Cypress EZ-USB® FX3™ SDK

EZ-USB Suite User Guide

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 EZ-USB Suite

1.1 Introduction

EZ-USB Suite is the integrated development environment provided by Cypress for firmware development and debugging using the EZ-USB FX3 and associated parts.

EZ-USB Suite is based on the standard Eclipse Kepler IDE for C/C++ Developers, and provides a few customizations that are FX3 device family specific.

1.2 IDE Features

The EZ-USB Suite IDE provides firmware development and debugging support for FX3 applications. It also provides a mechanism to create new FX3 firmware projects, and plug-ins for through which other FX3 applications such as Control Center and GPIF-II Designer can be accessed.

The IDE is integrated with the GNU ARM tool-chain for firmware compilation, linking and debugging. The ARM GNU Eclipse plug-in is used for managed builds and debugging.

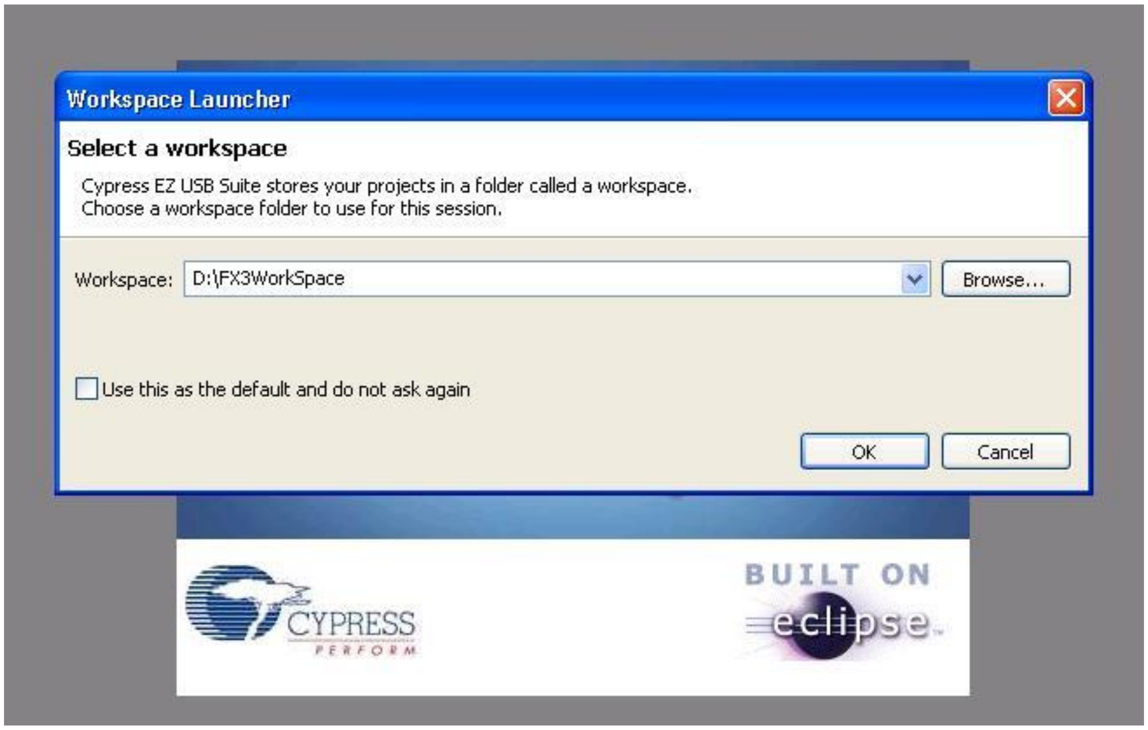
The following chapters provide more detailed description on the IDE features.

2 Creating an FX3 Firmware Project

This chapter outlines the steps involved in creating a FX3 firmware application, using the GpifToUsb project from the SDK as a reference.

2.1 Starting the EZ-USB Suite IDE

Launch the EZ-USB Suite IDE by following the “All Programs -> Cypress -> Eclipse -> EZ USB Suite” path from the Start Menu.


Workspace Launcher

Select a workspace

Cypress EZ USB Suite stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: D:\FX3WorkSpace

☐ Use this as the default and do not ask again

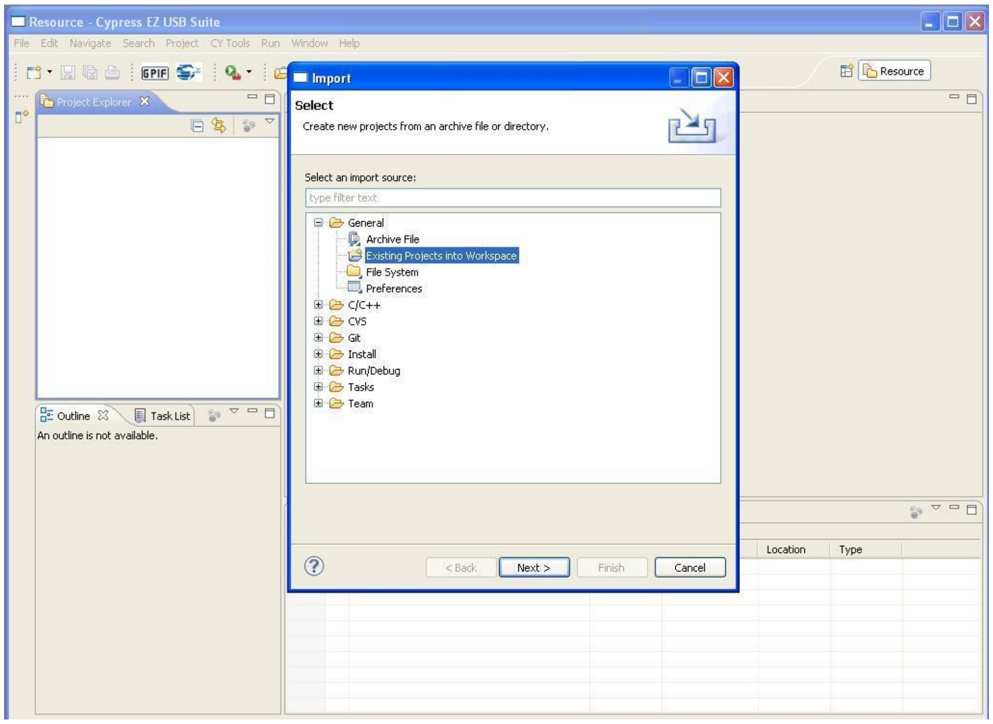
Figure 2-1: Workspace Selection Dialog

The IDE will prompt for a Workspace to be opened. If you have already created a workspace, select it from the dropdown box. Otherwise, provide the path where the new workspace has to be created and Click OK.

2.2 Creating the Project

The easiest way to create a new FX3 firmware project under the Eclipse IDE, is to import an existing project and make changes to it.

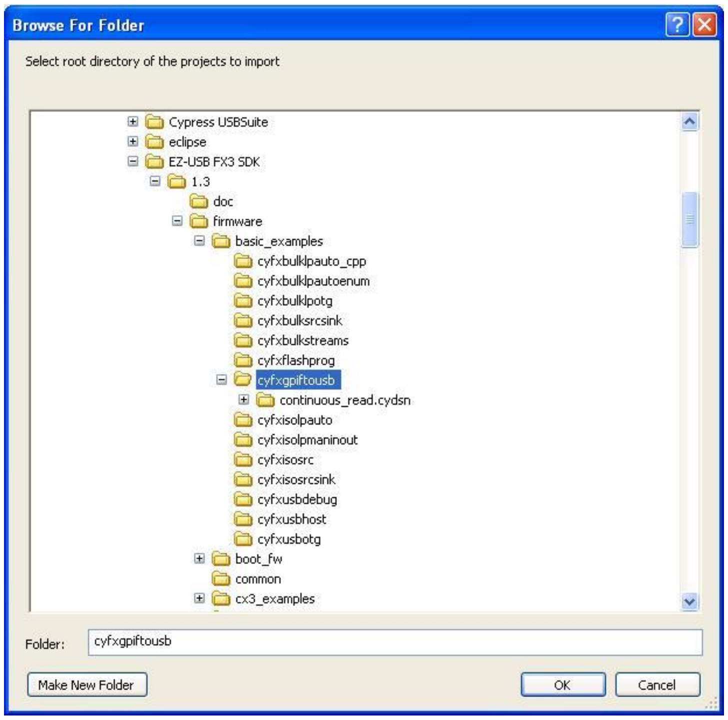
1. Choose the Import option under the File menu to bring up the Import options window. Choose “Existing Projects into Workspace” from the “General” group.



An outline is not available.

Figure 2-2: Project Import Dialog

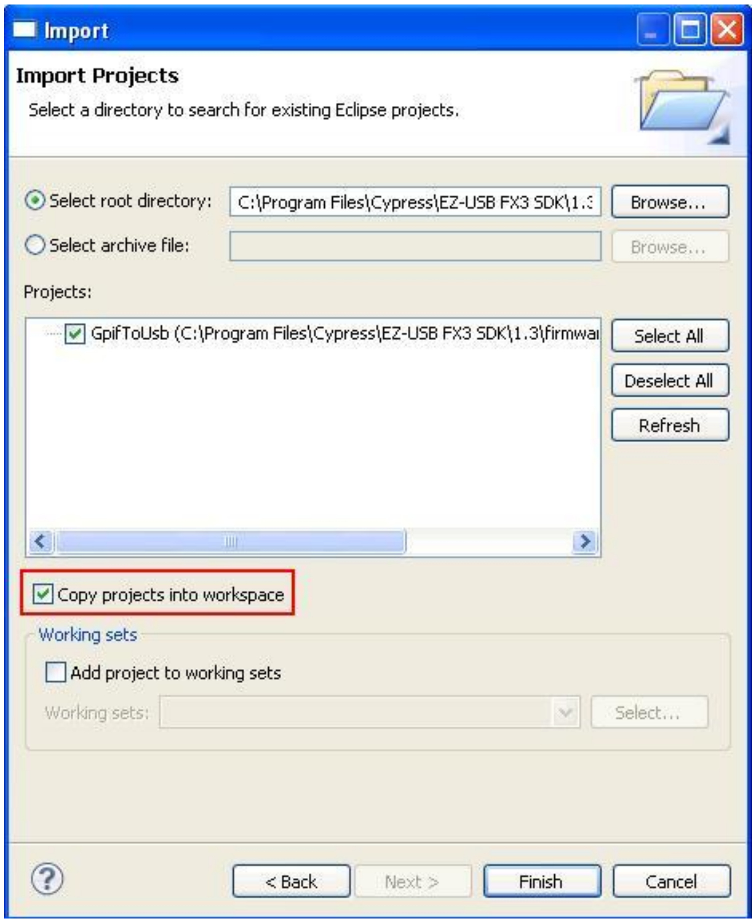
2. In the “Browse for Folder” window that pops up, navigate into the firmware/basic_examples/cyfxgpiftousb folder in the FX3 SDK installation.



Folder: cyfxgpiftousb

Figure 2-3: Folder Browse Dialog

3. Select the “Copy projects into workspace” option in the Import window, so that we get a new copy of the source files that can be modified.



Import Projects

Select a directory to search for existing Eclipse projects.

Select root directory: C:\Program Files\Cypress\EZ-USB FX3 SDK\1.3

Select archive file:

Projects:

GpifToUsb (C:\Program Files\Cypress\EZ-USB FX3 SDK\1.3\firmwa

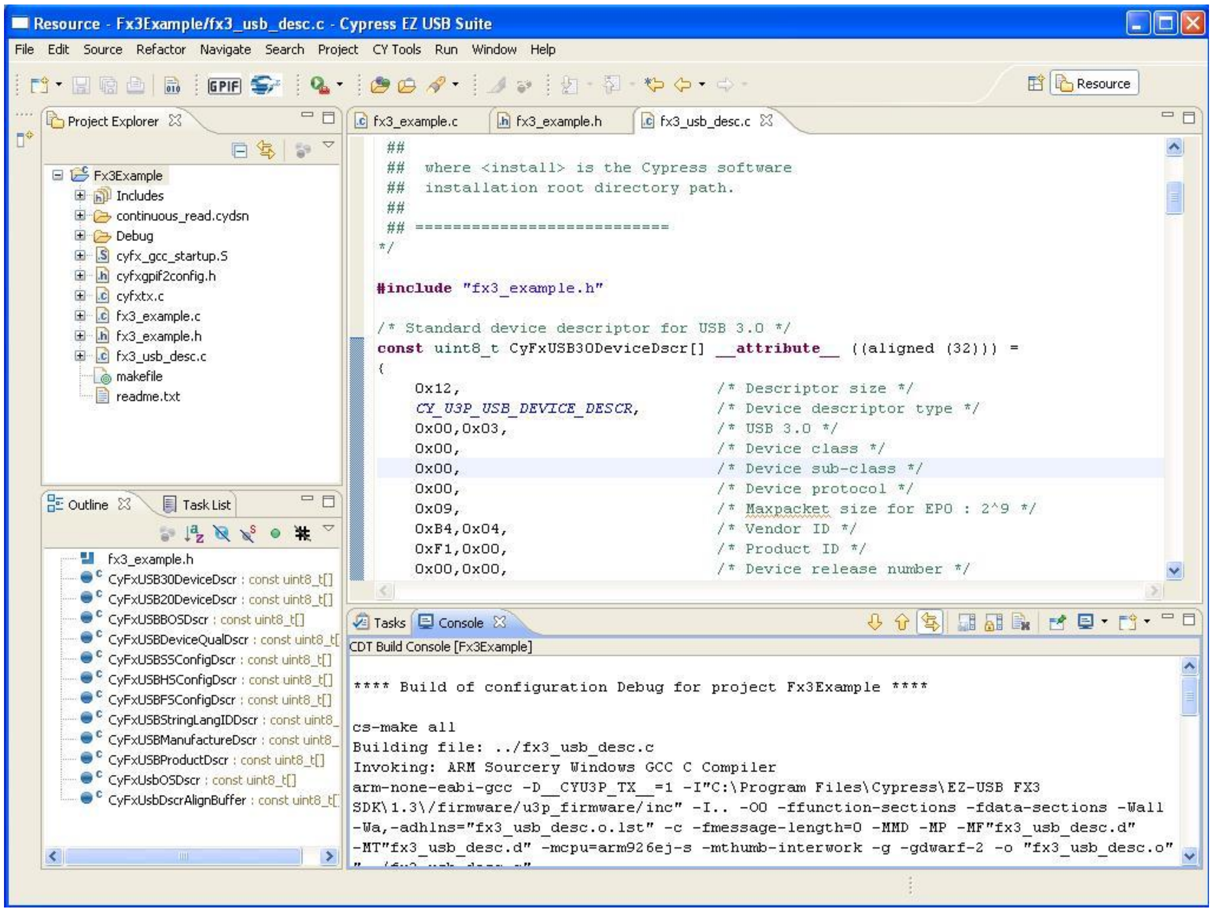
[✓] Copy projects into workspace

Working sets

[ ] Add project to working sets

Figure 2-4: Import Projects Dialog

4. If desired, right-click the project name and use the “Rename” option to rename the project. The respective source files in the project can also be renamed as desired. If the header files are renamed, references to the header file in the source files will need to be updated.


Resource - Fx3Example/fx3_usb_desc.c - Cypress EZ USB Suite
File Edit Source Refactor Navigate Search Project CY Tools Run Window Help
Resource
Project Explorer
Fx3Example
Includes
continuous_read.cydsn
Debug
cyfx_gcc_startup.S
cyfxgpif2config.h
cyfxtx.c
fx3_example.c
fx3_example.h
fx3_usb_desc.c
makefile
readme.txt
Outline
Task List
fx3_example.h
CyFxUSB30DeviceDscr : const uint8_t[]
CyFxUSB20DeviceDscr : const uint8_t[]
CyFxUSBBOSDscr : const uint8_t[]
CyFxUSBDeviceQualDscr : const uint8_t[
CyFxUSBSSConfigDscr : const uint8_t[]
CyFxUSBHSConfigDscr : const uint8_t[]
CyFxUSBFSConfigDscr : const uint8_t[]
CyFxUSBStringLangIDDscr : const uint8_
CyFxUSBManufactureDscr : const uint8_
CyFxUSBProductDscr : const uint8_t[]
CyFxUsbOSDscr : const uint8_t[]
CyFxUsbDscrAlignBuffer : const uint8_t[
fx3_example.c
fx3_example.h
fx3_usb_desc.c
Tasks
Console
CDT Build Console [Fx3Example]

```
 ##
 ##  where <install> is the Cypress software
 ##  installation root directory path.
 ##
 ## ===========================
*/

#include "fx3_example.h"

/* Standard device descriptor for USB 3.0 */
const uint8_t CyFxUSB30DeviceDscr[] __attribute__ ((aligned (32))) =
{
    0x12,                           /* Descriptor size */
    CY_U3P_USB_DEVICE_DESCR,        /* Device descriptor type */
    0x00,0x03,                      /* USB 3.0 */
    0x00,                           /* Device class */
    0x00,                           /* Device sub-class */
    0x00,                           /* Device protocol */
    0x09,                           /* Maxpacket size for EP0 : 2^9 */
    0xB4,0x04,                      /* Vendor ID */
    0xF1,0x00,                      /* Product ID */
    0x00,0x00,                      /* Device release number */
```

```
**** Build of configuration Debug for project Fx3Example ****

cs-make all
Building file: ../fx3_usb_desc.c
Invoking: ARM Sourcery Windows GCC C Compiler
arm-none-eabi-gcc -D__CYU3P_TX__=1 -I"C:\Program Files\Cypress\EZ-USB FX3
SDK\1.3\/firmware/u3p_firmware/inc" -I.. -O0 -ffunction-sections -fdata-sections -Wall
-Wa,-adhlns="fx3_usb_desc.o.lst" -c -fmessage-length=0 -MMD -MP -MF"fx3_usb_desc.d"
-MT"fx3_usb_desc.d" -mcpu=arm926ej-s -mthumb-interwork -g -gdwarf-2 -o "fx3_usb_desc.o"
```

Figure 2-5: EZ USB Suite Eclipse Based IDE

5. Add additional source files as required. This can be done in one of two ways:
    a. Right-click the project name and select the “New -> Header File” or “New -> Source File” option as appropriate.

