# Procedure to Debug using GDB in Windows CMD

## Prerequisites

1. Windows PC
2. OpenOCD setup
3. GDB-Multiarch setup

## OpenOCD Setup

To install the environment for GDB debugging in Windows, follow the instructions described in sections: *Prerequisites for Eclipse* and *Add Paths to the Environment Variable* of the document: UG\_Eclipse\_Setup\_Windows.docx (*sdk\_x.y\doc\user\_guides\ug\_eclipse\_setup\_windows\*).

## GDB-Multiarch

MSYS2 is a collection of tools and libraries, which provides an easy-to-use environment for building, installing and running in native Windows software. MSYS2 allows user to install GDB-Multiarch in windows machine.

Download the installer from the following link: <https://www.msys2.org/>.

Follow the installation procedure available in the above link. After completing the installation, click on Finish, which will create a popup for MSYS2 CMD line interface.

A screen shot of a computer

Description automatically generated

Figure 15: Running MSYS2

Run the following command in MSYS2 terminal and proceed with installation.

|  |
| --- |
| pacman -Syu |

A computer screen shot of a black screen

Description automatically generated

Figure 16: Installing mingw setup

Once the installation is complete, the window will be automatically closed. Run MSYS2 MSYS from the Start menu and run the following command in terminal to update the rest of the base packages.

Proceed with installation.

|  |
| --- |
| pacman -Syu |

After completing the installation, run the following command to install GDB-Multiarch:

|  |
| --- |
| pacman -S --needed base-devel mingw-w64-x86\_64-toolchain |

Enter a selection number, for GDB-Multiarch as shown in figure.

A screenshot of a computer

Description automatically generated

Figure 17: Iinstalling gdb-multiarch

Add MSYS2 path in environmental variable to access GDB-Multiarch in command line. To add path to environment variable, follow the steps mentioned in section: *Add Paths to the Environment Variable* of the document for MSYS2: UG\_Eclipse\_Setup\_Windows.pdf ((*sdk\_x.y\doc\user\_guides\ug\_eclipse\_setup\_windows\*).

A screenshot of a computer program

Description automatically generated

Figure 18: Adding environment variable

## Procedure to Debug using GDB

Following is the procedure to debug the VM-based applications using GDB:

1. Open the SDK folder in windows command line and type the following command to start OpenOCD:

|  |
| --- |
| openocd -s .\conf -f ftdi.cfg -f t2.cfg |

Console output:

A computer screen shot of a black screen

Description automatically generated

Figure 19: Running Openocd in windows CMD

1. Use the Download Tool to flash the virtual image from the SDK directory.

For example: Consider wifi\_connect.elf.

**Note**: For the GDB to work, ELF needs to be loaded. By default, the SDK package contains ELF files in the bin folder (which are stripped ELF files). Hence, the user needs to build the sample application, generate the ELF file (by default, the ELF gets generated in the out folder) and load this ELF for debugging.

For building in windows, follow the steps described in section: *Building Application in Eclipse* of the document: UG\_Eclipse\_Setup\_Windows.pdf ((*sdk\_x.y\doc\user\_guides\ug\_eclipse\_setup\_windows\*).

Execute the make for using\_wifi example application (*sdk\_x.y\examples\using\_wifi*) to generate the ELFs under the out folder.

A screen shot of a computer screen

Description automatically generated

Figure 20: Running make command in windows CMD

.gdbinit initialization file contains the information on Talaria TWO’s memory and the required scripts of the GDB sources. gdbinit file is present under the *apps\* folder. To start the GDB session, gdb-multiarch should be started from this folder.

Manual method of configuring the gdbinit file:

If there are any warnings as shown in Figure 6, the gdb-multiarch will not work for GDB commands. Hence, create a file named gdbinit in the home directory to allow auto-load.

A computer screen shot of a program

Description automatically generated

Figure 21: Warning for .gdbinit file

In the created gdbinit file, add the following path:

add-auto-load-safe-path C:\Users\innop\Music\sdk\_2.5alpha\apps\.gdbinit

A close-up of a computer screen

Description automatically generated

Figure 22: Configuring the gdbinit file

## Start a GDB session

In a separate terminal, run the following command from the *sdk\_x.y\apps* folder. In this directory, there is a .gdbinit file that configures the GDB. Here, the RAM portion of the ELF gets loaded.

