OPENING, BUILDING, AND DEBUGGING A PROJECT IN VS CODE IDE

Relevant products: VA416xx

DECEMBER 5, 2022

VORAGO TECHNOLOGIES

Contents

1 Introduction

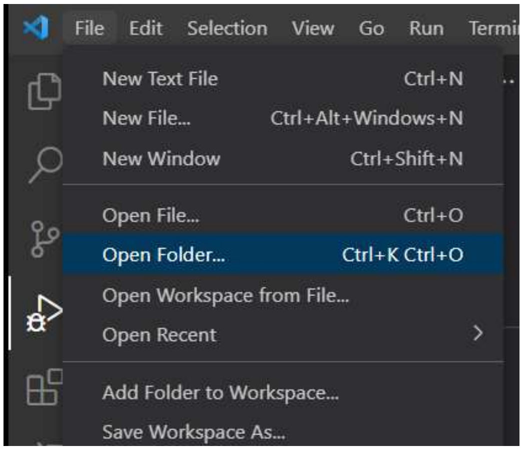
1.1 Purpose of Document

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_Users_Manual section 3.2.3 have been followed to install the toolchain.

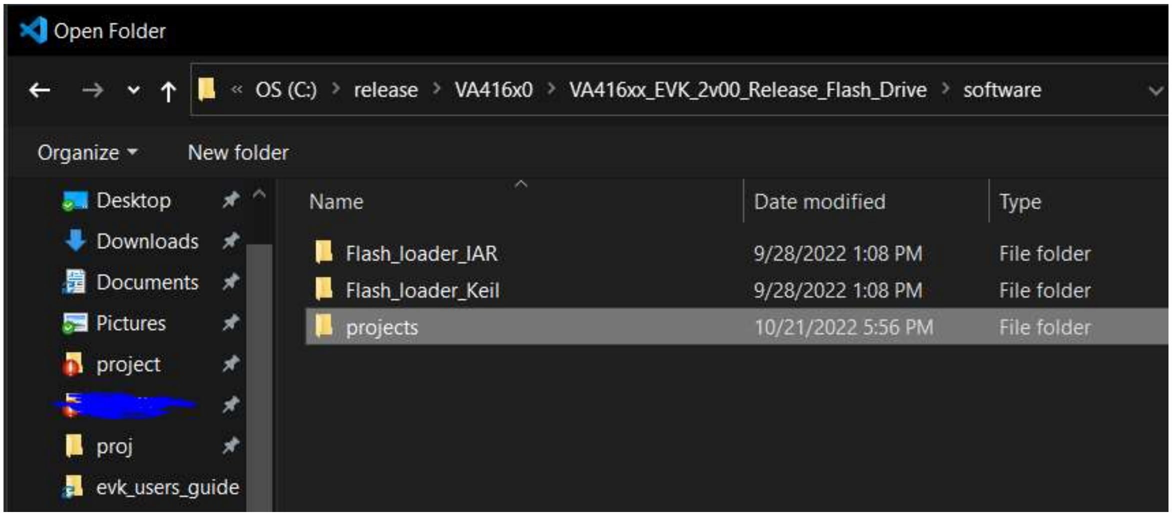
2 Opening and building a project

2.1 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”.



Open Folder

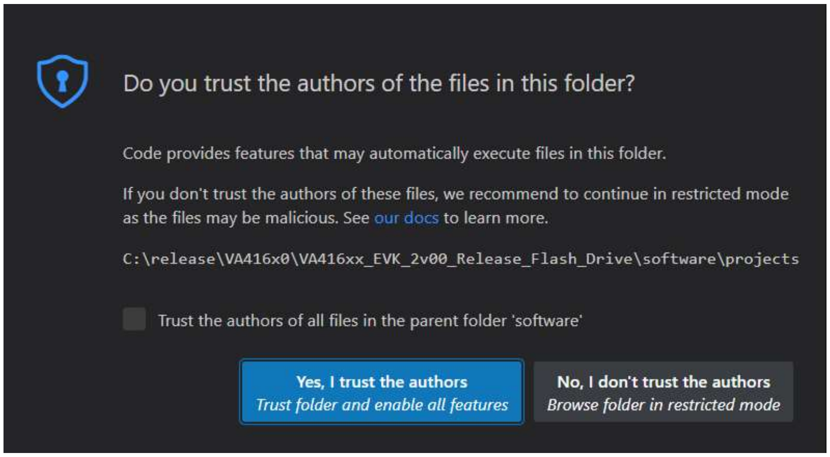
« OS (C:) › release › VA416x0 › VA416xx_EVK_2v00_Release_Flash_Drive › software

Organize ▾ New folder

- Desktop
- Downloads
- Documents
- Pictures
- project
- [redacted]
- proj
- evk_users_guide

Name	Date modified	Type
Flash_loader_IAR	9/28/2022 1:08 PM	File folder
Flash_loader_Keil	9/28/2022 1:08 PM	File folder
projects	10/21/2022 5:56 PM	File 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”.



Do you trust the authors of the files in this folder?

Code provides features that may automatically execute files in this folder.

If you don't trust the authors of these files, we recommend to continue in restricted mode as the files may be malicious. See our docs to learn more.

```
C:\release\VA416x0\VA416xx_EVK_2v00_Release_Flash_Drive\software\projects
```

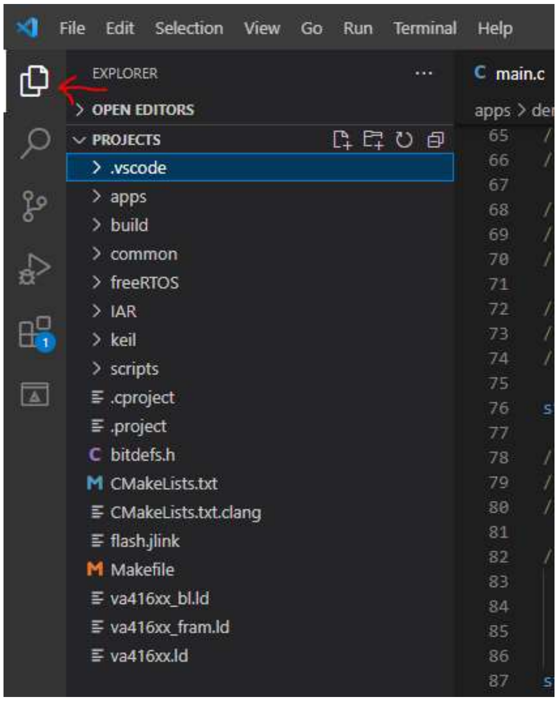
□ Trust the authors of all files in the parent folder 'software'

Yes, I trust the authors
Trust folder and enable all features
No, I don't trust the authors
Browse folder in restricted mode

The workspace is now open and is ready for development.

