



Cypress FX3™ SDK

Using the FX3 SDK on Linux Platforms

Version 1.3.1

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
Phone (USA): 800.858.1810
Phone (Intl): 408.943.2600
<http://www.cypress.com>

Copyrights

Copyright © 2013 Cypress Semiconductor Corporation. All rights reserved.

FX3 and FX3S are the trademarks of Cypress Semiconductor. All other trademarks or registered trademarks referenced herein are the property of their respective owners.

The information in this document is subject to change without notice and should not be construed as a commitment by Cypress. While reasonable precautions have been taken, Cypress assumes no responsibility for any errors that may appear in this document. No part of this document may be copied or reproduced in any form or by any means without the prior written consent of Cypress. Made in the U.S.A.

Disclaimer

CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

License Agreement

Please read the license agreement during installation.

Contents



1	Introduction	4
1.1	SDK Components	5
2	SDK Installation.....	7
2.1	Pre-requisites	7
2.2	SDK Installation.....	7
3	Building the Firmware Examples	9
4	Debugging	11
4.1	J-Link GDB Server	11
4.2	Debug Configuration on the Eclipse IDE	12

1 Introduction

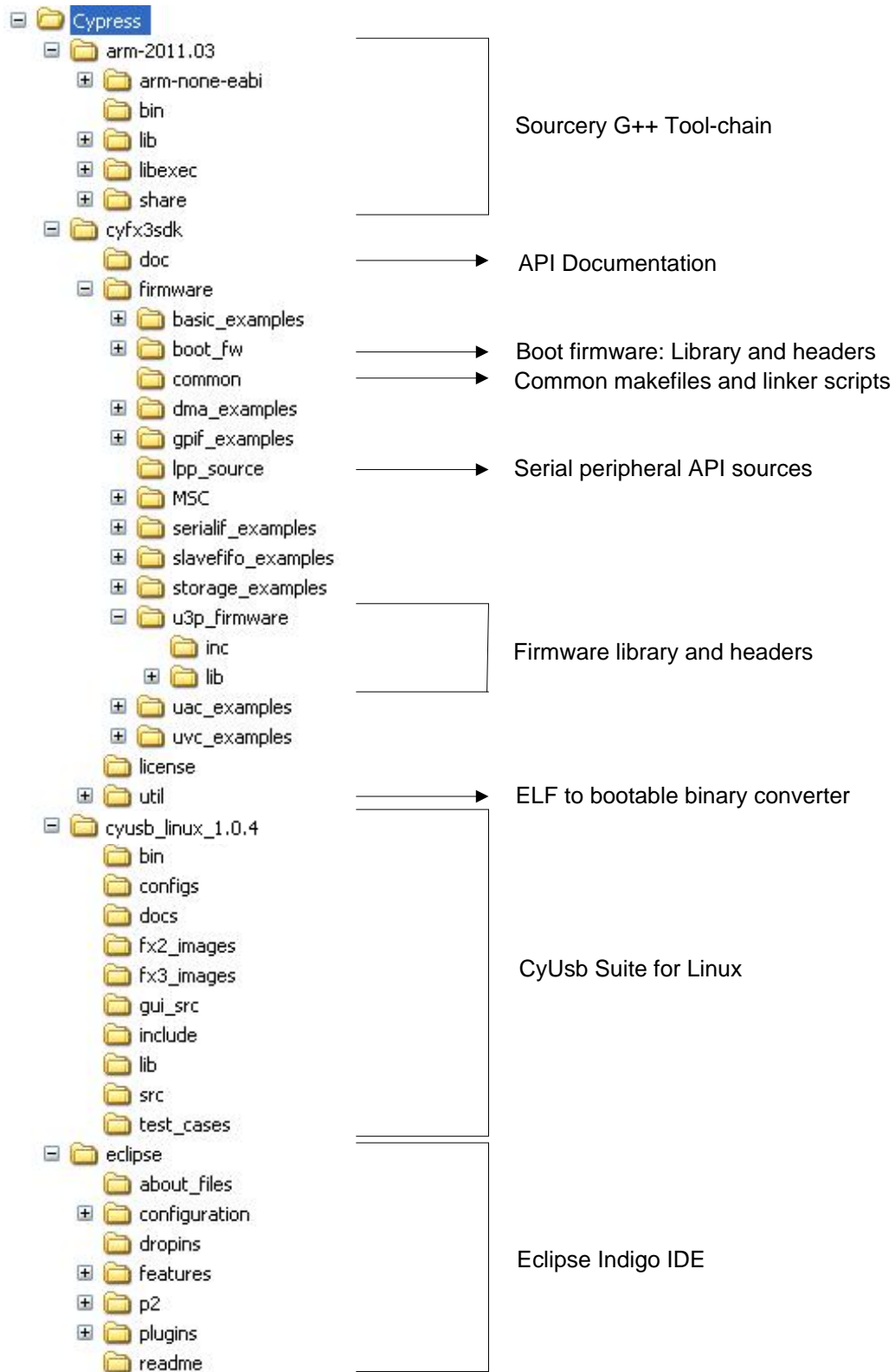


The FX3 SDK includes the FX3/FX3S firmware libraries and example firmware applications which can be used with the EZ-USB FX3 DVK, along with documentation on the firmware API and the programming model. All of the firmware examples have an Eclipse project associated with them and can be compiled using the Sourcery G++ Lite ARM EABI tool chain and the Eclipse IDE for C/C++ developers.

The FX3 SDK also includes a CyUSB library for Linux that allows users to develop user space applications to talk to generic USB devices; as well as a set of applications using this library that allow programming and testing data transfers with the FX3 device.

This 1.3.1 release of the FX3 SDK supports firmware development using the Eclipse IDE and debugging using the J-Link JTAG debugger probe on a Linux platform.

1.1 SDK Components





The FX3 SDK package for Linux is a gzipped tar archive that contains:

1. The FX3 firmware libraries and header files.
2. FX3 firmware API documentation and programmer's manual.
3. Firmware examples
4. CyUSB Suite for Linux – API library and applications for talking to generic USB devices connected to a host computer running Linux.
5. Sourcery G++ Lite – ARM EABI tool chain for compilation of FX3 firmware applications.
6. Eclipse Indigo package with the following plugins:
 - a. GNU ARM Eclipse Plugin for managed firmware builds.
 - b. Zylind Embedded CDT Plugin for debug support

