VORAGO VA108x0 GCC IDE application note

April 2, 2019 Version 1.1

VA10800/VA10820

Abstract

ARM has provided support for the GCC (GNU C compiler) and GDB (GNU DeBug) tools such that it is now a very reliable and often used development environment for ARM Cortex-M based MCUs. This application note covers all the steps necessary to get the suite running specifically with the VA108x0 MCU from VORAGO. This includes:

- Instructions on how to install the Eclipse+JRE (Java runtime environment), GCC and J-Link GDB server packages,
- Information on unique linker (*.LD), make (Makefile) and startup (*.S) files
- Example project demonstrations

During the development, several challenges were encountered with download, options and tool nuances. These are addressed in the frequently asked questions. It is highly recommended to review this list if you have plans to “open-the-hood” and modify linkers, make files, startup or other components in the environment.

Table of Contents

1 Overview of GCC and GDB development environment

Eclipse is an integrated development environment (IDE) that has special support for C programming. The compiler and debugger interface are specified inside the project. Eclipse

plug-ins are used to access some of the debugger functions. Figure 1 shows a high-level interconnect diagram of all the pieces used. The following sections provide more depth into each block.

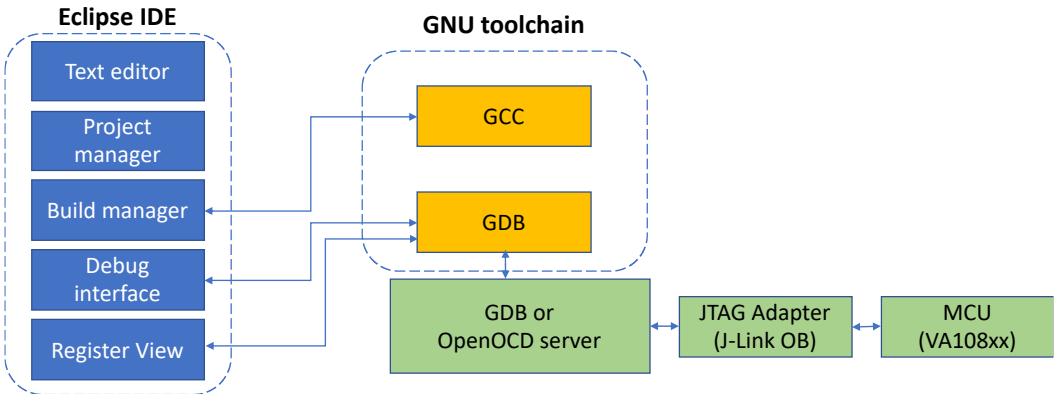


Figure 1- Development environment interconnect diagram

1.1 GCC

The GNU Compiler Collection includes front ends for C, C++, Objective-C, Fortran, Ada, and Go, as well as libraries for these languages (libstdc++,...). The home URL is:

https://gcc.gnu.org

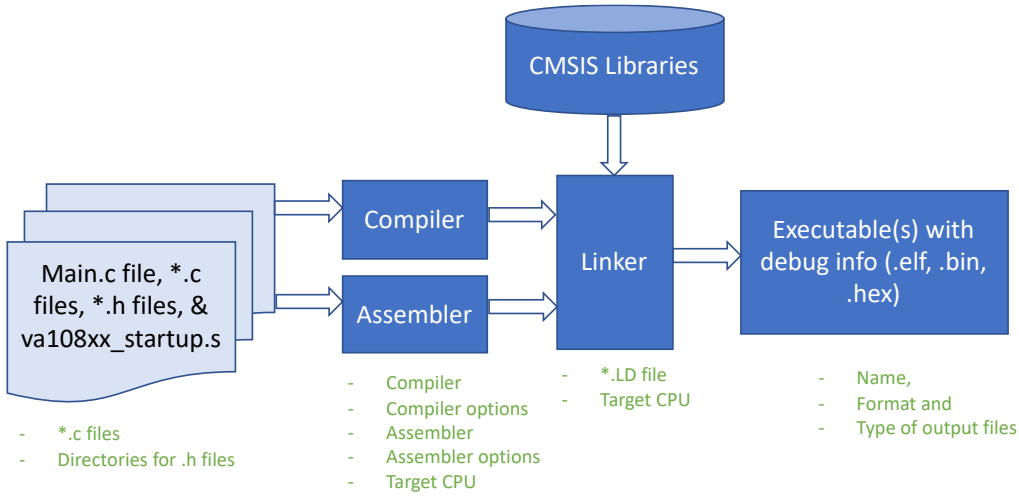
There is extensive documentation available for GCC on-line at: https://gcc.gnu.org/onlinedocs/. This information can sometimes come in very handy but by following the steps listed in this application note, the need to read the GCC documentation should be limited.

ARM has invested time and effort in improving the compiler specific to the ARM architectures. The ARM version of GCC has been proven on many MCUs and improved over the last 15 years. Information on this development and access to the source code can be found at:

https://developer.arm.com/open-source/gnu-toolchain/gnu-rm.

This application note will use “arm-none-eabi-gcc”. This is an embedded application binary interface (eabi) for the “arm” instruction set with no (“none”) IC vendor specific information. The project specific linker and “Makefile” files specify which ARM core is used.

Figure 2 provides a summary of the flow and processing of information to create an executable file that the debugger can utilize. ARM has created a software interface standard known as “CMSIS”, Cortex-M Software Interface Standard. This has multiple layers and a wide breadth but in general, it offers an API to lower level drivers for peripherals like UART, SPI and I2C. This is intended to make code very portable from one ARM based processor to another.



"Makefile" (A text file inside the project) calls out all of the above information in green.
*The Linker uses a *.LD file to receive: the ARM core, the memory map placement for instructions/data and size of stack.*

Figure 2 - Summary of information processing to create an executable file

1.2 GDB - GNU Project Debugger

GDB is intended to provide a powerful user interface for debugging software on the target MCU. It includes the following capabilities:

- Download executable files to either RAM or NVM on the target MCU
- Start program from any program counter location
- Stop program execution by the use of break points or when a STOP instruction is executed
- Examine and modify memory mapped register and memory contents
- Examine and modify CPU registers (not memory mapped)

More information can be found at: https://www.gnu.org/software/gdb/

In order to support different JTAG / SWI probes, a GDB client server concept has been implemented. When the "debug" command is entered in Eclipse, the GDB client is launched. For our case, the client will be "arm-none-eabi-gdb". A port number is specified in the debug configuration Debugger menu. A PC will direct the communication traffic to a separate task controlling the probe. For the Segger J-Link, the port ID is 2331. The OpenOCD port ID is 3333. The GDB server can be started via a script from Eclipse or separately prior to entering the debug command.

