



TELINK SEMICONDUCTOR

Application Note : Telink gdb Tool User Guide

AN-16082900-E4

Ver 1.3.0

2016/9/7

Brief:

This document is the user guide for Telink gdb tool.

Published by
Telink Semiconductor

Bldg 3, 1500 Zuchongzhi Rd,
Zhangjiang Hi-Tech Park, Shanghai, China

© Telink Semiconductor
All Right Reserved

Legal Disclaimer

Telink Semiconductor reserves the right to make changes without further notice to any products herein to improve reliability, function or design. Telink Semiconductor disclaims any and all liability for any errors, inaccuracies or incompleteness contained herein or in any other disclosure relating to any product.

Telink Semiconductor does not assume any liability arising out of the application or use of any product or circuit described herein; neither does it convey any license under its patent rights, nor the rights of others

The products shown herein are not designed for use in medical, life-saving, or life-sustaining applications. Customers using or selling Telink Semiconductor products not expressly indicated for use in such applications do so entirely at their own risk and agree to fully indemnify Telink Semiconductor for any damages arising or resulting from such use or sale.

Information:

For further information on the technology, product and business term, please contact Telink Semiconductor Company (www.telink-semi.com).

For sales or technical support, please send email to the address of:

telinkcnsales@telink-semi.com

telinkcnsupport@telink-semi.com

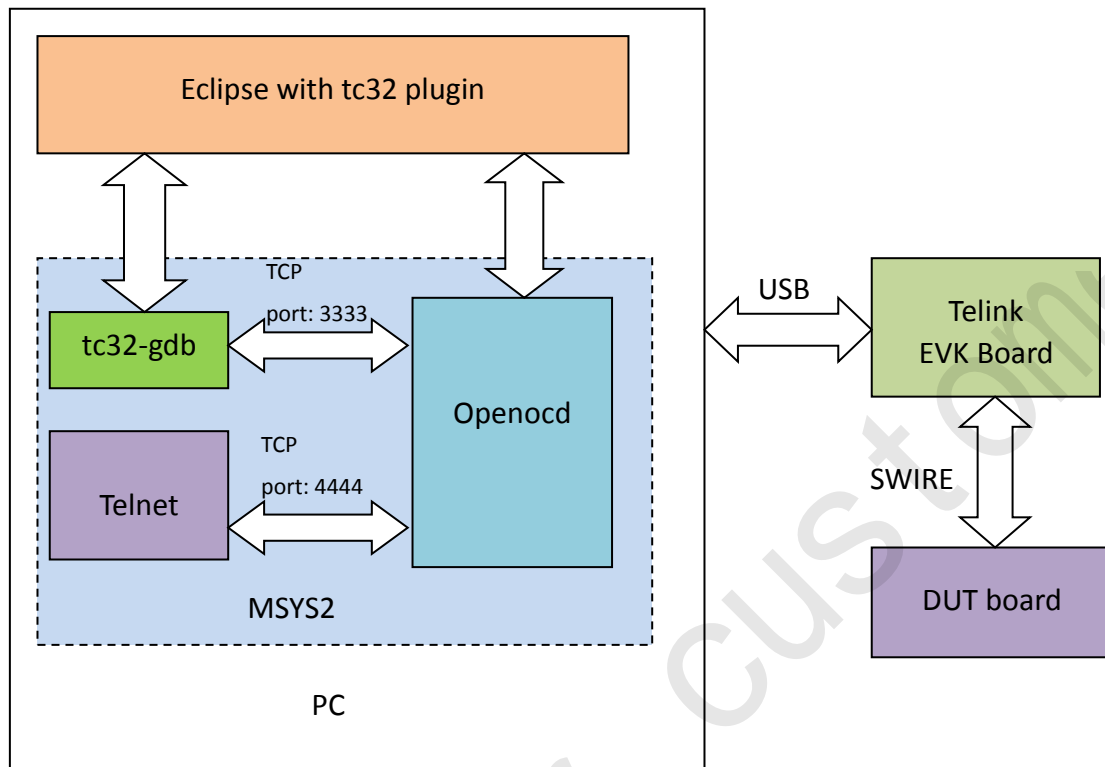
Revision History

Version	Major Changes	Date	Author
1.0.0	Initial release	2016/8	Z.X.D., Cynthia
1.1.0	Added useful OpenOCD commands.	2016/8	Peter, Z.X.D., Cynthia
1.2.0	Added driver installment introduction for initial usage of Telink gdb tool.	2016/9	Z.X.D., Cynthia
1.3.0	Added debug guide for Eclipse with tc32 plugin; Updated debug guide for tc32-gdb.	2016/9	L.J.W., Z.X.D., Cynthia

Table of contents

1	Tool Architecture	4
2	Debug Guide for tc32-gdb and Telnet	6
3	Debug Guide for Eclipse with tc32 Plugin	13
4	Useful OpenOCD Commands.....	17
4.1	Telink OpenOCD Commands.....	17

1 Tool Architecture

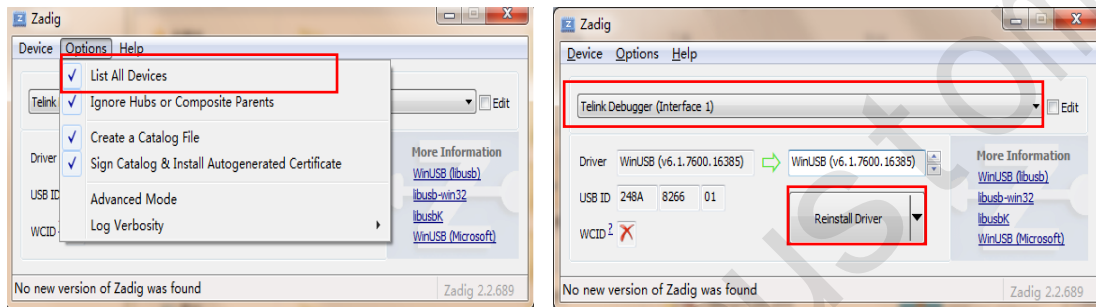


Notes:

1. Now the tc32 plugin is still under development.
2. The firmware download function is not available in current tc32-gdb.