Note: The Sourcery G++ Lite package is originally from Mentor Graphics (Code Sourcery) and provided here for convenience. The Eclipse Indigo package is from the Eclipse foundation and provided here along with the required plugins for convenience.

2 SDK Installation



2.1 Pre-requisites

The following tools are required for proper functioning of various components of the FX3 SDK.

1. Java Runtime Environment – The Eclipse IDE requires a recent version of the Java Runtime Environment (JRE) or Java Development Kit (JDK).

Note: We have tested the packaged versions with OpenJDK Java 6 Runtime and OpenJDK Java 7 Runtime versions.

2. A native C compiler for the host computer is required to compile the elf2img converter program that converts ELF firmware binaries into the bootable .img file format.
3. If the Linux installation on the host computer is a 64-bit OS version, 32-bit system libraries need to be installed for the GNU ARM toolchain to work. Please see <https://sourcery.mentor.com/GNUToolchain/kbentry62> for more details.

Note: As the GUI elements of the Sourcery G++ Lite are not being used, only the 32-bit system libraries are required and there is not need to install Xulrunner. On a Debian/Ubuntu Linux distribution, you can install the required libraries by executing the command:

```
apt-get install ia32-libs
```

4. The J-Link driver library and GDB server program for Linux are required if you wish to run a JTAG debug session from the Linux platform. A beta version of the J-Link driver and GDB server can be downloaded from <http://www.segger.com/jlink-software.html>. Please follow the instructions in the associated README file for installing the J-Link software.

2.2 SDK Installation

The EZ-USB FX3 SDK for Linux is released in the form of a gzipped tar archive called FX3_SDK.tar.gz. On extraction, this tar archive contains four other gzipped tar archives:

1. FX3_Firmware.tar.gz: The FX3 firmware library and examples.
2. ARM_GCC.tgz: Sourcery ARM GNU toolchain.
3. eclipse_x86.tgz: Eclipse IDE for 32-bit Linux OS installations.
4. eclipse_x64.tgz: Eclipse IDE for 64-bit Linux OS installations.
5. cyusb_linux_1.0.4.tar.gz: CyUSB Suite for Linux OS.

The installation procedure involves extraction of these archives and the setting of a couple of environment variables, and is enumerated below.

1. Extract the contents of the FX3_SDK.tar.gz archive at a preferred location, say, \$HOME/Cypress.
2. Change to the install location (\$HOME/Cypress) and extract the contents of the FX3_Firmware.tar.gz, ARM_GCC.tgz, cyusb_linux_1.0.4.tar.gz and eclipse_x86.tgz (or eclipse_x64.tgz) files. This will create a directory structure as shown in the figure under section 1.1.