As a side note, the fish sketch often accompanying GDB information is in remembrance of Fred Fish (November 4, 1952 – April 20, 2007) and his significant contributions to the GDB effort.

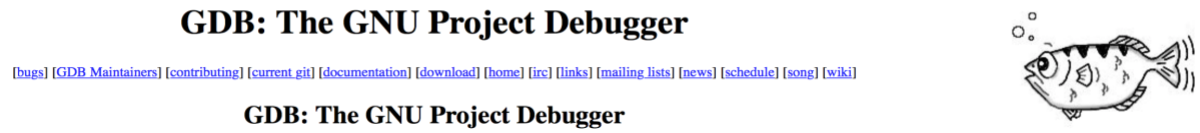


Figure 3 – GDB home page with tribute to Fred Fish.

2 Installing GCC and conducting a test run

ARM has taken the basic GCC suite and tailored it to the ARM architectures. Access to the latest version can be found at:

https://developer.arm.com/open-source/gnu-toolchain/gnu-rm/downloads

The name of the .exe file should look something similar to "gcc-arm-none-eabi-X_X-XXXXX... .exe". See below for screen capture of the download site. Newer versions may be available when you visit the site.

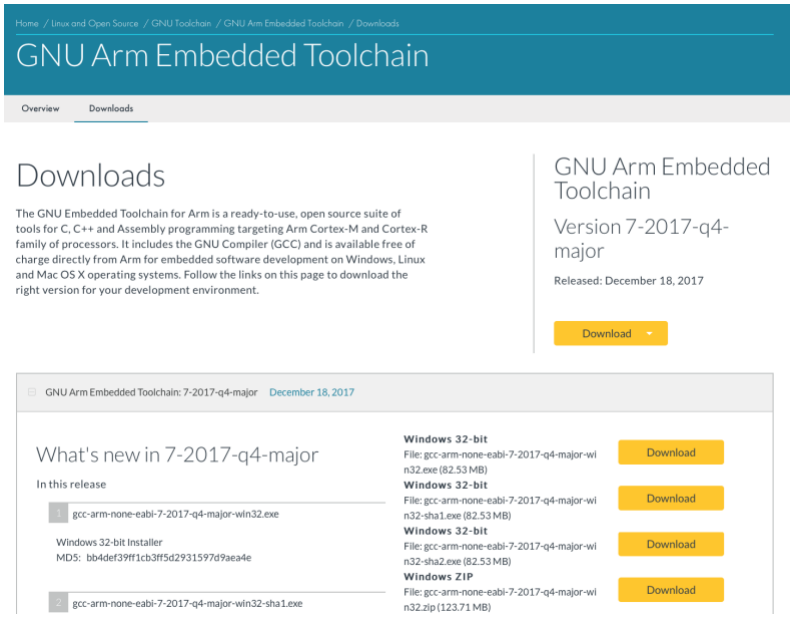


Figure 4 - ARM GNU download site screen capture

Double click on the *.exe file and click through the install instructions. Make certain that you click the “Add path to environment variable”. In order for Eclipse to access the GCC software, the global path must be changed to include the folder containing GCC executables.

2.1 GCC test run

Open a command prompt (CMD) window and enter the following command to show the GCC version information and confirm it was installed: “arm-none-eabi-gcc -v”. This command will show the version information. The below figure shows an example screen capture.

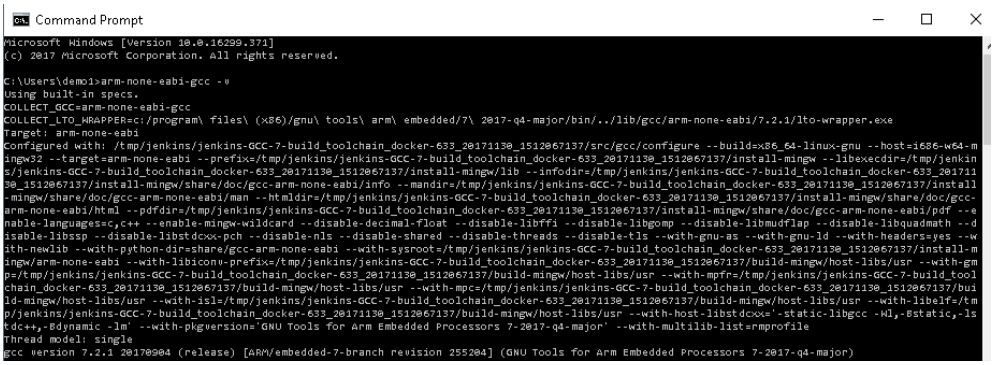


Figure 5 - Command prompt invoking GCC and showing the version information.

The GDB client software was also installed with the GCC download. You can confirm this by entering: “arm-none-eabi-gdb -v” at a command prompt.

3 Installing Eclipse CDT

Eclipse is a free IDE (integrated debug environment) that supports several different programming languages. For our purpose, we will focus on the CDT (C/C++ Development tooling) version. Please visit the below URL for access to the latest version.

https://www.eclipse.org/downloads/

As of April 2, 2019, the download filename was: eclipse-inst-win64.exe.

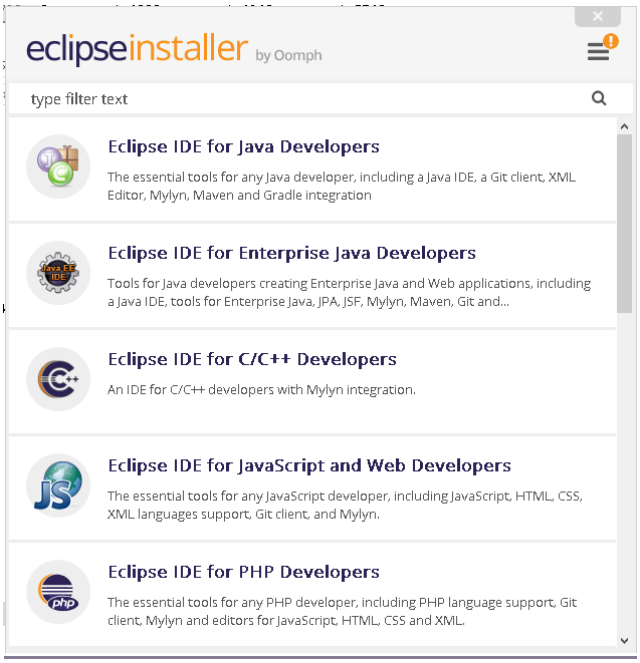


Figure 6 – Eclipse install screen capture

Download Eclipse for your platform. Run the executable file and select "Eclipse IDE for C/C++ Developers" as shown in the Figure 6. You will want to remember the folder location where you extracted the files. We used Windows 64-bit and placed them in a folder called "c:/EclipseCDT" but any folder name should work.