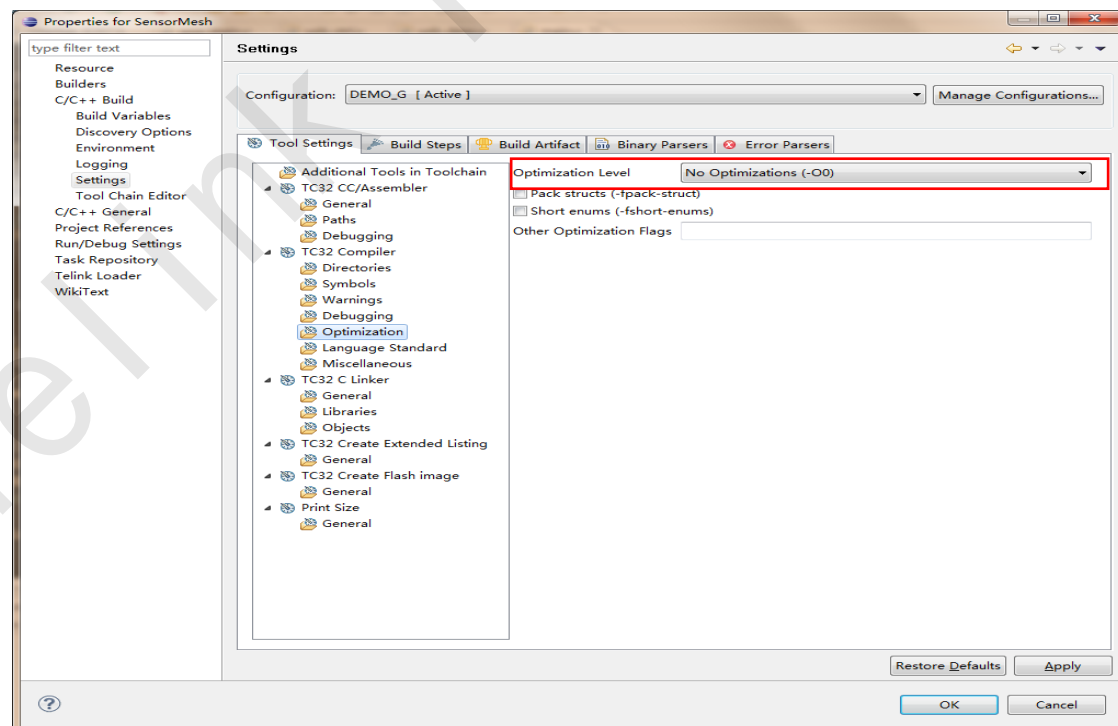
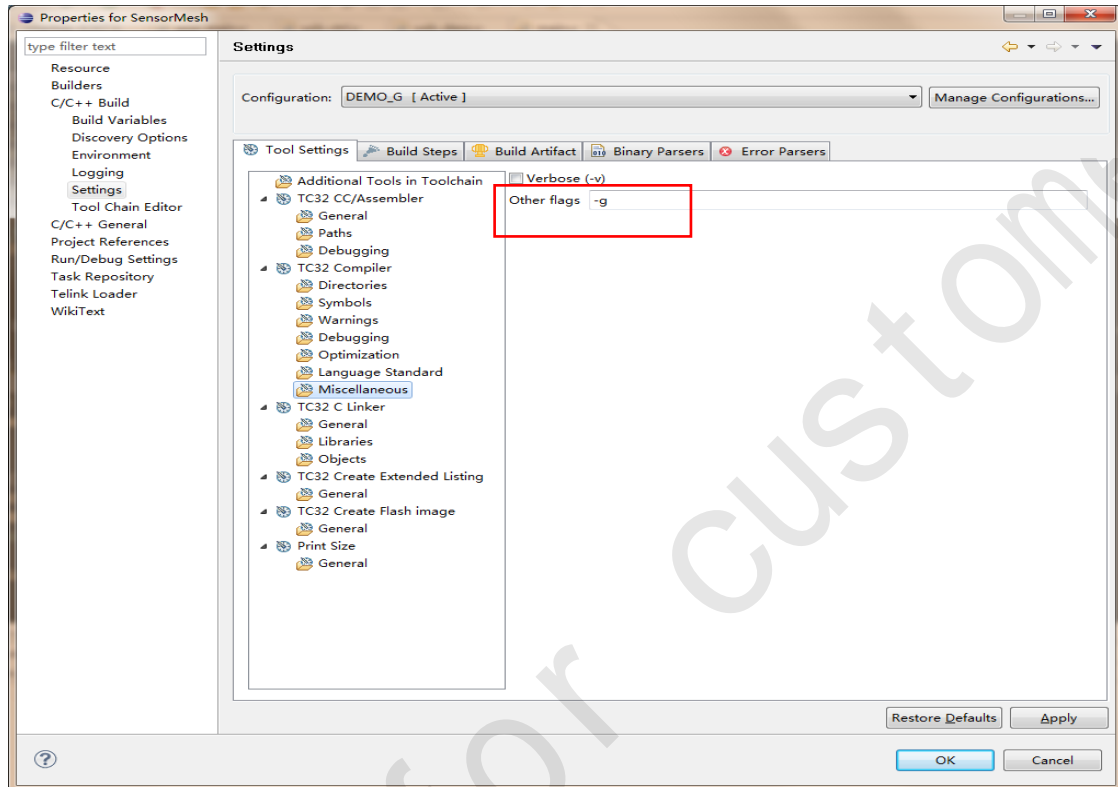
To start using Telink gdb tool for the first time, make sure the "usb print" device is converted to "libusb" driver through the "zadig_2.2.exe". Please follow the steps below:

1. Connect 8266 EVK (burned with 8266_tlink.bin) to PC via USB.
2. Start "zadig_2.2.exe", tick the "List All Devices" under the "Options", select the driver of Telink Debugger (interface 1), and click the "Reinstall Driver" button to install the driver.

