

Configuring a zephyr eclipse project for VEGA

The following guide was written to give instructions on how to load an zephyr project for VEGAboard onto eclipse for debugging using the openOCD server and a JLink probe.

The links below are general instructions on how to debug on the VEGA board (without zephyr):

Debugging on VEGA board (Follows instructions from open-isa website)

<https://mcuoneclipse.com/2019/03/05/debugging-the-rv32m1-vega-risc-v-with-eclipse-and-mcuxpresso-ide/>

How debugging works. Gdb server and client working together. (first part of this article):

<https://mcuoneclipse.com/2013/07/22/diy-free-toolchain-for-kinetis-part-3-debugger-gdb-server-with-pe-and-segger/>

More information can be found on the eclipse guide on the zephyr documentation:

<https://docs.zephyrproject.org/latest/application/index.html#eclipse-debugging>, although this guide describes using a pyOCD debugger. The VEGA board uses OpenOCD. The guide written here pulls from that document above as well as the getting started page on open-isa.org: <https://open-isa.org/get-started/>

Enabling Debug optimizations:

Before building the project, I would recommend enabling debug compiler optimizations.

This can be done by editing the <project directory>\prj.conf file or the <project directory>\boards\rv32m1_vega_riscy.conf and adding the following line to it:

```
CONFIG_DEBUG_OPTIMIZATIONS=y
```

Otherwise, when debugging eclipse will jump lines and optimize out many variables, making it very inconvenient to debug.

There is no way to change debug optimizations after the project has been imported into eclipse.

Generating the eclipse project files:

From the %userprofile% directory, create a file named zephyrrc.cmd and make sure the following two lines of code are present:

```
set ZEPHYR_TOOLCHAIN_VARIANT=cross-compile
```

```
set CROSS_COMPILE=C:\riscv32-unknown-elf-gcc\bin\riscv32-unknown-elf-
```

The CROSS_COMPILE file above can be obtained from the toolchain downloaded from the downloads tab of the open-isa website.

Now, Open terminal/command prompt and go into the user profile directory.

Run the following commands:

```
Cd zephyrproject/zephyr
```

```
Zephyr-env.cmd (this will set the correct cross_compile path)
```

```
Cd %userprofile% or cd ../../
```

If a build directory exists here, delete it, or run *ninja pristine* from within the build directory

```
Mkdir build & cd build
```

```
Type NUL > empty.ld
```

Run Cmake to build the project:

```
cmake -G"Eclipse CDT4 - Ninja" -DBOARD=rv32m1_vega_riscy -DCMAKE_REQUIRED_FLAGS=-Wl,-dT=%cd%\empty.ld %ZEPHYR_BASE%<project_directory>
```

In this project, I will run:

```
cmake -G"Eclipse CDT4 - Ninja" -DBOARD=rv32m1_vega_riscy -DCMAKE_REQUIRED_FLAGS=-Wl,-dT=%cd%\empty.ld %ZEPHYR_BASE% \samples\synchronization
```

Please note after you have built your project in CMake:

Any code changes that require rerunning cmake (Kconfigs, device trees, CMakeLists) will require regenerating your Eclipse .project file and reloading it in Eclipse. Code changes in .c and .h files do not require rerunning cmake

```
ninja
```

This will generate a .project file* in the %userprofile%/build directory

Then, import the project into eclipse by opening eclipse and going to file > import > existing projects:

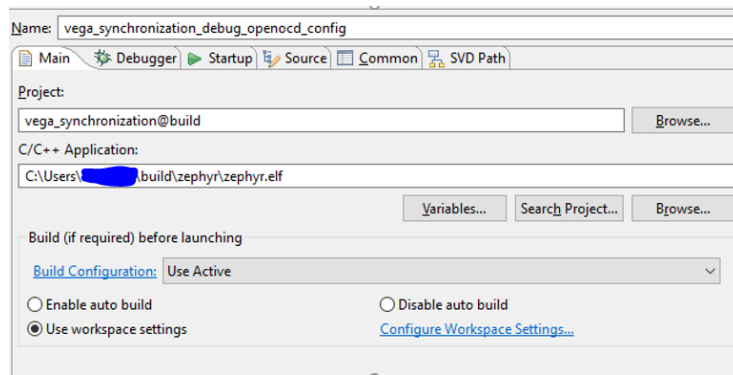
Now we will configure the debugging settings for the VEGA board which uses OpenOCD:

Go to run > debug configurations

Create a new openOCD configuration by right clicking on "GDB OpenOCD debugging" and selecting "new configuration".