Note: Only one of the eclipse archives need to be extracted. Select the appropriate archive based on the OS type (32 bit or 64 bit).

3. Add the folder containing the ARM GNU toolchain binaries to the PATH environment variables.

e.g.: `export PATH=$PATH:$HOME/Cypress/arm-2011.03/bin`

4. Create an environment variable called FX3_INSTALL_PATH that points to the directory where the FX3 firmware package has been extracted.

e.g.: `export FX3_INSTALL_PATH=$HOME/Cypress/cyfx3sdk`

5. Create an environment variable called ARMGCC_INSTALL_PATH that points to the directory where the ARM GNU toolchain has been extracted.

e.g.: `export ARMGCC_INSTALL_PATH=$HOME/Cypress/arm-2011.03`

6. Create an environment variable called ARMGCC_VERSION which is set to the Sourcery Lite GNU ARM toolchain version (4.5.2 for this release).

e.g.: `export ARMGCC_VERSION=4.5.2`

7. Change to the Cypress/cyfx3sdk/util/elf2img folder and compile the elf2img program that converts ELF firmware binaries into the .img binaries that can be used to boot the FX3 device.

e.g.: `cd $FX3_INSTALL_PATH/util/elf2img`
`gcc elf2img.c -o elf2img -Wall`

8. Change the Cypress/cyusb_linux_1.0.4 folder and follow the instructions in the README file for compiling the CyUSB library and cyusb_linux GUI application. Refer to the documentation in the Cypress/cyusb_linux_1.0.4/doc folder for instructions on how to use these tools.

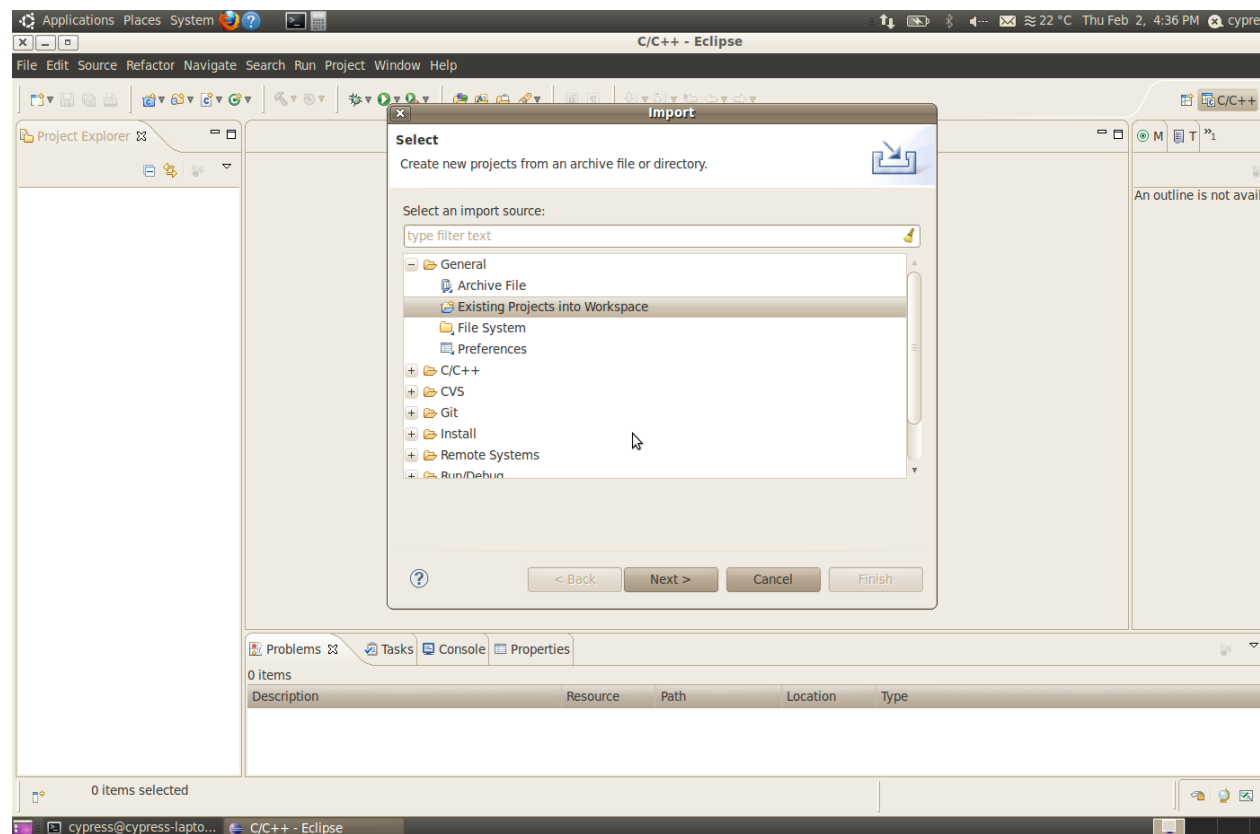
9. Create an environment variable called CYUSB_ROOT that points to the directory where the CYUSB package has been extracted. This variable is used by the cyusb_linux utility to look for the FX3 I2C/SPI programming firmware.