A screenshot of a computer program

Description automatically generated

Figure 23 Running GDB

1. Connect to OpenOCD by running ocd in the GDB prompt.

|  |
| --- |
| ocd |

1. Set a break point at main:

|  |
| --- |
| b main |

1. Run the application by executing:

|  |
| --- |
| R |

1. Information on the break points set can be seen by issuing:

|  |
| --- |
| info b |

1. A break point at a line number of a particular source file can be set using:

|  |
| --- |
| b <filename>: <linenum> |

1. If the line to be executed is a function call, GDB will step into that function and start executing its code one line at a time.

|  |
| --- |
| s |

1. If the entire function needs to be executed with one key press, type next or n.

|  |
| --- |
| next |

1. Continue running the program (after stopping, for example at a breakpoint).

|  |
| --- |
| continue |

1. Step out is the operation that resumes execution after the function the program is executing terminates. The debugger will stop at the statement after the function call.

|  |
| --- |
| finish |

Example 1: Following is the output while debugging the wifi\_connect.elf using GDB:

|  |
| --- |
| (gdb) ocd  warning: A handler for the OS ABI "Windows" is not built into this configuration  of GDB. Attempting to continue with the default armv7 settings.  0x00023f36 in ?? ()  (gdb) b main  Breakpoint 1 at 0x150e04: file src/wifi\_connect.c, line 79.  Note: automatically using hardware breakpoints for read-only addresses.  (gdb) R  JTAG tap: talaria\_two.cpu tap/device found: 0x4ba00477 (mfg: 0x23b (ARM Ltd.), part: 0xba00, ver: 0x4)  target halted due to debug-request, current mode: Thread  xPSR: 0x01000000 pc: 0x00020f90 msp: 0x00041a78  Loading section .text, size 0x13778 lma 0x42000  Loading section .data, size 0x520 lma 0x55778  Loading section .virt0, size 0x10a28 lma 0x2000000  Loading section .virt1, size 0x17c98 lma 0x3000000  Loading section .virt2, size 0x22824 lma 0x4000000  Loading section .virt3, size 0x628 lma 0x5000000  Loading section .virt4, size 0x5704 lma 0x6000000  Loading section .virt5, size 0x2ec lma 0x7000000  Start address 0x00047d00, load size 412564  Transfer rate: 71 KB/sec, 13308 bytes/write.  Program received signal SIGTRAP, Trace/breakpoint trap.  shutdown () at arm/entry.S:196  196 arm/entry.S: No such file or directory.  (gdb) info b  Num Type Disp Enb Address What  1 breakpoint keep y 0x00150e04 in main at src/wifi\_connect.c:79  (gdb) del 1  (gdb) info b  No breakpoints or watchpoints.  (gdb) b main.c:29  Breakpoint 2 at 0x112704: file core/main.c, line 30.  (gdb) R  JTAG tap: talaria\_two.cpu tap/device found: 0x4ba00477 (mfg: 0x23b (ARM Ltd.), part: 0xba00, ver: 0x4)  target halted due to debug-request, current mode: Thread  xPSR: 0x01000000 pc: 0x00020f90 msp: 0x00041a78  Loading section .text, size 0x13778 lma 0x42000  Loading section .data, size 0x520 lma 0x55778  Loading section .virt0, size 0x10a28 lma 0x2000000  Loading section .virt1, size 0x17c98 lma 0x3000000  Loading section .virt2, size 0x22824 lma 0x4000000  Loading section .virt3, size 0x628 lma 0x5000000  Loading section .virt4, size 0x5704 lma 0x6000000  Loading section .virt5, size 0x2ec lma 0x7000000  Start address 0x00047d00, load size 412564  Transfer rate: 71 KB/sec, 13308 bytes/write.  Program received signal SIGTRAP, Trace/breakpoint trap.  shutdown () at arm/entry.S:196  196 in arm/entry.S  (gdb) |