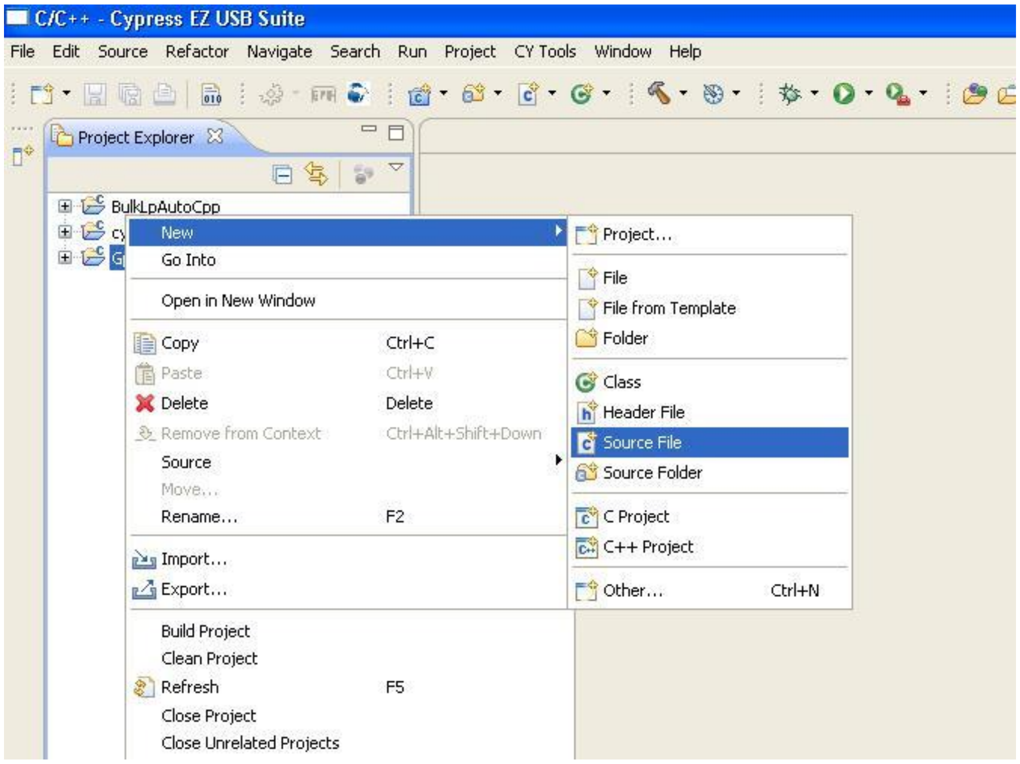


Figure 2-6: Adding new source files to a project

   b. Copy the file into the project folder inside the workspace and use the Refresh (F5 key) option to make it visible in the project.

6. The build settings for the project will already be functional. Right-Click on the project name and use the “Properties” menu item to view and modify the build settings.
7. Build the project to verify that the project import and renaming is successful.

2.3 Creating a New Project Based on Templates

Another way to create a new FX3 firmware project is to use the Create Project based on Templates option provided by EZ-USB Suite.

1. Select the New -> Project option from the File menu.

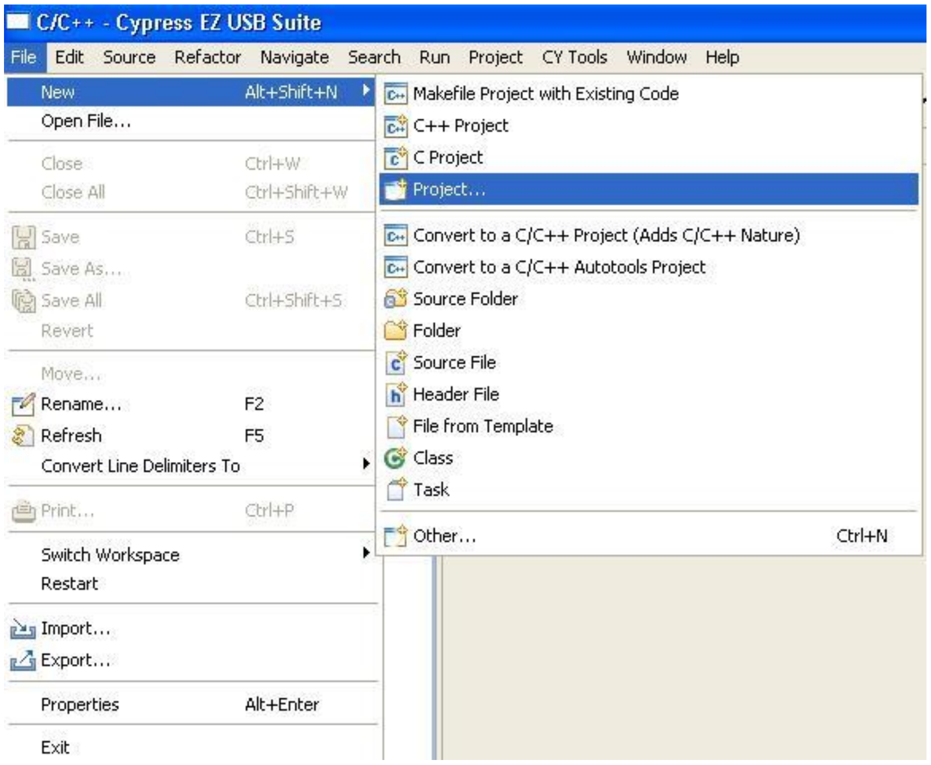


Figure 2-7: Selecting the New Project Menu Item

2. Choose the “Cypress -> FX3 Project” option from the resulting popup window and click on “Next”.

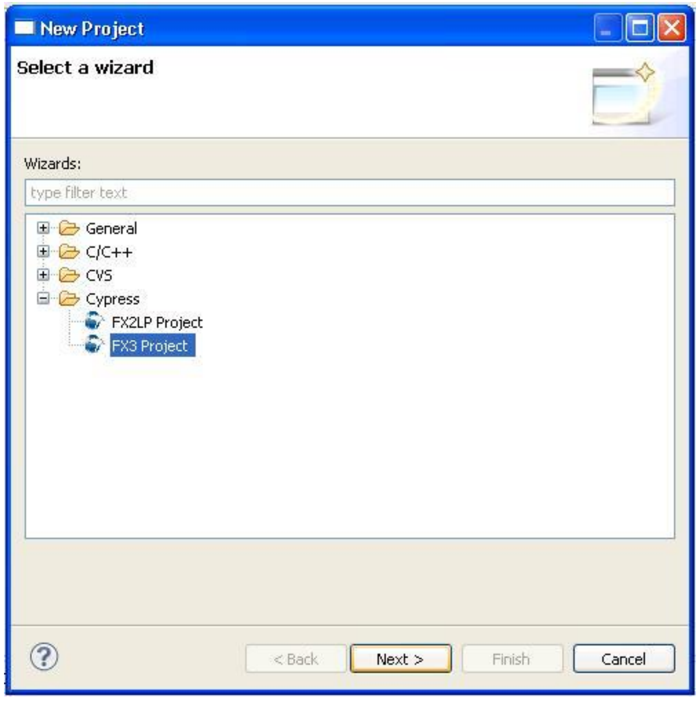
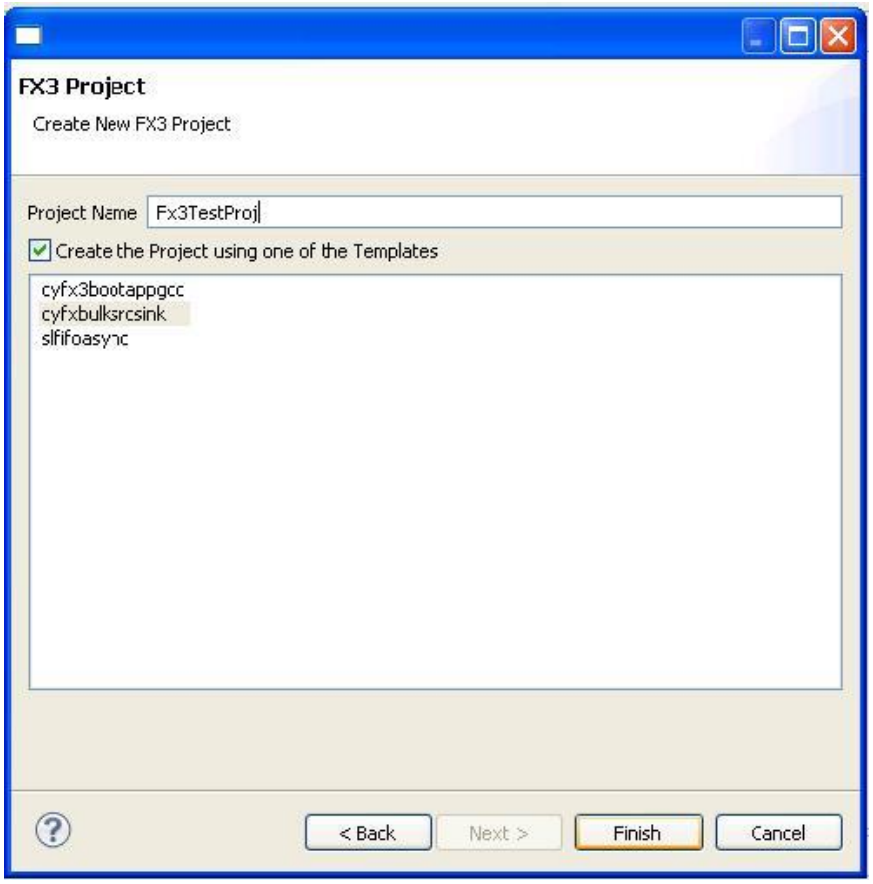


Figure 2-8: Choosing project type to be created

3. Three project templates are provided. The cyfx3bootappgcc template creates a project based on the FX3 boot library. The cyfxbulksrcsink template creates a project that does not include a GPIF-II configuration. The slfifoasync template creates a project that includes a GPIF-II configuration. Select the “Create the Project using one of the Templates” checkbox, select the desired template, provide a name for the project; and click “Finish” to create the project.


FX3 Project

Create New FX3 Project

Project Name: Fx3TestProj

☑ Create the Project using one of the Templates

cyfx3bootappgcc
cyfxbulksrcsink
slfifoasync

Figure 2-9: Creating the template based project

2.4 Using the FX2LP Firmware Templates

The ezUsbSuite also integrates SDCC compiler suite based plug-ins for build of firmware targeting the FX2LP device which has an 8051 MCU core.

To use the ezUsbSuite to develop firmware for FX2LP, you will first need to download and install the SDCC compiler suite. The latest release version (3.6.0) for Windows platforms can be found at:

https://sourceforge.net/projects/sdcc/files/sdcc-win64/3.6.0/ (for Windows 64-bit)

https://sourceforge.net/projects/sdcc/files/sdcc-win32/3.6.0/ (for Windows 32-bit)

2.4.1 Importing the FX2LP Firmware Template

The SDK includes only one FX2LP template project which is a bulkloop (data loopback using Bulk endpoints) application. Use the File → New → Project menu option and select the FX2LP Project option in the dialog.

Select the ***Create the project using one of the Templates*** check-box, select the Bulkloop_SDCC project, name the project as desired in the text box and select Finish to create a copy of the template project.

You can use the Project → Build Project menu option to compile the code and generate the HEX file.

Note: The SDCC compiler and assembler syntax is different from the syntax used by the ARM (Keil) uVision IDE. You cannot directly compile uVision based FX2LP projects using this tool-chain and will have to port the code to SDCC.

3 JTAG Debugging of Firmware Projects

The ARM9 core on the FX3 and related devices support the standard ARM JTAG TAP block and all of the JTAG pins are made available on the device package. This means that any standard JTAG debugger and tools can be used to debug FX3 firmware projects.

The following sub-sections provide instructions on how the Segger J-Link Debug Probe and the Cypress CY7C65215 USB serial device can be used for debugging firmware projects. Similar sequences can be followed when using other debug probes.

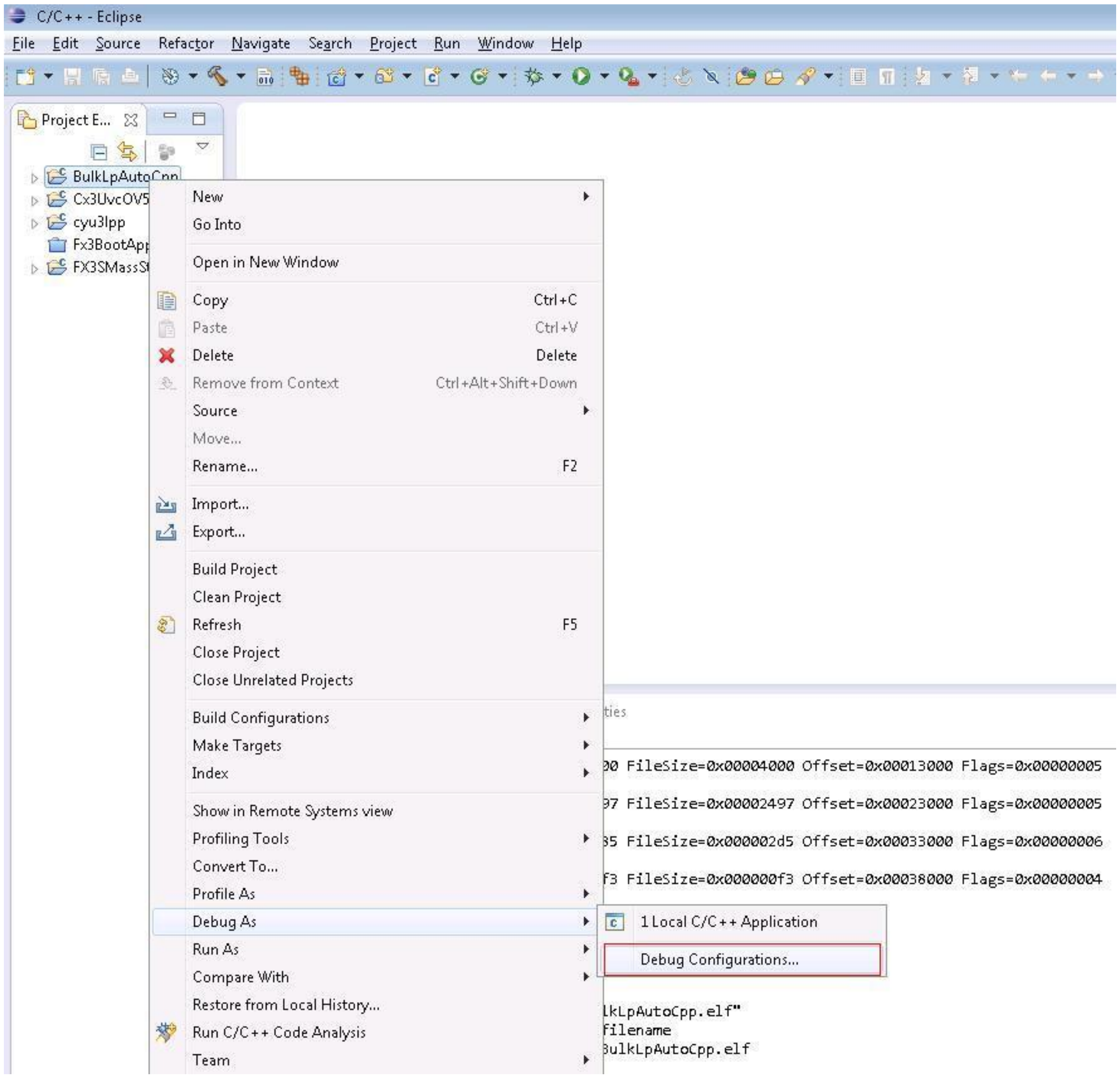
3.1 Debugging using Segger J-Link

The following procedure can be used to debug FX3 projects from the Eclipse IDE on all supported platforms (Windows, MacOS and Linux).

Download and install the latest J-Link GDB Server software from the Segger web page: https://www.segger.com/jlink-gdb-server.html

The command line version of the GDB server program is used for the following steps.

