Cypress EZ-USB® FX3™ SDK

Using the FX3 SDK on Linux Platforms

Version 1.3.4

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

Contents

1 Introduction

The FX3 SDK includes the FX3/FX3S/CX3 firmware libraries and example firmware applications which can be used with the CYUSB3KIT-003 kit from Cypress, along with documentation on the firmware API and the programming model. All 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 1.3.4 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

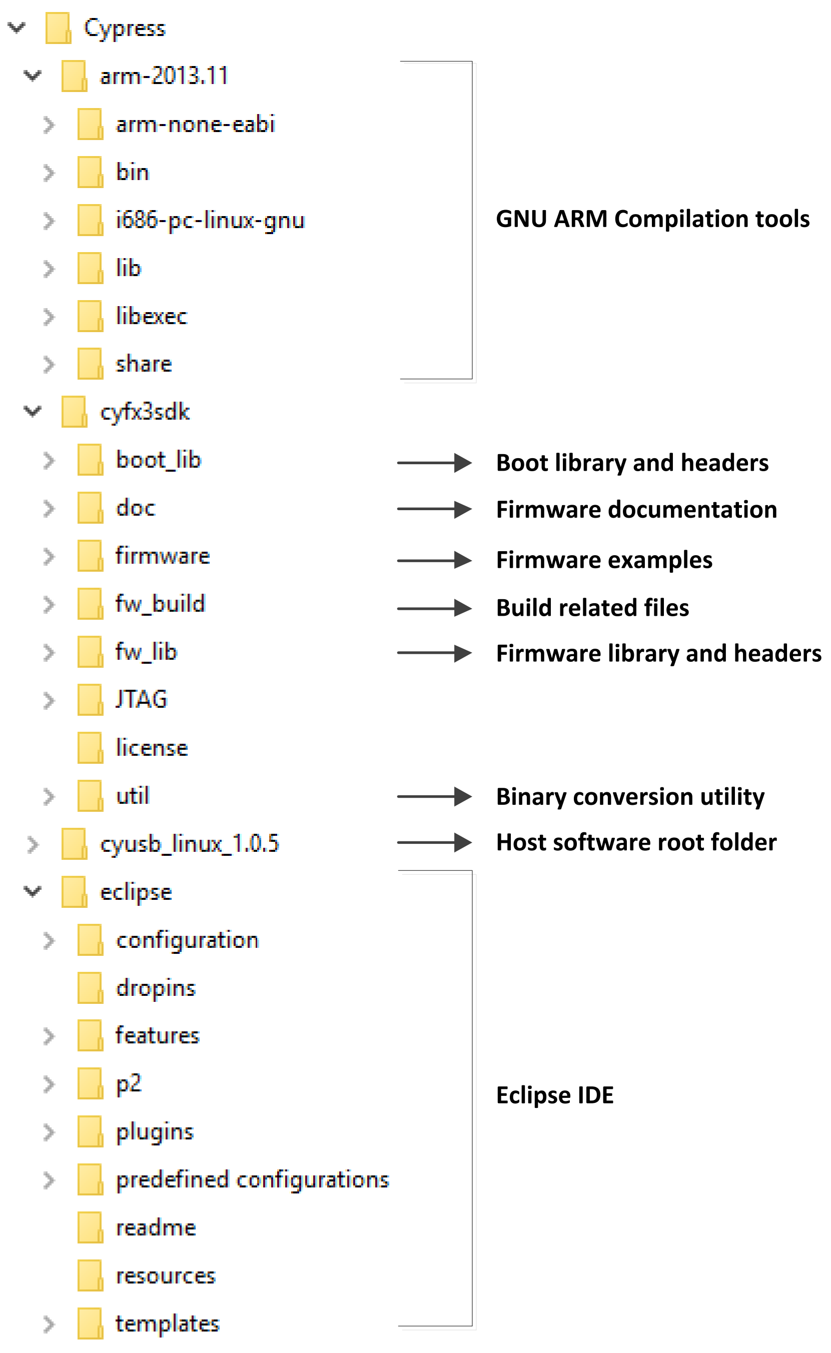


Figure 1-1: FX3 SDK directory structure

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 IDE package with the following plugins:
    a. GNU ARM Eclipse Plugin for managed firmware builds and debugging.