e.g.: `export CYUSB_ROOT=$HOME/Cypress/cyusb_linux_1.0.4`

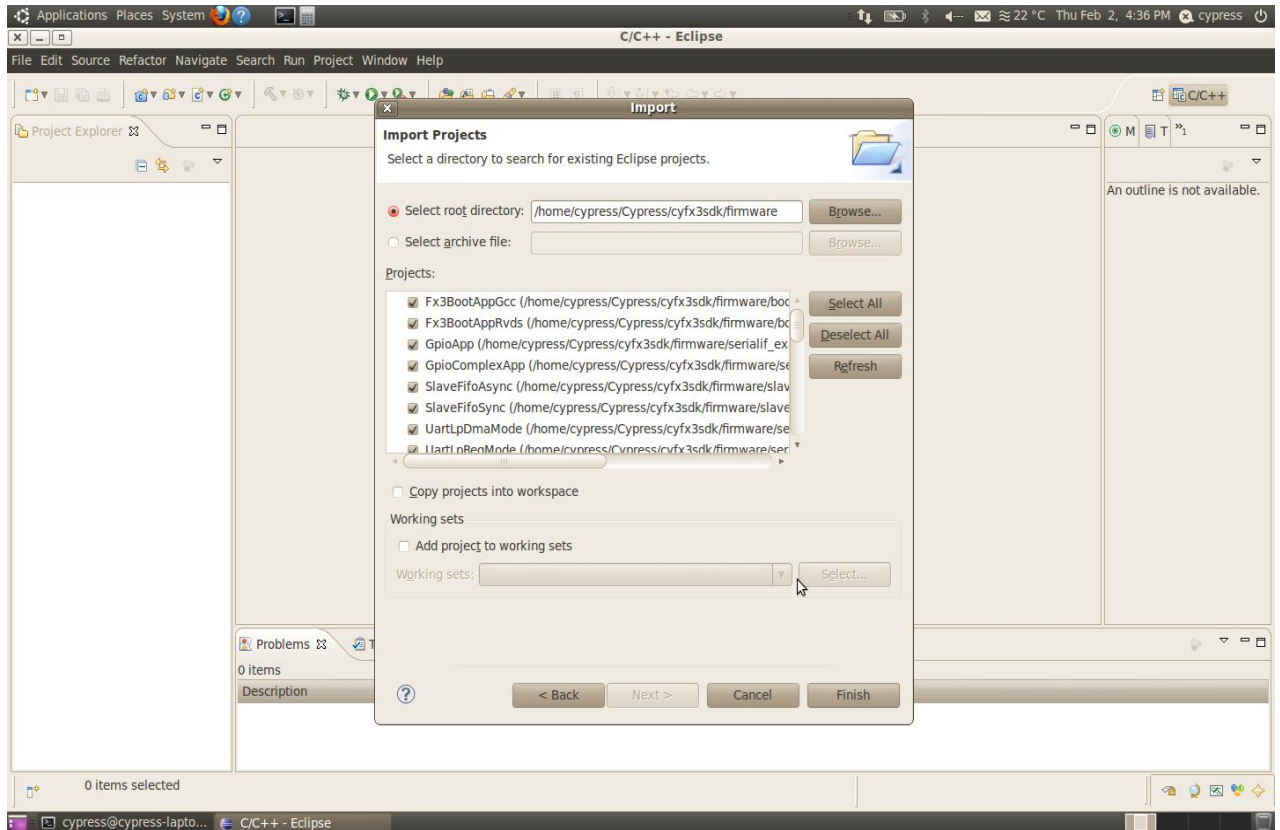
3 Building the Firmware Examples

Once the FX3 SDK components have all been installed, the Eclipse IDE can be used to open and build all the firmware example projects.