Eclipse requires Java runtime environment, so you will have to also download JRE from http://java.com. After downloading the "jre-8u171-windows-x64.exe" run the program and click the "change destination" box. Before starting the install, create a folder "jre" in the "eclipse" folder created in the above paragraph. Change the install location to the "jre" folder just created.

Alternatively, it is possible to modify the eclipse.ini file to point to the JRE path location. In that case, use the default location of c:\ProgramFiles\Java\jre1.8.0_171.

Detailed information on CDT can be found at:
https://wiki.eclipse.org/CDT/User/NewIn90#Release

4 Installing Segger J-LINK GDB server

Since the REB-1 development board comes with J-LINK OB, we are using the J-LINK GDB server to route information from the GDB tool to the MCU. The JLINK GDB server is part of the larger J-LINK download available at: https://www.segger.com/jlink-software.html

Select Software and documentation pack for Windows V6.10n [23,162 KB]. If a later version is available, please use it.

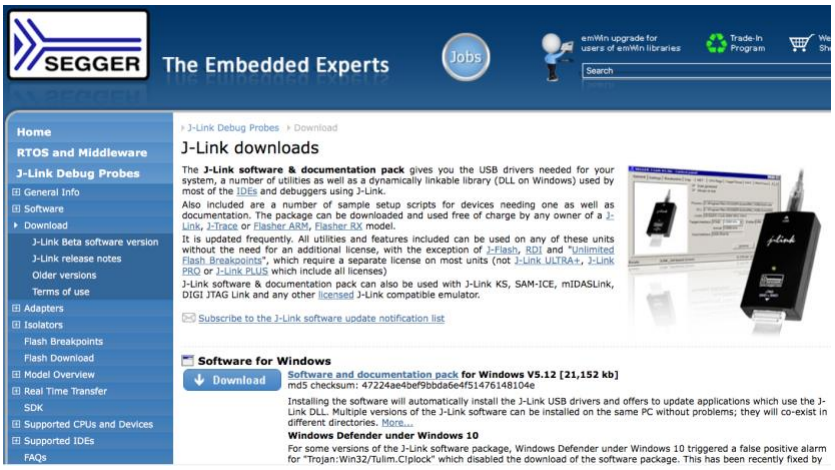


Figure 7 - Segger J-Link download site screen capture

5 Downloading an Example VA108x0 project

Two projects are available on GitHub: Blink_only and the full REB1 BSP. They can be found at: https://github.com/Voragotech

Download both projects and put them in the same directory. For our example, we used: c://Vorago_projects. The next section will access these projects through Eclipse.

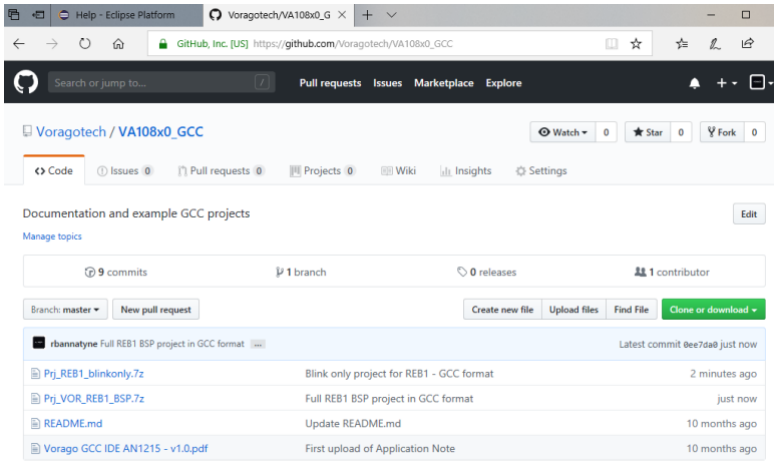


Figure 8 - VORAGO github page

6 Introduction to Eclipse

The organization of a project is somewhat unique in Eclipse and deserves a quick explanation. Eclipse uses a “workspace” concept which groups together the following in a folder:

1. Some configuration pertaining to all these projects in metadata
2. Some settings for Eclipse itself
3. Optionally one or more related projects

This happens by creating a directory/folder and storing files in it that provides Eclipse this information when the workspace is opened or “switched” to. See below for the contents of an example Workspace folder.

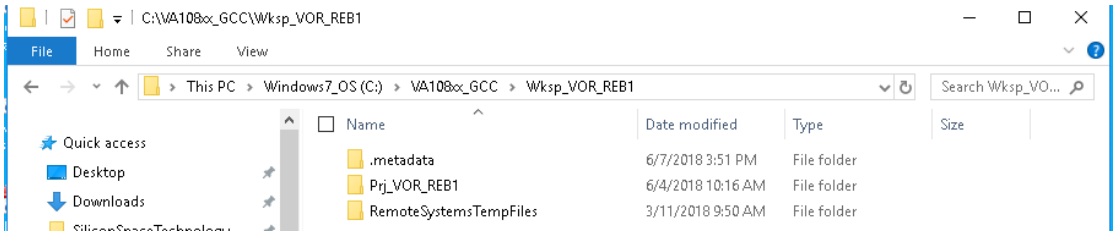


Figure 9 - Contents of Workspace directory

Copying a workspace from one machine to another can be dangerous as the metadata can reference information that is not available on the second machine. Instead it is recommended that a new workspace be created and then reference the project.

Eclipse makes it easy to create a new workspace and to import an existing project. With Eclipsed already launched, Use the “File” pull down menu to select “Switch Workspace”. You can either point to an established workspace or to an empty folder.

With Eclipse already launched and a Workspace active, pull down the following menu: File -> New -> Makefile Project with Existing Code. You should get a screen similar to Figure 10 below. Unclick the C++ box and select “Cross GCC”.

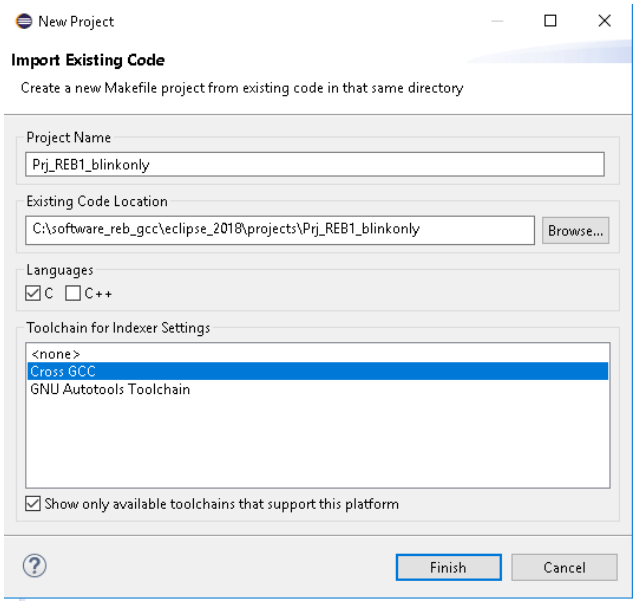


Figure 10 - Eclipse New -> Makefile Project with Existing Code screen capture

The workspace is now associated with the referenced project and you can start your code development from a solid starting point.