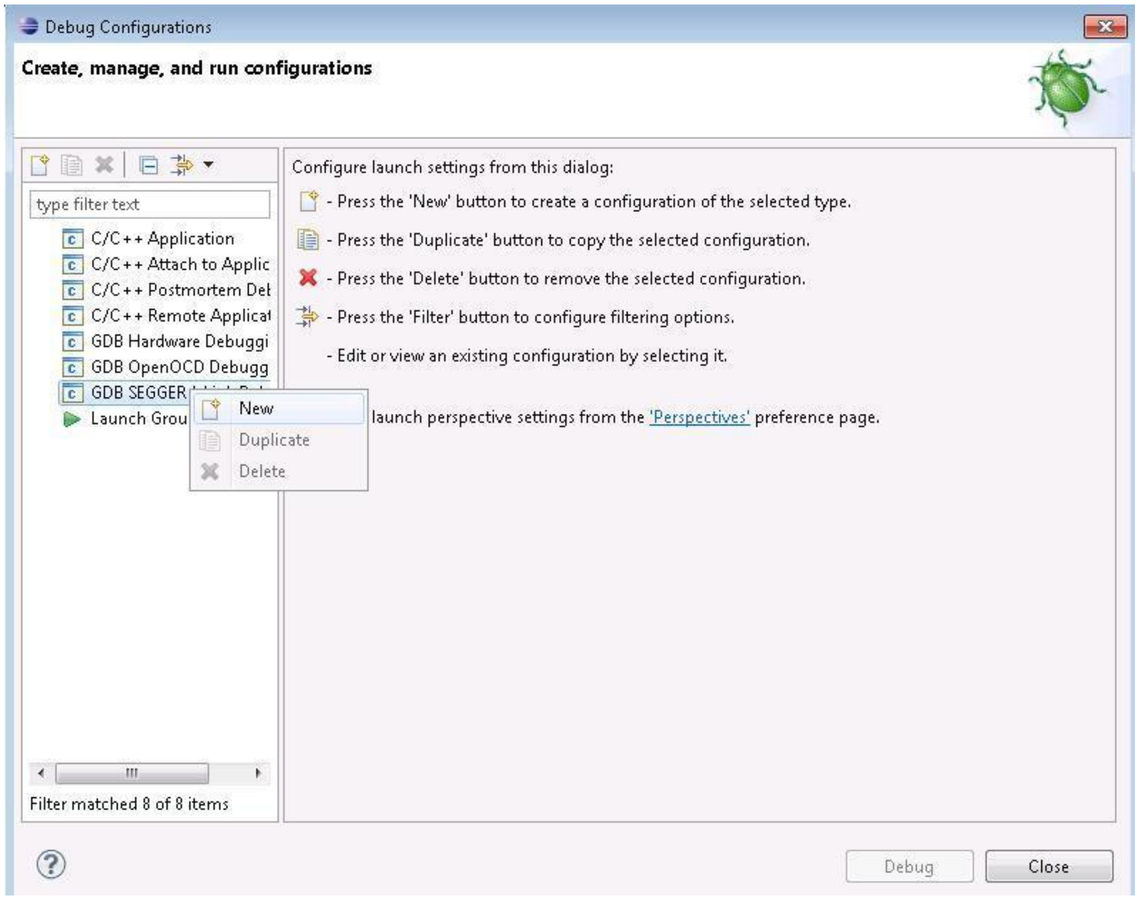
1. Select and right-click on the project to be debugged, and select the "Debug As -> Debug Configurations…" option.



```
00 FileSize=0x00004000 Offset=0x00013000 Flags=0x00000005
97 FileSize=0x00002497 Offset=0x00023000 Flags=0x00000005
35 FileSize=0x000002d5 Offset=0x00033000 Flags=0x00000006
f3 FileSize=0x000000f3 Offset=0x00038000 Flags=0x00000004
lkLpAutoCpp.elf"
filename
BulkLpAutoCpp.elf
```

Figure 3-1: Creating debug configuration: Start

2. In the Debug Configurations popup window, right-click the “GDB Segger J-Link debugging” option and select “New”.



Debug Configurations

Create, manage, and run configurations

Configure launch settings from this dialog:

- Press the 'New' button to create a configuration of the selected type.

- Press the 'Duplicate' button to copy the selected configuration.

- Press the 'Delete' button to remove the selected configuration.

- Press the 'Filter' button to configure filtering options.

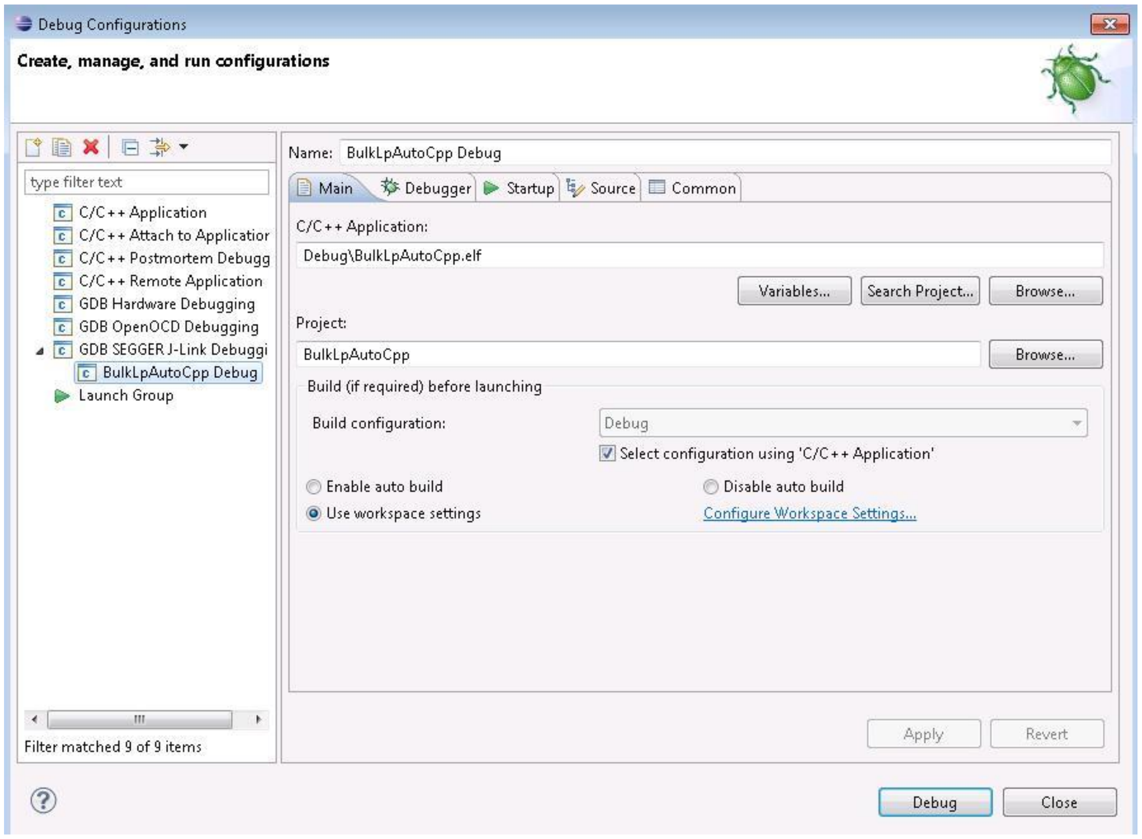
- Edit or view an existing configuration by selecting it.

launch perspective settings from the 'Perspectives' preference page.

Figure 3-2: Creating debug configuration: Segger J-Link

3. A “<ProjName> Debug” configuration will be created. The settings for the configuration need to be entered as below.

4. No changes are required under the “Main” tab.


Debug Configurations

Create, manage, and run configurations

type filter text

- C/C++ Application
- C/C++ Attach to Applicatior
- C/C++ Postmortem Debugg
- C/C++ Remote Application
- GDB Hardware Debugging
- GDB OpenOCD Debugging
- GDB SEGGER J-Link Debuggi
  - BulkLpAutoCpp Debug
- Launch Group

Filter matched 9 of 9 items

Name: BulkLpAutoCpp Debug

Main | Debugger | Startup | Source | Common

C/C++ Application:
Debug\BulkLpAutoCpp.elf

Variables... Search Project... Browse...

Project:
BulkLpAutoCpp

Browse...

Build (if required) before launching

Build configuration: Debug

[✓] Select configuration using 'C/C++ Application'

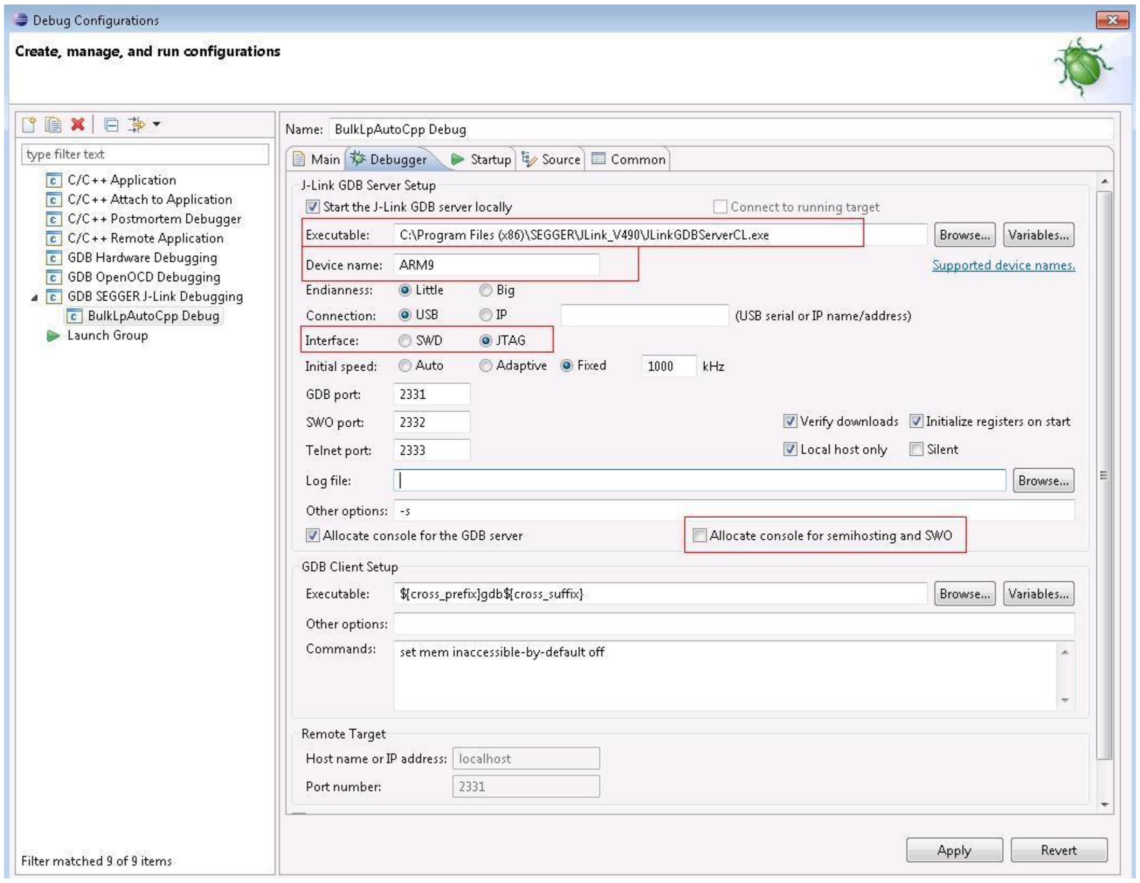
○ Enable auto build ○ Disable auto build
● Use workspace settings Configure Workspace Settings...

Apply Revert

Debug Close

Figure 3-3: Creating debug configuration: Main tab

5. On the “Debugger” tab, browse and select the path to the J-Link GDB Server executable, specify “ARM9” as the Device Name and select JTAG as the debug interface. Please note that the *{cross_prefix}* variable must be set to *arm-none-eabi-* in some cases if the default settings don’t work.


Debug Configurations

Create, manage, and run configurations

type filter text

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

Filter matched 9 of 9 items

Name: BulkLpAutoCpp Debug

Main | Debugger | Startup | Source | Common

J-Link GDB Server Setup

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

Executable:	C:\Program Files (x86)\SEGGER\JLink_V490\JLinkGDBServerCL.exe
Device name:	ARM9
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
Log file:	
Other options:	-s

- [✓] Verify downloads
- [✓] Initialize registers on start
- [✓] Local host only
- [ ] Silent
- [✓] Allocate console for the GDB server
- [ ] Allocate console for semihosting and SWO

GDB Client Setup

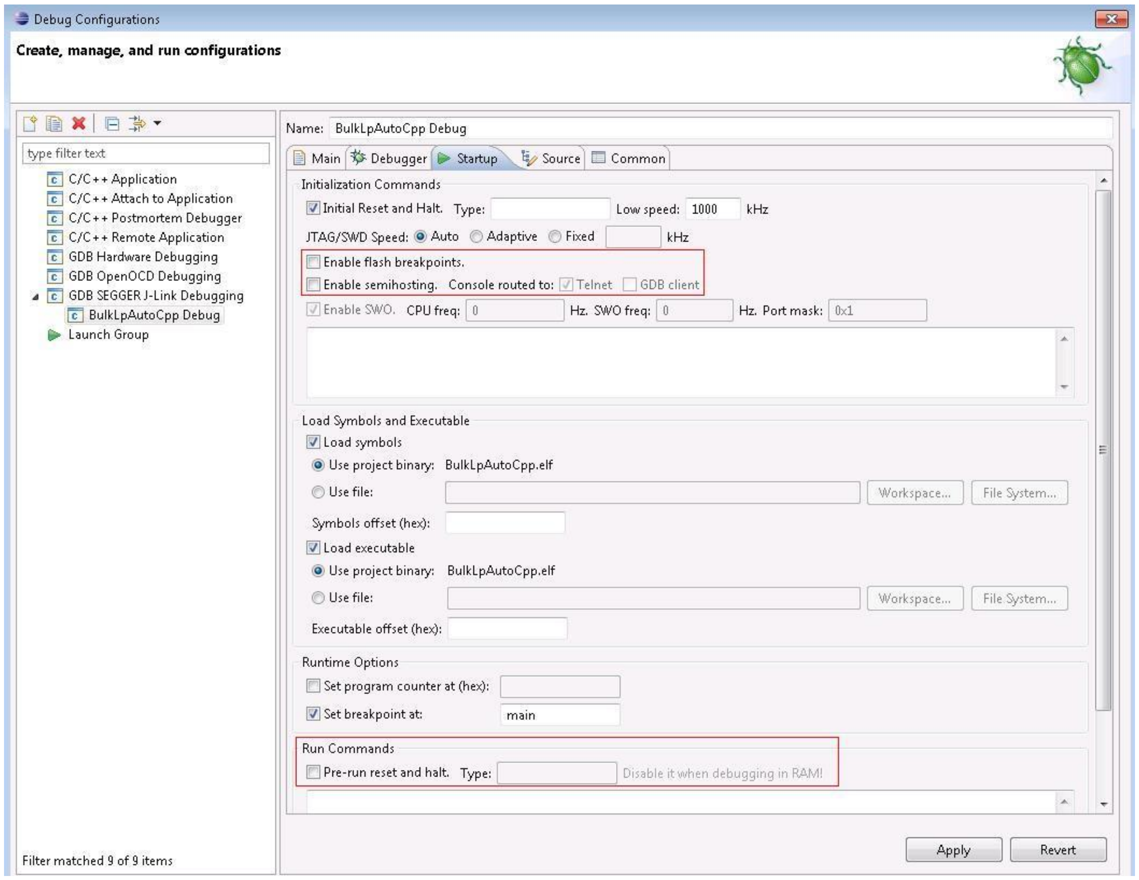
Executable:	${cross_prefix}gdb${cross_suffix}
Other options:	
Commands:	set mem inaccessible-by-default off

Remote Target

Host name or IP address:	localhost
Port number:	2331

Figure 3-4: Creating debug configuration: Debugger tab

6. On the “Startup” tab, disable the “Enable flash breakpoints”, “Enable semihosting” and “Pre-run reset and halt” options using the corresponding checkboxes.


Debug Configurations

Create, manage, and run configurations

type filter text

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

Filter matched 9 of 9 items

Name: BulkLpAutoCpp Debug

Main | Debugger | Startup | Source | Common

Initialization Commands

[✓] Initial Reset and Halt. Type: Low speed: 1000 kHz

JTAG/SWD Speed: [● Auto] ○ Adaptive ○ Fixed kHz

[ ] Enable flash breakpoints.

[ ] Enable semihosting. Console routed to: [✓] Telnet [ ] GDB client

[✓] Enable SWO. CPU freq: 0 Hz. SWO freq: 0 Hz. Port mask: 0x1

Load Symbols and Executable

[✓] Load symbols

[● Use project binary:] BulkLpAutoCpp.elf

○ Use file: Workspace... File System...

Symbols offset (hex):

[✓] Load executable

[● Use project binary:] BulkLpAutoCpp.elf

○ Use file: Workspace... File System...

Executable offset (hex):

Runtime Options

[ ] Set program counter at (hex):

[✓] Set breakpoint at: main

Run Commands

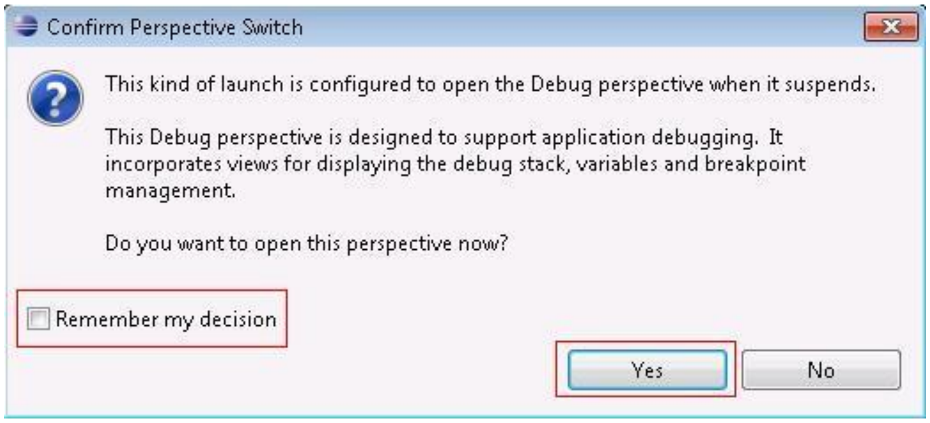
[ ] Pre-run reset and halt. Type: Disable it when debugging in RAM!