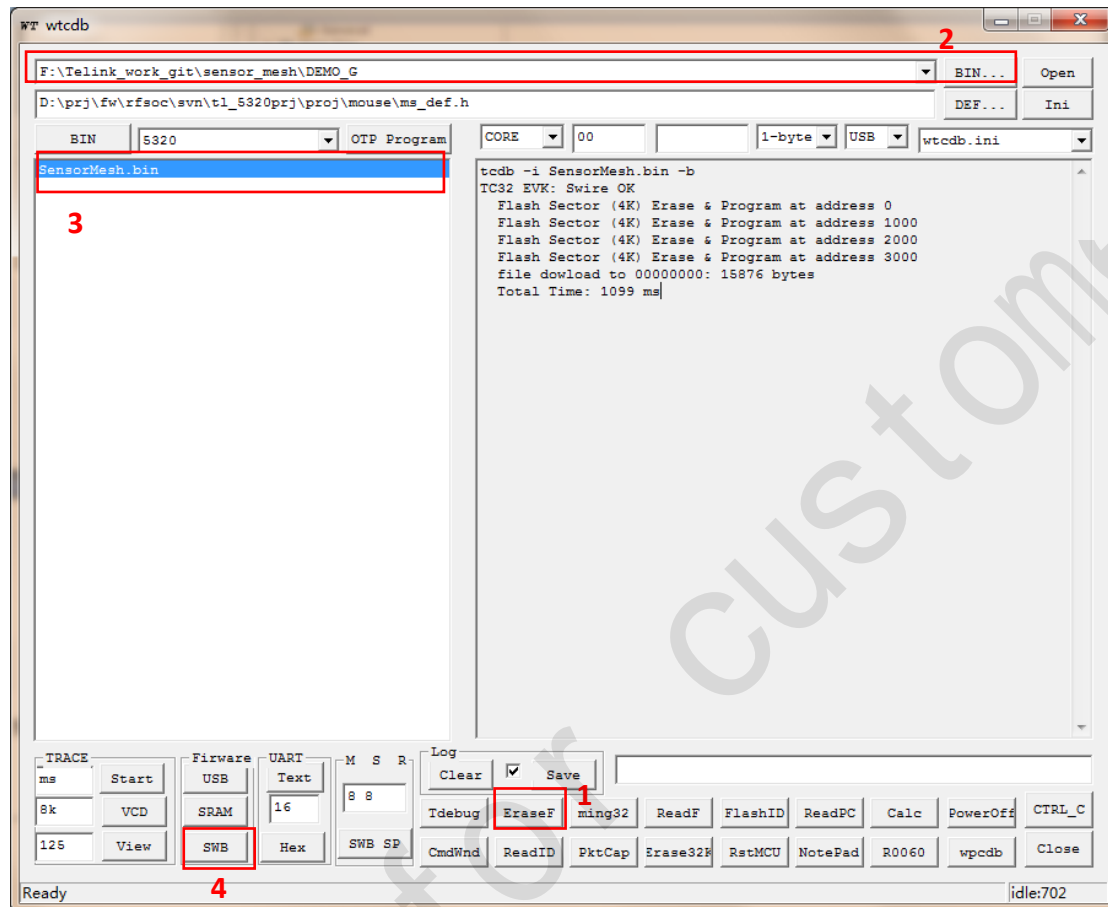


2 Debug Guide for tc32-gdb and Telnet

Step 1: Add "-g" options to the project, and select "No Optimizations (-O0)" for "Optimization Level" option.



Step 2: Download target test firmware into the DUT board via Telink WtcdB tool and a burning EVK.

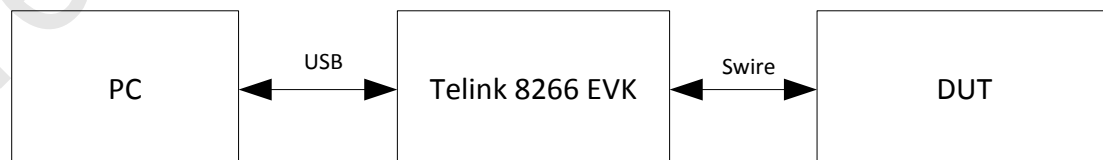


Step 3: Connect hardware

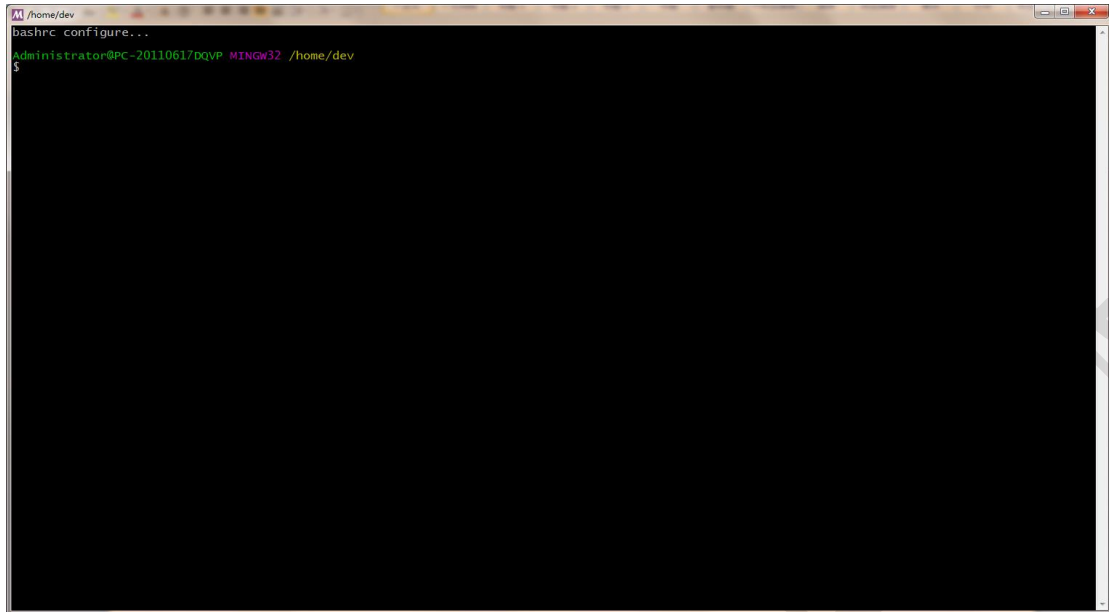
Connect the DUT board with Telink 8266 EVK board via Swire interface. (Telink-supplied EVK board is already preloaded with the firmware needed.)

Connect the miniUSB interface of the EVK board with PC USB via an USB cable.

The figure below shows the hardware connection chart.



Step 4: Double click the “msys.bat” file to start MSYS2 environment.



It's allowed to open more than one MSYS2 environment window at the same time.

Step 5: Open openocd.

Type a command "openocd -f interface/tlink.cfg" in current MSYS2 environment, and click the "Enter" key.

If an error such as "can't find the tlink.cfg" occurs, please use the following command:

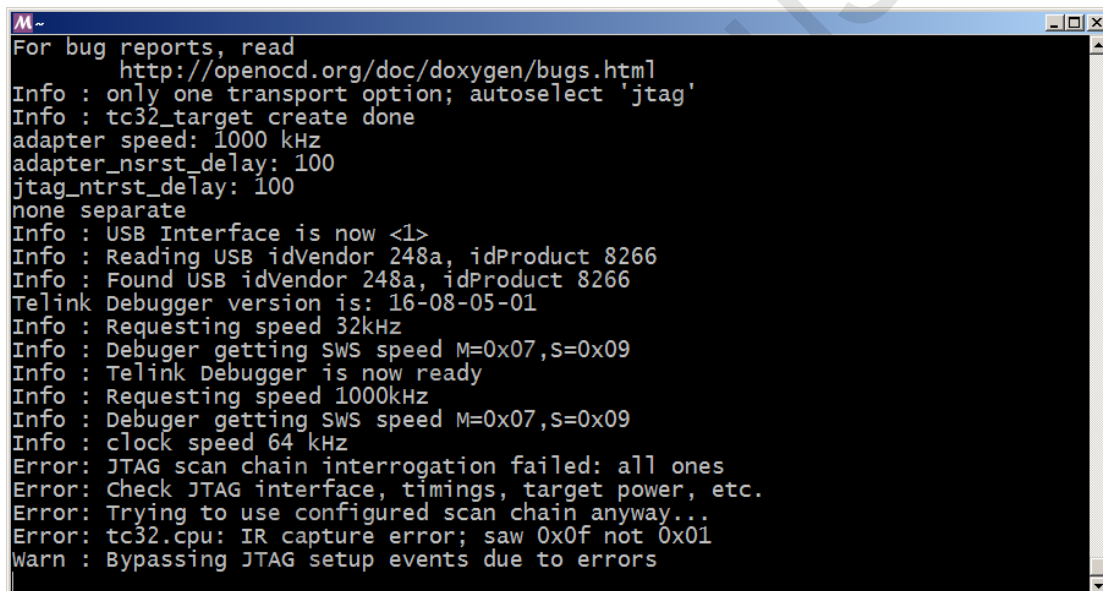
```
"openocd -s /usr/local/share/openocd/scripts -f interface/tlink.cfg"
```

or

```
"openocd -f interface/tlink.cfg -s /usr/local/share/openocd/scripts"
```

Note: "/usr/local/share/openocd/scripts" indicates the script path.

The window below shows the debugging message.