Note: The Sourcery G++ Lite package is originally from Mentor Graphics (Code Sourcery) and provided here for convenience. The Eclipse 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 Version 7 of the Java Runtime Environment (JRE) or Java Development Kit (JDK).
2. A native C compiler (such as gcc) 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 lib32z1
   ```

4. The J-Link driver library and GDB server program for Linux are required if you wish to run a JTAG debug session using the Segger J-Link debug probe. 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.
5. The SDCC compiler suite is required if you intend to use the Eclipse IDE to develop firmware for the FX2LP parts. This can be downloaded from the SDCC site at http://sdcc.sourceforge.net/. On a Debian/Ubuntu Linux distribution, you can install the latest version by executing the command:

   ```
   apt-get install sdcc
   ```

2.2 SDK Installation

The EZ-USB FX3 SDK for Linux is released in the form of a gzipped tar archive called FX3_SDK_Linux.tar.gz. On extraction, this tar archive contains four other gzipped tar archives:

1. fx3_firmware_linux.tar.gz: The FX3 firmware library and examples.
2. ARM_GCC.tar.gz: Sourcery ARM GNU toolchain.
3. eclipse_x86.tar.gz: Eclipse IDE for 32-bit Linux OS installations.
4. eclipse_x64.tar.gz: Eclipse IDE for 64-bit Linux OS installations.
5. cyusb_linux_1.0.5.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 detailed below.

1. Extract the contents of the FX3_SDK_Linux.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_linux.tar.gz, ARM_GCC.tar.gz, cyusb_linux_1.0.5.tar.gz and eclipse_x64.tar.gz (or eclipse_x86.tar.gz) 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-2013.11/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-2013.11`
6. Create an environment variable called ARMGCC_VERSION which is set to the Sourcery Lite GNU ARM toolchain version (4.8.1 for this release).

   e.g.: `export ARMGCC_VERSION=4.8.1`
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_1.0.5 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_1.0.5`
10. Please note that the environment updates made above will only be visible to processes started from the console where these commands are executed. The variables can be made persistent across sessions and accessible to applications launched from the dock by setting them in the /etc/environment file.

3 Building the Firmware Examples

Please refer to chapter 2 in the EZUSBSuite_UG.pdf document in the cyfx3sdk/doc/firmware folder.

