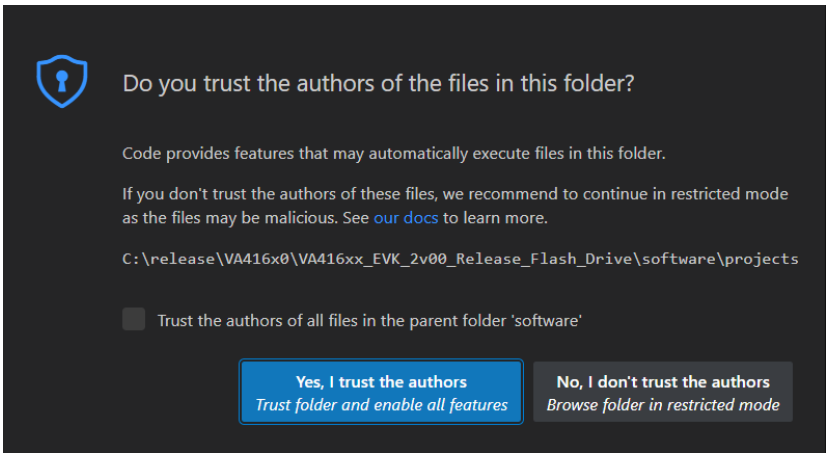
Instead of using a project file the way Keil or IAR uses, Visual Studio Code keeps all projects, files, and settings in a *workspace*, which is essentially a folder that contains all the necessary files. To open a VS Code workspace, run VS Code, and choose File->Open Folder.



Navigate to where the 'VA416xx_EVK_2vxx_Release_Flash_Drive' zip file was extracted. Go into the 'software' folder and select 'projects'. Click "select folder".



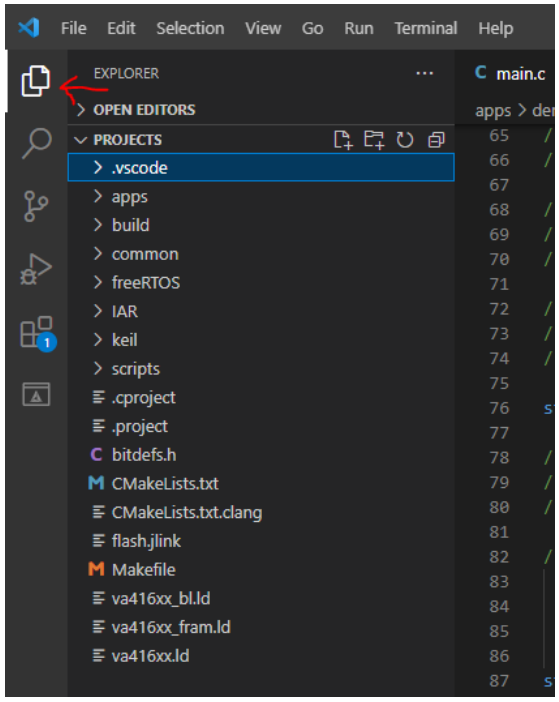
If it is the first time opening a workspace, you may see a message asking if you trust the authors of the files in this folder. Click "Yes, I trust the authors".



The workspace is now open and is ready for development.

1.2 Navigating through the projects and their files

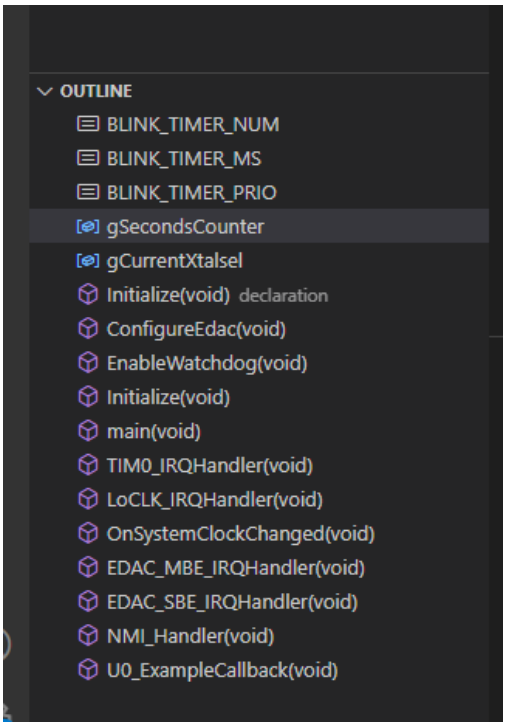
Navigating through the source files in VS Code is done by using the Explorer tab. This tab is accessed by clicking on the icon that looks like two files on the upper left hand sidebar, or by typing Control+Shift+E.



There are many folders and files within the VS Code workspace. The '.vscode' folder contains VS Code's configuration JSON files. The 'apps' folder contains source code for application projects. The 'build' folder contains the project build outputs. The 'common' folder contains

device driver source files that are common across multiple projects. The ‘freeRTOS’ folder contains the source files for freeRTOS. The ‘IAR’ and ‘keil’ folders contain the IAR and keil project files, and their build outputs, for when using these IDEs instead of VS Code. The scripts folder contains a jlink file and and the SVD register definition files for debugging. The top level workspace folder also contains some text files such as CMakeLists.txt, the Makefile, and *.ld linker scripts.