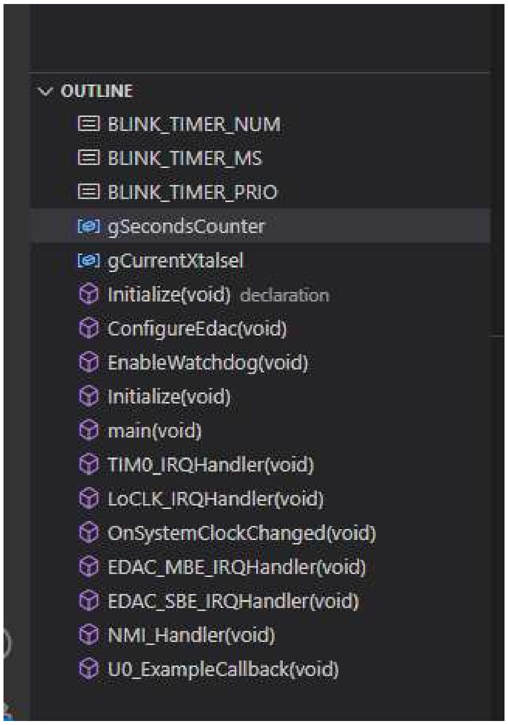
2.2 Navigating through the projects and their files

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.

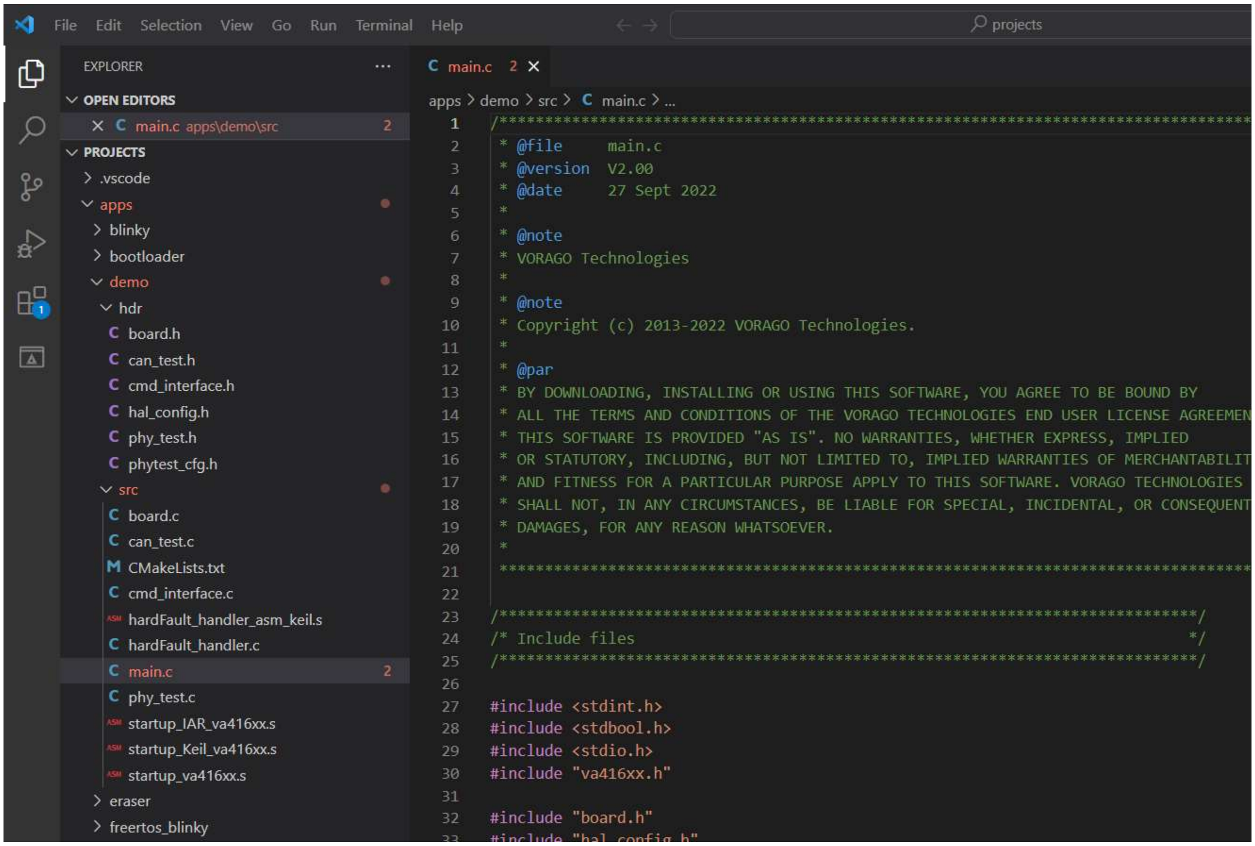


OUTLINE
- BLINK_TIMER_NUM
- BLINK_TIMER_MS
- BLINK_TIMER_PRIO
- gSecondsCounter
- gCurrentXtalsel
- Initialize(void) declaration
- ConfigureEdac(void)
- EnableWatchdog(void)
- Initialize(void)
- main(void)
- TIM0_IRQHandler(void)
- LoCLK_IRQHandler(void)
- OnSystemClockChanged(void)
- EDAC_MBE_IRQHandler(void)
- EDAC_SBE_IRQHandler(void)
- NMI_Handler(void)
- U0_ExampleCallback(void)

2.3 Opening / editing the EVK ‘demo’ project

The demo project is the 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.

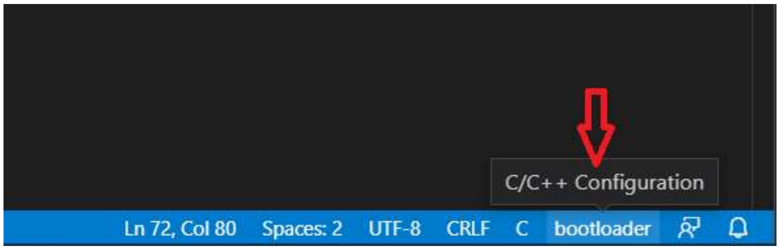
To view and edit the demo source files, expand the *apps* folder, then expand *demo,* and expand the *hdr* and *src* folders. Double click on *main.c*.