6.1 Eclipse first time use

Open Eclipse by double clicking on the eclipse.exe file that was downloaded per instructions in Chapter 4. After a few seconds a menu will appear asking for a workspace. Point to ../VA108X0_GCC_project/Workspace-VORAGO-REB1-blinkonly folder downloaded in section 5. Note that once Eclipse has been used, it will remember the last used Workspace and will go directly to it. If a prompt for a Workspace is not shown, use the “file” pulldown menu and select “switch workspace”, then proceed to the above-mentioned workspace.

You should see something similar to the below screen capture.

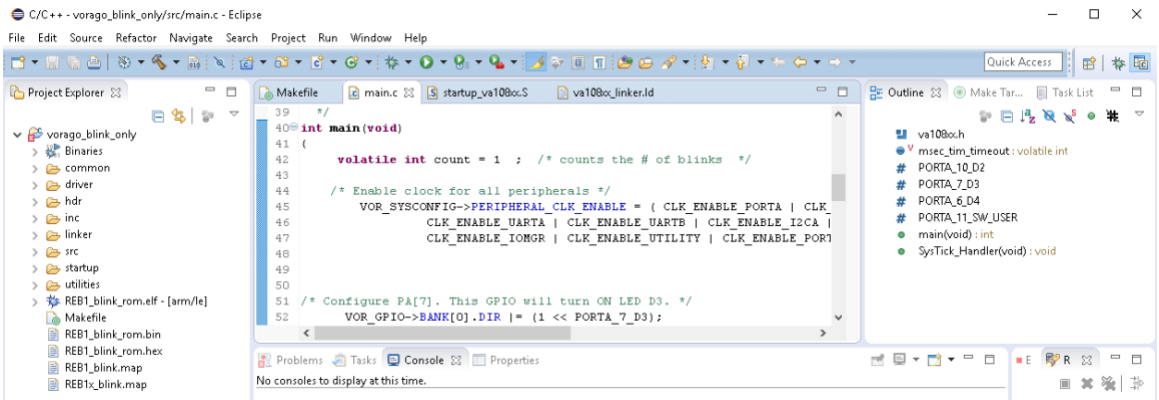


Figure 11 - Blink_only project screen capture

6.2 Updates and Installing New Software

The native download of Eclipse does not support the “GDB Hardware Debugger” Configuration and an extra step is necessary to enable GDB use.

To make certain Eclipse is using the latest tools for C development it is recommended that you pull in the latest updates. This is done using the following pull-down menu: Help -> Check for update.

For some versions of Eclipse such as Mars, it might be necessary to use the install new software option instead of the update option. In that case, install updates for CDT. This will enable the creation of new GDB debug configurations.

7 Unique files for VA108x0 – make, startup and linker files

Inside each project are three text files that determines which files are included in the project, how the MCU boots and which memory space the program should use. Each file can be viewed and modified in the eclipse environment using the project explorer. The following three sections describe these files.

7.1 Calling out project options in “Makefile”

The “makefile” is a text file normally >200 lines long that contains the files and options needed to build a project. Included in this list are:

- Compiler to be used
- Target device CPU information. i.e. cortex-m0.
- Linker file to be used
- List of *.c files to include
- Directories to find the *.h files.
- Output file name and list of formats i.e. .hex, .bin, .elf.

We have provided a makefile for the two example projects. It resides in the root directory of the project. If you decide to add source files or add directories for .h files, you will probably need to modify this file.

7.2 Defining reset / interrupt vectors and initializing MCU before main{} is called – “startup_VA108x0.S”

Explicit definitions for the reset and interrupt vectors are called out in this file. The Cortex-M0 has 16 core related vectors and 32 User interrupt vectors. IRQ[0:31]. VORAGO has designated the first 12 interrupt vectors to be specific to a specific peripheral on the device

such as SPI0. This is not a convention that must be followed but for ease in understanding these were named with the peripheral assigned to the vector.

The reset handler and the default interrupt handler are also included in the startup_VA108x0.s file. This is a text file with assembly language instructions and directives. Prior to the main.c code executing structure information is moved from ROM space to RAM space. On some MCUs it is necessary to have some parameters setup and a system_init routine called. Commonly the clock must be configured as the first step. The VA108x0 device has a straightforward clock system that does not need to be setup and the reset handler moves a section of ROM data to RAM, initializes the first 4 registers then jumps to the main routine.

7.3 VA108x0_linker.ld

The linker file provides information that allows the build process to attach object file information to specific addresses in the MCU. The linker file is a text file that can be easily modified. It contains directives for certain pieces of information to be grouped together such as data and text. The linker file has labels that are used by the startup.s file to move information from ROM to RAM.

For first time GCC users, it is recommended that the linker file be left untouched. After the environment is better understood, changes to the linker file can be attempted.

8 Build and debug first project

As with the time-honored tradition of many other MCU evaluation kits the first project will be a simple LED blink routine. The project is located in the folder of the downloaded Prj_REB1_blinkonly.7z file. Follow the steps in the next three sections to launch, build and execute the project.

8.1 Start Eclipse and load the project

Find the directory with Eclipse and double click on the eclipse.exe file. After a few moments, Eclipse should open and ask for a “workspace”. Using Windows File Explorer create an empty folder and title it “Wksp_VOR_Blink” or any similar title. Navigate to the directory and hit the OK button. At this point, the project files are not associated with the workspace. Under the eclipse “file” menu, select “new” and then “Makefile project with Existing Code”. Set the existing code location to point to the directory with material from the download in section 5, “Prj_REB1_blinkonly”. Unclick the “C++” box and select “Cross GCC” as the toolchain. Then hit finish.