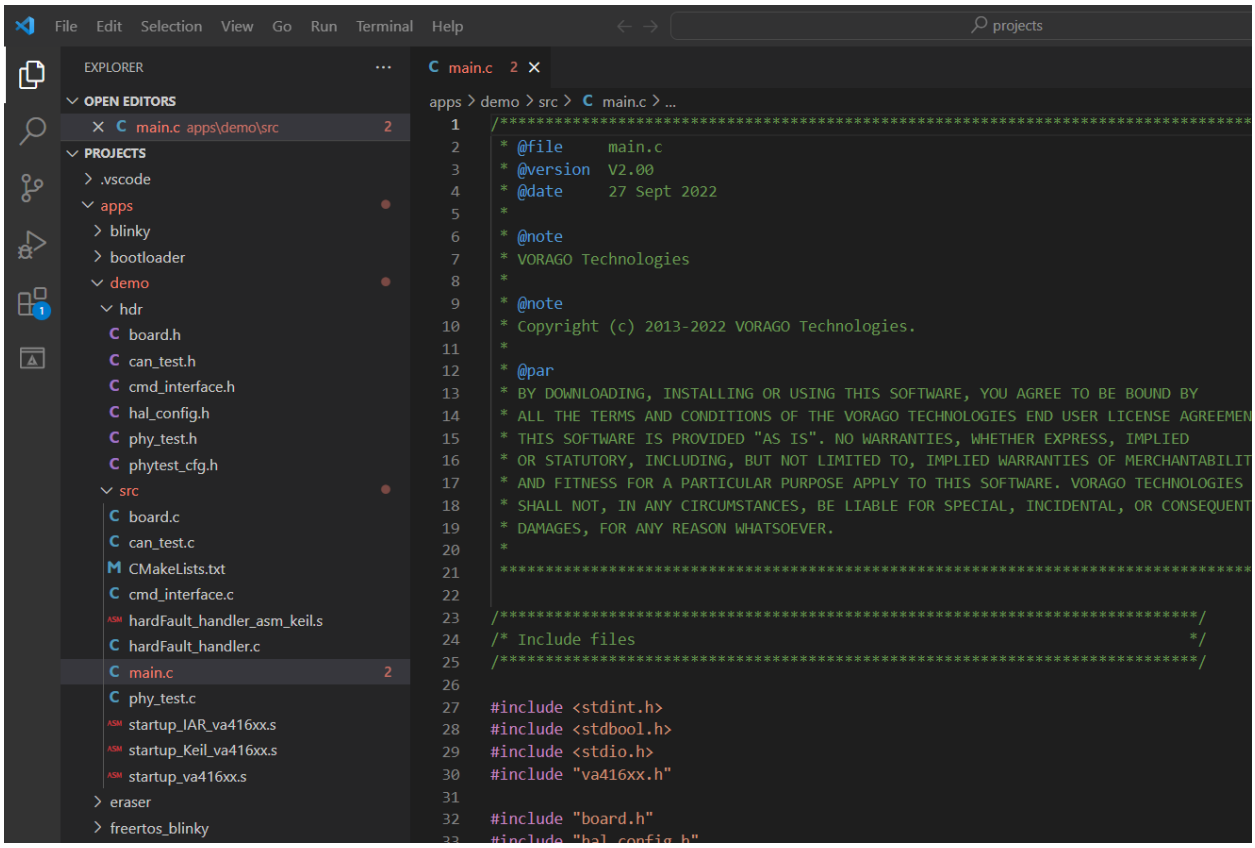
When a source file is open for viewing/editing, the *outline* tab will show a list of the macros, global variables, and function names in the active file. Clicking on one of these will jump to the definition in the file.



1.3 Opening and editing the PEB1 EVK ‘demo’ project

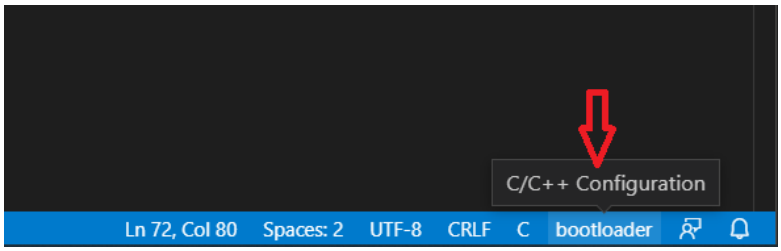
The demo project is a software application that is pre-loaded onto the PEB1 EVK board when it is shipped. This project sets up a command line interface to the board using the Segger RTT terminal. See the PEB1_Users_Manual document for a more detailed description of the demo project’s functionality and usage.

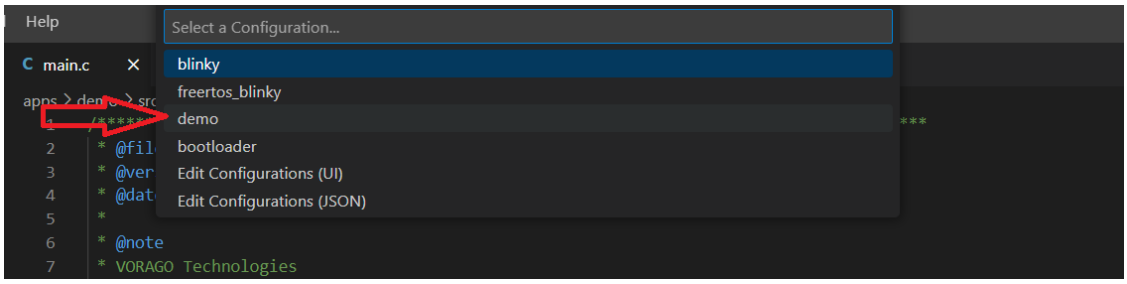
To view and edit the demo source files, expand the *apps* folder, then expand *demo,* and expand the *hdr* and *src* folders. Double click on *main.c*.



There may be a lot of red squiggly lines indicating that VS code's IntelliSense cannot find the required include files / it doesn't know which project's include files to reference. Since *demo* is the current project being edited, the include paths for *demo* must be selected. This is done by changing the C/C++ configuration to *demo*.

In the lower right corner of the VS Code window, click on the current C/C++ configuration, then select *demo* from the options that appear in the 'select a configuration' dropdown menu.

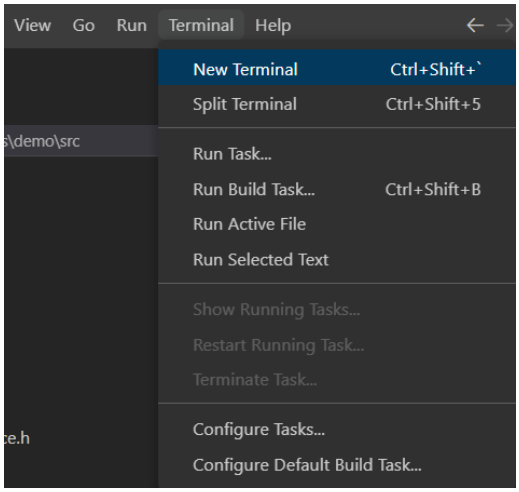




This will allow the VS Code Intellisense features to work correctly, such as autocompletion, “go to definition”, etc. If moving to work on a different project within the workspace, change the configuration from *demo* to one of the other options associated with the project to be edited (a configuration needs to be created when making a new project).