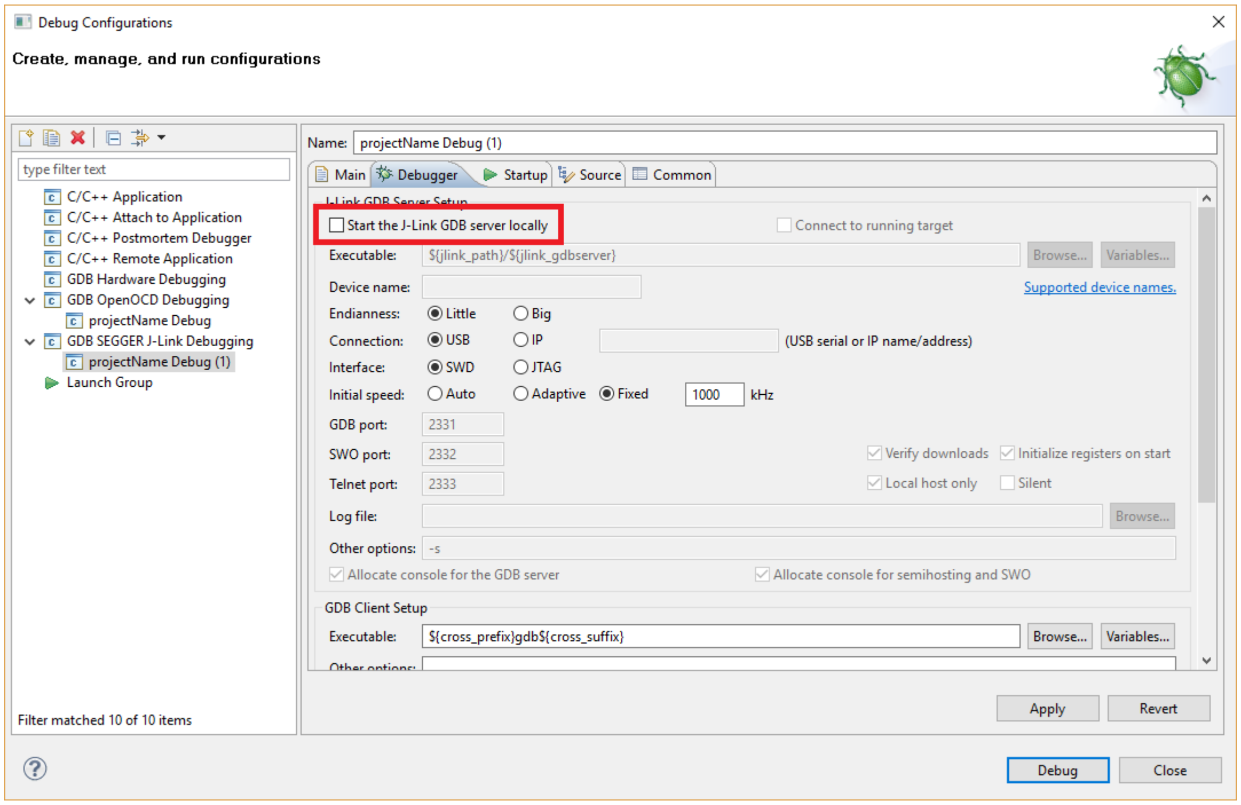
4 Debugging

Please refer to Chapter 3 in the EZUSBSuite_UG.pdf document in the cyfx3sdk/doc/firmware folder for instructions on setting up the debug options in the Eclipse IDE.

4.1 Debugging using Segger J-Link

If the Segger J-Link debugger is being used for debugging, download and install the J-Link GDB server program from https://www.segger.com/downloads/jlink/#J-LinkSoftwareAndDocumentationPack

On Linux platforms, it is recommended that you start the GDB server program as a separate process and configure the Eclipse IDE to not start the J-Link GDB server locally.



Debug Configurations

Create, manage, and run configurations

- C/C++ Application
- C/C++ Attach to Application
- C/C++ Postmortem Debugger
- C/C++ Remote Application
- GDB Hardware Debugging
- GDB OpenOCD Debugging
  - projectName Debug
- GDB SEGGER J-Link Debugging
  - projectName Debug (1)
- Launch Group

Filter matched 10 of 10 items

Name: projectName Debug (1)

Main | Debugger | Startup | Source | Common

J-Link GDB Server Setup

- [ ] Start the J-Link GDB server locally
- [ ] Connect to running target

Executable: ${jlink_path}/${jlink_gdbserver}

Device name:

Endianness: [✓ Little] Big

Connection: [✓ USB] IP (USB serial or IP name/address)

Interface: [✓ SWD] JTAG

Initial speed: Auto Adaptive [✓ Fixed] 1000 kHz

GDB port: 2331

SWO port: 2332

Telnet port: 2333

- [x] Verify downloads
- [x] Initialize registers on start
- [x] Local host only
- [ ] Silent

Log file:

Other options: -s

- [x] Allocate console for the GDB server
- [x] Allocate console for semihosting and SWO

GDB Client Setup

Executable: ${cross_prefix}gdb${cross_suffix}

Other options:

Figure 2: Debug configuration to be used when running GDB server separately

4.2 Debugging using OpenOCD on the CYUSB3KIT-003

4.2.1 Preparing to run the OpenOCD binary

The SDK includes pre-compiled OpenOCD binaries using the Cypress USB-Serial device on the CYUSB3KIT-003 as the JTAG transport. The binaries can be found in the cyfx3sdk/JTAG/OpenOCD/Linux/x64 and cyfx3sdk/JTAG/OpenOCD/Linux/x86 folders.

These binaries depend on libusb and libcyusb dynamic libraries that are provided in the same directory. These libraries (.so files) will need to be copied into the system library folder (/usr/local/lib or similar) for the application to work properly. Please refer to the README.txt file in these folders for instructions to complete this.

You can test whether the above step is successful by running the command from a terminal:

```
./Linux/x64/openocd.exe -f Config/arm926ejs_fx3.cfg
```

In some cases, the /usr/local/lib folder is not part of the library search path used by the Operating System. If so, you may receive an error like the following:

```
openocd: error while loading shared libraries: libcyusbserial.so:
cannot open shared object file: No such file or directory
```

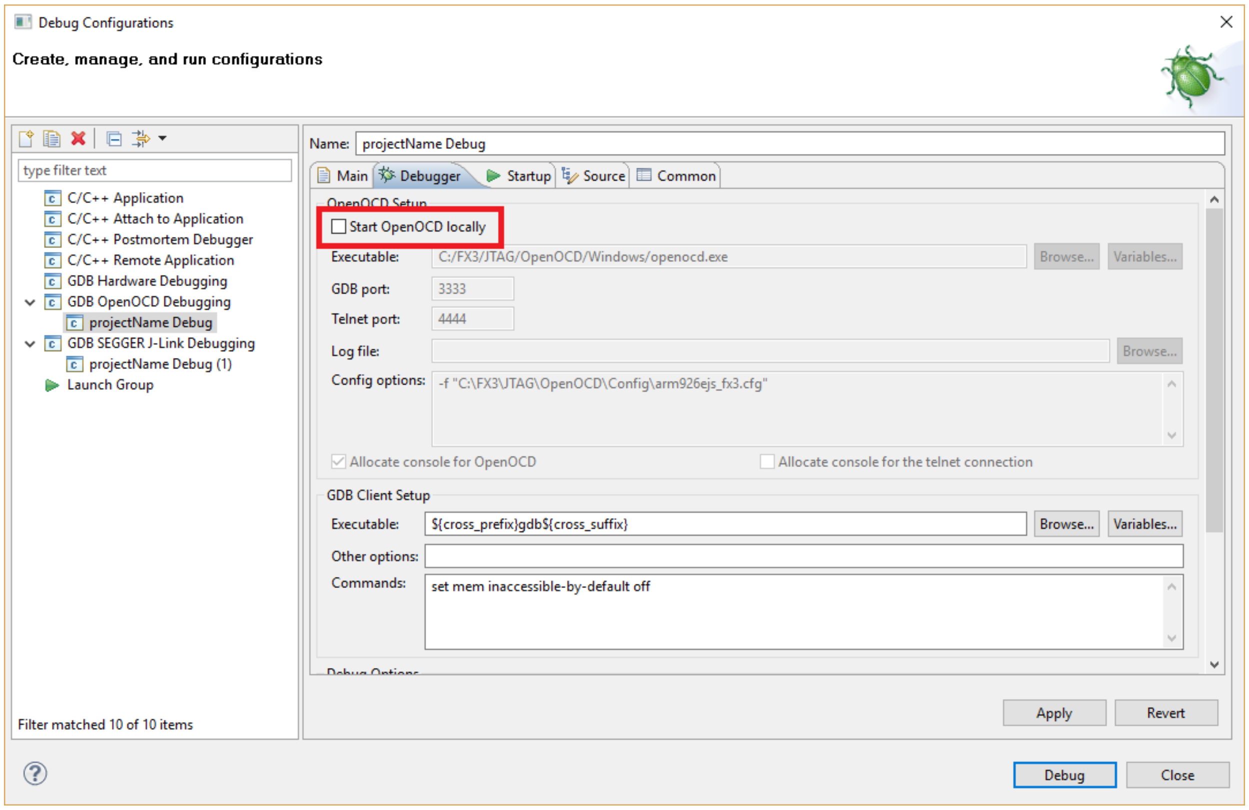
In such a case, please use ldconfig to ensure that /usr/local/lib is added to the library search path. You can refer to https://stackoverflow.com/questions/4743233/is-usr-local-lib-searched-for-shared-libraries for instructions.

4.2.2 Running the debug session

As in the case of the J-Link debugger, it is recommended that you run OpenOCD as a separate process and then configure the Eclipse debug configuration to not start the OpenOCD process.

Run the following command from a Terminal session and then update the debug settings as shown in Figure 3.

```
./Linux/x64/openocd.exe -f Config/arm926ejs_fx3.cfg
```



Debug Configurations

Create, manage, and run configurations

Name: projectName Debug

Main | Debugger | Startup | Source | Common

OpenOCD Setup

[ ] Start OpenOCD locally

Executable: C:/FX3/JTAG/OpenOCD/Windows/openocd.exe

GDB port: 3333

Telnet port: 4444

Log file:

Config options: -f "C:\FX3\JTAG\OpenOCD\Config\arm926ejs_fx3.cfg"

[✓] Allocate console for OpenOCD   [ ] Allocate console for the telnet connection

GDB Client Setup

Executable: ${cross_prefix}gdb${cross_suffix}

Other options:

Commands: set mem inaccessible-by-default off

- C/C++ Application
- C/C++ Attach to Application
- C/C++ Postmortem Debugger
- C/C++ Remote Application
- GDB Hardware Debugging
- GDB OpenOCD Debugging
  - projectName Debug
- GDB SEGGER J-Link Debugging
  - projectName Debug (1)
- Launch Group

Filter matched 10 of 10 items

Figure 3: Debug configuration to be used when running OpenOCD separately