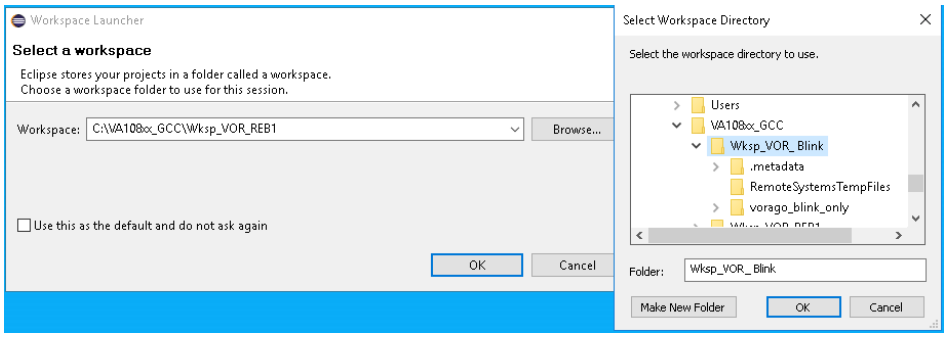


Figure 12 - Eclipse Workspace Launch screen capture

8.2 Build the project

The “blink_only” project has a build configuration file. It is possible to either use the pull-down menu (Project -> Build All) or click on the build icon (Hammer shape) as shown below. Different versions of Eclipse have slightly different appearance and this capture may not be identical to yours.

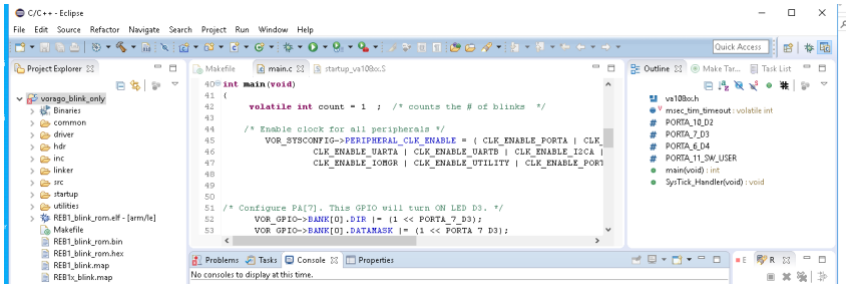


Figure 13 - Blink_only project "C" window capture

8.3 Download and run the program

The “blink_only” project may not have debug configuration file. It takes three quick steps to setup a configuration as shown in the following three figures. Start by selecting “Debug Configuration” under the “Run” menu. In the left-hand menu select “GDB Hardware Debug” and then the “New” icon. Follow the instructions in the next three sections.

8.3.1 Select Project and Application.

Hit the “Browse” button to the right of the project field. It should show the active project immediately. Then hit the “Search Project” button by the application field.

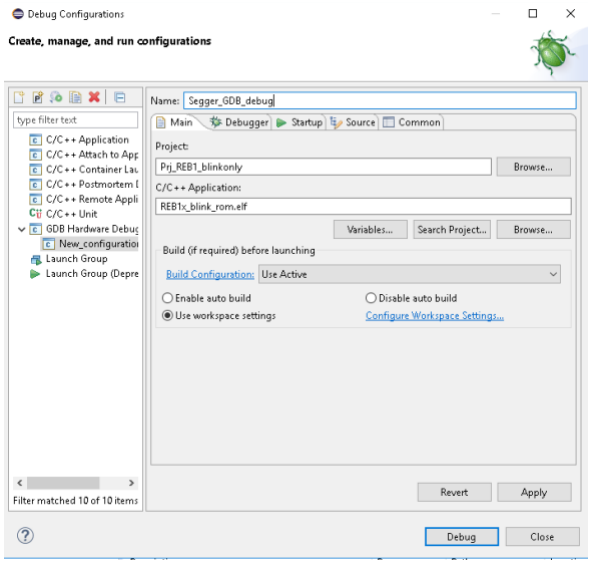


Figure 14 - Screen 1 of debugger setup

8.3.2 Set hardware

Click the “Debugger” tab and enter information as shown. The “arm-none-eabi-gdb” will invoke the ARM specific GDB interface software. Port 2331 is the Segger GDB server.

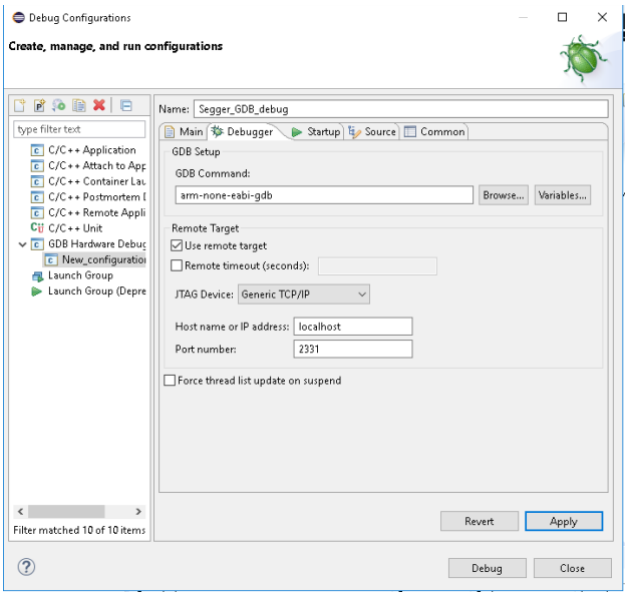


Figure 15 - Screen 2 of debugger setup

8.3.3 Set Startup parameters

Later versions of the eclipse IDE do not support the Reset & Delay button. It is necessary to manually enter the two initialization commands for the Segger GDB server. If this is not done, the debugger will attempt to run code without resetting the MCU. This can lead to unpredicted

results with peripherals still running from the last processor run. Hit the “Apply” button to save this configuration.

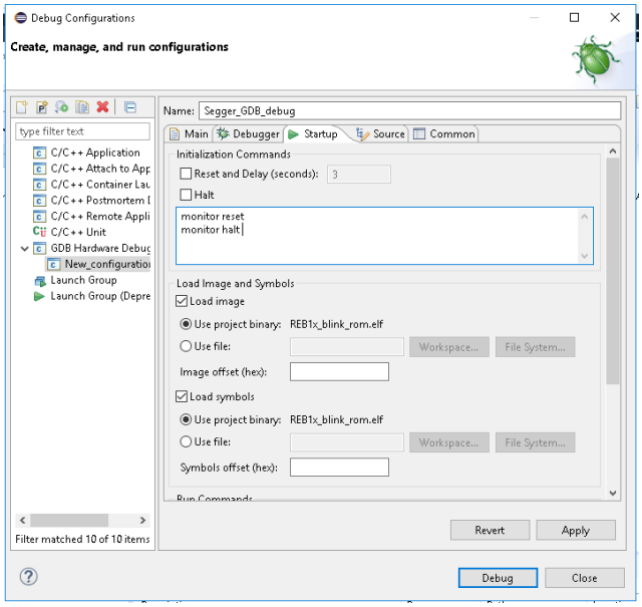


Figure 16 - Screen 3 of debugger setup

Before invoking the debugger (by hitting the “Debug” button) , please have the REB1 board attached to the PC USB port and start the Segger JLink GDB server by navigating to the Windows Explorer -> Segger ->J-Link GDB Server V6.30i. Enter “Cortex-M0” if prompted for a target device and hit the OK button.