1.4 Building the PEB1 EVK demo project

Building a project involves running commands from the VS Code terminal window. Open a new terminal by clicking ‘New Terminal’ in the Terminal dropdown menu.



The first time a workspace is opened, or after modifying any of the makefiles, a ‘make init’ must be performed. This cleans the build folder and re-generates all the generated makefiles. The output of this command should looks like the following:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\release\VA416x0\VA416xx_EVK_2v00_Release_Flash_Drive\software\projects> make init
-- The C compiler identification is GNU 10.3.1
-- The CXX compiler identification is GNU 10.3.1
-- The ASM compiler identification is GNU
-- Found assembler: C:/Program Files (x86)/GNU Arm Embedded Toolchain/10 2021.10/bin/arm-none-eabi-gcc.exe
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: C:/Program Files (x86)/GNU Arm Embedded Toolchain/10 2021.10/bin/arm-none-eabi-gcc.exe - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: C:/Program Files (x86)/GNU Arm Embedded Toolchain/10 2021.10/bin/arm-none-eabi-g++.exe - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: C:/release/VA416x0/VA416xx_EVK_2v00_Release_Flash_Drive/software/projects/build
PS C:\release\VA416x0\VA416xx_EVK_2v00_Release_Flash_Drive\software\projects>
```

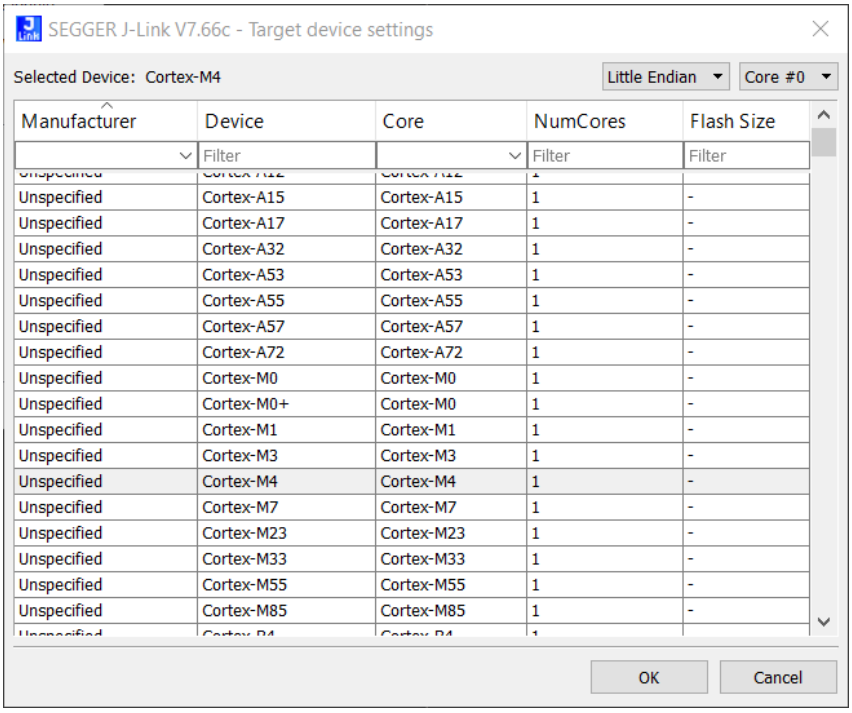
To build the *demo* project, type ‘make demo’. It should build with no errors or warnings, and the terminal output should look like the following:

```
Scanning dependencies of target demo
make[4]: Leaving directory `C:/release/VA416x0/VA416xx_EVK_2v00_Release_Flash_Drive/software/projects/build'
make[4]: Entering directory `C:/release/VA416x0/VA416xx_EVK_2v00_Release_Flash_Drive/software/projects/build'
[ 15%] Building C object CMakeFiles/demo.dir/apps/demo/src/board.c.o
[ 15%] Building C object CMakeFiles/demo.dir/apps/demo/src/can_test.c.o
[ 19%] Building C object CMakeFiles/demo.dir/apps/demo/src/cmd_interface.c.o
[ 23%] Building C object CMakeFiles/demo.dir/apps/demo/src/hardFault_handler.c.o
[ 26%] Building C object CMakeFiles/demo.dir/apps/demo/src/main.c.o
[ 26%] Building C object CMakeFiles/demo.dir/apps/demo/src/phy_test.c.o
[ 30%] Building ASM object CMakeFiles/demo.dir/apps/demo/src/startup_va416xx.s.o
[ 34%] Building C object CMakeFiles/demo.dir/common/drivers/src/va416xx_debug.c.o
[ 34%] Building C object CMakeFiles/demo.dir/common/drivers/src/va416xx_hal.c.o
[ 38%] Building C object CMakeFiles/demo.dir/common/drivers/src/va416xx_hal_adc.c.o
[ 42%] Building C object CMakeFiles/demo.dir/common/drivers/src/va416xx_hal_adc_swcal.c.o
[ 42%] Building C object CMakeFiles/demo.dir/common/drivers/src/va416xx_hal_canbus.c.o
[ 46%] Building C object CMakeFiles/demo.dir/common/drivers/src/va416xx_hal_clkgen.c.o
[ 50%] Building C object CMakeFiles/demo.dir/common/drivers/src/va416xx_hal_dac.c.o
[ 50%] Building C object CMakeFiles/demo.dir/common/drivers/src/va416xx_hal_dma.c.o
[ 53%] Building C object CMakeFiles/demo.dir/common/drivers/src/va416xx_hal_ethernet.c.o
[ 57%] Building C object CMakeFiles/demo.dir/common/drivers/src/va416xx_hal_i2c.c.o
[ 57%] Building C object CMakeFiles/demo.dir/common/drivers/src/va416xx_hal_ioconfig.c.o
[ 61%] Building C object CMakeFiles/demo.dir/common/drivers/src/va416xx_hal_irqrouter.c.o
[ 65%] Building C object CMakeFiles/demo.dir/common/drivers/src/va416xx_hal_spi.c.o
[ 65%] Building C object CMakeFiles/demo.dir/common/drivers/src/va416xx_hal_spw.c.o
[ 69%] Building C object CMakeFiles/demo.dir/common/drivers/src/va416xx_hal_timer.c.o
[ 73%] Building C object CMakeFiles/demo.dir/common/drivers/src/va416xx_hal_uart.c.o
[ 76%] Building C object CMakeFiles/demo.dir/common/mcu/src/system_va416xx.c.o
[ 76%] Building C object CMakeFiles/demo.dir/common/utils/src/circular_buffer.c.o
[ 80%] Building C object CMakeFiles/demo.dir/common/utils/src/dac_sine.c.o
[ 84%] Building C object CMakeFiles/demo.dir/common/utils/src/segger_rtt.c.o
[ 84%] Building C object CMakeFiles/demo.dir/common/utils/src/segger_rtt_printf.c.o
[ 88%] Building C object CMakeFiles/demo.dir/common/utils/src/spi_fram.c.o
[ 92%] Building C object CMakeFiles/demo.dir/common/BSP/evk/src/evk_board.c.o
[ 92%] Building C object CMakeFiles/demo.dir/common/BSP/evk/src/i2c_adxl343.c.o
[ 96%] Building C object CMakeFiles/demo.dir/common/BSP/evk/src/i2c_lis2de12.c.o
[100%] Linking C executable demo.elf
Memory region         Used Size  Region Size  %age Used
            FLASH:       35012 B       256 KB     13.36%
              RAM:        5656 B        32 KB     17.26%
           CCMRAM:         128 B        32 KB      0.39%
   text    data     bss     dec     hex filename
  33780    1228    4560   39568    9a90 demo.elf
make[4]: Leaving directory `C:/release/VA416x0/VA416xx_EVK_2v00_Release_Flash_Drive/software/projects/build'
[100%] Built target demo
make[3]: Leaving directory `C:/release/VA416x0/VA416xx_EVK_2v00_Release_Flash_Drive/software/projects/build'
make[2]: Leaving directory `C:/release/VA416x0/VA416xx_EVK_2v00_Release_Flash_Drive/software/projects/build'