1. Open the Eclipse IDE (execute `$HOME/Cypress/eclipse/eclipse`) and select a workspace location such as `$HOME/workspace`
2. Select the File -> Import menu option to bring up the Import Dialog and choose the General -> Existing Projects into Workspace option as shown below.



3. In the file browser popup that comes up, select `$HOME/Cypress/cyfx3sdk/firmware` (or the corresponding folder based on your installation) as the root folder to search for projects.



4. On selecting OK, all of the firmware example projects that are part of the FX3 SDK will be imported into the workspace.
5. Select the Project -> Build All menu option to build all of the projects, or right click on a project name and select Build Project to build only the selected project. The Debug configuration for the Project will be selected by default and can be changed to Release through the Build Configurations -> Set Active menu option.
6. The firmware binary in img format can be copied onto a Windows USB host machine and loaded onto the FX3 device RAM or a connected EEPROM or FLASH device using the Cypress Superspeed USB Suite tools.

4 Debugging

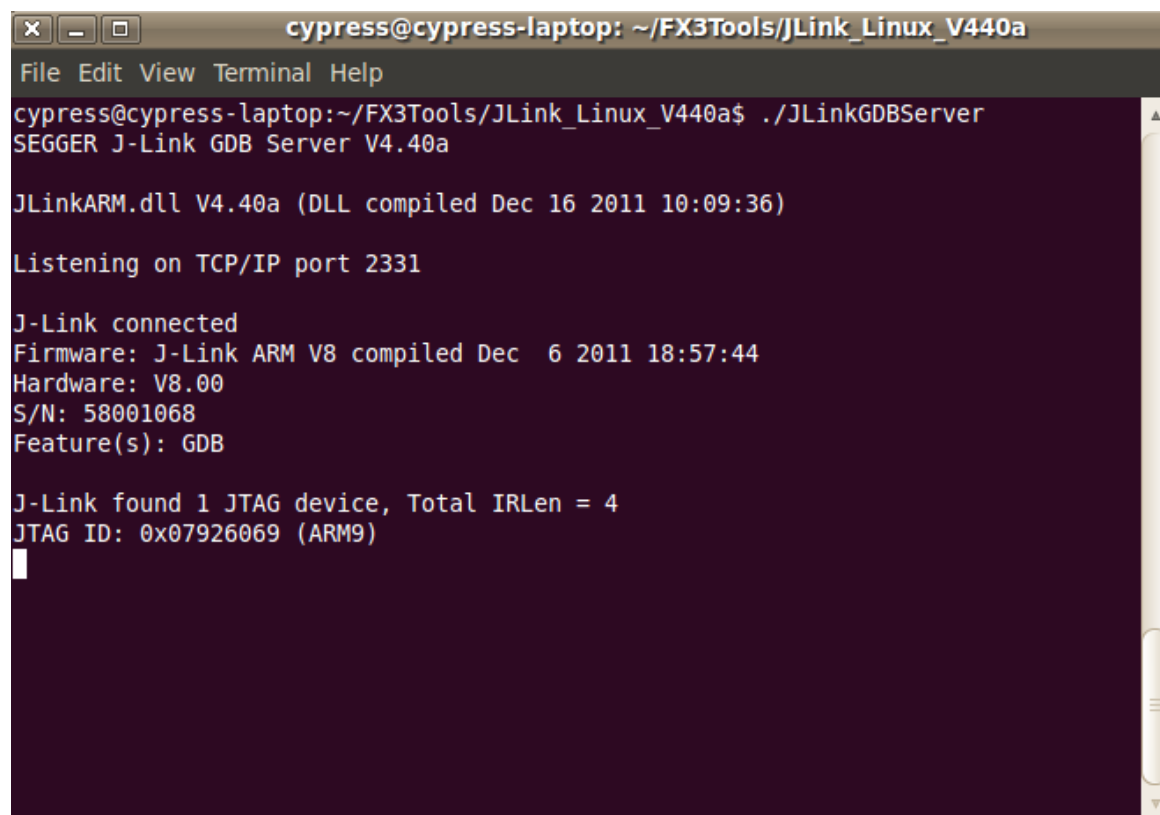
Once the firmware binaries have been built, the J-Link JTAG probe and the GDB debugger can be used to debug the firmware running on the FX3 DVK.