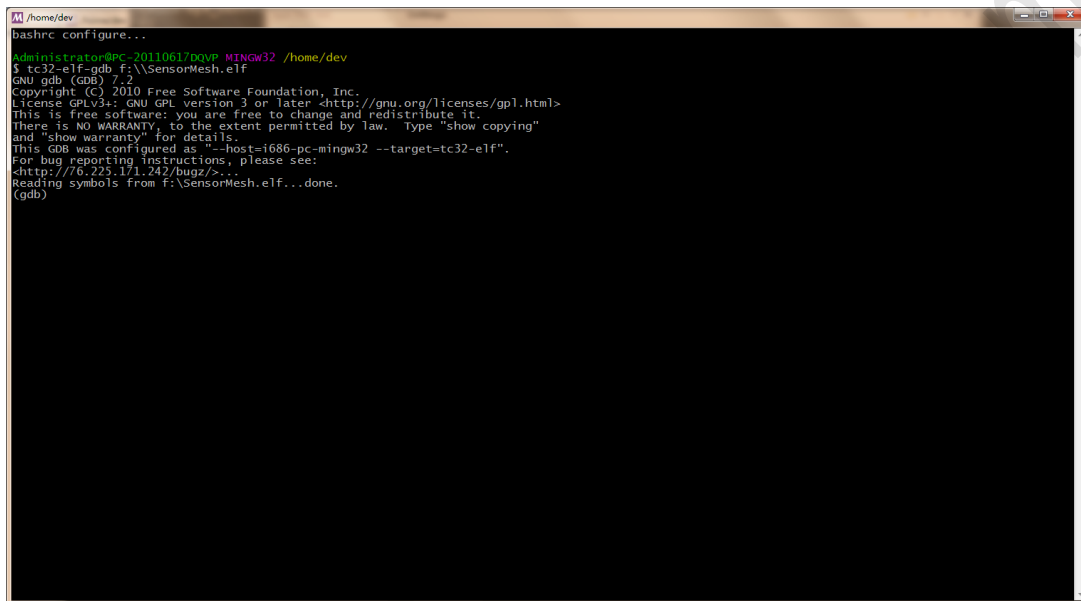
```
M~
For bug reports, read
  http://openocd.org/doc/doxygen/bugs.html
Info : only one transport option; autoselect 'jtag'
Info : tc32_target create done
adapter speed: 1000 kHz
adapter_nsrst_delay: 100
jtag_ntrst_delay: 100
none separate
Info : USB Interface is now <1>
Info : Reading USB idVendor 248a, idProduct 8266
Info : Found USB idVendor 248a, idProduct 8266
Telink Debugger version is: 16-08-05-01
Info : Requesting speed 32kHz
Info : Debugger getting SWS speed M=0x07,S=0x09
Info : Telink Debugger is now ready
Info : Requesting speed 1000kHz
Info : Debugger getting SWS speed M=0x07,S=0x09
Info : clock speed 64 kHz
Error: JTAG scan chain interrogation failed: all ones
Error: Check JTAG interface, timings, target power, etc.
Error: Trying to use configured scan chain anyway...
Error: tc32.cpu: IR capture error; saw 0x0f not 0x01
Warn : Bypassing JTAG setup events due to errors
```

Step 6: Open tc32-gdb.

- 1) Start **another** MSYS2 environment, type a command "tc32-elf-gdb f:\\SensorMesh.elf" in the new MSYS2 environment and click the "Enter" key to open tc32-gdb.

The "SensorMesh.elf" is available at the folder containing the SensorMesh.bin.

In this example, the SensorMesh.elf file is copied to the "f:\\\" folder to simplify the path in this command.

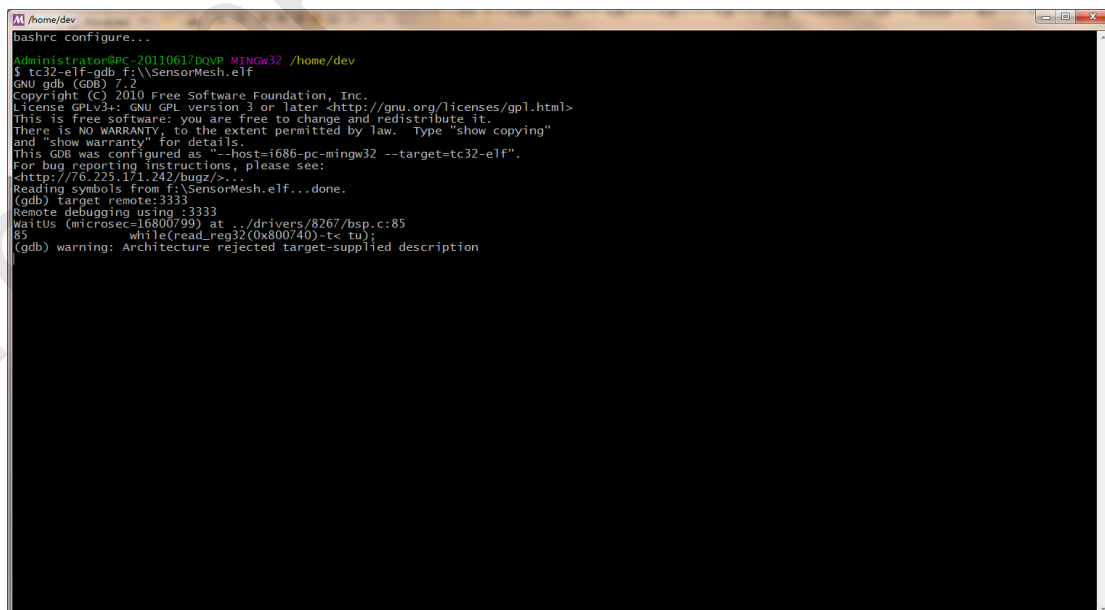


```

/home/dev
bashrc configure...
Administrator@PC-20110617QQWP MINGW32 /home/dev
$ tc32-elf-gdb f:\\SensorMesh.elf
GNU gdb (GDB) 7.2
Copyright (C) 2010 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-pc-mingw32 --target=tc32-elf".
For bug reporting instructions, please see:
<http://6.225.171.242/bugz/>...
Reading symbols from f:\\SensorMesh.elf...done.
(gdb)

```

- 2) Type a command "target remote:3333" in tc32-gdb window and click the "Enter" key to connect tc32-gdb with openocd.