make[1]: Leaving directory `C:/release/VA416x0/VA416xx_EVK_2v00_Release_Flash_Drive/software/projects/build'
PS C:\release\VA416x0\VA416xx_EVK_2v00_Release_Flash_Drive\software\projects>
```

2 Downloading and debugging the demo project

2.1 J-link setup

By default, the provided VS Code workspace is set up to use the J-link debug pod or the J-Link OB (on board) debug interface included on the PEB1 EVK board. At this point the Segger J-link software should be installed on your PC, and the “Jlink GDB server Vx.xx” tool run at least once. If the SEGGER software asks for a Target Device, select a generic Cortex-M4.



Manufacturer	Device	Core	NumCores	Flash Size
	Filter		Filter	Filter
Unspecified	Cortex-A15	Cortex-A15	1	-
Unspecified	Cortex-A17	Cortex-A17	1	-
Unspecified	Cortex-A32	Cortex-A32	1	-
Unspecified	Cortex-A53	Cortex-A53	1	-
Unspecified	Cortex-A55	Cortex-A55	1	-
Unspecified	Cortex-A57	Cortex-A57	1	-
Unspecified	Cortex-A72	Cortex-A72	1	-
Unspecified	Cortex-M0	Cortex-M0	1	-
Unspecified	Cortex-M0+	Cortex-M0	1	-
Unspecified	Cortex-M1	Cortex-M1	1	-
Unspecified	Cortex-M3	Cortex-M3	1	-
Unspecified	Cortex-M4	Cortex-M4	1	-
Unspecified	Cortex-M7	Cortex-M7	1	-
Unspecified	Cortex-M23	Cortex-M23	1	-
Unspecified	Cortex-M33	Cortex-M33	1	-
Unspecified	Cortex-M55	Cortex-M55	1	-
Unspecified	Cortex-M85	Cortex-M85	1	-

2.2 How the VA416xx boots and runs code

The VA416xx MCUs rely on an external SPI based memory device to boot from. The 128kbyte memory is transferred to the MCU's program RAM automatically during the MCU boot process by a hardware bootloader. The location of the code in the SPI memory device is the same as the location inside the MCU. For instance, the RESET vector information is located at address 0x00000000 in both devices. This means that for debugging in RAM or debugging by downloading the code to the nonvolatile memory, the code starting address is the same (0x00000000), and the same linker settings will be used. Effectively, the code always runs from instruction RAM at 0x00000000 out of a core reset, but the code can get there from one of three ways:

1. The IRAM is populated at bootup by reading the SPI NVM (if EBIBOOT=0), 256KB.
2. The IRAM is populated at bootup by reading memory from the external bus, copying 0x10000000-0x1003FFFF (256k) to 0x00000000-0x0003FFFF (if EBIBOOT=1).
3. The IRAM can be written by the debugger, and the code executed (debug mode).

2.3 Debug methods using VS Code

In the VS Code environment, two debug methods are provided:

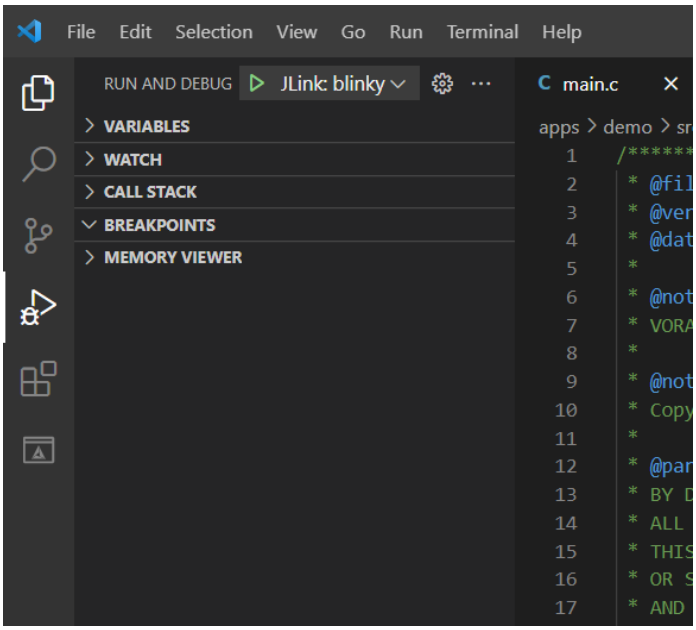
1. Executable code is loaded into instruction RAM, and execution begins. On the next processor reset, the contents from the NVM will be loaded and executed instead.
2. Executable code is loaded into instruction RAM. A loader program is loaded into DATA RAM. The loader program executes, writing the contents of the instruction RAM to the

SPI NVM (FRAM). Once the loader completes, execution of the main program begins (reset vector loaded from 0x00000000).

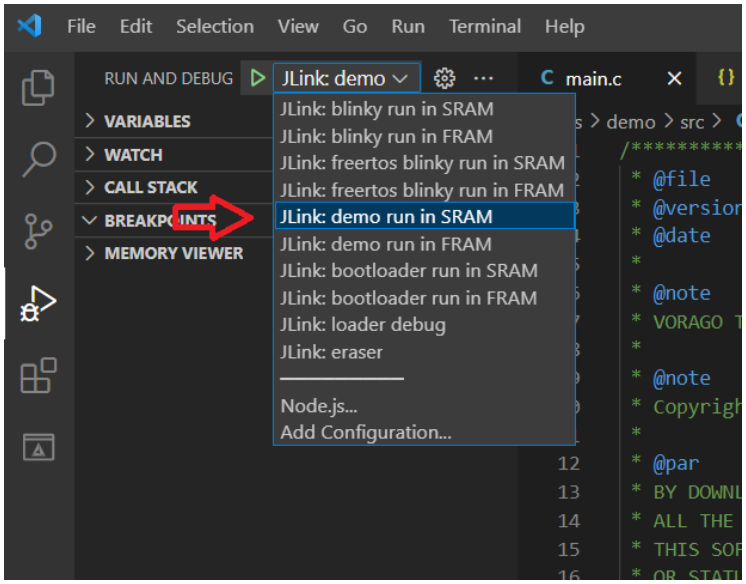
Note: the source code for the loader program is included in the workspace, so it can be easily modified to write to other NVM types on either the SPI or the external memory (EBI).

2.4 Entering debug mode

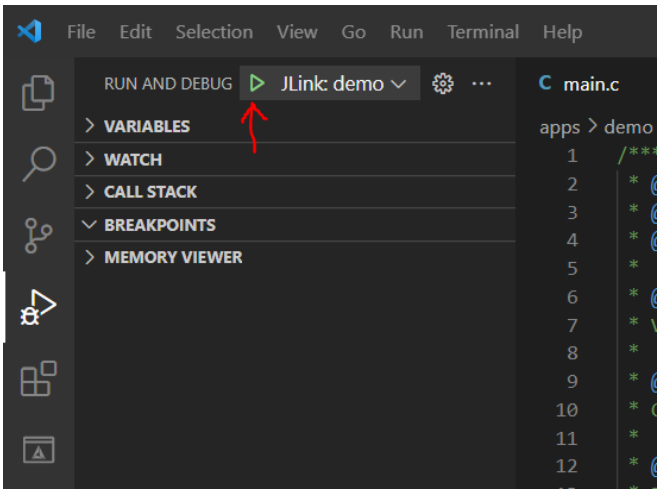
To debug the *demo* program, first navigate to the 'Run and Debug' tab on the left hand side of the VS Code window, or type Control+Shift+D.