Apply Revert

Figure 3-5: Creating debug configuration: Startup Tab

7. Click on “Apply” to save settings, and then click on “Debug” to start debugging.

8. If a “Confirm Perspective Switch” popup window (as shown below) appears, select “Yes” to switch to the Debug perspective. The “Remember my decision” checkbox can be ticked to prevent this popup from appearing in subsequent debug sessions.


Confirm Perspective Switch

This kind of launch is configured to open the Debug perspective when it suspends.

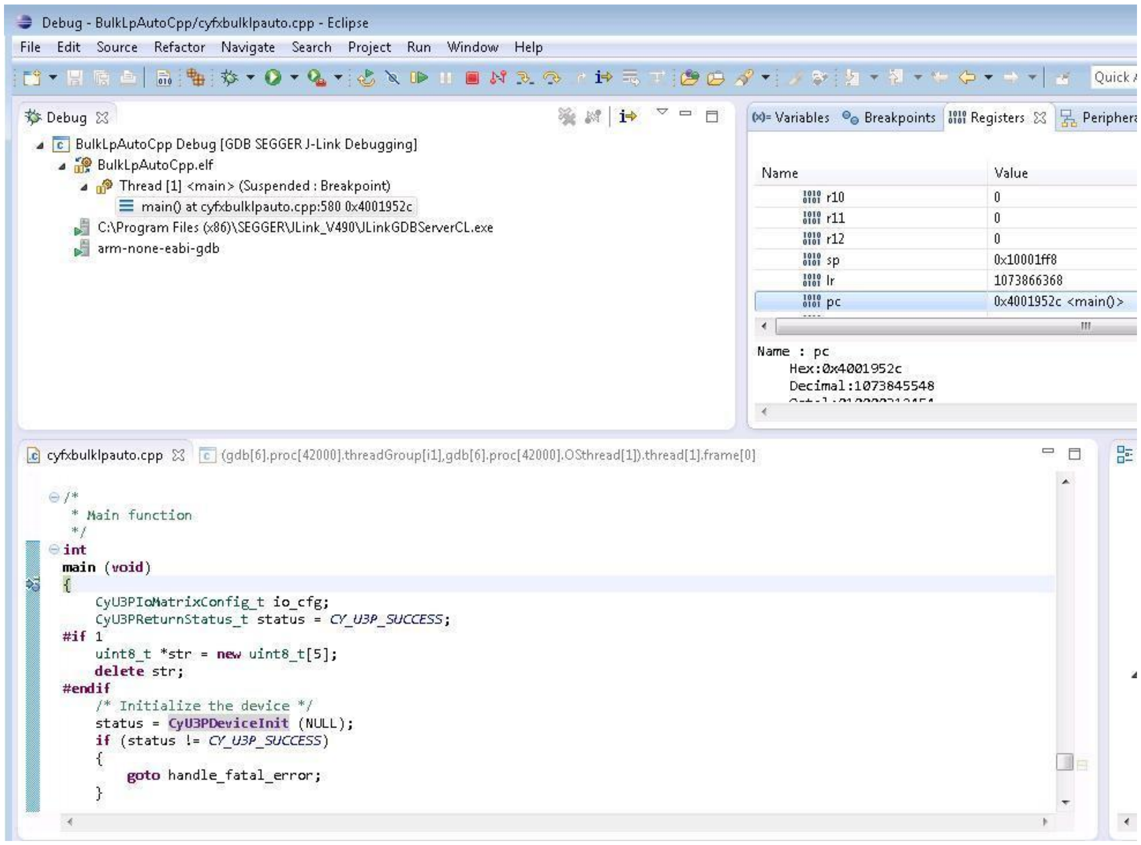
This Debug perspective is designed to support application debugging. It incorporates views for displaying the debug stack, variables and breakpoint management.

Do you want to open this perspective now?

☐ Remember my decision

Figure 3-6: Creating debug configuration: Perspective switch

9. Execution stops at the “main ()” function. Additional breakpoints can be placed at this stage, and the Resume, Step Into or Step Over actions can be used to run through the code.


Debug - BulkLpAutoCpp/cyfxbulklpauto.cpp - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick
Debug
Variables Breakpoints Registers Periphera

```
BulkLpAutoCpp Debug [GDB SEGGER J-Link Debugging]
  BulkLpAutoCpp.elf
    Thread [1] <main> (Suspended : Breakpoint)
      main() at cyfxbulklpauto.cpp:580 0x4001952c
  C:\Program Files (x86)\SEGGER\JLink_V490\JLinkGDBServerCL.exe
  arm-none-eabi-gdb
```

Name	Value
r10	0
r11	0
r12	0
sp	0x10001ff8
lr	1073866368
pc	0x4001952c <main()>

```
Name : pc
    Hex:0x4001952c
    Decimal:1073845548
```

cyfxbulklpauto.cpp
(gdb[6].proc[42000].threadGroup[i1],gdb[6].proc[42000].OSthread[1]).thread[1].frame[0]

```
/*
 * Main function
 */
int
main (void)
{
    CyU3PIoMatrixConfig_t io_cfg;
    CyU3PReturnStatus_t status = CY_U3P_SUCCESS;
#if 1
    uint8_t *str = new uint8_t[5];
    delete str;
#endif
    /* Initialize the device */
    status = CyU3PDeviceInit (NULL);
    if (status != CY_U3P_SUCCESS)
    {
        goto handle_fatal_error;
    }
```

Figure 3-7: Debug session stopped at breakpoint

10. Click on “Terminate” to stop the debug session.

3.2 Debugging with OpenOCD

Open On-Chip Debugger (OpenOCD) is an open source debugger implementation that supports a variety of debugger probes. The CY7C65215 Cypress USB Serial part on the CYUSB3KIT-003 development kit provides a debugger interface that works with OpenOCD.

A version of OpenOCD binary that supports debugging using the CY7C65215 part is provided with the FX3 SDK, under the OpenOCD folder. This binary is based on the OpenOCD 0.8.0 release and only supports the CY7C65215 part as a debug interface.

If any other OpenOCD compliant debug probe (such as the Olimex ARM JTAG debug probe) is being used, replace the OpenOCD binary provided with a version that supports the target debug probe. The rest of the instructions are interface independent, and apply to any OpenOCD compliant debug probe.

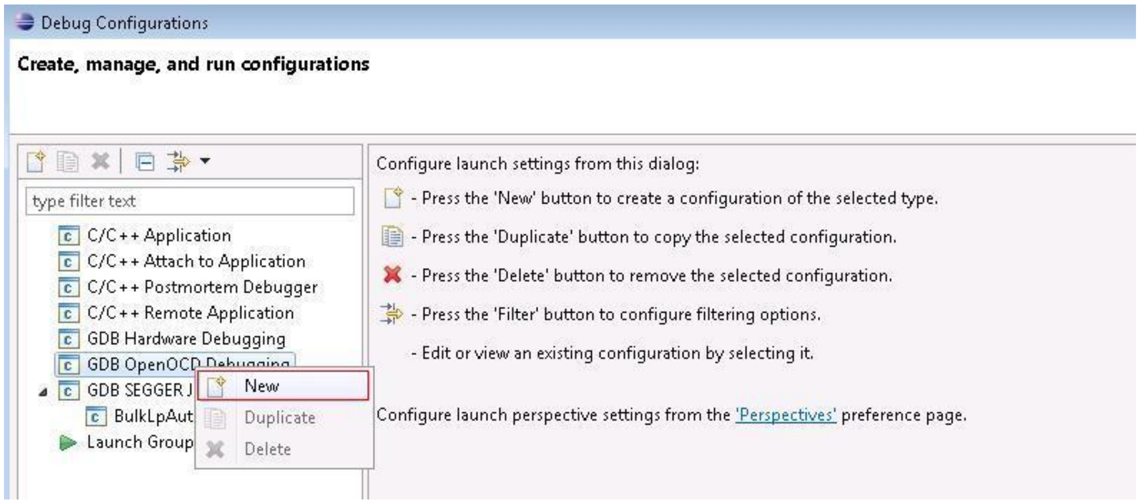
The latest version of OpenOCD sources can be downloaded from: http://sourceforge.net/projects/openocd/files/openocd/. Pre-compiled binaries for Windows can be obtained from: http://www.freddiechopin.info/en/download/category/4-openocd.

Note: When using OpenOCD on Linux or Mac platforms, the libraries that the OpenOCD binary depends on; need to be copied to the system folders. Please change to the <FX3_INSTALL_PATH>/OpenOCD/<platform> folder and run the script.sh script to do this. No specific installation steps are required on Windows platforms.

The procedure for setting up OpenOCD based JTAG debugging is shown below:

1. Select and right-click on the project to be debugged, and select the “Debug As -> Debug Configurations…” option. See Figure 3-1.

2. In the Debug Configurations popup window, right-click the “GDB OpenOCD debugging” option and select “New”.



Configure launch settings from this dialog:

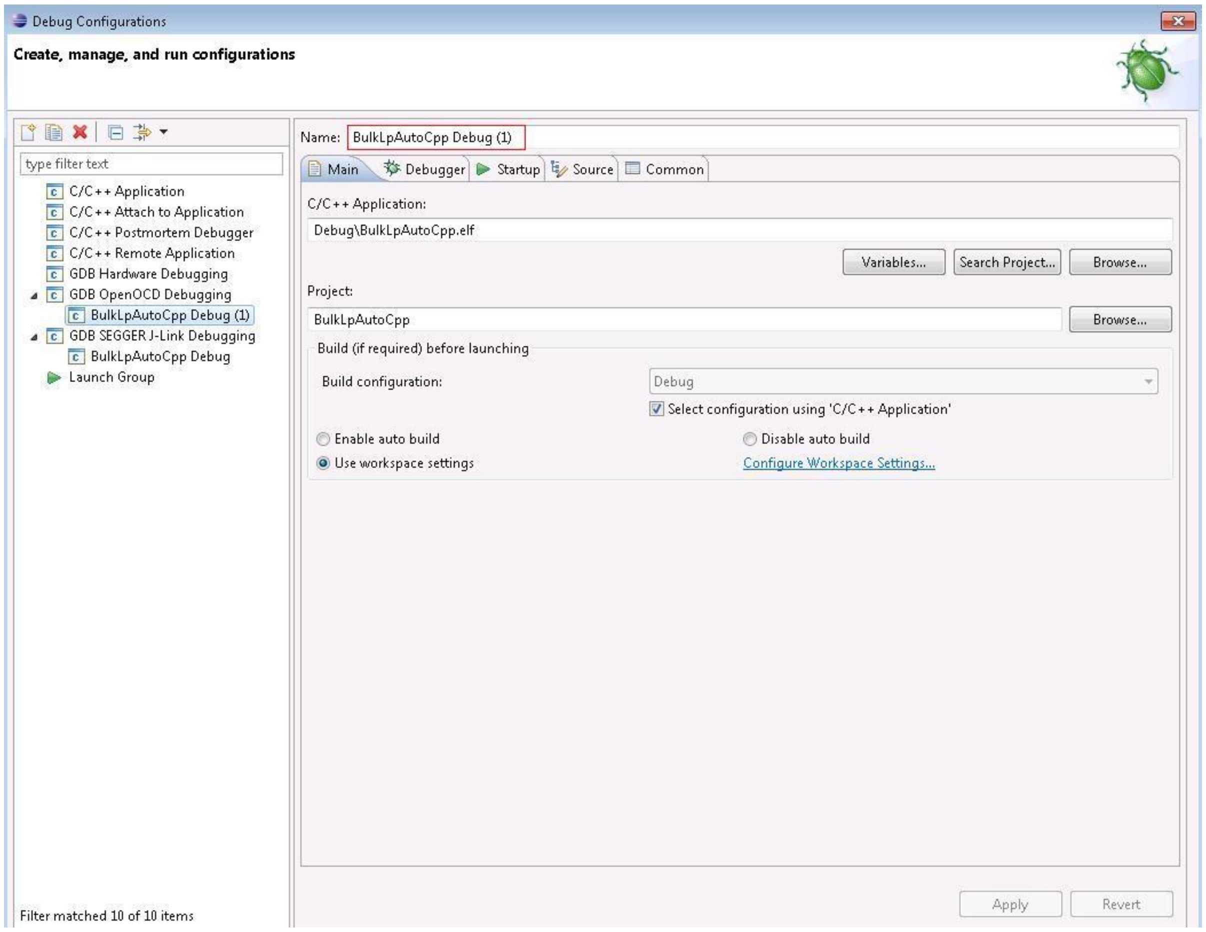
- Press the 'New' button to create a configuration of the selected type.
- Press the 'Duplicate' button to copy the selected configuration.
- Press the 'Delete' button to remove the selected configuration.
- Press the 'Filter' button to configure filtering options.
- Edit or view an existing configuration by selecting it.

Configure launch perspective settings from the 'Perspectives' preference page.

Figure 3-8: Creating debug configuration: OpenOCD

3. A “<ProjName> Debug” configuration will be created. The settings for the configuration need to be entered as described in the next steps.

4. No changes are required under the “Main” tab.


Debug Configurations

Create, manage, and run configurations

type filter text

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

Filter matched 10 of 10 items

Name: BulkLpAutoCpp Debug (1)

C/C++ Application: Debug\BulkLpAutoCpp.elf

Project: BulkLpAutoCpp

Build (if required) before launching

Build configuration: Debug

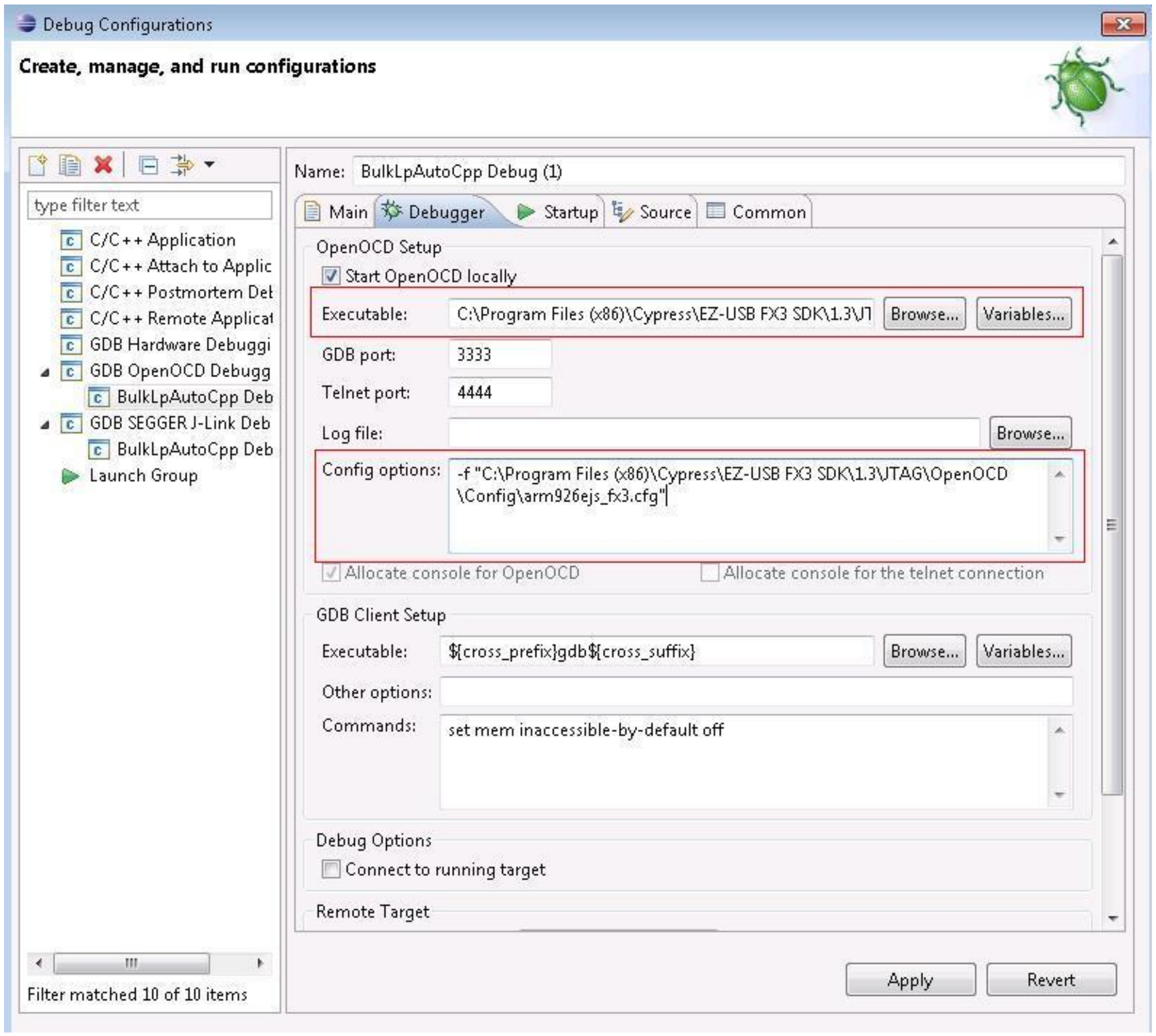
[✓] Select configuration using 'C/C++ Application'

○ Enable auto build ○ Disable auto build

● Use workspace settings

Figure 3-9: Creating OpenOCD debug configuration: Main tab

5. On the “Debugger” tab, browse and select the path to the OpenOCD executable, and selecting the OpenOCD configuration file provided with the SDK under “Config options”. Please note that the full path to the config file needs to be provided here, and may need to be enclosed in quotes if it includes spaces. Please note that the *{cross_prefix}* variable must be set to *arm-none-eabi-* in some cases if the default settings don’t work.