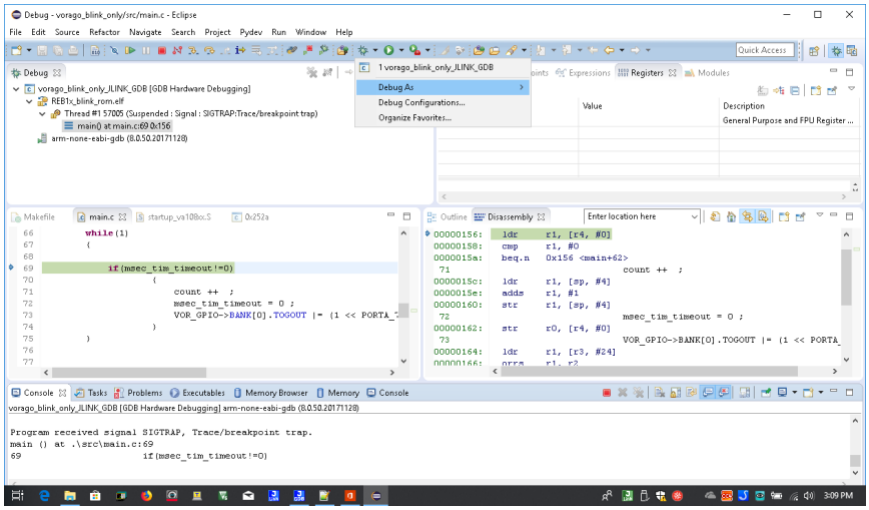


Figure 17 - Invoking the debugger screen capture

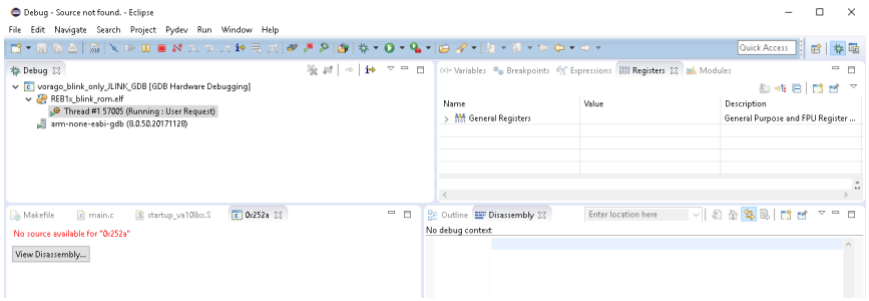


Figure 18 - Debug screen capture for Blink_only project

Start the program execution by hitting the “resume” button as shown in the below figure. Alternatively you can hit “F8” or use the “Run” pull-down menu.

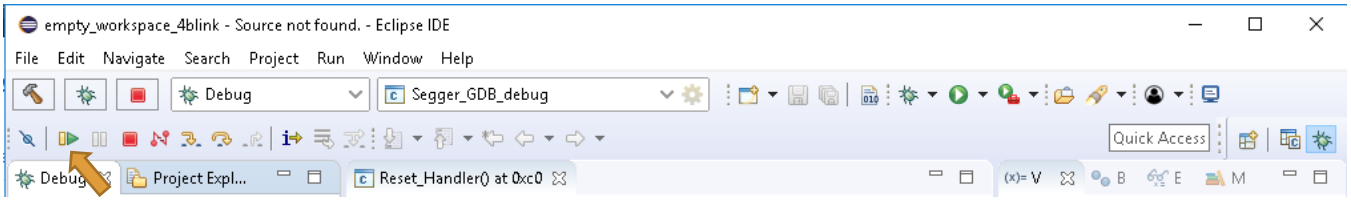


Figure 19 - Screen capture showing "Resume" icon to start code running

9 Build and debug the full REB1 BSP project

9.1 Start Eclipse and load the project

Find the directory with Eclipse and double click on the eclipse.exe file. After a few moments, Eclipse should open and ask for a “workspace”. Using Windows File Explorer create an empty folder and title it “Wksp_REB1_BSP” or any similar title. Navigate to the directory and hit the OK button. At this point, the project files are not associated with the workspace. Under the eclipse “file” menu, select “new” and then “Makefile project with Existing Code”. Set the existing code location to point to the directory with material from the download in section 5, “Prj_VOR_REB1_BSP”. Unclick the “C++” box and select “Cross GCC” as the toolchain. Then hit finish.

If Eclipse was already started, use the “File -> Switch Workspace” menu instead of restarting Eclipse.

9.2 Build the project

The “Prj_VOR_REB1_BSP” project has a build configuration file, “default”. This file calls out the GCC compiler and has library include paths. It is possible to either use the pull-down menu (Project -> Build All) or click on the build (hammer) icon to start the build process.

9.3 Download and run the program

The “Prj_VOR_REB1_BSP” project may not have a debug configuration file called “Prj_VOR_REB1”. If not, please follow identical instructions as called out in section 8.3. This file calls out which file to download, the debugger hardware and startup options. Before launching the debugger, please have the REB1 board attached to the PC USB port and start the Segger JLink GDB server by navigating to the Windows Explorer -> Segger ->J-Link GDB Server V6.30i. Enter Cortex-M0 is prompted for a target device and hit the OK button.

10 Creating your own project

Instead of starting from scratch to create your project, it is recommended to start from the REB1 project. One way of doing this is to copy the entire folder Prj_VOR_REB_BSP and renaming it. This provides a working file structure to start from.

11 Installing OpenOCD (On-chip Debugger)

Note: If you are not using an OpenOCD based setup, you can skip this section.

OpenOCD is standard to provide common debug and programming commands through an array of probes. OpenOCD can create a GDB server with port ID = 3333.