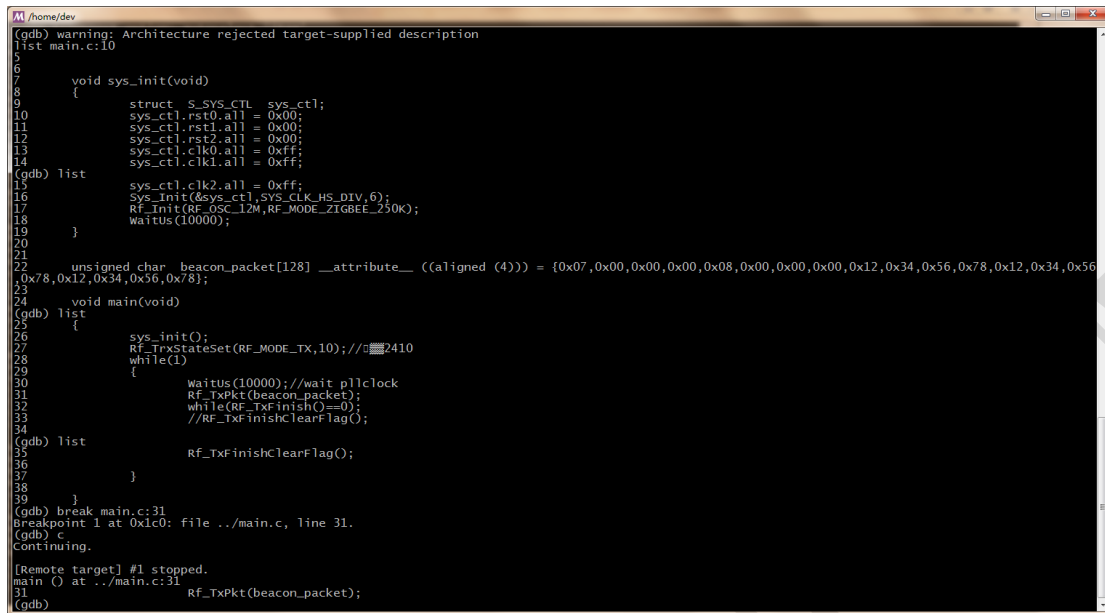
```

/home/dev
bashrc configure...
Administrator@PC-20110617QQWP MINGW32 /home/dev
$ tc32-elf-gdb f:\\SensorMesh.elf
GNU gdb (GDB) 7.2
Copyright (C) 2010 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-pc-mingw32 --target=tc32-elf".
For bug reporting instructions, please see:
<http://6.225.171.242/bugz/>...
Reading symbols from f:\\SensorMesh.elf...done.
(gdb) target remote:3333
Remote debugging using :3333
waitus (microsec=16800/99) at :./drivers/8267/bsp.c:85
85 while(read_reg32(0x800740)-tc tu);
(gdb) warning: Architecture rejected target-supplied description

```

3) User can use gdb commands to control and debug the DUT board.

Example:



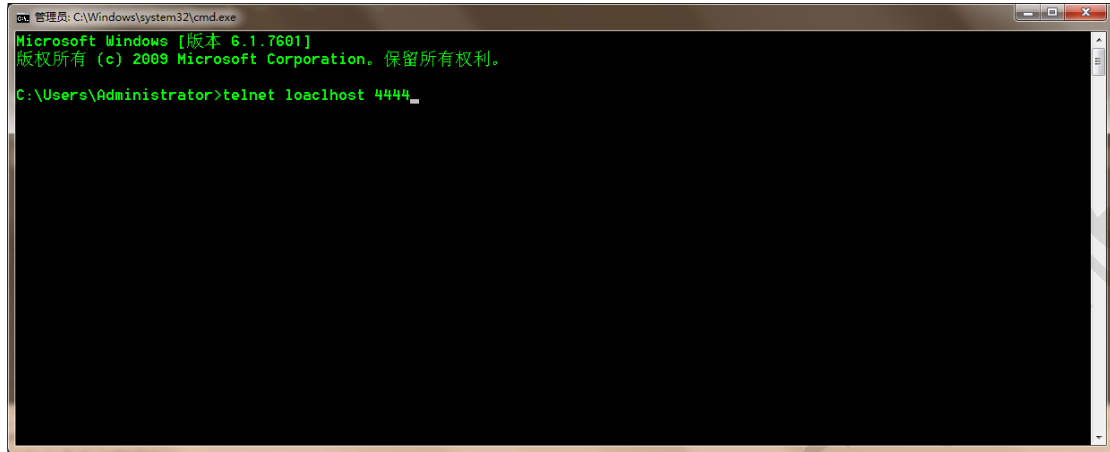
```

(gdb) warning: Architecture rejected target-supplied description
list main.c:10
5
6
7     void sys_init(void)
8     {
9         struct S_SYS_CTL sys_ctl;
10        sys_ctl.rst0.all = 0x00;
11        sys_ctl.rst1.all = 0x00;
12        sys_ctl.rst2.all = 0x00;
13        sys_ctl.clk0.all = 0xff;
14        sys_ctl.clk1.all = 0xff;
(gdb) list
15        sys_ctl.clk2.all = 0xff;
16        sys_init(&sys_ctl, SYS_CLK_HS_DIV, 6);
17        Rf_Init(RF_OSC_12M, RF_MODE_ZIGBEE_250K);
18        waitus(10000);
19    }
20
21
22    unsigned char beacon_packet[128] __attribute__((aligned(4))) = {0x07,0x00,0x00,0x00,0x08,0x00,0x00,0x00,0x12,0x34,0x56,0x78,0x12,0x34,0x56,
0x78,0x12,0x34,0x56,0x8};
23
24    void main(void)
(gdb) list
25    {
26        sys_init();
27        Rf_TrxStateSet(RF_MODE_TX,10);//2410
28        while(1)
29        {
30            waitus(10000);//wait pllclock
31            Rf_TxPkt(beacon_packet);
32            while(Rf_TxFinish()==0);
33            //Rf_TxFinishClearFlag();
34
(gdb) list
35            Rf_TxFinishClearFlag();
36
37        }
38    }
39
(gdb) break main.c:31
Breakpoint 1 at 0x1c0: file ../main.c, line 31.
(gdb) c
Continuing.

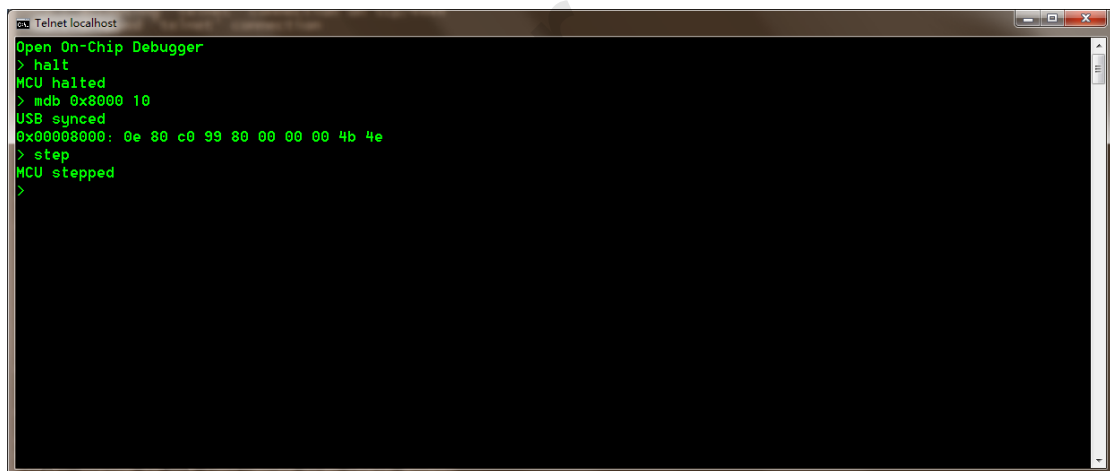
[Remote target] #1 stopped.
main () at ../main.c:31
31      Rf_TxPkt(beacon_packet);
(gdb)
  
```

Debug via Telnet:

Start windows command line environment "cmd.exe", and type a command "telnet localhost 4444" to open Telnet.