Debug Configurations

Create, manage, and run configurations

Name: BulkLpAutoCpp Debug (1)

OpenOCD Setup

[✓] Start OpenOCD locally

Executable: C:\Program Files (x86)\Cypress\EZ-USB FX3 SDK\1.3\JT

GDB port: 3333

Telnet port: 4444

Log file:

Config options: -f "C:\Program Files (x86)\Cypress\EZ-USB FX3 SDK\1.3\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

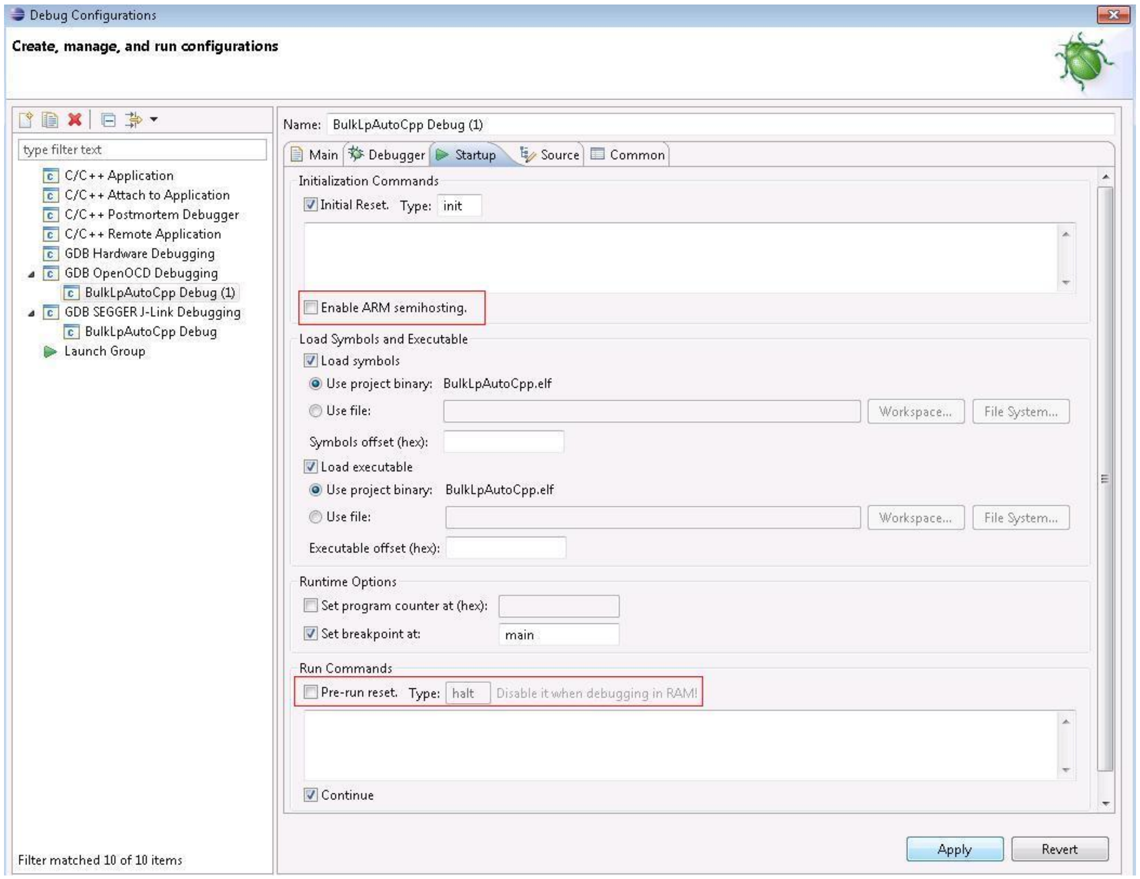
Debug Options

[ ] Connect to running target

Remote Target

Figure 3-10: Creating OpenOCD debug configuration: Debugger tab

6. On the “Startup” tab, disable the “Enable ARM semihosting” and “Pre-run reset” options using the corresponding checkboxes.


Debug Configurations

Create, manage, and run configurations

type filter text

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

Filter matched 10 of 10 items

Name: BulkLpAutoCpp Debug (1)

Main | Debugger | Startup | Source | Common

Initialization Commands

[✓] Initial Reset. Type: init

[ ] Enable ARM semihosting.

Load Symbols and Executable

[✓] Load symbols

(●) Use project binary: BulkLpAutoCpp.elf

( ) Use file: Workspace... File System...

Symbols offset (hex):

[✓] Load executable

(●) Use project binary: BulkLpAutoCpp.elf

( ) Use file: Workspace... File System...

Executable offset (hex):

Runtime Options

[ ] Set program counter at (hex):

[✓] Set breakpoint at: main

Run Commands

[ ] Pre-run reset. Type: halt Disable it when debugging in RAM!

[✓] Continue

Apply Revert

Figure 3-11: Creating OpenOCD debug configuration: Startup Tab

7. Click on “Apply” to save settings, and then click on “Debug” to start debugging.
8. If a “Confirm Perspective Switch” popup window (as shown below) appears, select “Yes” to switch to the Debug perspective. The “Remember my decision” checkbox can be ticked to prevent this popup from appearing in subsequent debug sessions. See Figure 3-6.
9. Please note that firmware download using the CY7C65215 debug probe is a slow process and can take about 30 to 40 seconds. Wait for the firmware download to be completed before the session is ready for debugging.
10. Execution stops at the “main ()” function. Additional breakpoints can be placed at this stage, and the Resume, Step Into or Step Over actions can be used to run through the code. See Figure 3-7.
11. Click on “Terminate” to stop the debug session.

4 CX3 Configuration Utility

This section provides an introduction to MIPI CSI-2 interface on the CX3 device and provides a usage guide for the CX3 Configuration tool available as part of the Cypress EZ USB Suite Eclipse based IDE. More details on the CX3 part are available on the Cypress website at http://www.cypress.com/cx3/

4.1 Introduction to the CX3 device

The CX3 is a variant of the FX3 device that features an integrated MIPI CSI-2 receiver mated to the GPIF as shown in the CX3 Logic block diagram. This device provides the ability to add USB 3.0 connectivity to Image Sensors implementing the MIPI CSI-2 interface.

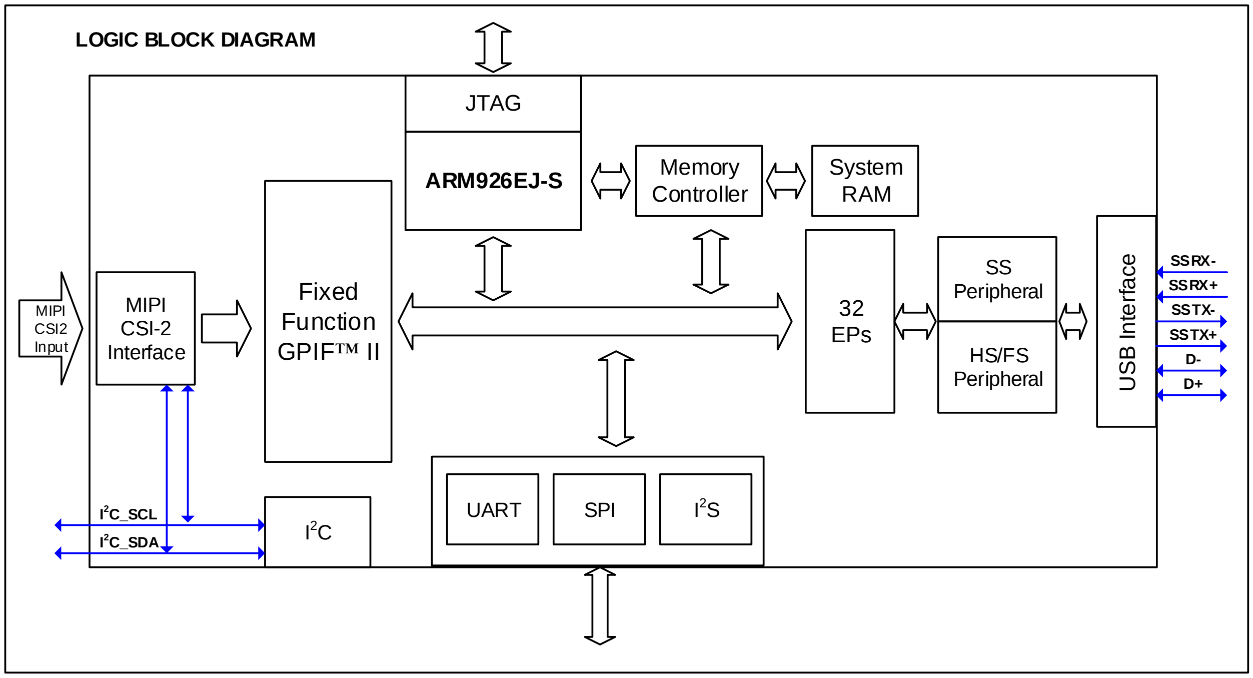


Figure 4-1: CX3 Device Block Diagram

The MIPI CSI-2 interface on the CX3 supports 1 to 4 CSI-2 data lanes and RAW8/10/12/14, YUV422, and RGB888/666/565 image formats. It reads image data from the sensor, de-packetizes it and sends it in to the parallel interface of the fixed-function GPIF-II interface on the CX3.

As the GPIF-II signals on CX3 are connected internally to the MIPI CSI-2 Interface block, it is not possible make use of the GPIF-II to talk to any external devices. The control signals on the interface are also fixed, and only the width of the data bus can be selected from the values of 8-bit, 16-bit and 24-bits. CX3 firmware applications typically make use of a pre-defined GPIF-II configuration with changes being made only to the data bus width and the counter limits used.

Support for the MIPI CSI-2 interface is provided through a new firmware library added to the SDK (cyu3mipicsi.a). The APIs provided by the library and the data structures and enumerations used by this interface are provided through the cyu3mipicsi.h header file.

4.2 CX3 MIPI CSI-2 interface

4.2.1 Data Lanes

The MIPI CSI-2 interface on the CX3 supports 1 to 4 CSI-2 data lanes each capable of transfers of up to 1 Gbps. The number of data lanes to be selected for a CX3 application will depend upon the number of data lanes provided by the Image Sensor and the total required data transfer rate.

4.2.2 MIPI CSI-2 stream formats supported by the CX3

The MIPI CSI-2 interface on the CX3 supports the following stream formats and output modes:

Table 4-1: Stream formats supported by CX3

Format and Mode	CX3 Enumeration Type	Description	CSI-2 Data Type	Output Stream
RAW8	CY_U3P_CSI_DF_RAW8	Bayer format 8-bits per pixel data stream.	0x2A	8-Bit Output: RAW[7:0]
RAW10	CY_U3P_CSI_DF_RAW10	Bayer format 10-bits per pixel data stream.	0x2B	16-Bit Output: 6’b0, RAW[9:0]
RAW12	CY_U3P_CSI_DF_RAW12	Bayer format 12-bits per pixel data stream.	0x2C	16-Bit Output: 4’b0,RAW[11:0]
RAW14	CY_U3P_CSI_DF_RAW14	Bayer format 14-bits per pixel data stream.	0x2D	16-Bit Output: 2'b0,RAW[13:0]
RGB888	CY_U3P_CSI_DF_RGB888	RGB 888 format 24- bits per pixel data stream.	0x24	24-Bit Output: R[7:0],G[7:0],B[7:0]
RGB666 Mode 0	CY_U3P_CSI_DF_RGB666_0	RGB 666 format 24- bits per pixel data stream.	0x23	24-Bit Output: 2’b0,R[5:0],2’b0,G[5:0], 2’b0,B[5:0]

Format and Mode	CX3 Enumeration Type	Description	CSI-2 Data Type	Output Stream
RGB666 Mode 1	CY_U3P_CSI_DF_RGB666_1	RGB 666 format 24- bits per pixel data stream.	0x23	24 Bit Output: 6’b0,R[5:0],G[5:0], B[5:0]
RGB565 Mode 0	CY_U3P_CSI_DF_RGB565_0	RGB 565 format 24- bits per pixel data stream.	0x22	24 Bit Output: 2’b0,R[4:0],3’b0,G[5:0], 2’b0,B[4:0],1’b0
RGB565 Mode 1	CY_U3P_CSI_DF_RGB565_1	RGB 565 format 24- bits per pixel data stream.	0x22	24 Bit Output: 3’b0,R[4:0],2’b0,G[5:0], 3’b0,B[4:0]
RGB565 Mode 2	CY_U3P_CSI_DF_RGB565_2	RGB 565 format 16-bits per pixel data stream.	0x22	16 Bit Output: R[4:0],G[5:0], B[4:0]
YUV422 8-Bit Mode 0	CY_U3P_CSI_DF_YUV422_8_0	YUV422 format 8-bits per pixel data stream.	0x1E	8 Bit Output: P[7:0] Data Order: U1,Y1,V1,Y2,U3,Y3,....
YUV422 8-Bit Mode 1	CY_U3P_CSI_DF_YUV422_8_1	YUV422 format 8-bits per pixel data stream.	0x1E	16 Bit Output: P[15:0] Data Order: {U1,Y1},{V1,Y2},{U3,Y3},{V3,Y4}...
YUV422 8-Bit Mode 2	CY_U3P_CSI_DF_YUV422_8_2	YUV422 format 8-bits per pixel data stream.	0x1E	16 Bit Output: P[15:0] Data Order: {Y1,U1},{Y2,V1},{Y3,U3},{Y4,V3}....
YUV422 10-Bit	CY_U3P_CSI_DF_YUV422_10	YUV422 format 10-bits per pixel data stream.	0x1F	16 Bit Output: 6'b0,P[9:0] Data Order: U1,Y1,V1,Y2,U3,Y3,V3,Y4.

If the GPIF interface used is larger than the width of the output stream (e.g. if a 24-bit GPIF II interface is used for the CY_U3P_CSI_DF_YUV422_8_1 type) the upper bits on the GPIF II interface will be padded with 0s.

4.2.3 MIPI CSI-2 interface clocks

The primary clocks on the MIPI CSI-2 interface of the CX3 are shown in Figure 4-2 below.

The interface takes a reference clock as its input and generates the required clocking using a PLL and multiple clock dividers on the PLL generated clock. Clock configuration parameters are a part of the *CyU3PMipicsiCfg_t* structure which passed to the *CyU3PMipicsiSetIntfParams()* API to configure the CSI-2 interface.

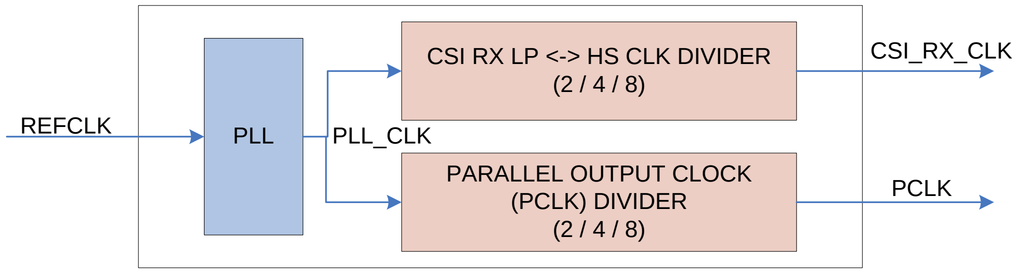

REFCLK
PLL
PLL_CLK
CSI RX LP <-> HS CLK DIVIDER (2 / 4 / 8)
CSI_RX_CLK
PARALLEL OUTPUT CLOCK (PCLK) DIVIDER (2 / 4 / 8)
PCLK

Figure 4-2: CX3 MIPI CSI-2 Interface Clocks

A brief description of each of the clocks is provided below:

1. **Reference Clock (REFCLK)**

This is the input reference clock provide to the MIPI-CSI interface. This input clock should be between 6 and 40 MHz.

2. **PLL Clock (PLL_CLK)**

The PLL_CLK is the primary clock on the MIPI CSI-2 interface. The minimum legal value for PLL clock is 62.5 MHz and maximum legal value for the PLL is 1 GHz.

All other internal/output clocks are derived from this clock.

The PLL clock frequency is generated from the Input Reference Clock using the following equation:

PLL_CLK = **REFCLK** * [(**pllFbd** + 1)/(**pllPrd** + 1)] /(2^**pllFRS**)

where

pllPrd is the input divider whose range is between 0 and 0x0F.

pllFbd is the feedback divider whose range is between 0 and 0x1FF.

pllFRS is the frequency range selection parameter which takes the following values:

0 if PLL Clock is between 500MHz- 1GHz.

1 if PLL Clock is between 250MHz- 500MHz.

2 if PLL Clock is between 125MHz- 250MHz.

3 if PLL Clock is between 62.5MHz- 125MHz.

E.g.:

If RefClk is 19.2 MHz, pllFbd is 69, pllPrd is 1, and PLL Clock range is in the 500MHz-1GHz range (i.e. pllFrs = 0),

PLL_Clock (MHz) = 19.2 * [(69+1)/(1+1)]/(2^0)

= 19.2 * [70/2]/1

= 672 MHz

For the same values, changing PLL frequency range to 125-250MHz (pllFrs = 2) will change the PLL Clock value to

PLL_Clock (MHz) = 19.2 * [(69+1)/(1+1)]/(2^2)

= 19.2 * [70/2]/4

= 168 MHz

3. CSI RX LP←→HS Transition Clock

This clock is used for detecting the CSI link LP<->HS transition. It is generated by dividing the PLL_Clock by a value of 2, 4 or 8.

The minimum value for this clock is 10Mhz and maximum value for this clock is 125MHz.

4. Output Parallel Clock (PCLK)

This clock is the PCLK output which drives the fixed-function GPIF interface on the CX3. It is generated by dividing the PLL_Clock by a value of 2, 4 or 8.