4.1 J-Link GDB Server

The J-Link GDB server software from Segger systems is required to enable this. A beta version of the GDB server program and the J-Link driver can be downloaded from <http://www.segger.com/cms/jlink-software.html>

Connect the J-Link JTAG probe to the host computer using a USB cable and connect the JTAG connector to the 20 pin JTAG port on the FX3 DVK. Power up the FX3 DVK using the 5V power adapter.

Now, start the GDB server program. If all connections are proper and the drivers are in place, the GDB server output will look as below.



```
cyress@cypress-laptop: ~/FX3Tools/JLink_Linux_V440a
File Edit View Terminal Help
cyress@cypress-laptop:~/FX3Tools/JLink_Linux_V440a$ ./JLinkGDBServer
SEGGER J-Link GDB Server V4.40a

JLinkARM.dll V4.40a (DLL compiled Dec 16 2011 10:09:36)

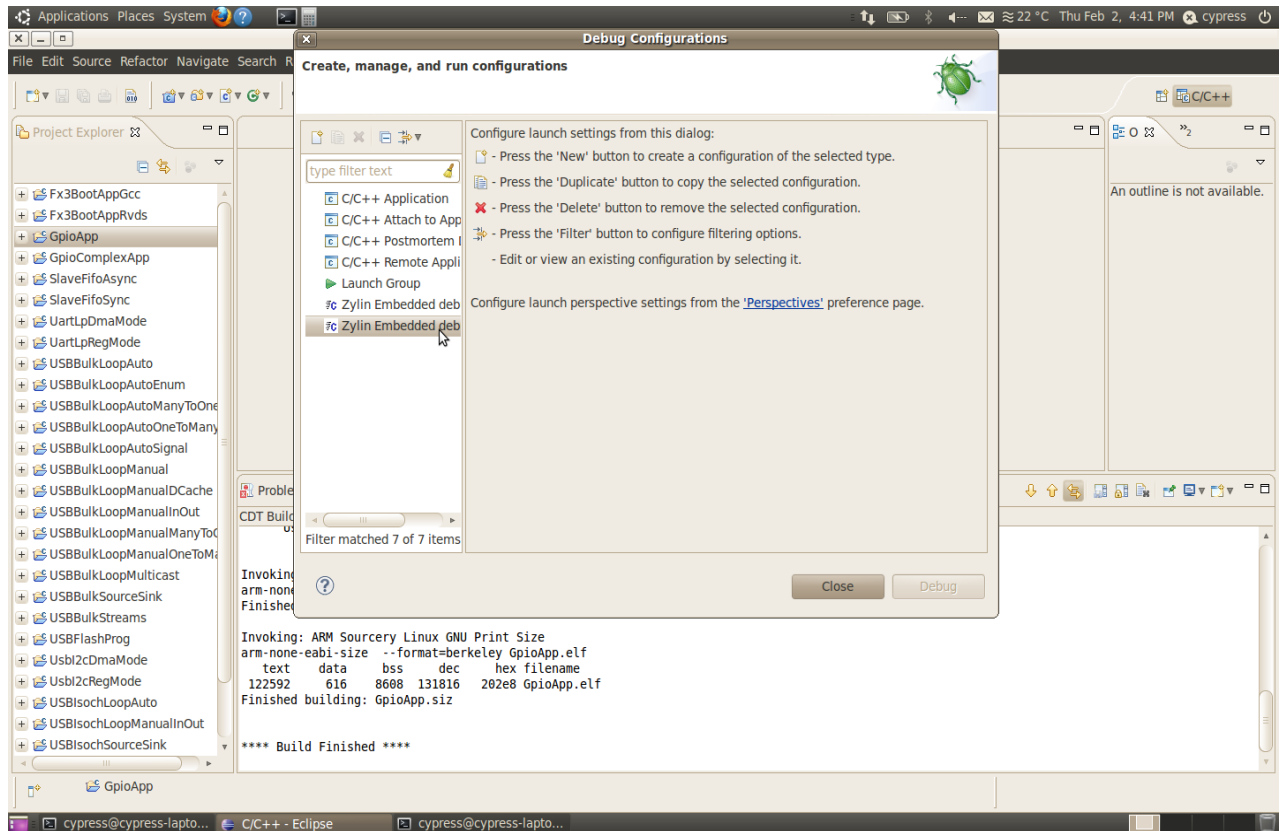
Listening on TCP/IP port 2331

J-Link connected
Firmware: J-Link ARM V8 compiled Dec  6 2011 18:57:44
Hardware: V8.00
S/N: 58001068
Feature(s): GDB

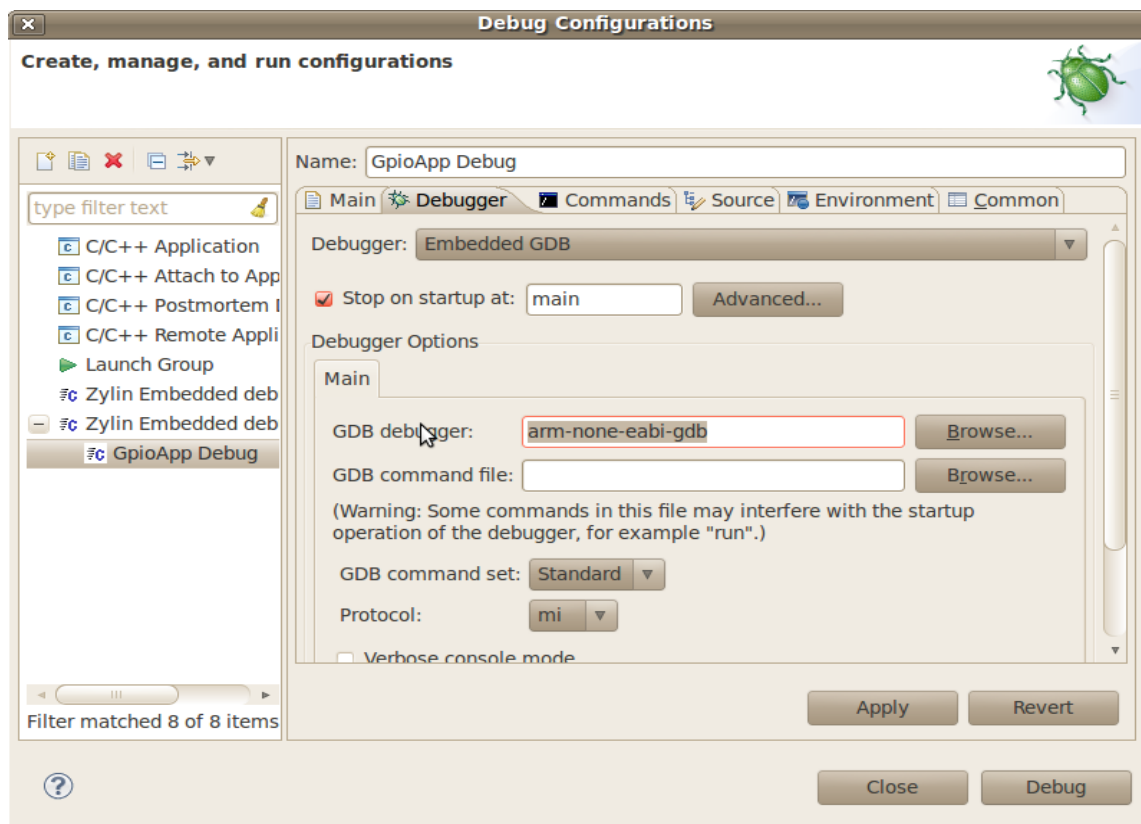
J-Link found 1 JTAG device, Total IRLen = 4
JTAG ID: 0x07926069 (ARM9)
```