Then user can also use telnet commands to control and debug the DUT. Please refer to the "openocd.pdf" for the command list.



3 Debug Guide for Eclipse with tc32 Plugin

Besides command lines debugging by gdb and Telnet, user can also use Eclipse with tc32 plugin to debug the DUT.

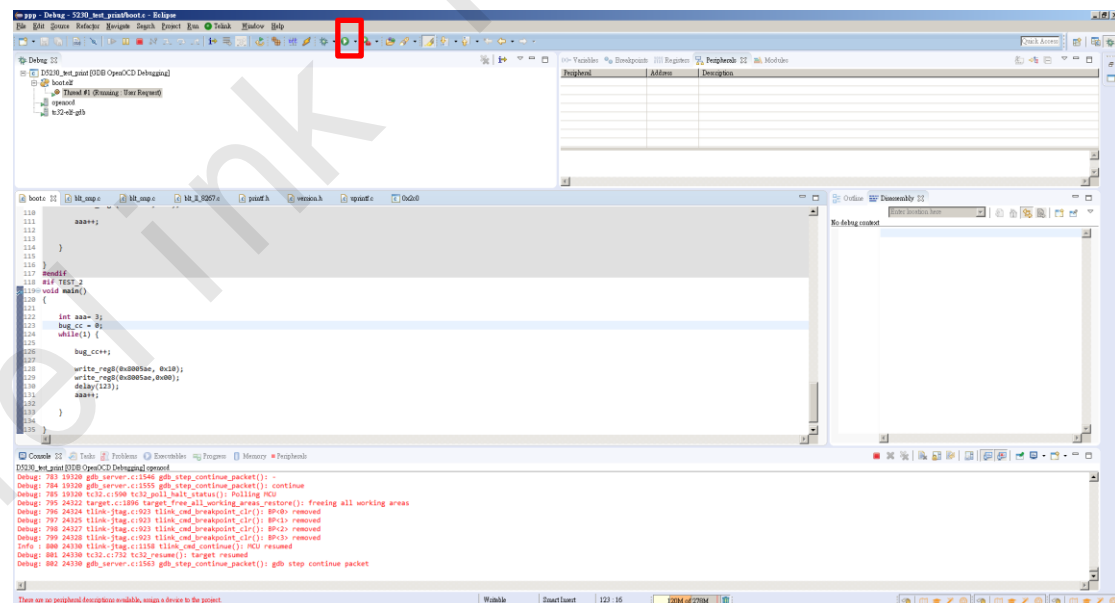
Step 1: Make sure the test firmware is already downloaded into the DUT board, and connect the DUT with PC via Telink 8266 EVK (refer to **Step 3** in **Section 2**).

Step 2: Double click the “msys.bat” file to start MSYS2 environment.

Step 3: Type a command “/eclipse &” in the MSYS2 environment, and click the “Enter” key to open Eclipse window with tc32 plugin.

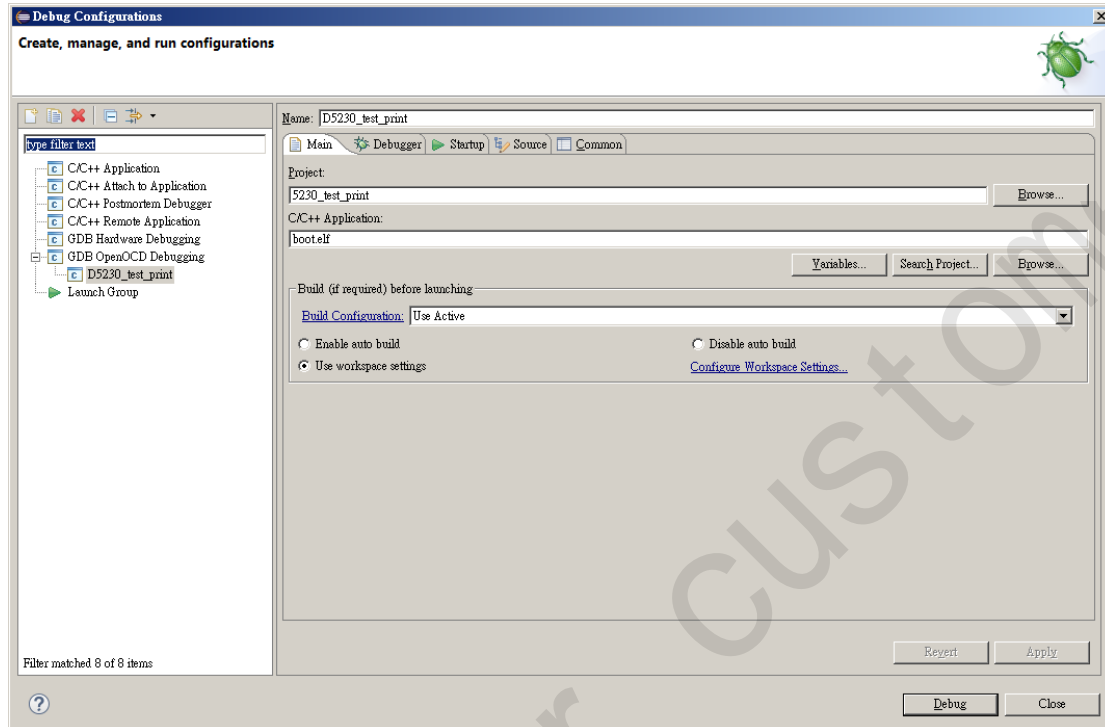
Step 4: Add “-g” options to the project, and select “No Optimizations (-O0)” for “Optimization Level” option. Refer to **Step 1** in **Section 2**.

Step 5: Click “Debug configuration...” under the debug icon (as shown in the figure below). The “Debug Configurations” window will pop up.