There is an organization that supports OpenOCD with a URL: http://openocd.org. The download location for the server software is: https://sourceforge.net/projects/openocd/files/

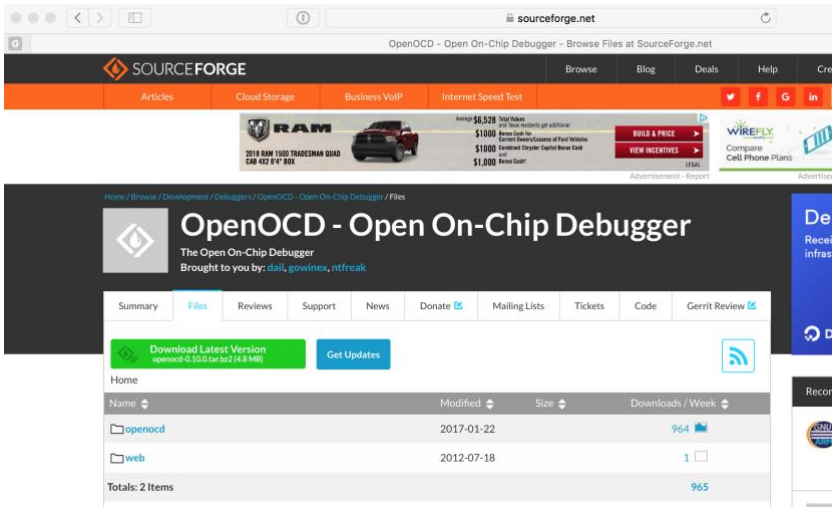


Figure 20 - OpenOCD download site screen capture

11.1 Instructions for using the OpenOCD

There is a plug-in for OpenOCD. See: https://gnu-mcu-eclipse.github.io/debug/openocd/. By following the steps, it is possible to create a debug configuration which will configure all the specifics of the probe, GDB server and OpenOCD.

11.1.1 Adding OpenOCD to path environment

When GCC was loaded, there was an option to have the Environment path updated. For OpenOCD, this must be done manually. In Windows 10, open a Windows Explorer screen and right click on “This PC”. Click on “Properties”.

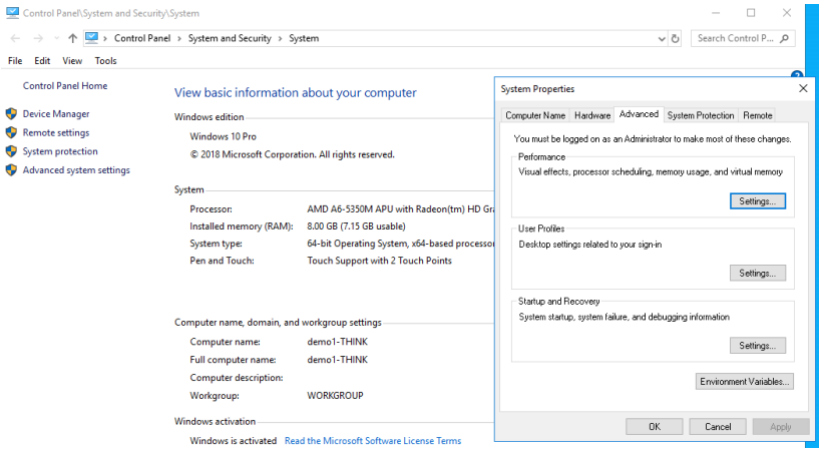


Figure 21 - System variable change navigation procedure

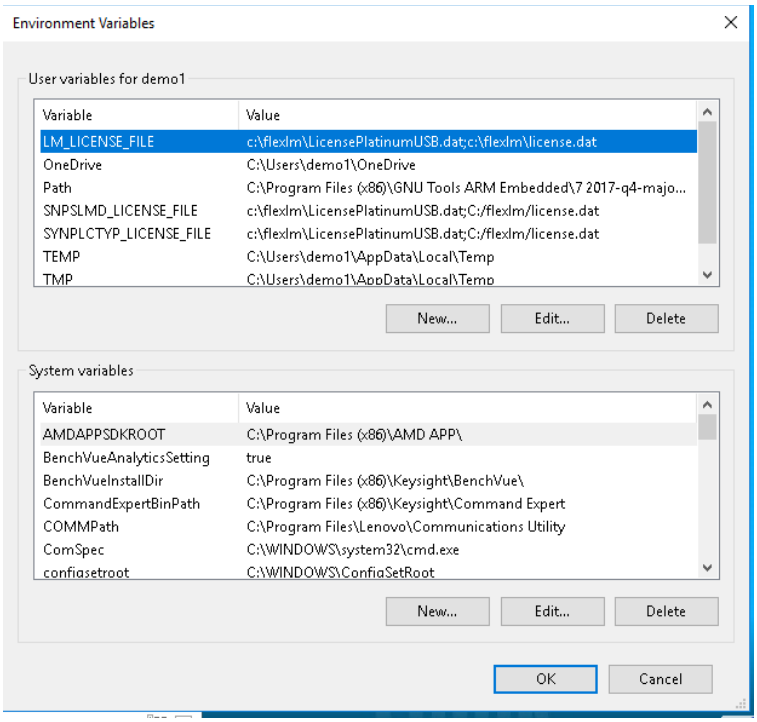


Figure 22 - System variable change -> path edit

Select the “Path” line and hit the “Edit” button. Add the OpenOCD binary directory which should look similar to ..\openocd-0.6.0\bin-x64”.

12 Alternative probe - Black Magic Probe (BMP)

There are several low-cost (<$100) JTAG probes that can be used with GDB. Information on one of the more popular ones, Black Magic Probe (BMP), can be found here: https://github.com/blacksphere/blackmagic/wiki.

A unique debug configuration (“vorago_blink_with_BMP”) has been made for the Black Magic probe for the “blink_only” file. Instead of using the “remote target” with port address, BMP uses a virtual communication port (USB CDC ACM). There is no need to start a GDB server prior to starting a debug section. The following three screen captures show the debugger setup menus.