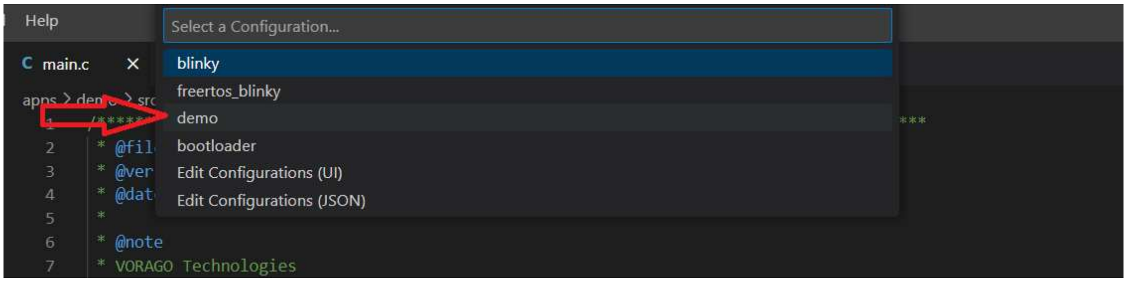


```
/*************************************************************************************
 * @file     main.c
 * @version  V2.00
 * @date     27 Sept 2022
 *
 * @note
 * VORAGO Technologies
 *
 * @note
 * Copyright (c) 2013-2022 VORAGO Technologies.
 *
 * @par
 * BY DOWNLOADING, INSTALLING OR USING THIS SOFTWARE, YOU AGREE TO BE BOUND BY
 * ALL THE TERMS AND CONDITIONS OF THE VORAGO TECHNOLOGIES END USER LICENSE AGREEMEN
 * THIS SOFTWARE IS PROVIDED "AS IS". NO WARRANTIES, WHETHER EXPRESS, IMPLIED
 * OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILIT
 * AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. VORAGO TECHNOLOGIES
 * SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL, OR CONSEQUENT
 * DAMAGES, FOR ANY REASON WHATSOEVER.
 *
 *************************************************************************************

/*****************************************************************************/
/* Include files                                                             */
/*****************************************************************************/

#include <stdint.h>
#include <stdbool.h>
#include <stdio.h>
#include "va416xx.h"

#include "board.h"
#include "hal_config.h"
```

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.

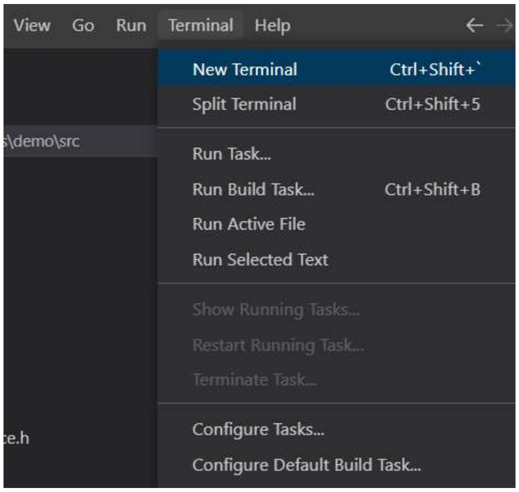




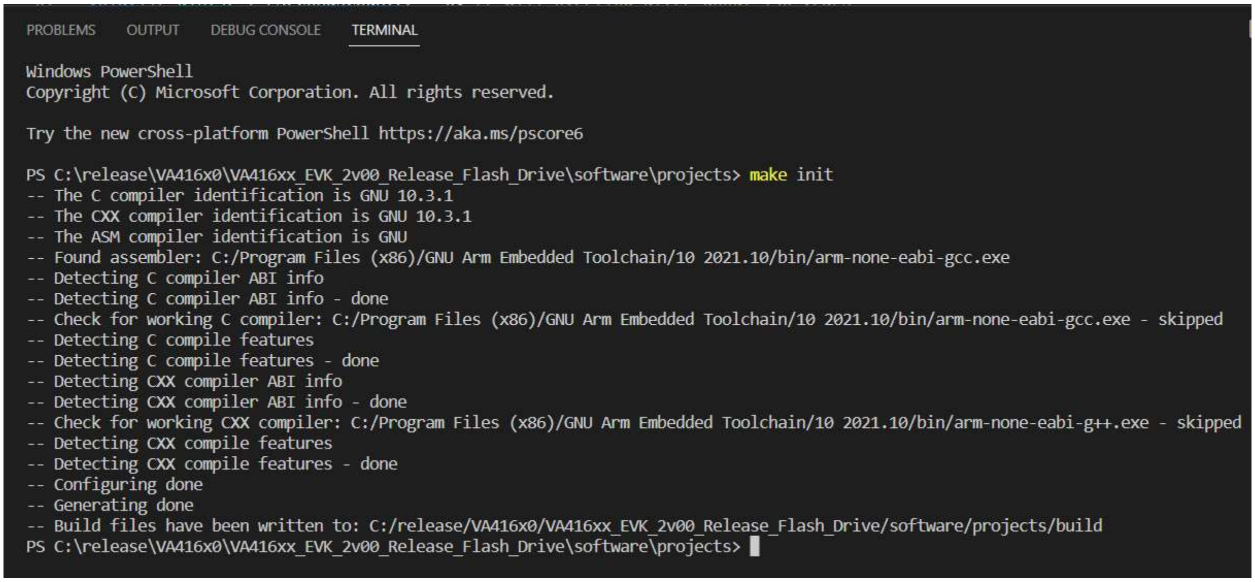
This will allow the VS Code Intellisense features to work correctly, such as autocompletion, “go to definition”, etc. If moving to work on a different project within the workspace, change the configuration from *demo* to one of the other options associated with the project to be edited (a configuration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