The maximum value for this clock is 100MHz.

4.3 Configuring the CX3 Device

The typical CX3 application is a USB Video Class (UVC) compliant Camera application that streams video or still image data captured by an Image Sensor to a host PC. Most parts of a CX3 system design are common, with the only variations being in the Image Sensor and data formats chosen.

The typical steps involved in UVC implementations using the CX3 device are:

1. Configuring the image sensor as required.
2. Configuring the MIPI CSI-2 interface on the CX3 device based on the image sensor settings.
3. Defining the USB descriptors for the application based on the image formats to be supported.
4. Implementing the actual image streaming logic using CX3/FX3 APIs.
5. Implementing the sensor control operations required to handle requests on the Video Control interface (e.g. brightness, pan – tilt – zoom etc.)

The EZ-USB Suite application provides a CX3 Configuration utility that helps in steps 2, 3 and 4. The actual image sensor control (step 1) and the control interface handling (step 5) are sensor dependent, and not handled by the tool.

4.3.1 MIPI CSI-2 Configuration

The MIPI CSI-2 interface on CX3 is configured using the *CyU3PMipicsiSetIntfParams()* API. This API takes an input parameter of type *CyU3PMipicsiCfg_t* which contains various configurations like output data format, clock dividers and configuration parameters, number of CSI data lanes to use, horizontal resolution etc.

The CSI Configuration Utility generates the data required to configure the interface in the form of C code which can be embedded into the firmware application.

4.3.2 Defining the USB Descriptors

The UVC specification defines the format for the USB configuration descriptors for all UVC compliant devices. The size of the configuration descriptor can stretch to 200 bytes or more depending on the number and types of video formats and resolutions supported. Putting together the descriptors in a fully spec compliant manner across all three USB connection speeds (SuperSpeed, Hi-Speed and Full Speed) is a cumbersome process.

The CSI Configuration Utility generates the USB descriptors based on the settings provided. The descriptor data is generated in the form of C code which can be embedded into the firmware application.

4.3.3 Video Streaming Logic

The firmware design and implementation to handle video streaming includes elements such as:

1. Defining the buffering requirements for the video stream.
2. Setting up the DMA connection from the GPIF-II interface to the USB endpoint
3. Handling Video Streaming control requests to negotiate the video format and resolution.
4. Managing the actual video data transfer with UVC header addition.

The CSI configuration tool generates C header and source files that implement this logic. These files can be used along with the CSI-2 configuration file and the USB descriptor file to create the UVC firmware project.

The next section describes the usage of the MIPI CSI-2 Configuration tool provided as part of the EZ-USB Suite IDE.

4.4 Using the CSI Configuration Utility

4.4.1 Creating a new CX3 Configuration Project

From the File menu of the EZ USB Suite IDE, select the **New → Other** option.

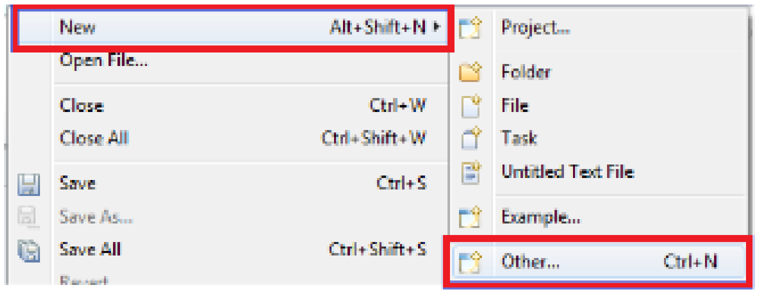


Figure 4-3: CX3 Configuration File Creation Menu

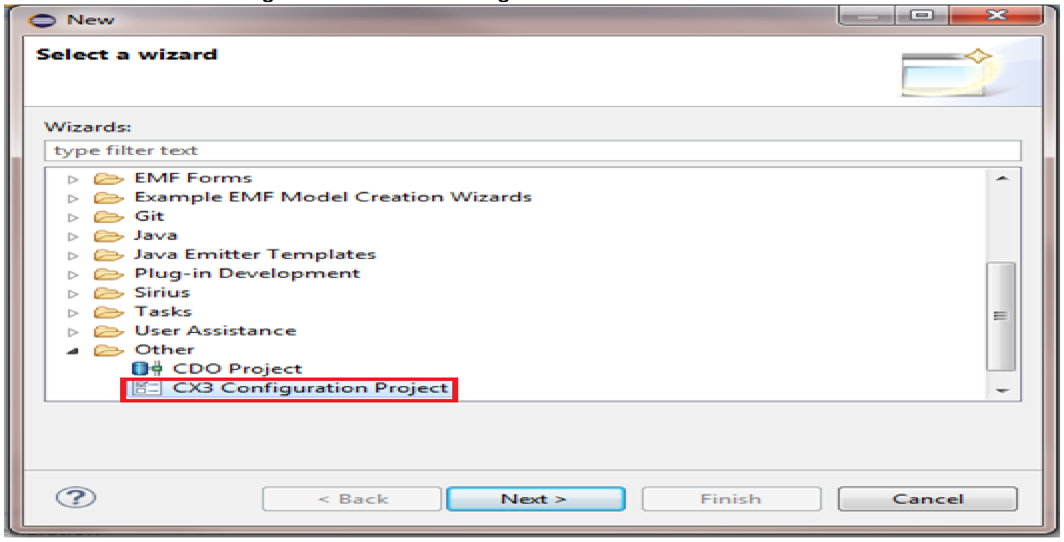


Figure 4-4: CX3 Configuration Project Creator

4.4.2 Image Sensor Selection and Configuration

The first step in configuring the CX3 UVC project is to define the properties of the image sensor being used in the system. This includes properties like the type of sensor, still image and video capture support, the format and resolution of the images etc.

Configurations for the Aptina AS0260 and OmniVision OV5640 sensors are provided as part of the tool, and you can use these directly if these sensors are being used. If other sensors are being used, the wizard can be used to input the sensor properties for conversion into the required format.

4.4.2.1 CX3 Configuration Project Creation

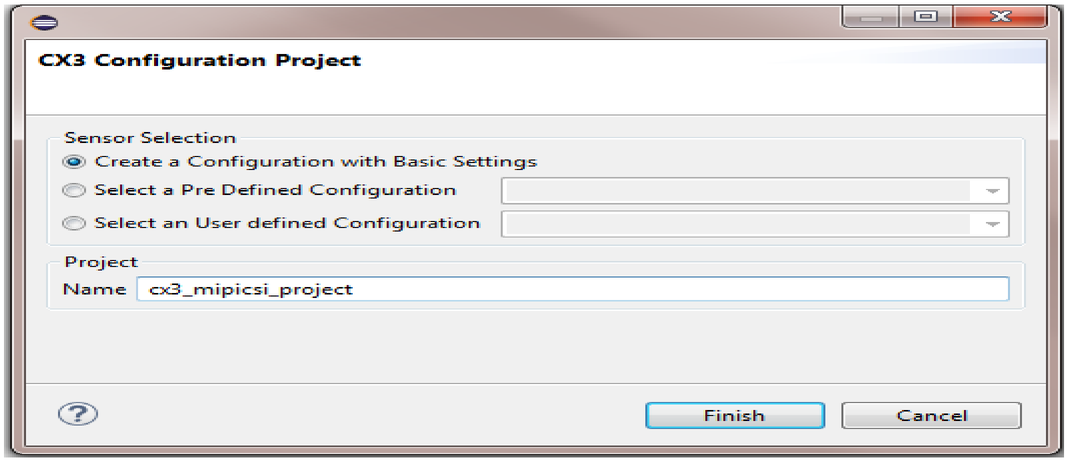
The CX3 Configuration Project Creation Wizard provides three possible alternatives for configuring the newly created project:

1. Create a Configuration with Basic Settings
2. Select a Pre-Defined Configuration
3. Select an User defined Configuration

The steps for configuring the MIPI CSI-2 interface and image sensor using each of the three possible options are described in the following sub-sections.

4.4.2.1.1 Configuration with Basic Settings

1. Click on **Create a Configuration with Basic Settings** Radio Button and provide a new name to the project. Click on **Finish** button to start generating relevant configuration data.



CX3 Configuration Project

Sensor Selection

(●) Create a Configuration with Basic Settings

( ) Select a Pre Defined Configuration

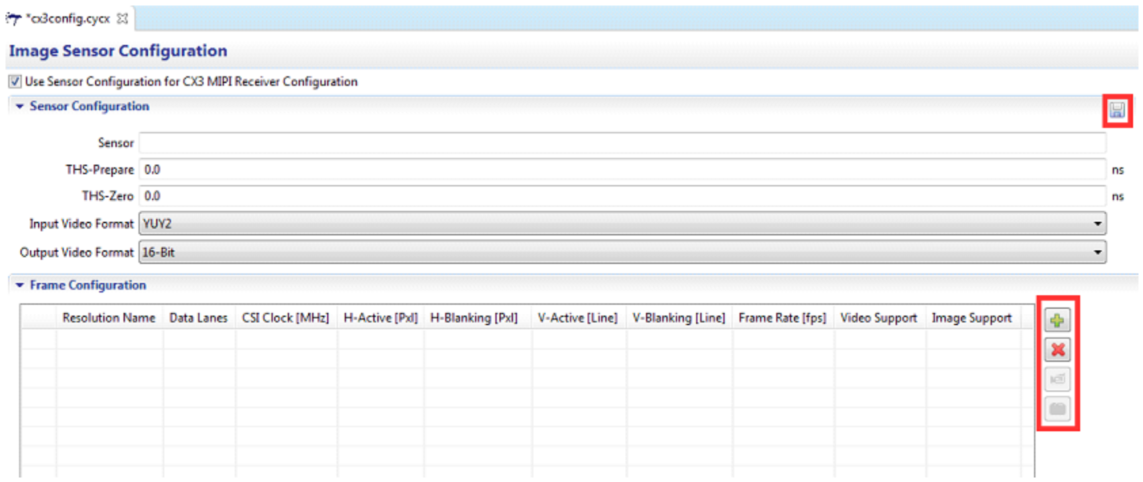
( ) Select an User defined Configuration

Project

Name: cx3_mipicsi_project

Figure 4-5: CX3 Configuration Project Selector

2. An empty **Image Sensor Configuration** window with no pre-defined sensor and frame opens up. A new CX3 Project is created under the Eclipse workspace as well.

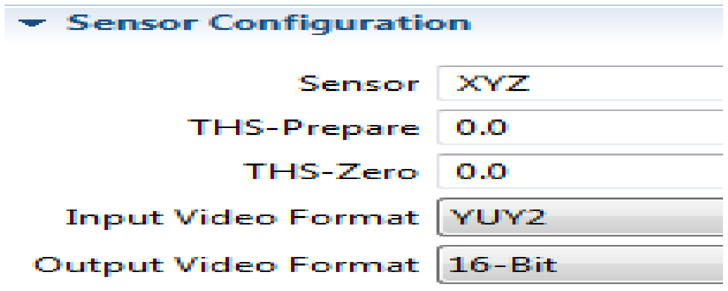

*cx3config.cycx
Image Sensor Configuration
Use Sensor Configuration for CX3 MIPI Receiver Configuration
Sensor Configuration
Frame Configuration

Sensor	
THS-Prepare	0.0 ns
THS-Zero	0.0 ns
Input Video Format	YUY2
Output Video Format	16-Bit

Resolution Name	Data Lanes	CSI Clock [MHz]	H-Active [Pxl]	H-Blanking [Pxl]	V-Active [Line]	V-Blanking [Line]	Frame Rate [fps]	Video Support	Image Support

Figure 4-6: Image Sensor Configuration Stage

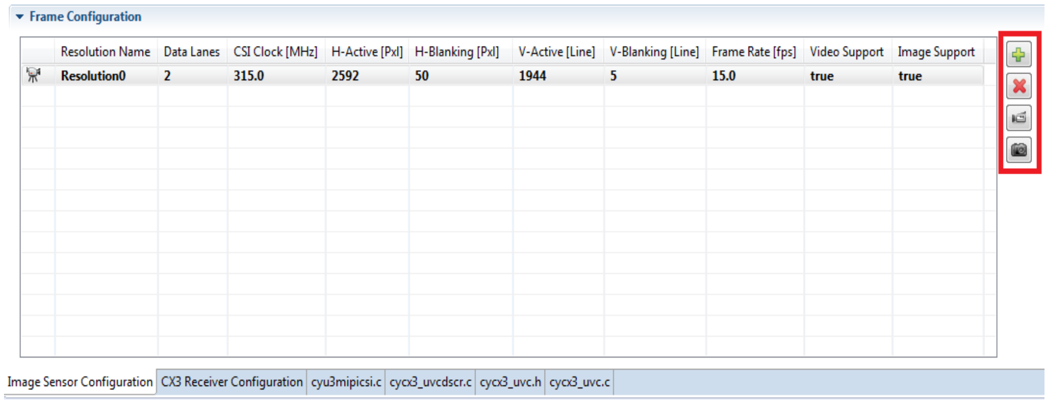
3. Input the desired sensor configuration and select the appropriate video format from the dropdown list provided in the configuration window.
   a. A name string can be provided for the sensor.
   b. The THS-Prepare value represents the duration for which the sensor drives the CSI data lane LP-00 line state before starting the HS transmission. This value can be determined from the image sensor datasheet.
   c. The THS-Zero value represents the duration for which the sensor drives the HS-0 state before transmitting the sync sequence. This value can also be determined from the image sensor datasheet.
   d. The Input Video Format represents the video picture encoding format supported by the sensor. This has to be selected from the drop-down list.
   e. The Output Video Format represents the data size of each pixel in the image stream. This has to be selected from the drop-down list.



Sensor	XYZ
THS-Prepare	0.0
THS-Zero	0.0
Input Video Format	YUY2
Output Video Format	16-Bit

Figure 4-7: Image Sensor Configuration

4. Each Image Sensor may be capable of supporting multiple video resolutions and frame rates. The Frame Configuration area of the Image Sensor configuration wizard can be used to input the supported (desired) frame resolutions and frame rates. The properties to be specified for each frame configuration includes:
   a. A name string to identify the configuration.
   b. The number of CSI-2 data lanes used.
   c. The CSI-2 clock frequency used.
   d. The H-Active value represents the frame width in number of pixels.
   e. The H-Blanking value represents the horizontal blanking period after each line of image data is transferred.
   f. The V-Active value represents the frame height in number of pixels.
   g. The V-Blanking value represents the vertical blanking period after each frame is transferred, and is represented in terms of number of lines.
   h. The streaming rate in terms of number of frames per second.
   i. Support for video and/or still image capture.



Resolution Name	Data Lanes	CSI Clock [MHz]	H-Active [Pxl]	H-Blanking [Pxl]	V-Active [Line]	V-Blanking [Line]	Frame Rate [fps]	Video Support	Image Support
Resolution0	2	315.0	2592	50	1944	5	15.0	true	true

Figure 4-8: Frame Configuration

Multiple user defined frame configurations can be added by pressing **(+)** button for every configuration required. Frame configurations can also be deleted from the current project by pressing (**X)** button. Deleted configurations can be reinstated by pressing using the undo key (**Ctrl+Z)**. Still Image and Video support in CX3 project can be enabled or disabled using the buttons below remove (X) button.

Entering any invalid configuration parameters will trigger warnings at the next stage, i.e., CX3 receiver configuration.

The configuration parameters can be stored as a custom configuration by pressing the **Save** button located at upper right corner of the window. Once saved, these configurations will be available for selection as User defined configurations in other projects.

4.4.2.1.2 Using Pre-Defined Configurations

1. Click on the **Select a Pre-Defined Configuration** Radio Button and select any of the pre-defined configurations from the dropdown list. Configurations for the Aptina AS0260 and OmniVision 5640 sensors are provided in this version of the configuration utility. Provide a name for the newly created project and click on **Finish** button to start generating project configuration data.