4.2 Debug Configuration on the Eclipse IDE

1. Select the firmware project to be debugged, right-click and select the Debug As -> Debug Configurations option to bring up a dialog box as shown below.



2. Right-click on the Zylin Embedded debug (Native) option and select New to create a new debug configuration.
3. On the debugger tab, specify arm-none-eabi-gdb as the GDB debugger program and leave the GDB command file option empty.



4. On the Commands tab, enter the following set of commands as the 'Initialize' commands:

```
set prompt (arm-gdb)
# This connects to a target on this PC's tcp port 2331
target remote localhost:2331
monitor speed 1000
monitor endian little
set endian little
monitor reset

# Set the processor to SVC mode
monitor reg cpsr =0xd3

# Disable all interrupts
monitor memU32 0xFFFFF014 =0xFFFFFFFF

# Code to enable the TCMS
monitor memU32 0x40000000 =0xE3A00015
monitor memU32 0x40000004 =0xEE090F31
monitor memU32 0x40000008 =0xE240024F
monitor memU32 0x4000000C =0xEE090F11

# If required, set the FX3 system clock to faster than 400 MHz.
# This is done here to prevent debug session errors due to the
# clock being changed at runtime.
# Update with the correct value from list below.
# Clock input is 19.2 MHz: Value = 0x00080015
# Clock input is 38.4 MHz: Value = 0x00080115
```