2.4 Building the 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.

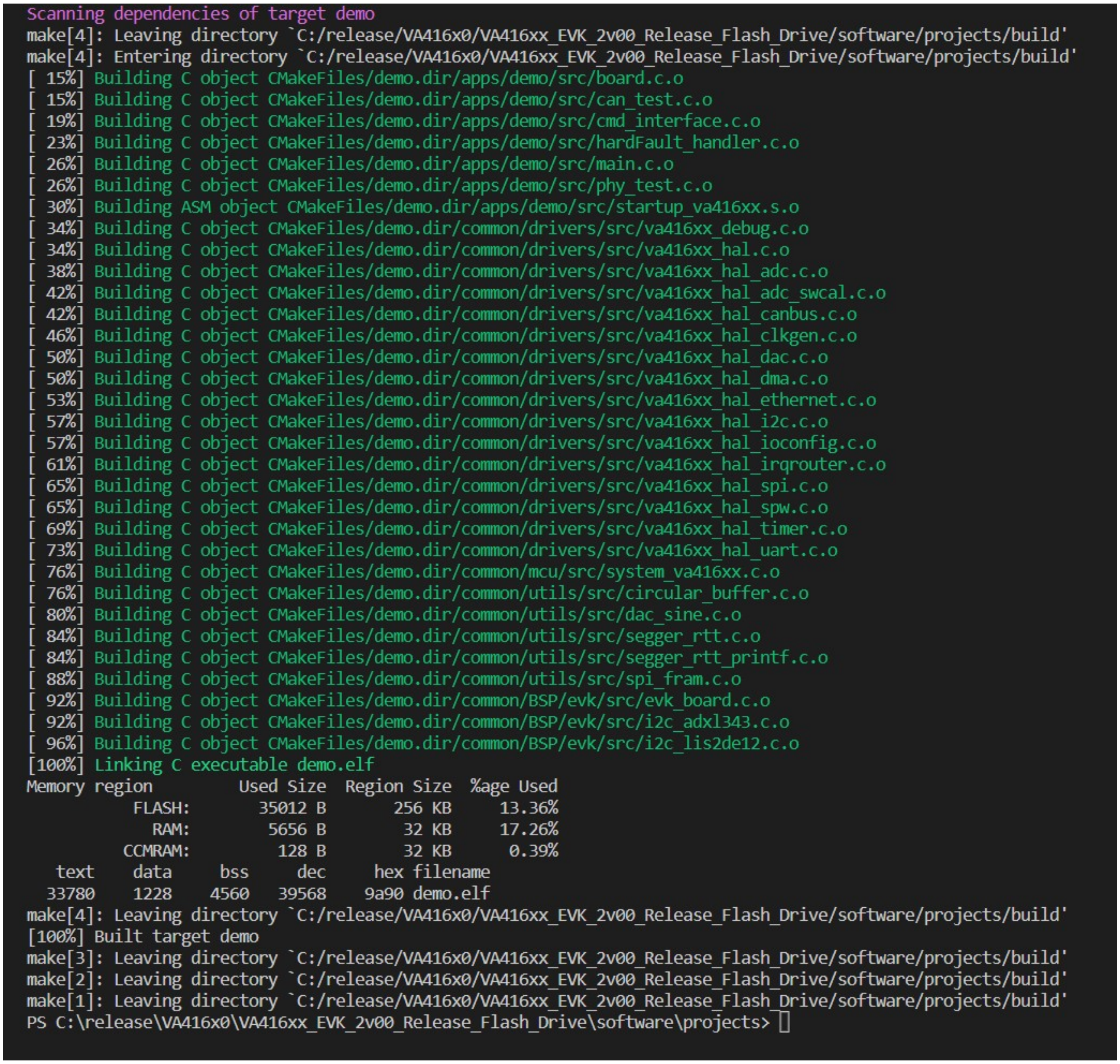


The first time a workspace is opened, or after modifying any of the makefiles, a ‘make init’ must be performed. This cleans the build folder and re-generates all the generated makefiles. The output of this command should looks like the following:



```
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:

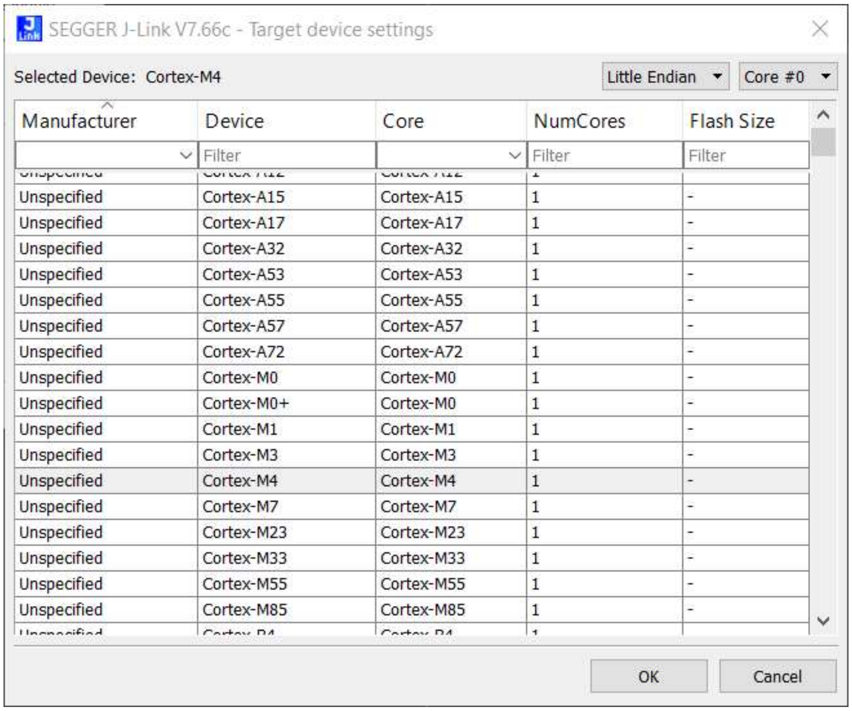


```
Scanning dependencies of target demo
make[4]: Leaving directory `C:/release/VA416x0/VA416xx_EVK_2v00_Release_Flash_Drive/software/projects/build'
make[4]: Entering directory `C:/release/VA416x0/VA416xx_EVK_2v00_Release_Flash_Drive/software/projects/build'
[ 15%] Building C object CMakeFiles/demo.dir/apps/demo/src/board.c.o
[ 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> 
```

2.5 Running and debugging the demo project

2.5.1 Jlink 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.



Selected Device: Cortex-M4

Manufacturer	Device	Core	NumCores	Flash Size
	Filter		Filter	Filter
Unspecified	Cortex-A12	Cortex-A12	1	
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	-
Unspecified	Cortex-R4	Cortex-R4	1	

2.5.2 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).

2.5.3 Debug methods using VS Code

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

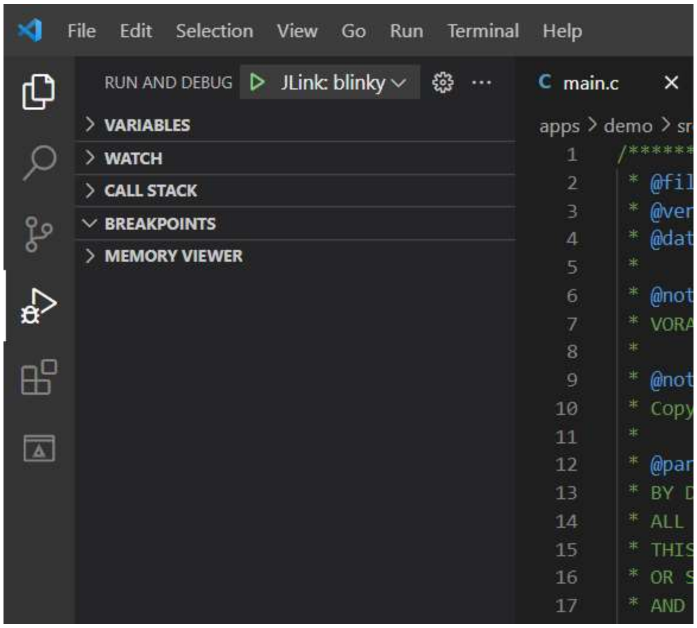
1. Executable code is loaded into instruction RAM, and execution begins. On the next processor reset, the contents from the NVM will be loaded and executed instead.

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).

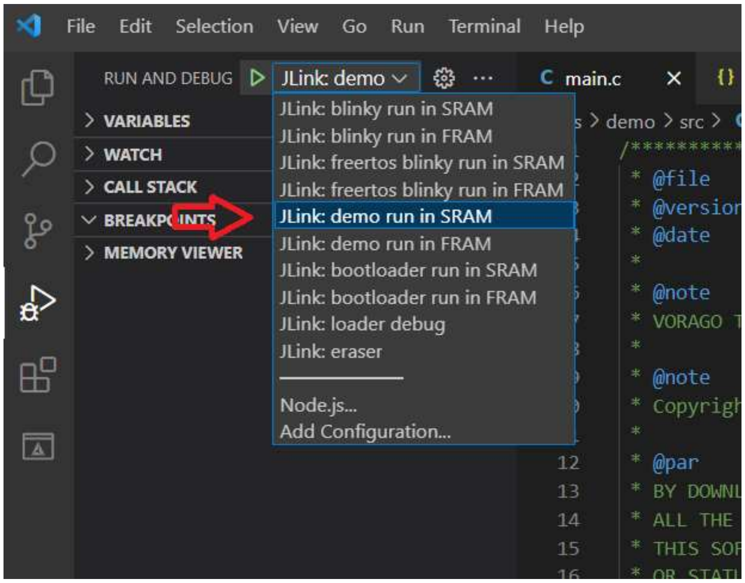
2.5.4 Entering debug mode

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.

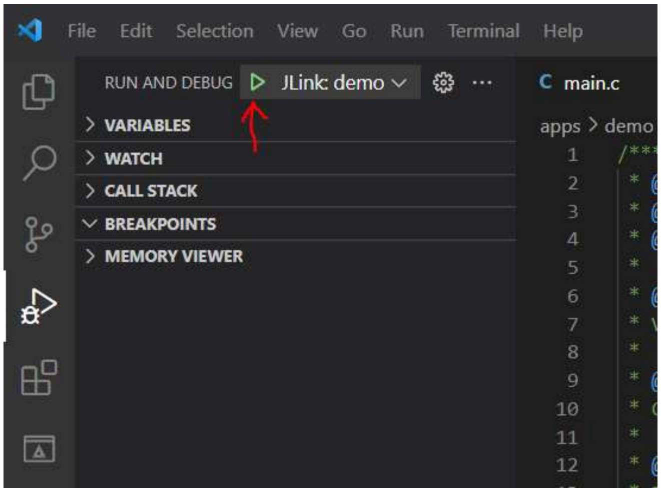