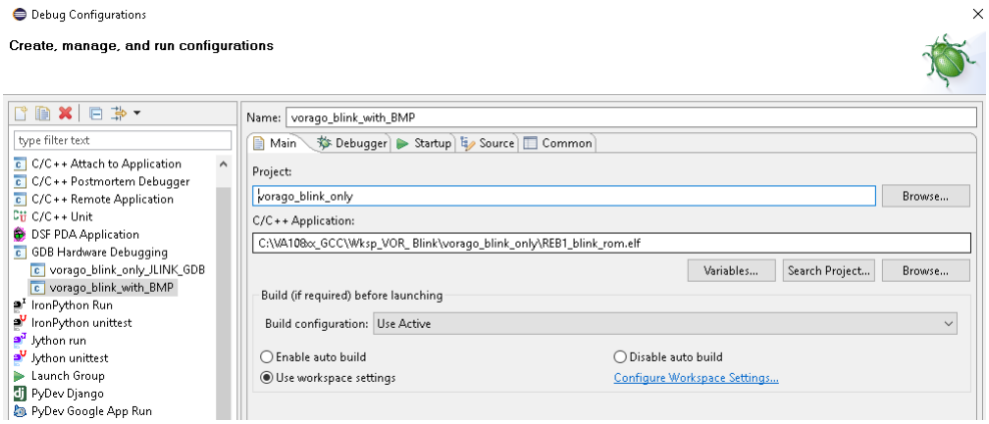


Figure 23 - Main debug configuration menu

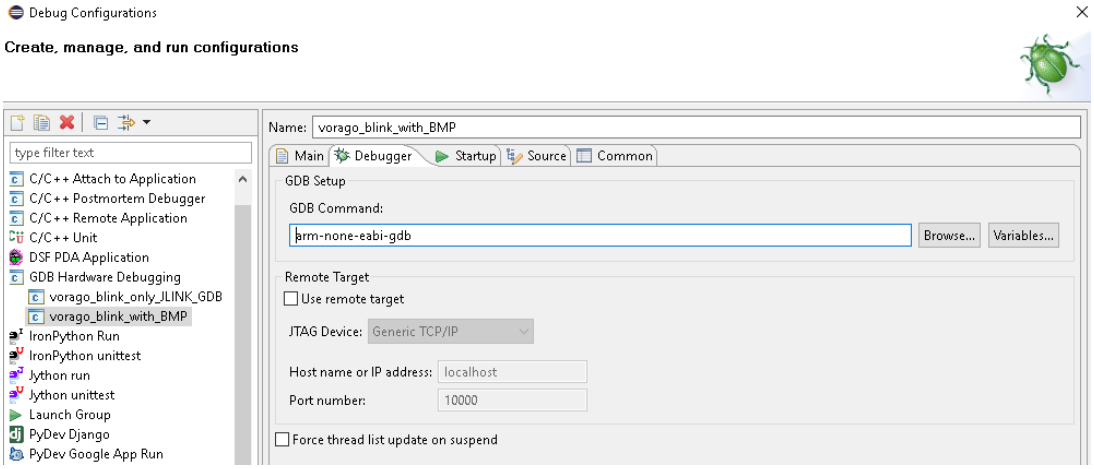


Figure 24 - Debugger debug configuration menu

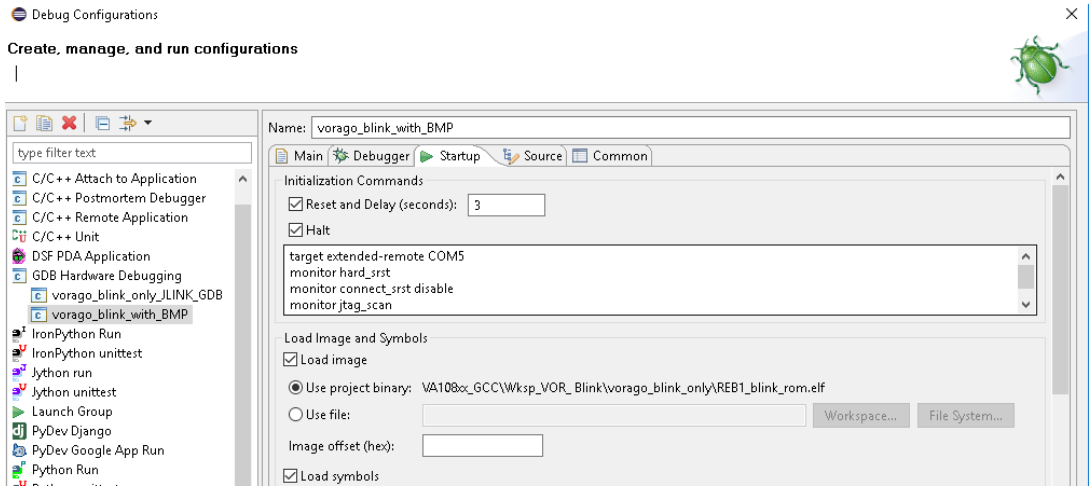


Figure 25 - Startup debug configuration menu

Need to add “connect 1” on line below “monitor jtag_scan”.

13 Programming SPI NVM for the VA108x0

There is currently there is no programming support for the GDB environment. We expect to offer this support in the future.

14 Conclusions

This application note has provided all the steps necessary to install and demonstrate GCC using the Eclipse IDE. Two examples were provided for simple and complex projects. You should be able to develop your own projects using the foundation provided.

15 Common questions and issues

1. In Eclipse what is the difference between a workspace and a project?

   1a: A workspace is the folder / directory containing information in metadata format for the entire project. This includes the path to the project files. A project is a collection of files including: *.c, *.h, startup, linker and make files. The project can be stored in the workspace directory or outside of it. Different companies like to organize their projects differently depending upon shared libraries and re-use policies. To reduce confusion, it is recommended that the folder name have useful information with either “workspace” or “project” being part of the name.

2. When starting a new project, should I import an older one or use “new - > makefile from existing“?

   2a. Starting new project from scratch can be a long and laborious task. Copying the entire workspace of the REB1 project and renaming it is a good way to start. Otherwise using the “new -> makefile from existing” option is the second option.

3. Inside the debug window, when and why use “relaunch”?

   3a. If debugging code included modifying memory or registers, it normally a good idea to reset the part and to reload the program periodically. The relaunch command will reset the MCU and download the active project to Instruction RAM on the VA108x0.

4. Can MCU peripheral registers be viewed by the debugger?

   4a - This feature has not been implemented at this time. VORAGO plans on adding this feature in the future.

5. Is it possible to program SPI memory from the debugger?

   5a - Ans: Yes this is possible but support for it is not available at this time. VORAGO plans on adding this feature in the near future.

6. The MCU will not respond to any interrupt requests like it had in previous debug sessions. What is going on?

   6a. The MCU can react to a hard fault problem in a way that is not recoverable until the unit is powered down. To check for this condition, view location 0xE000ED04. If the least significant 11 bits are 0x003, the device is in this condition. Most likely the cause of this is code trying to access invalid memory space. To correct the condition a POR reset is required.

7. I have modified the source code in the edit window, but the compiled output does not reflect the changes. Why is that?

   7a. Eclipse will always use the saved files and not the version you are viewing. Save the file and rebuild the project to remedy this issue.

16 Other Resources

VORAGO VA108x0 programmers guide & VORAGO MCU products:
http://www.voragotech.com/VORAGO-products

VORAGO Application notes: http://www.voragotech.com/resources

VORAGO VA108x0 REB1board user guide: Part of Board Support Package (BSP)
http://www.voragotech.com/products/reb1

MCU on Eclipse – Excellent reference for Eclipse https://mcuoneclipse.com/category/eclipse-2/

Revision log:

June 11, 2018 – Initial release

April 2, 2019 – Rev 1.1

- Eclipse support for Reset command to debugger not working. Need to add work around.
- Better explanation of workspace
- Minor corrections and improvements to diagrams.
- Screen captures of latest download pages.