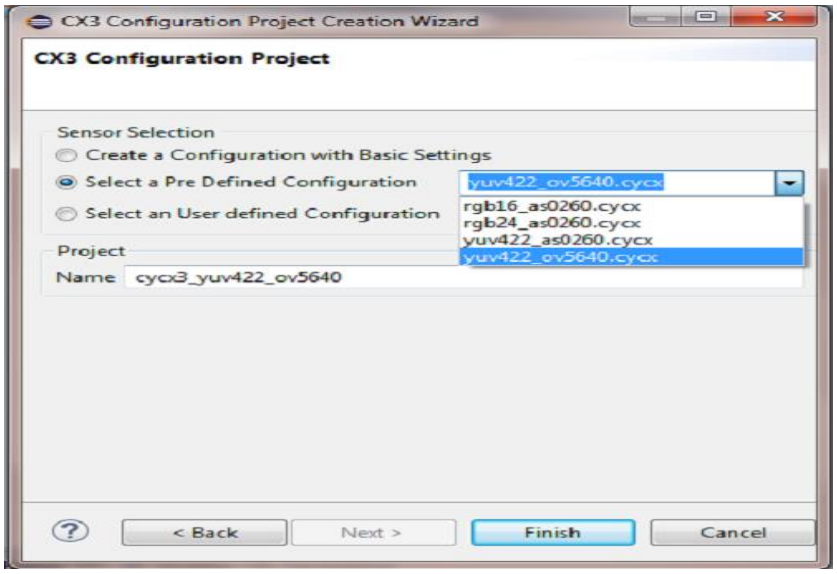
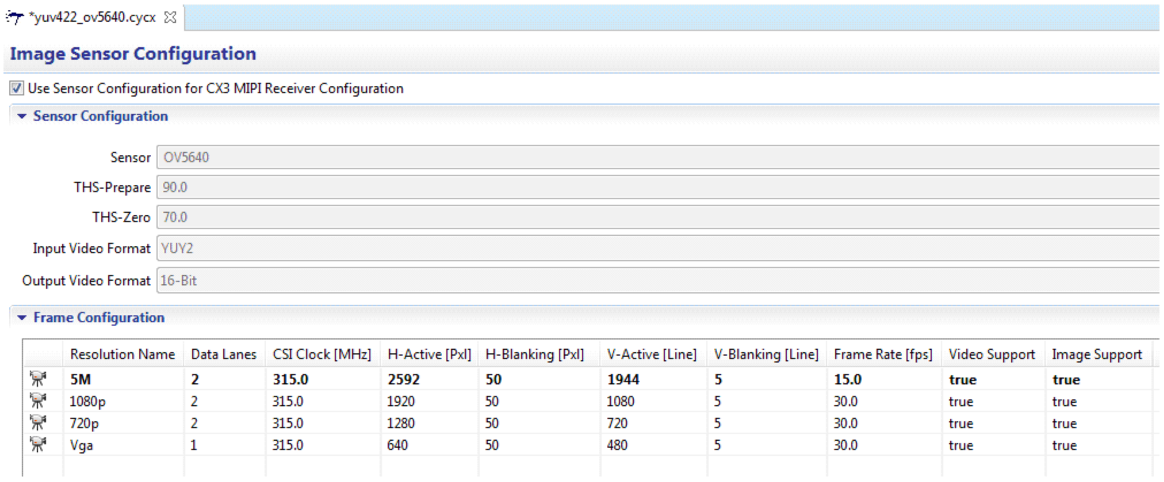

CX3 Configuration Project Creation Wizard
CX3 Configuration Project
Sensor Selection
Create a Configuration with Basic Settings
Select a Pre Defined Configuration
Select an User defined Configuration
yuv422_ov5640.cycx
rgb16_as0260.cycx
rgb24_as0260.cycx
yuv422_as0260.cycx
yuv422_ov5640.cycx
Project
Name
cycx3_yuv422_ov5640
< Back
Next >
Finish
Cancel

Figure 4-9: CX3 Configuration Project Selector

2. **The Image Sensor Configuration** window with pre-loaded sensor and frame configurations pops up. A new firmware project is created under the Eclipse workspace as well.


*yuv422_ov5640.cycx
Image Sensor Configuration
Use Sensor Configuration for CX3 MIPI Receiver Configuration
Sensor Configuration
Frame Configuration

Sensor: OV5640
THS-Prepare: 90.0
THS-Zero: 70.0
Input Video Format: YUY2
Output Video Format: 16-Bit

Resolution Name	Data Lanes	CSI Clock [MHz]	H-Active [Pxl]	H-Blanking [Pxl]	V-Active [Line]	V-Blanking [Line]	Frame Rate [fps]	Video Support	Image Support
5M	**2**	**315.0**	**2592**	**50**	**1944**	**5**	**15.0**	**true**	**true**
1080p	2	315.0	1920	50	1080	5	30.0	true	true
720p	2	315.0	1280	50	720	5	30.0	true	true
Vga	1	315.0	640	50	480	5	30.0	true	true

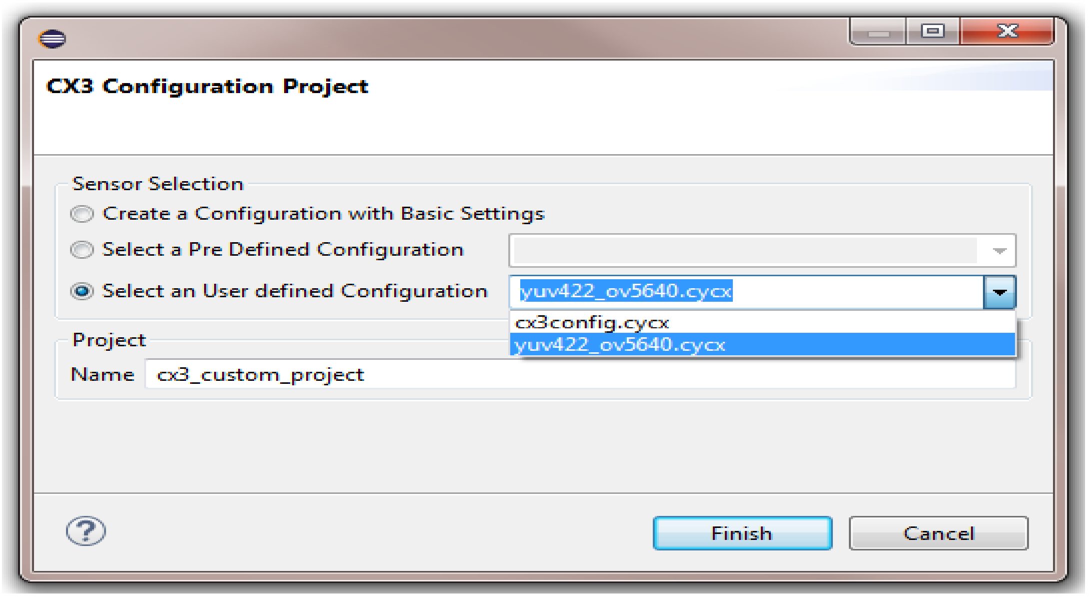
Figure 4-10: Using pre-defined Image Sensor Configurations

The Image Sensor Configuration window lists the pre-defined sensor and frame configurations for the selected Image Sensor. Listed frame configurations can be deleted from the current project but cannot be altered with new values. Deleted configurations can be reinstated using the undo key (**Ctrl+Z)**.

Configuration parameters cannot be modified in pre-defined configuration option, but can be stored as a custom configuration by pressing **Save** button located at upper right corner of window.

4.4.2.1.3 User Defined Configuration

A sensor configuration created and saved using the steps in Section 4.4.2.1.1, can be re-used to create new CX3 projects. To do this, click on **Select an User-Defined Configuration** Radio Button and select the desired configuration from the dropdown box. Name the newly created project and click on **Finish** button to start generating project configuration data.


CX3 Configuration Project

Sensor Selection
- Create a Configuration with Basic Settings
- Select a Pre Defined Configuration
- [✓ Select an User defined Configuration] yuv422_ov5640.cycx
  - cx3config.cycx
  - yuv422_ov5640.cycx

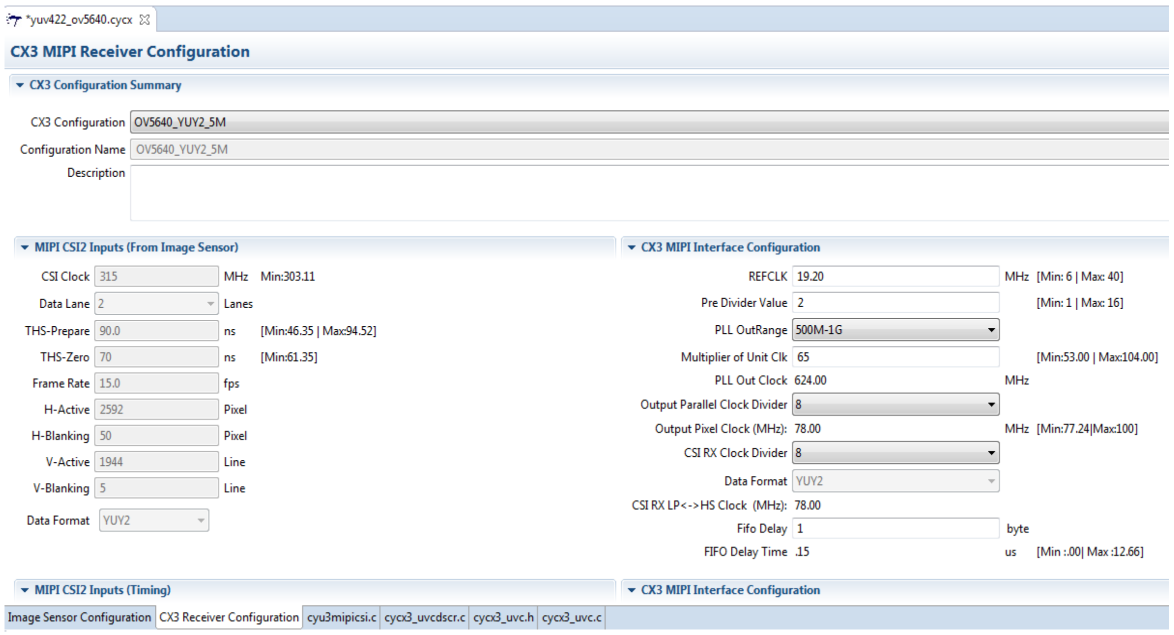
Project
Name: cx3_custom_project

Figure 4-11: Selecting a user defined image sensor configuration

Refer to Section 4.4.2.1.2 for the steps to modify the image sensor configuration.

4.4.3 CX3 MIPI Receiver Configuration

Once the Image Sensor Configuration is complete, select the **CX3 MIPI Receiver Configuration** tab positioned at the bottom of currently opened window.



*yuv422_ov5640.cycx

CX3 MIPI Receiver Configuration

CX3 Configuration Summary

CX3 Configuration: OV5640_YUY2_5M
Configuration Name: OV5640_YUY2_5M
Description:

MIPI CSI2 Inputs (From Image Sensor)

CSI Clock	315	MHz	Min:303.11
Data Lane	2	Lanes	
THS-Prepare	90.0	ns	[Min:46.35 | Max:94.52]
THS-Zero	70	ns	[Min:61.35]
Frame Rate	15.0	fps	
H-Active	2592	Pixel	
H-Blanking	50	Pixel	
V-Active	1944	Line	
V-Blanking	5	Line	
Data Format	YUY2		

CX3 MIPI Interface Configuration

REFCLK	19.20	MHz	[Min: 6 | Max: 40]
Pre Divider Value	2		[Min: 1 | Max: 16]
PLL OutRange	500M-1G		
Multiplier of Unit Clk	65		[Min:53.00 | Max:104.00]
PLL Out Clock	624.00	MHz	
Output Parallel Clock Divider	8		
Output Pixel Clock (MHz):	78.00	MHz	[Min:77.24|Max:100]
CSI RX Clock Divider	8		
Data Format	YUY2		
CSI RX LP<->HS Clock (MHz):	78.00		
Fifo Delay	1	byte	
FIFO Delay Time	.15	us	[Min :.00| Max :12.66]

MIPI CSI2 Inputs (Timing) | CX3 MIPI Interface Configuration

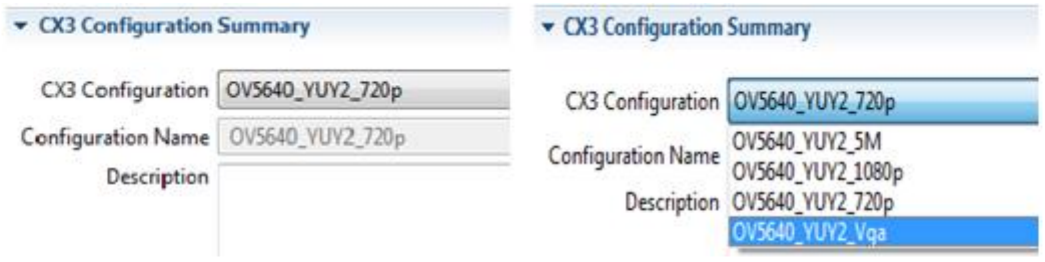
Image Sensor Configuration | CX3 Receiver Configuration | cyu3mipicsi.c | cycx3_uvcdscr.c | cycx3_uvc.h | cycx3_uvc.c

Figure 4-12: CX3 MIPI Receiver Configuration Stage

The CX3 MIPI Receiver Configuration Window is split into the following sub-sections:

4.4.3.1 CX3 Configuration Summary

The CX3 Configuration Summary section allows you to choose from among the frame configurations entered in the Image Sensor Configuration phase. When a frame configuration is selected from the drop-down list, the CSI-2 interface properties for the corresponding frame configuration will be displayed in the other sections of the window. The description text box allows you to provide a description for the configuration which will be added to the generated firmware source in the form of comments. Therefore, the description text should not contain ‘/*’ or ‘*/’ character combinations.



CX3 Configuration Summary

CX3 Configuration: OV5640_YUY2_720p
Configuration Name: OV5640_YUY2_720p
Description:

CX3 Configuration Summary

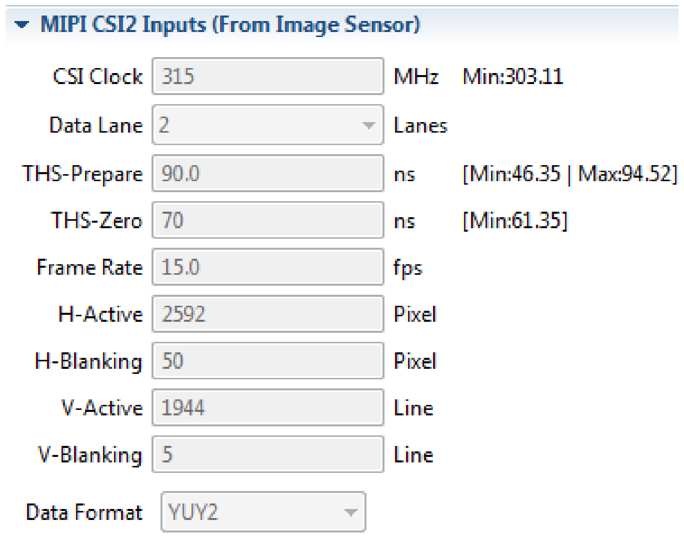
CX3 Configuration: OV5640_YUY2_720p
Configuration Name
Description

OV5640_YUY2_5M
OV5640_YUY2_1080p
OV5640_YUY2_720p
OV5640_YUY2_Vga

Figure 4-13: CX3 Configuration Summary

4.4.3.2 MIPI CSI-2 Inputs

The MIPI CSI-2 inputs section will display values passed from Image Sensor Configuration Stage for matching the Image Sensor's output parameters. These values are provided for information, and cannot be modified in this tab. You have to go back to the Image Sensor Configuration tab in order to change any of these settings.



MIPI CSI2 Inputs (From Image Sensor)

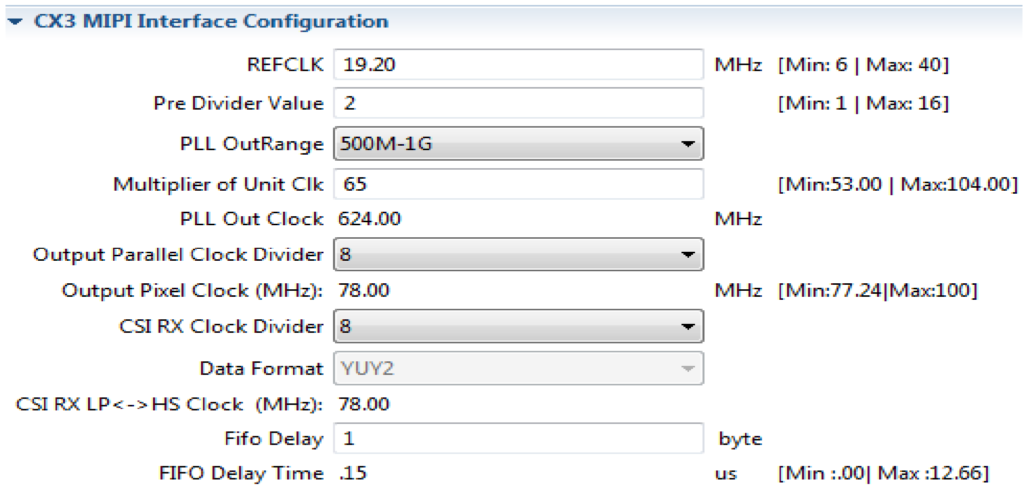
CSI Clock	315	MHz	Min:303.11
Data Lane	2	Lanes	
THS-Prepare	90.0	ns	[Min:46.35 | Max:94.52]
THS-Zero	70	ns	[Min:61.35]
Frame Rate	15.0	fps	
H-Active	2592	Pixel	
H-Blanking	50	Pixel	
V-Active	1944	Line	
V-Blanking	5	Line	
Data Format	YUY2		

Figure 4-14: MIPI CSI-2 Inputs

The CSI Clock frequency and the Data Lane entries along with the frame properties are used to compute the total input data throughput on the GPIF-II interface from the MIPI CSI-2 block.

4.4.3.3 CX3 MIPI Interface Configuration

The MIPI Interface Configuration section lists the CX3 MIPI CSI-2 configuration that has been computed based on the Image Sensor and frame properties. These value typically do not need to be modified, though the tool allows the user to make modifications.



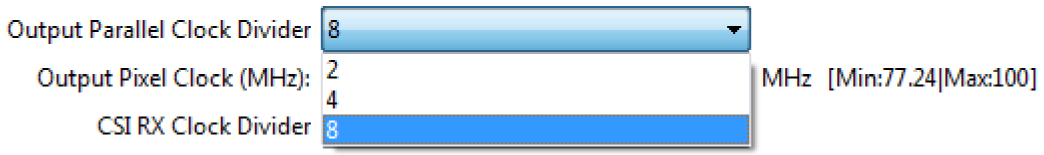
CX3 MIPI Interface Configuration

REFCLK	19.20	MHz	[Min: 6 | Max: 40]
Pre Divider Value	2		[Min: 1 | Max: 16]
PLL OutRange	500M-1G		
Multiplier of Unit Clk	65		[Min:53.00 | Max:104.00]
PLL Out Clock	624.00	MHz	
Output Parallel Clock Divider	8		
Output Pixel Clock (MHz):	78.00	MHz	[Min:77.24|Max:100]
CSI RX Clock Divider	8		
Data Format	YUY2		
CSI RX LP<->HS Clock (MHz):	78.00		
Fifo Delay	1	byte	
FIFO Delay Time	.15	us	[Min :.00| Max :12.66]

Figure 4-15: CX3 MIPI Interface Configurations

The first field in this section is the REFCLK frequency being provided to the block. This value along with the Pre Divider Value, PLL Out Range and Multiplier of Unit Clock is used to generate the pllPRD, pllFbd and pllFrs parameters which provide the PLL_CLK frequency using the equation listed in Section 4.2.3.