```
/******
 * @fil
 * @ver
 * @dat
 *
 * @not
 * VORA
 *
 * @not
 * Copy
 *
 * @par
 * BY D
 * ALL
 * THIS
 * OR S
 * AND
```

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.

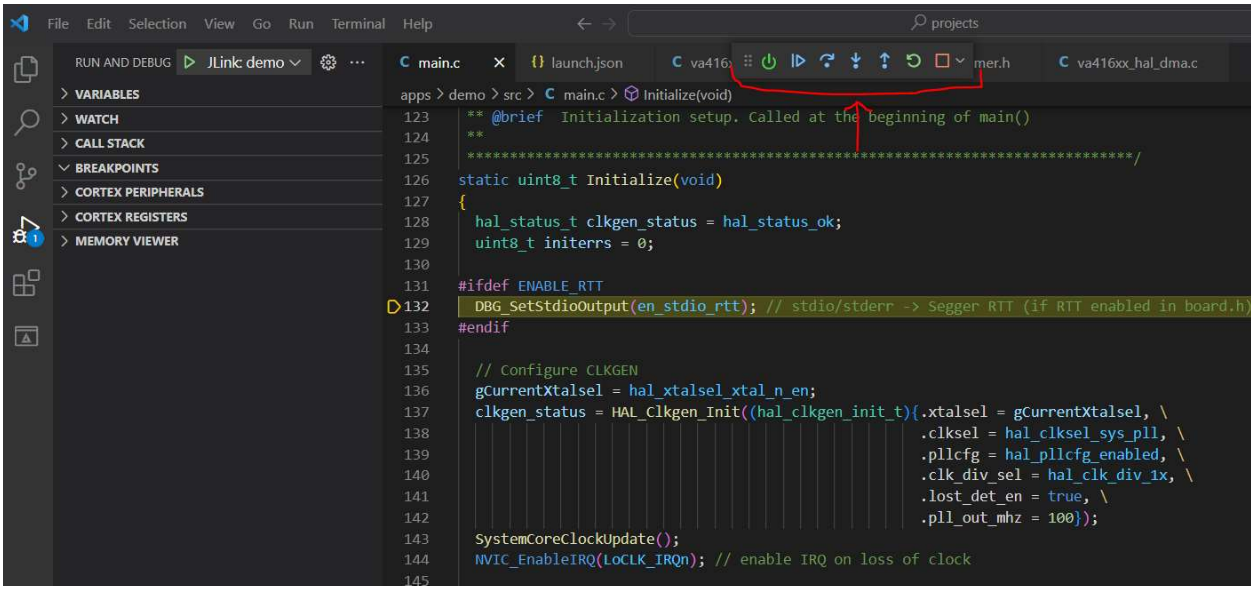


With the PEB1 board plugged into the PC with the micro USB cable, click the green triangle to begin a debug session.



2.5.5 Running, stopping, and stepping through code

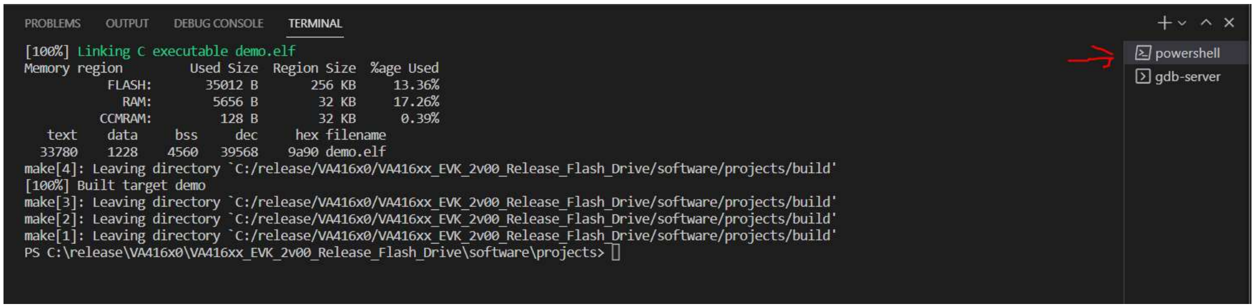
Once the connection to the board has been established, the code is downloaded to the MCU RAM. A breakpoint is set at the beginning of main(), and will be waiting to begin. In the upper middle of the window the debugger control buttons will appear, reset, run/stop, step over, step into, step out, restart,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



```
 ** @brief  Initialization setup. Called at the beginning of main()
 **
 ******************************************************************************/
static uint8_t Initialize(void)
{
  hal_status_t clkgen_status = hal_status_ok;
  uint8_t initerrs = 0;

#ifdef ENABLE_RTT
  DBG_SetStdioOutput(en_stdio_rtt); // stdio/stderr -> Segger RTT (if RTT enabled in board.h)
#endif