In the dropdown menu next to the green triangle, select "Jlink: demo run in SRAM". If you would like the code download to be persistent, and overwrite the current NVM contents, choose "Jlink: demo run in FRAM" instead.

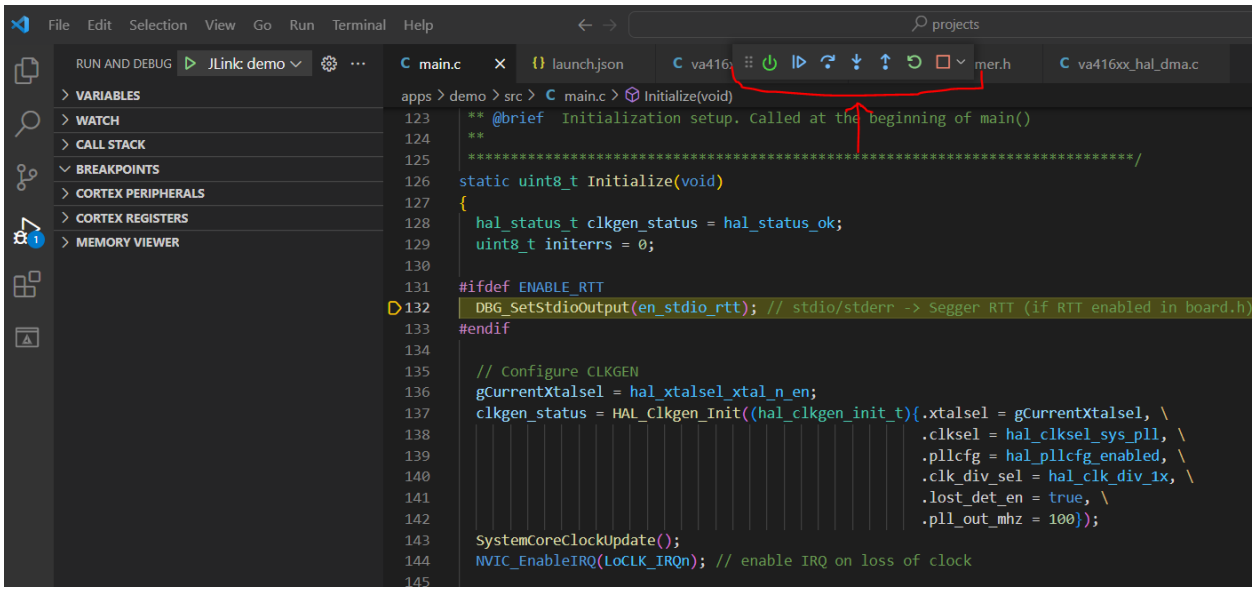


With the PEB1 board plugged into the PC with the micro USB cable, click the green triangle to begin a debug session.



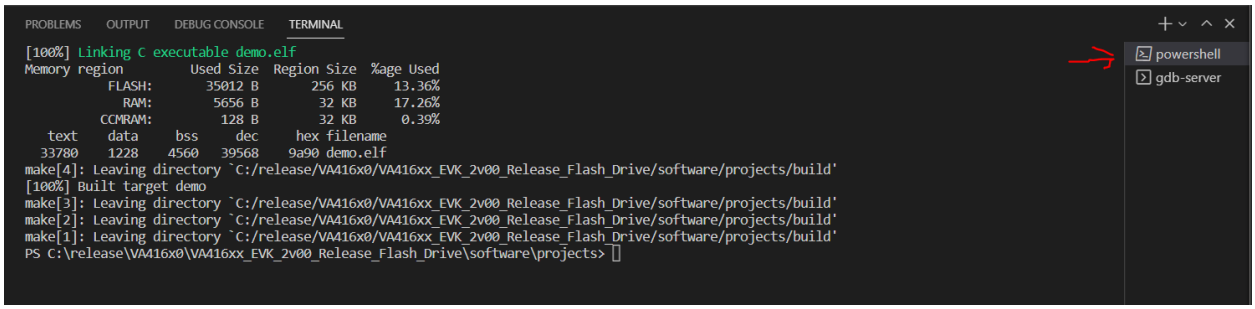
2.5 Running, stopping, and stepping through code

Once the connection to the board has been established, the code is downloaded to the MCU RAM. A breakpoint is set at the beginning of main(), and will be waiting to begin. In the upper middle of the window the debugger control buttons will appear, reset, run/stop, step over, step into, step out, restart, and stop. Clicking the run button will make the project run, and the PG5 LED on the PEB1 top board should start blinking.



2.6 Exiting debug mode

Press the orange square button to end a debug session. To get access to the terminal window again to run another ‘make init’ or ‘make $project_name’, either kill the gdb-server process, or click on the powershell process to bring it to the foreground.



2.7 The debug console

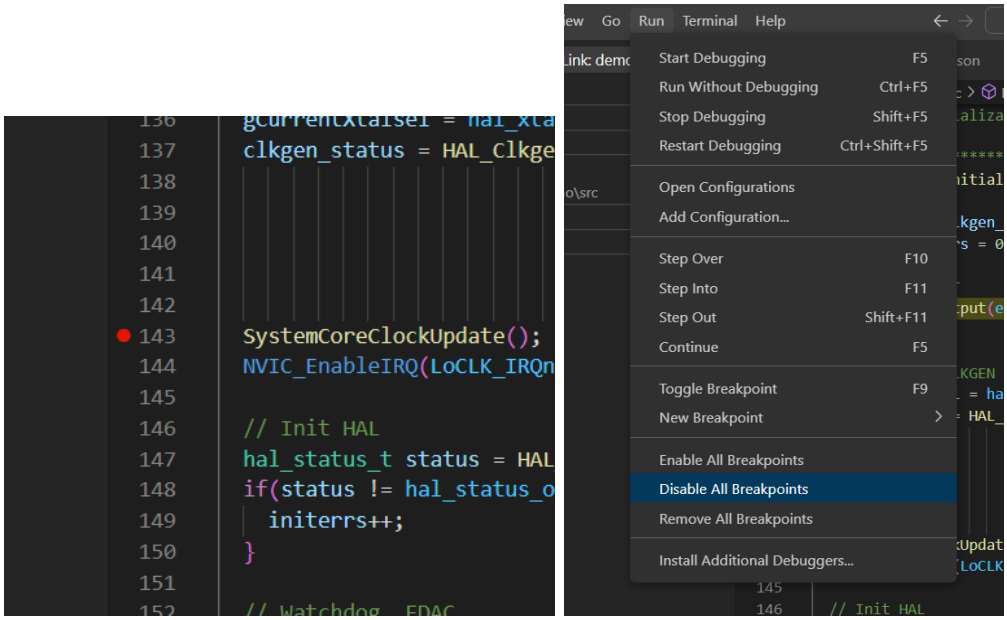
The debug console at the bottom of the VS Code window will show the debugger status. If there is a problem with the connection to the board, information here should help track down the cause of the issue.

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL                                   Filter (e.g. text, !exclude)
Cortex-Debug: VSCode debugger extension version 1.6.7 git(b0f5563). Usage info: https://github.com/Marus/cortex-debug#usage
Reading symbols from arm-none-eabi-objdump.exe --syms -C -h -w C:/release/VA416x0/VA416xx_EVK_2v00_Release_Flash_Drive/software/projects/build/demo.e
lf
Reading symbols from arm-none-eabi-nm.exe --defined-only -S -l -C -p C:/release/VA416x0/VA416xx_EVK_2v00_Release_Flash_Drive/software/projects/build/
demo.elf
Launching GDB: arm-none-eabi-gdb.exe -q --interpreter=mi2
    IMPORTANT: Set "showDevDebugOutput": "raw" in "launch.json" to see verbose GDB transactions here. Very helpful to debug issues or report problems
Launching gdb-server: JLinkGDBServerCL.exe -singlerun -nogui -if swd -port 50000 -swoport 50001 -telnetport 50002 -device Cortex-M4
    Please check TERMINAL tab (gdb-server) for output from JLinkGDBServerCL.exe
Finished reading symbols from objdump: Time: 86 ms
Finished reading symbols from nm: Time: 90 ms
C:\Program Files (x86)\GNU Arm Embedded Toolchain\10 2021.10\bin\arm-none-eabi-gdb.exe: warning: Couldn't determine a path for the index cache direct
ory.
0x00002058 in Initialize () at C:/release/VA416x0/VA416xx_EVK_2v00_Release_Flash_Drive/software/projects/apps/demo/src/main.c:220
220       status = HAL_DMA_Init(NULL, false, false, false);
Program stopped, probably due to a reset and/or halt issued by debugger
Resets core & peripherals via SYSRESETREQ & VECTRESET bit.
Loading section .isr_vector, size 0x350 lma 0x0
Loading section .text, size 0x6b1c lma 0x350
Loading section .rodata, size 0x1588 lma 0x6e70
Loading section .init_array, size 0x4 lma 0x83f8
Loading section .fini_array, size 0x4 lma 0x83fc
Loading section .data, size 0x4c4 lma 0x8400
Start address 0x000003a4, load size 35008
Transfer rate: 148 KB/sec, 5001 bytes/write.

Temporary breakpoint 1, main () at C:/release/VA416x0/VA416xx_EVK_2v00_Release_Flash_Drive/software/projects/apps/demo/src/main.c:132
132       DBG_SetStdioOutput(en_stdio_rtt); // stdio/stderr -> Segger RTT (if RTT enabled in board.h)
```