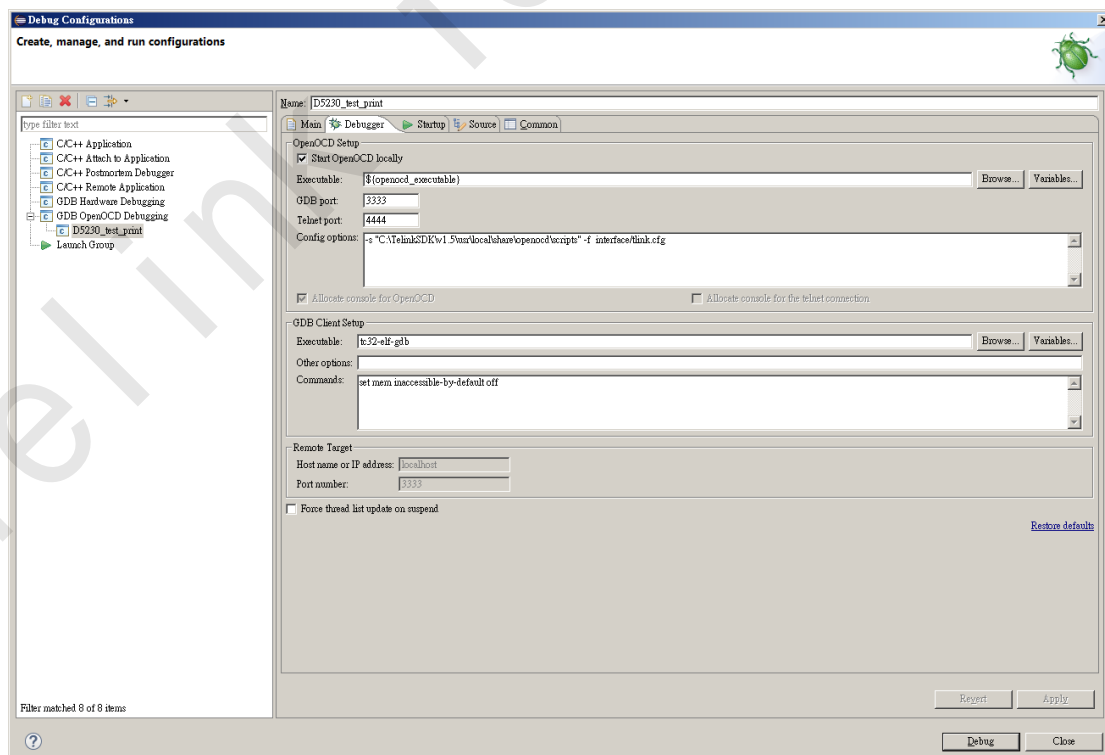


Step 6: Set the following openocd & tc32-elf-gdb parameters in the “Debug Configurations” window.

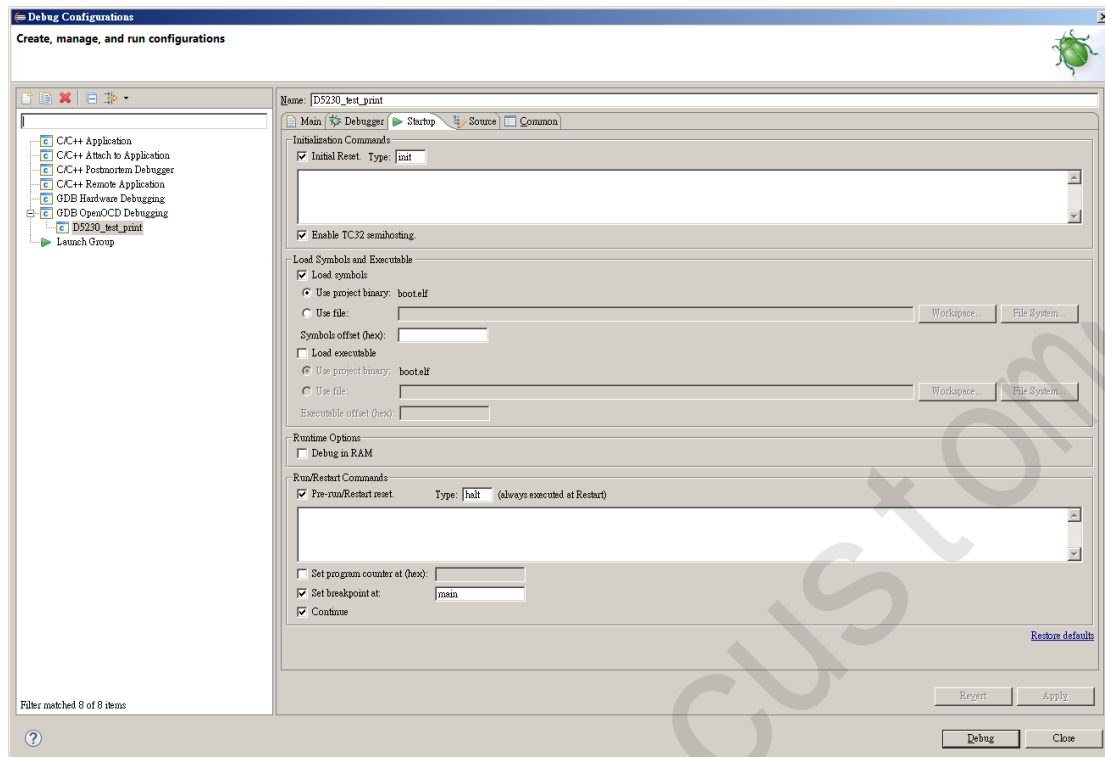
- 1) Set openocd for project, as shown below:



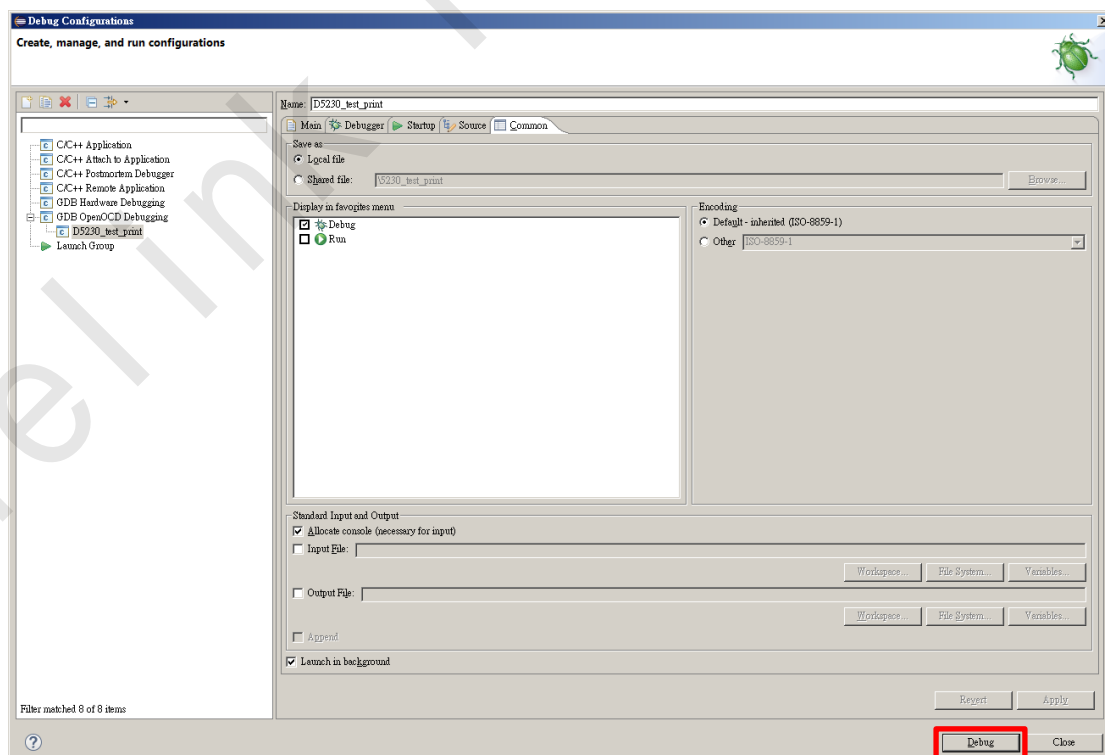
- 2) Set openocd for debugger, as shown below:



3) Set openocd for gdb init, as shown below:



4) (Optional) User can add menu shortcut for the specific project as needed, as shown below:



- 5) Click the “Debug” button, and wait until the debugging window pops up.