  // Configure CLKGEN
  gCurrentXtalsel = hal_xtalsel_xtal_n_en;
  clkgen_status = HAL_Clkgen_Init((hal_clkgen_init_t){.xtalsel = gCurrentXtalsel, \
                                                     .clksel = hal_clksel_sys_pll, \
                                                     .pllcfg = hal_pllcfg_enabled, \
                                                     .clk_div_sel = hal_clk_div_1x, \
                                                     .lost_det_en = true, \
                                                     .pll_out_mhz = 100});
  SystemCoreClockUpdate();
  NVIC_EnableIRQ(LoCLK_IRQn); // enable IRQ on loss of clock
```

2.5.6 Exiting debug mode

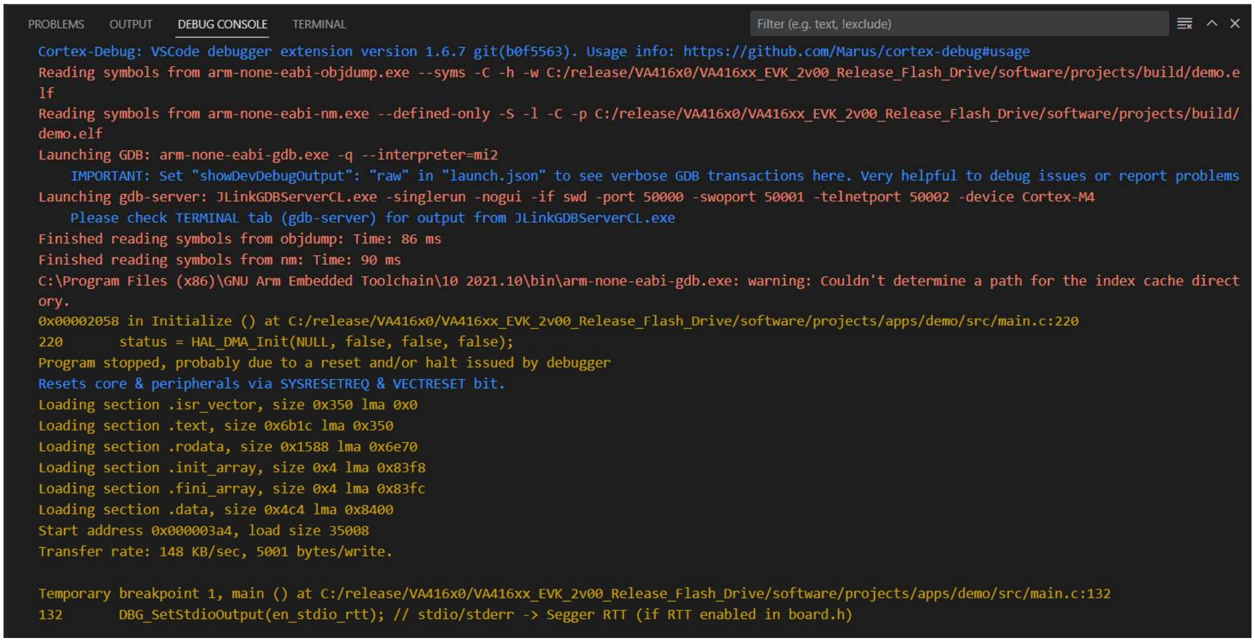
Press the orange square button to end a debug session. To get access to the terminal window again to run another 'make init' or 'make $project_name', either kill the gdb-server process, or click on the powershell process to bring it to the foreground.



```
[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> 
```

2.5.7 The debug console

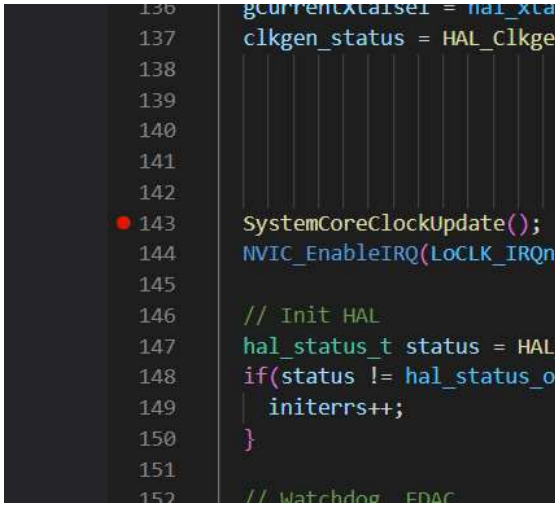
The debug console at the bottom of the VS Code window will show the debugger status. If there is a problem with the connection to the board, information here should help track down the cause of the issue.



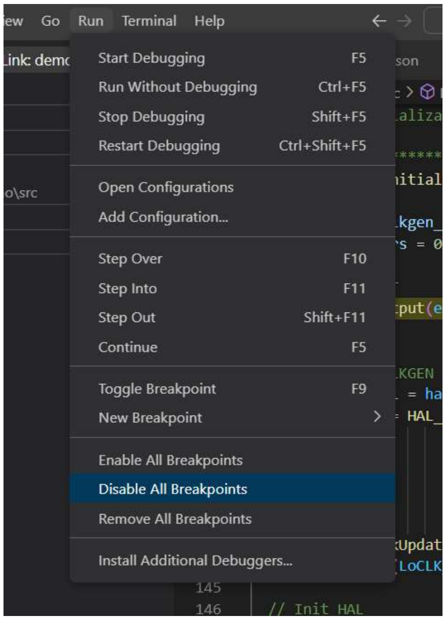
```
Cortex-Debug: VSCode debugger extension version 1.6.7 git(b0f5563). Usage info: https://github.com/Marus/cortex-debug#usage
Reading symbols from arm-none-eabi-objdump.exe --syms -C -h -w C:/release/VA416x0/VA416xx_EVK_2v00_Release_Flash_Drive/software/projects/build/demo.e
lf
Reading symbols from 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)
```

2.5.8 Setting and removing breakpoints

To set a breakpoint, click just to the left of the line number. Breakpoints can be removed by clicking the Run dropdown menu, then selecting Disable All Breakpoints.

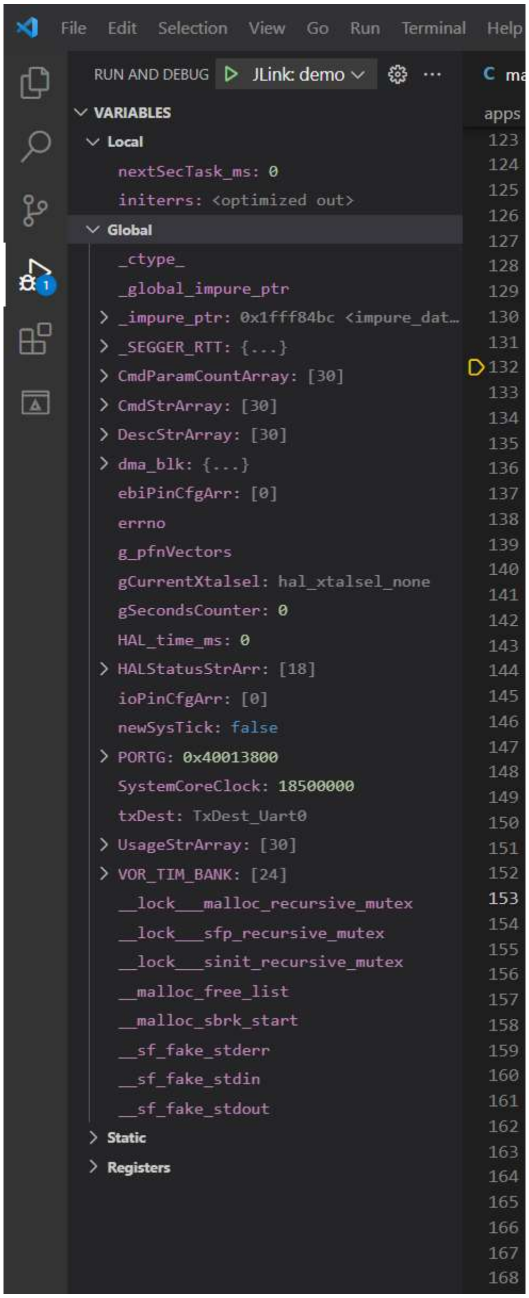