Set up the debug configurations as follows: (the blue box should point to %userprofile%)

Under the main tab, name your project and give it the zephyr.elf file:



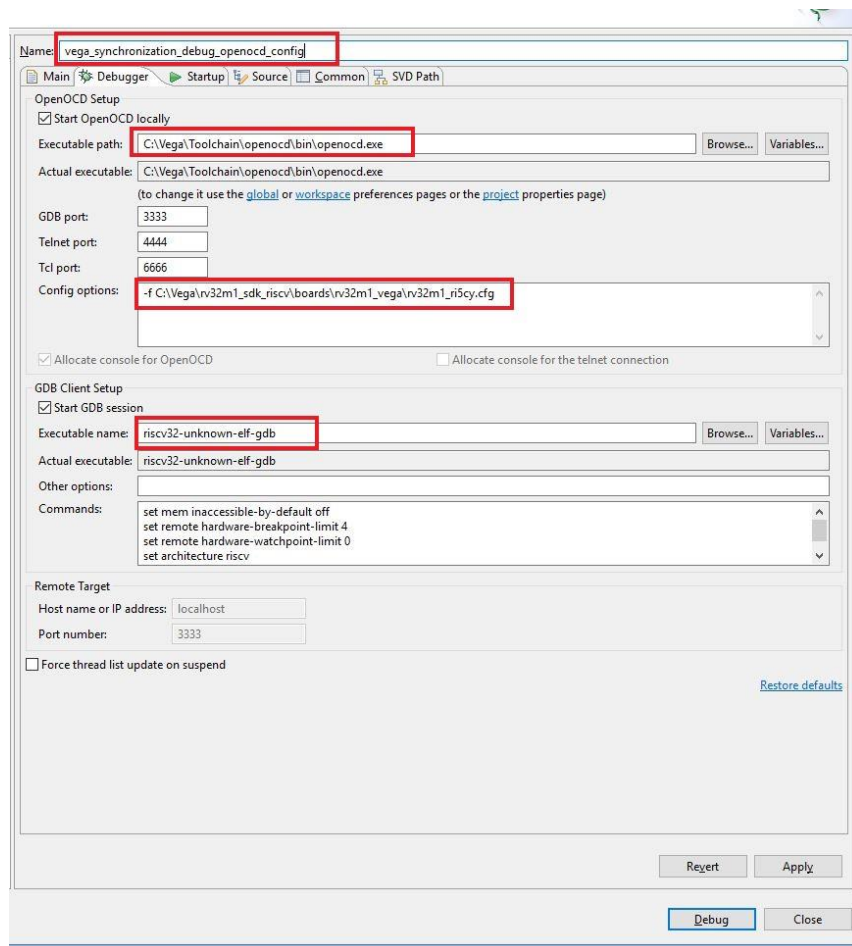
Then, in the debugger tab, configure the following:

Executable path:

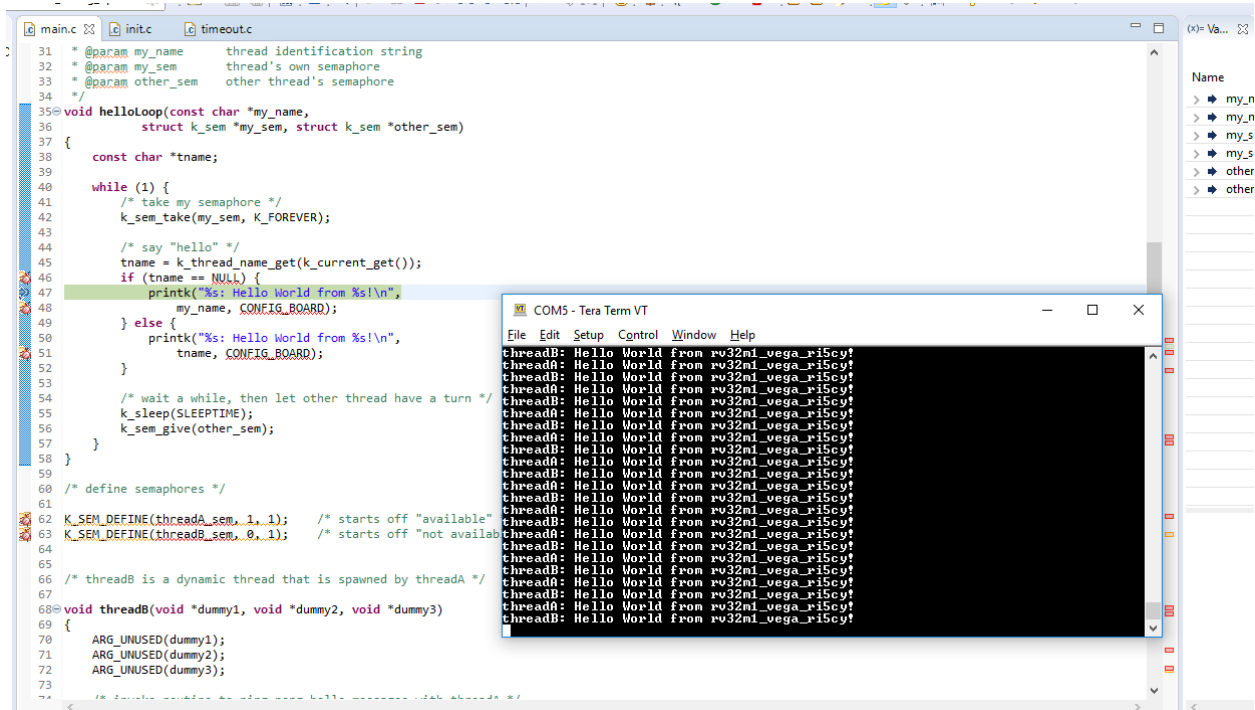
C:\Vega\Toolchain\openocd\bin\openocd.exe

Config options:

-f C:\Vega\rv32m1_sdk_riscv\boards\rv32m1_vega\rv32m1_ri5cy.cfg



After configuring the debug settings, you can open a serial terminal built into eclipse by window > show view > terminal and set the correct speed to 115200 and select the correct COM port. Alternatively, you can also choose to use your own terminal program. Click on the debug button of the debug configuration tab and you should be able to step through code! **



The screenshot shows the Eclipse IDE with a C program open in the editor. The program is a multi-threaded application that prints "Hello World" from different threads. The code is as follows:

```
main.c  init.c  timeout.c
31  * @param my_name      thread identification string
32  * @param my_sem       thread's own semaphore
33  * @param other_sem     other thread's semaphore
34  */
35  void helloLoop(const char *my_name,
36                struct k_sem *my_sem, struct k_sem *other_sem)
37  {
38      const char *tname;
39
40      while (1) {
41          /* take my semaphore */
42          k_sem_take(my_sem, K_FOREVER);
43
44          /* say "hello" */
45          tname = k_thread_name_get(k_current_get());
46          if (tname == NULL) {
47              printk("%s: Hello World from %s!\n",
48                    my_name, CONFIG_BOARD);
49          } else {
50              printk("%s: Hello World from %s!\n",
51                    tname, CONFIG_BOARD);
52          }
53      }
54      /* wait a while, then let other thread have a turn */
55      k_sleep(SLEEPTIME);
56      k_sem_give(other_sem);
57  }
58
59  /* define semaphores */
60  K_SEM_DEFINE(threadA_sem, 1, 1); /* starts off "available"
61  K_SEM_DEFINE(threadB_sem, 0, 1); /* starts off "not available"
62
63  /* threadB is a dynamic thread that is spawned by threadA */
64
65  void threadB(void *dummy1, void *dummy2, void *dummy3)
66  {
67      ARG_UNUSED(dummy1);
68      ARG_UNUSED(dummy2);
69      ARG_UNUSED(dummy3);
70
71      /* ... */
72  }
```

The terminal window (COM5 - Tera Term VT) shows the output of the program, which is a series of "Hello World" messages from different threads, indicating that the program is running successfully.

*Note if you would like to import a project with the same name (but for a different board, for example), you would have to rename that project. After building the project go to the directory and open the .project file. You can edit the name of the project here to be specific to whichever board you are using.

**Also note that when debugging on the VEGAboard, there is a limit of two hardware breakpoints. These can be seen in the breakpoints tab in debugger mode. If you are stepping through code it is recommended to delete all other debug breakpoints.