Step 7: In the debugging window, user can set breakpoint and do step-in (step-over) debugging line by line.

***Note:** If any error (usually "can't find the tlink.cfg") occurs in the debugging window, please refer to the solution introduced in **Step 5** of **Section 2**.

4 Useful OpenOCD Commands

OpenOCD telnet provides a useful and fast way to explore and debug your program.

4.1 Telink OpenOCD Commands

- helpme

```
Open On-Chip Debugger
[> helpme
Command: tinfo - Telink AlmightyTlink debugger (EVK2)
Usage: tinfo
Command: usync - sync up debugger USB
Usage: usync
Command: stop - halt the MCU
Usage: stop
Command: halt - halt the MCU
Usage: halt
Command: resume - continue
Usage: resume
Command: cont - continue
Usage: cont
Command: c - continue
Usage: c
Command: restart - restart MCU
Usage: restart
Command: until - Run until <hex>
Usage: until <hex>
Command: breakclear - clear all breakpoints
Usage: breakclear
Command: break - insert break at <hex>
Usage: break <hex>
Command: breakc - clear breakpoint <bp#>
Usage: breakc <bp#>
Command: breakr - remove breakpoint at <hex>
Usage: breakr <hex>
Command: breaks - breakpoint status
Usage: breaks
Command: mcu_status - MCU status
Usage: mcu_status
Command: reached - breakpoint reached status
Usage: reached <bp#>
Command: reachat - breakpoint <hex> reached status
Usage: reachat <hex>
Command: usbdebug - toggle usb debugging
Usage: usbdebug
```

- debug_level

This is a system level command. Useful debug level is between 0 ~ 3. Debug level 3 will give you the most debugging outputs.

It's advised that you can save your debugging output to a log file while you start your openocd.

```
openocd -f tlink.cfg -l debug.log
```

```
usage: usbddebug
>
> debug_level 3
debug_level: 3
>
```

- stop or halt

Both commands will halt the MCU. This is normally your first step of debugging.

- reg

This is also a system level command. You can only read the DUT board register values when your MCU is halted, except the PC value which you can read at anytime even when your program is running.

```
> halt
MCU halted
> reg
> TC32 registers
(0) r0 (/32): 0x00000000
(1) r1 (/32): 0x00000000
(2) r2 (/32): 0x00000000
(3) r3 (/32): 0x00000000
(4) r4 (/32): 0x00000000
(5) r5 (/32): 0x00000000
(6) r6 (/32): 0x00000000
(7) r7 (/32): 0x00000000
(8) r8 (/32): 0x00000000
(9) r9 (/32): 0x00000000
(10) r10 (/32): 0x00000000
(11) r11 (/32): 0x00000000
(12) r12 (/32): 0x00000000
(13) sp_usr (/32): 0x00000000
(14) lr_usr (/32): 0x00000000
(15) pc (/32): 0x0000010E
(16) status (/32): 0x00000000
(17) sp_irq (/32): 0x00000000
>
```

- resume or cont or c

Resume your MCU running.

```
[> resume
MCU resumed
> █
```

- poll on/off

Poll is a system level command. It does many things including polling MCU status to the cache. It also provides the heart beats to many of our OpenOCD debugging features.

It's suggested that we start the polling all the time if not by default started.

```
[> poll on
[> reg
> TC32 registers
(0) r0 (/32): 0x00000000
(1) r1 (/32): 0x00000000
(2) r2 (/32): 0x00000000
(3) r3 (/32): 0x00000000
(4) r4 (/32): 0x00000000
(5) r5 (/32): 0x00000000
(6) r6 (/32): 0x00000000
(7) r7 (/32): 0x00000000
(8) r8 (/32): 0x00000000
(9) r9 (/32): 0x00000000
(10) r10 (/32): 0x00000000
(11) r11 (/32): 0x00000000
(12) r12 (/32): 0x00000000
(13) sp_usr (/32): 0x00000000
(14) lr_usr (/32): 0x00000000
(15) pc (/32): 0x0000010C
(16) status (/32): 0x00000000
(17) sp_irq (/32): 0x00000000
```

- break <address>

Set a breakpoint at the hex address.

Unlike GDB, openocd does not have your source code symbol table. It won't know your symbol or source code line address. Useful place to obtain the correct address will be your .lst file created when you compile your program with Telink Eclipse IDE.

```
SensorMesh.bin  SensorMesh.lst  drivers      main.o      objects.mk  subdir.mk
SensorMesh.elf  boot      interrupt.o  makefile    sources.mk
Avoir:DEMO_G peters$

    3a0e:  977d 99d7    tjl dc0 <sys_init>
/Users/peters/sensor_mesh/DEMO_G/./main.c:17
    3a12:  0001    tmovs  r0, #1
    3a14:  a102    tmovs  r1, #2
    3a16:  97fe 9fab    tjl 2970 <Rf_Init>
/Users/peters/sensor_mesh/DEMO_G/./main.c:18
    WaitUs(10000);
    3a1a:  0b03    tloadr r3, [pc, #12] ; (3a28 <sys_init+0x4c>)
    3a1c:  ec18    tadds  r0, r3, #0
    3a1e:  97fc 9b67    tjl f0 <WaitUs>
/Users/peters/sensor_mesh/DEMO_G/./main.c:19
}
    3a22:  06bd    tmov   sp, r7
    3a24:  6007    tadd   sp, #28
    3a26:  6d80    tpop   {r7, pc}
    3a28:  00002710 .word  0x00002710

00003a2c <main>:
    .
    .

[> break 0x3a12
BP1 inserted at 0x00003A12
>
```

- breaks

Show the breakpoints.

TC32 has 4 hardware assisted breakpoints. Our first version of the OpenOCD only supports one breakpoint at this moment. Future releases will allow you to use more breakpoints.

```
[> breaks
BP0 addr 0x00000000 en[0] rc[0]
BP1 addr 0x00003A12 en[1] rc[0]
BP2 addr 0x00000000 en[0] rc[0]
BP3 addr 0x00800000 en[0] rc[0]
>
```

- breakc <bp#>

Clear the breakpoint. Breakpoint is normally inserted at BP1. If BP0 rc flag is on, it means MCU reached at the breakpoint regardless which breakpoint.

- breakclear

Clear all the breakpoints.

- until <address>

Run MCU until the hex address.

This is a very useful feature that allows you to run the MCU until the break address you want.

As you can see in this example, BP1 breakpoint were set to 3A14 and MCU stopped at the address. When rc flag is on, it means it reached the address.

To continue running, simply type resume.

```
[> until 3a14
Run until 0x00003a14
BP1 inserted at 0x00003A14
MCU halted
[> breaks
BP0 addr 0x00000000 en[0] rc[1]
BP1 addr 0x00003A14 en[0] rc[0]
BP2 addr 0x00800000 en[0] rc[0]
BP3 addr 0x00000000 en[0] rc[0]
> 
```

- breakr <address>

Remove breakpoint at address.

Same as breakc except you remove the breakpoint (turn off enable flag).