```
gCurrentXtalsel = hal_xta
clkgen_status = HAL_Clkge





SystemCoreClockUpdate();
NVIC_EnableIRQ(LoCLK_IRQn

// Init HAL
hal_status_t status = HAL
if(status != hal_status_o
  initerrs++;
}

// Watchdog  EDAC
```

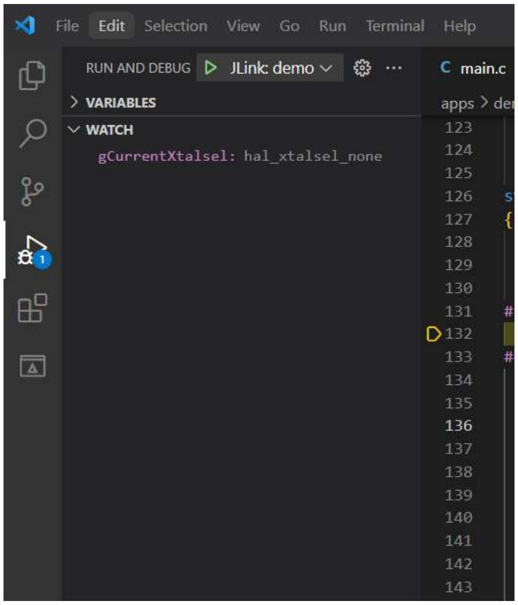


2.5.9 Viewing / watching variables

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.



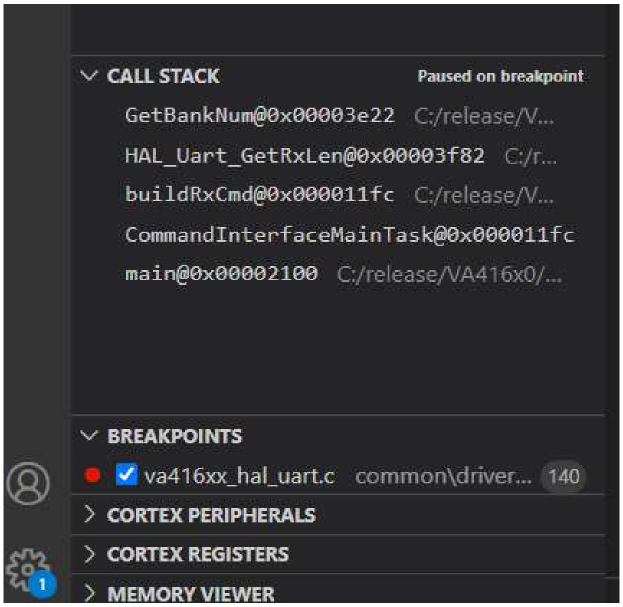
```
VARIABLES
  Local
    nextSecTask_ms: 0
    initerrs: <optimized out>
  Global
    _ctype_
    _global_impure_ptr
  > _impure_ptr: 0x1fff84bc <impure_dat…
  > _SEGGER_RTT: {...}
  > CmdParamCountArray: [30]
  > CmdStrArray: [30]
  > DescStrArray: [30]
  > dma_blk: {...}
    ebiPinCfgArr: [0]
    errno
    g_pfnVectors
    gCurrentXtalsel: hal_xtalsel_none
    gSecondsCounter: 0
    HAL_time_ms: 0
  > HALStatusStrArr: [18]
    ioPinCfgArr: [0]
    newSysTick: false
  > PORTG: 0x40013800
    SystemCoreClock: 18500000
    txDest: TxDest_Uart0
  > UsageStrArray: [30]
  > VOR_TIM_BANK: [24]
    __lock___malloc_recursive_mutex
    __lock___sfp_recursive_mutex
    __lock___sinit_recursive_mutex
    __malloc_free_list
    __malloc_sbrk_start
    __sf_fake_stderr
    __sf_fake_stdin
    __sf_fake_stdout
> Static
> Registers
```



```
> VARIABLES
v WATCH
    gCurrentXtalsel: hal_xtalsel_none
```

2.5.10 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.

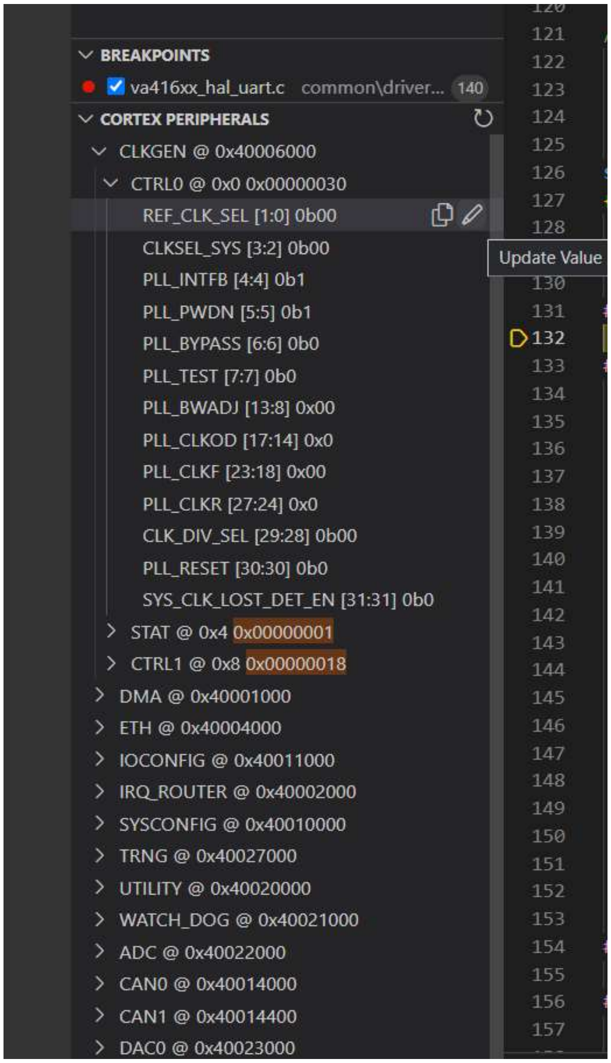