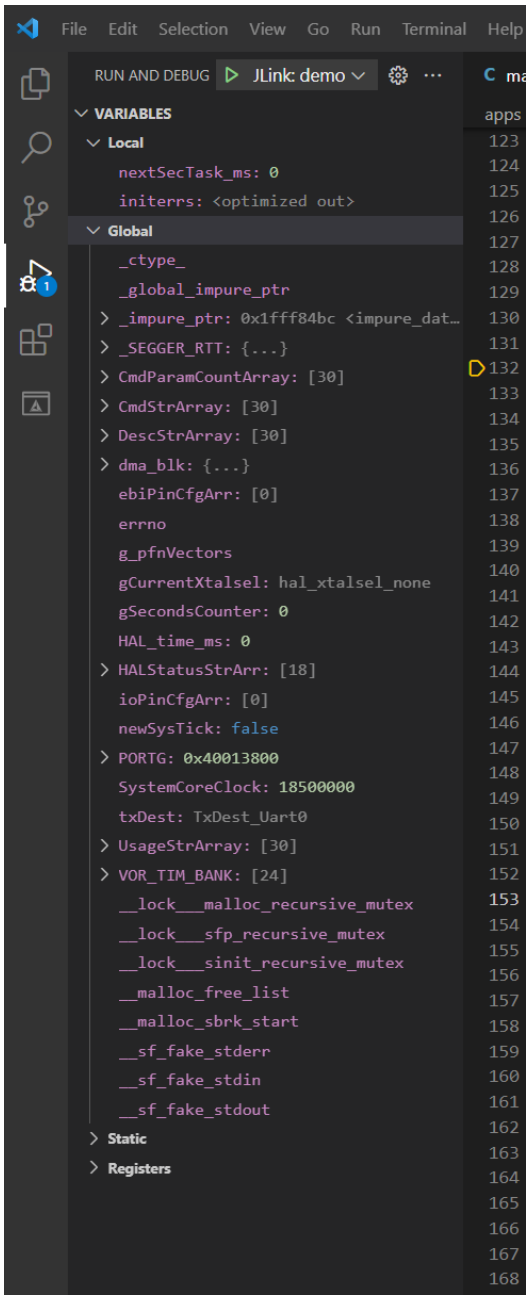
2.8 Setting and removing breakpoints

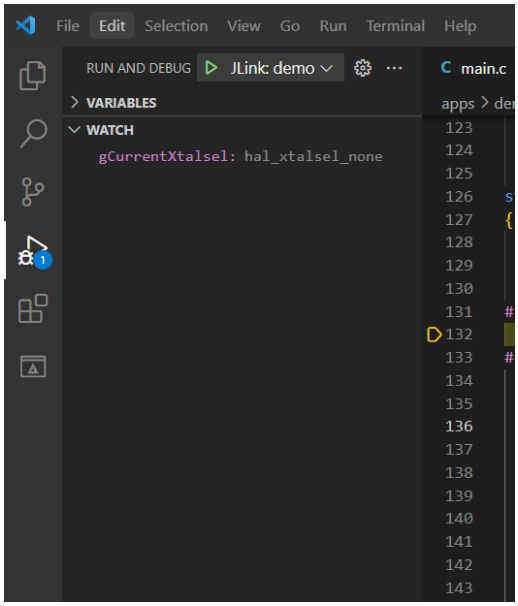
To set a breakpoint, click just to the left of the line number. Clicking the breakpoint again will remove it. All breakpoints can be disabled/removed by clicking the Run dropdown menu, then selecting Disable All Breakpoints, or Remove All Breakpoints.



2.9 Viewing / watching variables

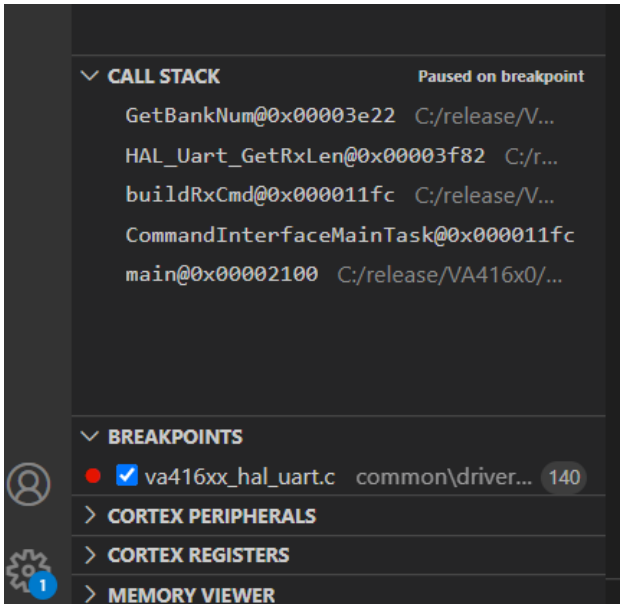
Program variables can be viewed using the ‘Variables’ tab in debug mode. To add a variable to the watch pane, click the “+” to add a new watch, type in the variable or structure name, and hit enter.