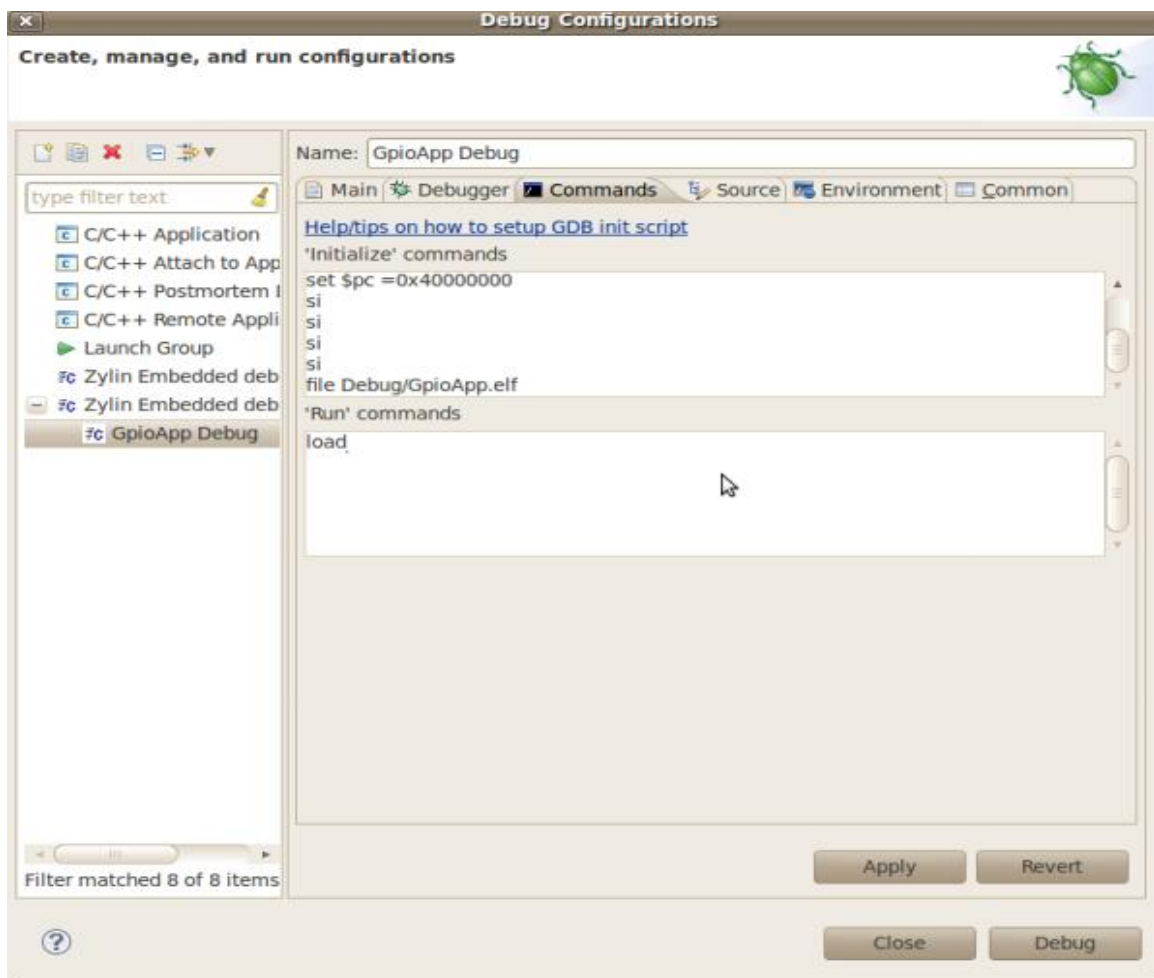
```
monitor memU32 0xE0052000 = 0x00080015
# Add a delay to let the clock stabilize.
monitor sleep 1000

# Execute the code to enable the TCMS
set $pc =0x40000000
si
si
si
si

# Select the file to be debugged
file Debug/GpioApp.elf
```

Note: The script shown above needs to be updated to select the correct value to be loaded into the register at 0xE0052000 as shown in the embedded comments. The elf filename to be loaded should also be changed to correspond with the project being debugged.

5. On the Commands tab, specify “load” as the only ‘Run’ command.



6. Select Apply and then click on Debug to start debugging the application.
7. The standard debug menu options can be used to set breakpoints, step into or over function calls etc.