```
GetBankNum@0x00003e22   C:/release/V...
HAL_Uart_GetRxLen@0x00003f82   C:/r...
buildRxCmd@0x000011fc   C:/release/V...
CommandInterfaceMainTask@0x000011fc
main@0x00002100   C:/release/VA416x0/...
```

● ☑ va416xx_hal_uart.c   common\driver...   140

2.5.11 Viewing and editing MCU peripheral registers

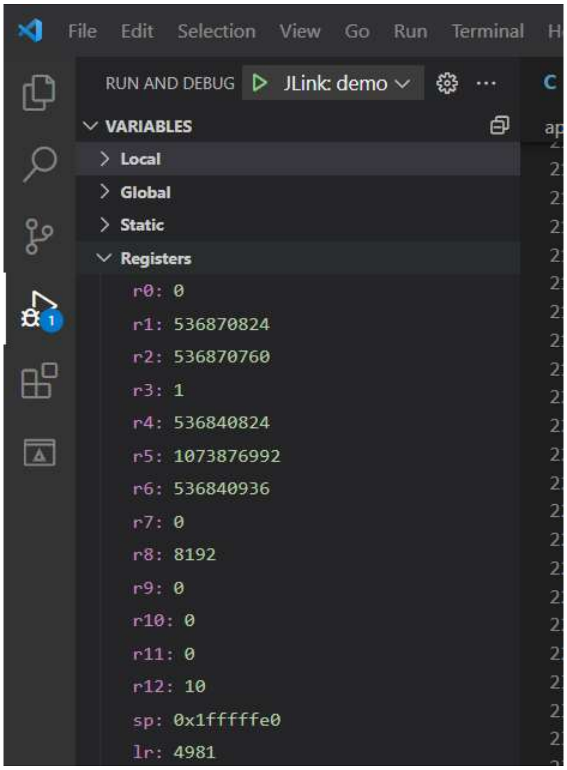
The Cortex Peripherals tab is used to view and modify peripheral registers. When mousing over a bit field or register, 2 buttons will appear to either copy the value, or edit it.



```
BREAKPOINTS
● ☑ va416xx_hal_uart.c  common\driver... 140
CORTEX PERIPHERALS
  CLKGEN @ 0x40006000
    CTRL0 @ 0x0 0x00000030
      REF_CLK_SEL [1:0] 0b00
      CLKSEL_SYS [3:2] 0b00
      PLL_INTFB [4:4] 0b1
      PLL_PWDN [5:5] 0b1
      PLL_BYPASS [6:6] 0b0
      PLL_TEST [7:7] 0b0
      PLL_BWADJ [13:8] 0x00
      PLL_CLKOD [17:14] 0x0
      PLL_CLKF [23:18] 0x00
      PLL_CLKR [27:24] 0x0
      CLK_DIV_SEL [29:28] 0b00
      PLL_RESET [30:30] 0b0
      SYS_CLK_LOST_DET_EN [31:31] 0b0
    STAT @ 0x4 0x00000001
    CTRL1 @ 0x8 0x00000018
  DMA @ 0x40001000
  ETH @ 0x40004000
  IOCONFIG @ 0x40011000
  IRQ_ROUTER @ 0x40002000
  SYSCONFIG @ 0x40010000
  TRNG @ 0x40027000
  UTILITY @ 0x40020000
  WATCH_DOG @ 0x40021000
  ADC @ 0x40022000
  CAN0 @ 0x40014000
  CAN1 @ 0x40014400
  DAC0 @ 0x40023000
```

2.5.12 Viewing and editing core registers

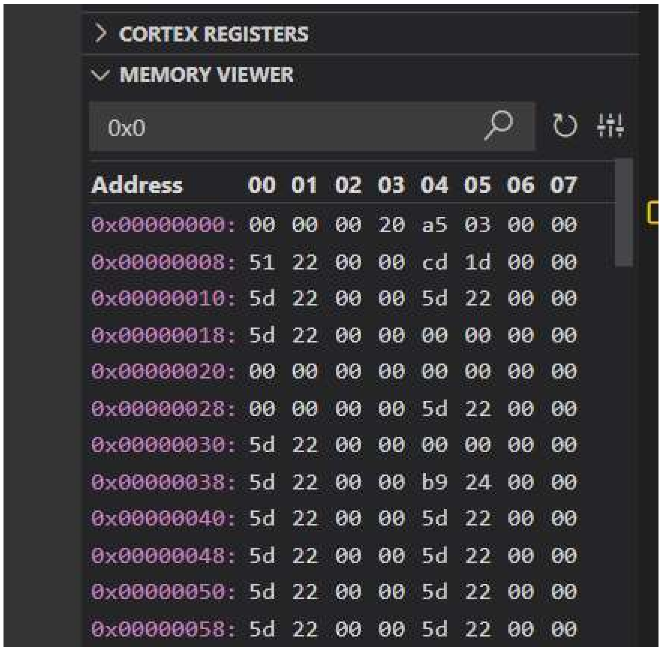
Core registers can be viewed through either the Cortex Registers tab, or through the Registers dropdown in the Variables tab. Core register values can only be modified through the Variables tab, however.



```
r0: 0
r1: 536870824
r2: 536870760
r3: 1
r4: 536840824
r5: 1073876992
r6: 536840936
r7: 0
r8: 8192
r9: 0
r10: 0
r11: 0
r12: 10
sp: 0x1fffffe0
lr: 4981
```

2.5.13 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.



```
Address     00 01 02 03 04 05 06 07
0x00000000: 00 00 00 20 a5 03 00 00
0x00000008: 51 22 00 00 cd 1d 00 00
0x00000010: 5d 22 00 00 5d 22 00 00
0x00000018: 5d 22 00 00 00 00 00 00
0x00000020: 00 00 00 00 00 00 00 00
0x00000028: 00 00 00 00 5d 22 00 00
0x00000030: 5d 22 00 00 00 00 00 00
0x00000038: 5d 22 00 00 b9 24 00 00
0x00000040: 5d 22 00 00 5d 22 00 00
0x00000048: 5d 22 00 00 5d 22 00 00
0x00000050: 5d 22 00 00 5d 22 00 00
0x00000058: 5d 22 00 00 5d 22 00 00
```

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.