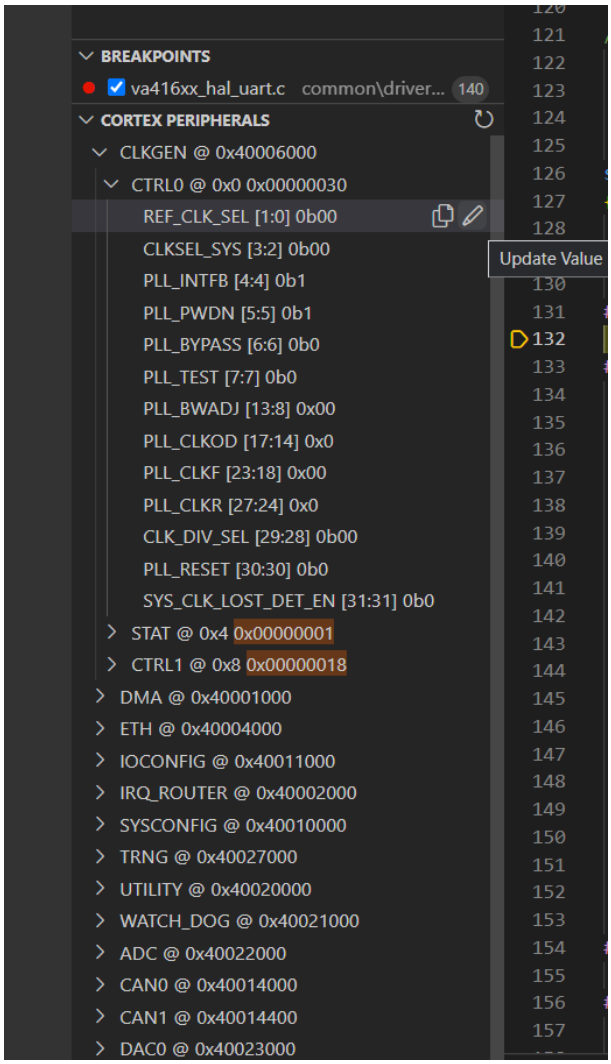
2.10 Call stack and breakpoints tabs

The call stack tab will show the current call stack within the program. Any active breakpoints will appear in the breakpoints tab.



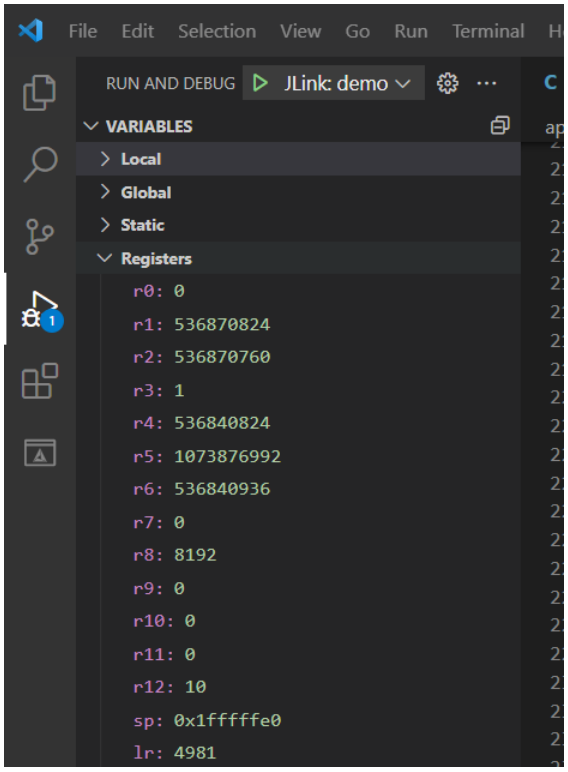
2.11 Viewing and editing MCU peripheral registers

The Cortex Peripherals tab is used to view and modify peripheral registers. When mousing over a bit field or register, 2 buttons will appear to either copy the value, or edit it.



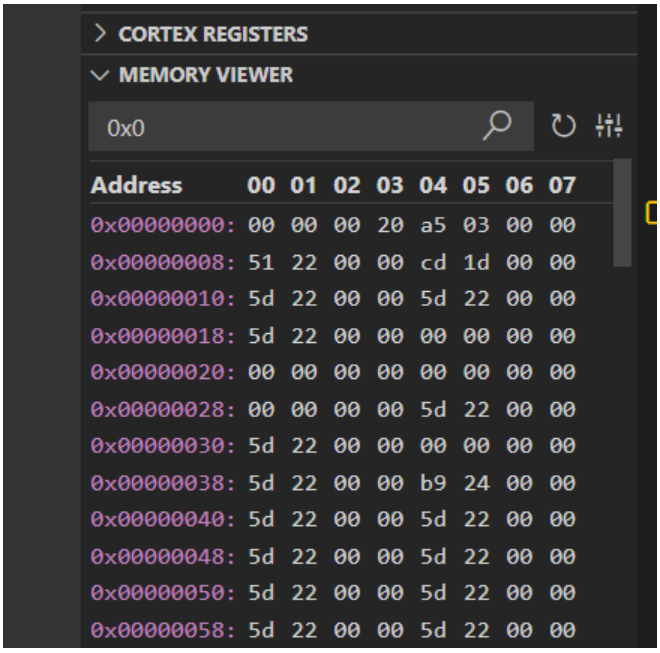
2.12 Viewing and editing MCU core registers

Core registers can be viewed through either the Cortex Registers tab, or through the Registers dropdown in the Variables tab. Core register values can only be modified through the Variables tab, however.



2.13 Viewing memory

The Memory Viewer tab can show the contents of accessible memory regions. Type a memory address into the box at the top of the tab.



3 Conclusion

These instructions show how to compile and run the VA416xx demo project in the VS Code IDE. It also shows the features and usage of the Cortex Debug plugin in VS Code for downloading and debugging projects.

4 Other Resources

Revision log:

August 13, 2024 – Initial template created (V0.1)