Once the PLL clock frequency has been set, the Output Pixel Clock (PCLK) and CSI RX clock frequency can be obtained by selecting the appropriate divider value as shown in the image below.



Output Parallel Clock Divider 8

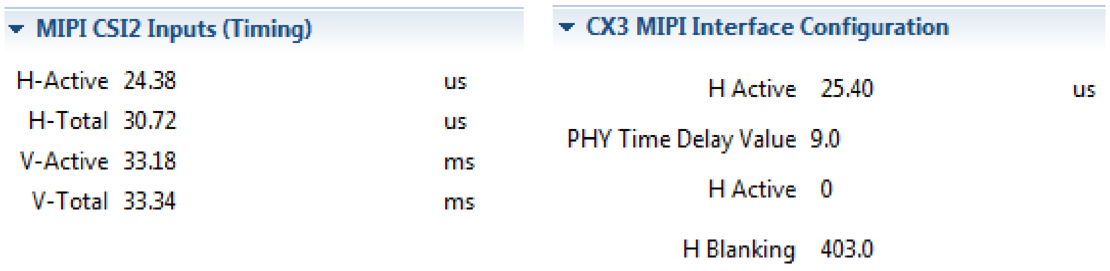
Output Pixel Clock (MHz): MHz [Min:77.24|Max:100]

CSI RX Clock Divider

Figure 4-16: Selecting Clock Dividers

4.4.3.4 *MIPI CSI2 Inputs / CX3 MIPI Interface Configuration*

These values are calculated by the utility and are fixed for each frame configurations. These values serve as input and output parameters for the MIPI interface part on CX3, and are provided here for information.



MIPI CSI2 Inputs (Timing)

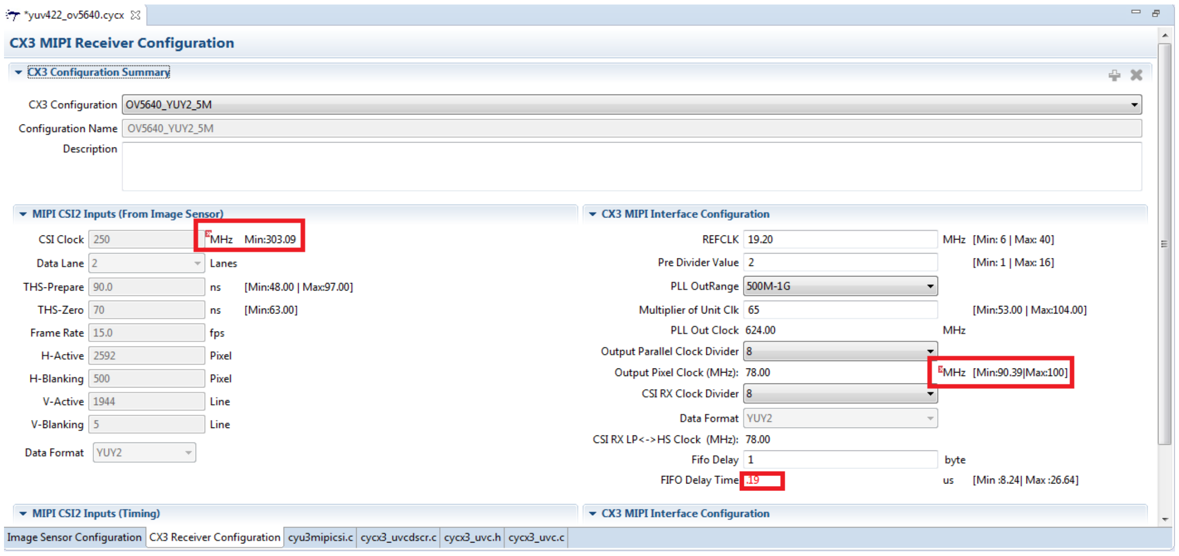
H-Active	24.38	us
H-Total	30.72	us
V-Active	33.18	ms
V-Total	33.34	ms

CX3 MIPI Interface Configuration

H Active	25.40	us
PHY Time Delay Value	9.0	
H Active	0	
H Blanking	403.0	

Figure 4-17: MIPI Inputs

The CX3 MIPI Receiver Configuration window will throw warnings if any invalid parameters had been provided while configuring the sensor or by modifying the values in the CX3 MIPI Interface Configuration section. These warnings can be seen by hovering the mouse cursor over the warning symbols painted in color red. These warnings can be rectified by providing acceptable values in the sensor configuration tab.


*yuv422_ov5640.cycx

CX3 MIPI Receiver Configuration

CX3 Configuration Summary

CX3 Configuration: OV5640_YUY2_5M
Configuration Name: OV5640_YUY2_5M
Description:

MIPI CSI2 Inputs (From Image Sensor)

CSI Clock: 250 MHz Min:303.09
Data Lane: 2 Lanes
THS-Prepare: 90.0 ns [Min:48.00 | Max:97.00]
THS-Zero: 70 ns [Min:63.00]
Frame Rate: 15.0 fps
H-Active: 2592 Pixel
H-Blanking: 500 Pixel
V-Active: 1944 Line
V-Blanking: 5 Line
Data Format: YUY2

CX3 MIPI Interface Configuration

REFCLK: 19.20 MHz [Min: 6 | Max: 40]
Pre Divider Value: 2 [Min: 1 | Max: 16]
PLL OutRange: 500M-1G
Multiplier of Unit Clk: 65 [Min:53.00 | Max:104.00]
PLL Out Clock: 624.00 MHz
Output Parallel Clock Divider: 8
Output Pixel Clock (MHz): 78.00 MHz [Min:90.39|Max:100]
CSI RX Clock Divider: 8
Data Format: YUY2
CSI RX LP<->HS Clock (MHz): 78.00
Fifo Delay: 1 byte
FIFO Delay Time: .19 us [Min :8.24| Max :26.64]

MIPI CSI2 Inputs (Timing)

CX3 MIPI Interface Configuration

Image Sensor Configuration | CX3 Receiver Configuration | cyu3mipicsi.c | cycx3_uvcdscr.c | cycx3_uvc.h | cycx3_uvc.c

Figure 4-18: Warnings shown in CX3 MIPI Receiver Configuration tab

Please note that the warnings do not block the tool from generating the interface configuration source code. These only indicate errors detected during sanity checks performed by the tool, and can be over-ridden if the user is sure that the changes are valid.

4.4.4 Adding the Configuration to the Firmware Project

As mentioned above, the CX3 configuration utility generates the following pieces of C source code, which can be added to the application firmware project.

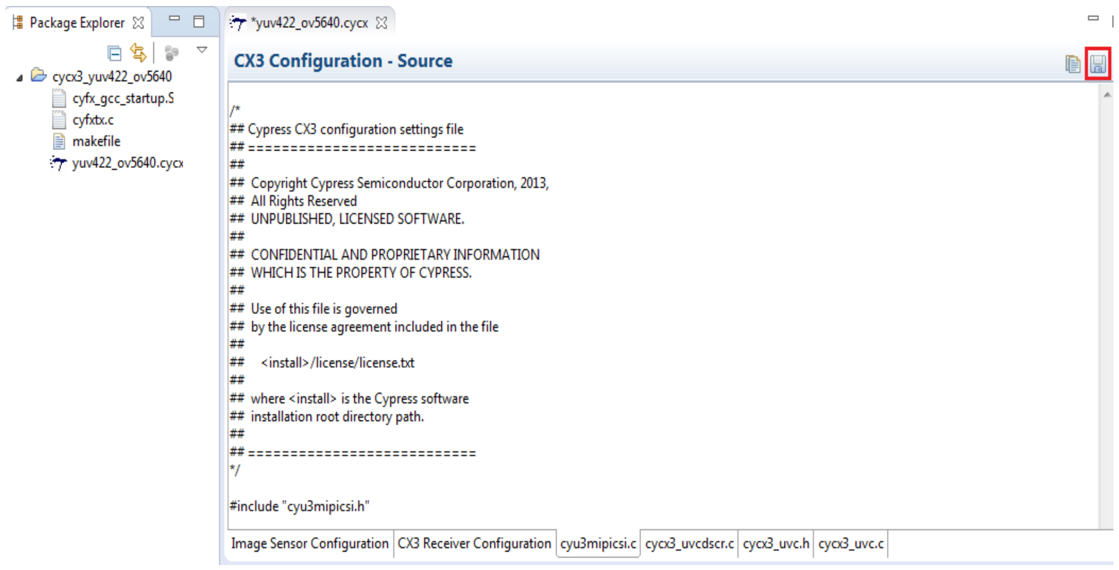
1. MIPI CSI-2 Configuration data for each frame configuration
2. USB descriptors for the UVC application
3. Header file defining the USB pipe and DMA buffer properties.
4. Source file implementing the actual video streaming logic.

The following sub-sections describe the procedure to add each of these to the firmware project.

4.4.4.1 Adding the CX3 CSI-2 Configurations

Move to the “cyu3mipicsi.c” tab once all CX3 MIPI CSI-2 Interface related configuration parameters are finalized. This tab contains the generated data structures that provide the CSI-2 configuration parameters corresponding to each frame configuration in the project.

This code can be saved to the related CX3 project, by pressing the **Save** button located at the upper right corner of window.

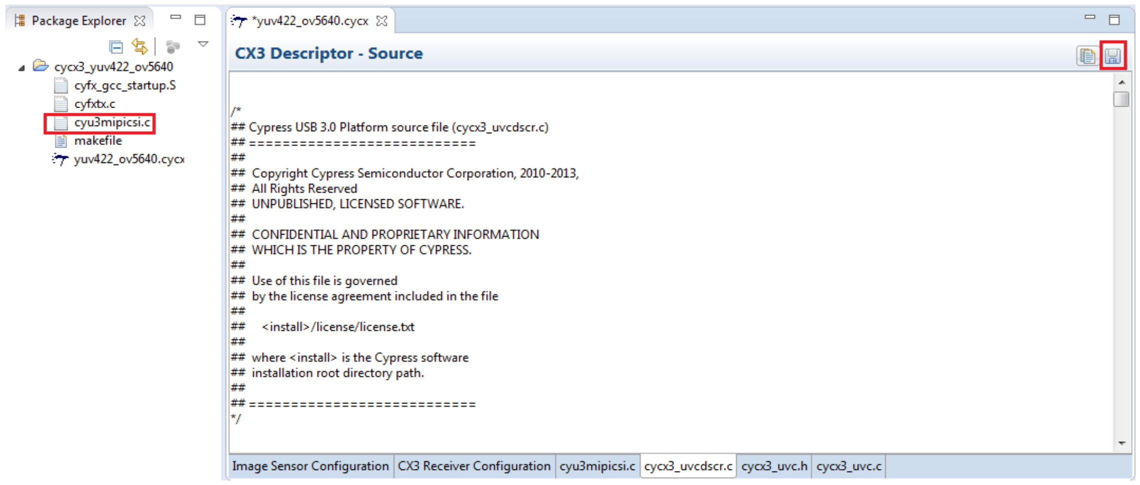


```
/*
## Cypress CX3 configuration settings file
## ===========================
##
## Copyright Cypress Semiconductor Corporation, 2013,
## All Rights Reserved
## UNPUBLISHED, LICENSED SOFTWARE.
##
## CONFIDENTIAL AND PROPRIETARY INFORMATION
## WHICH IS THE PROPERTY OF CYPRESS.
##
## Use of this file is governed
## by the license agreement included in the file
##
##    <install>/license/license.txt
##
## where <install> is the Cypress software
## installation root directory path.
##
## ===========================
*/

#include "cyu3mipicsi.h"
```

Figure 4-19: Utility generated CX3 Configuration Source file

4.4.4.2 *Adding the USB Descriptors*

Move to the “cycx3_uvcdscr.c” to view the USB descriptors that are generated based on the configuration provided. The USB configuration descriptors will contain frame descriptors corresponding to each of the frame configurations in the project. This file can be added to the related CX3 project by pressing the **Save** button located at the upper right corner of window.

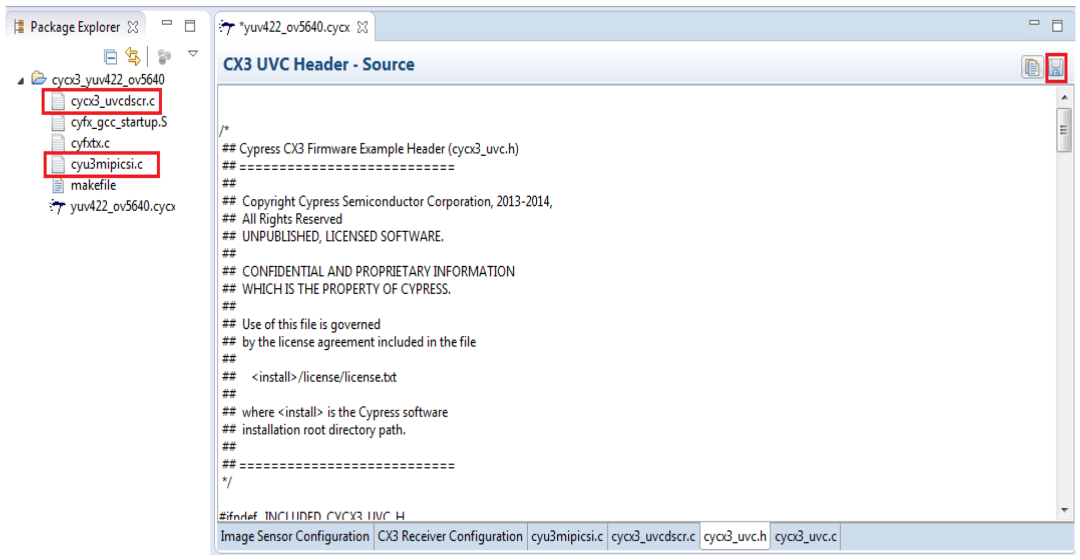


```
/*
## Cypress USB 3.0 Platform source file (cycx3_uvcdscr.c)
## ===========================
##
## Copyright Cypress Semiconductor Corporation, 2010-2013,
## All Rights Reserved
## UNPUBLISHED, LICENSED SOFTWARE.
##
## CONFIDENTIAL AND PROPRIETARY INFORMATION
## WHICH IS THE PROPERTY OF CYPRESS.
##
## Use of this file is governed
## by the license agreement included in the file
##
##    <install>/license/license.txt
##
## where <install> is the Cypress software
## installation root directory path.
##
## ===========================
*/
```

Figure 4-20: Utility generated CX3 Descriptor Source file

4.4.4.3 *Adding CX3 UVC Definitions*

Move to the “cycx3_uvc.h” tab to view the application definitions generated based on the streaming parameters. These definitions can be added to the related CX3 project by pressing the **Save** button located at the upper right corner of window.



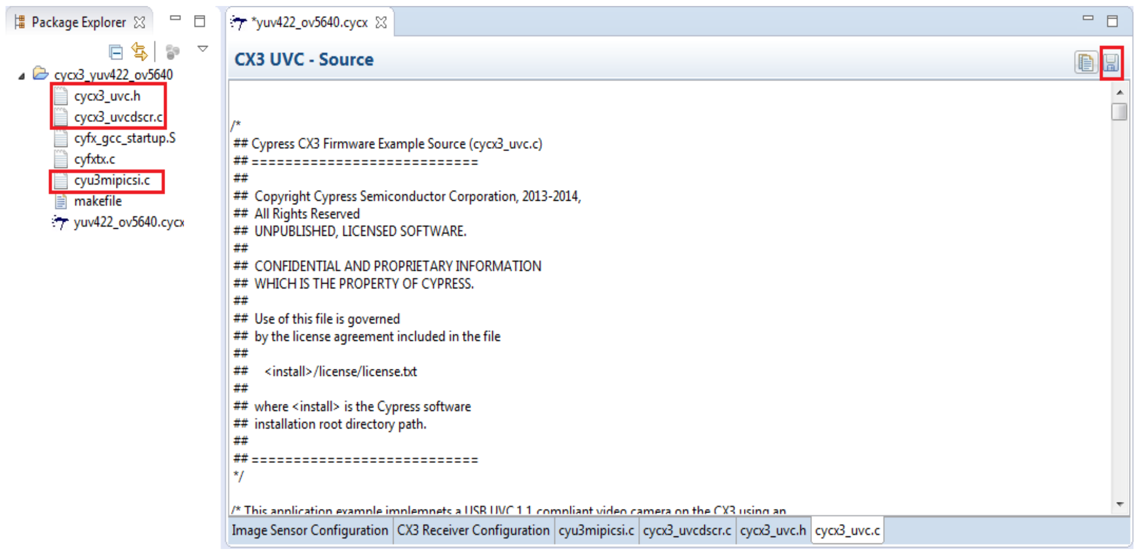
```
/*
## Cypress CX3 Firmware Example Header (cycx3_uvc.h)
## ===========================
##
## Copyright Cypress Semiconductor Corporation, 2013-2014,
## All Rights Reserved
## UNPUBLISHED, LICENSED SOFTWARE.
##
## CONFIDENTIAL AND PROPRIETARY INFORMATION
## WHICH IS THE PROPERTY OF CYPRESS.
##
## Use of this file is governed
## by the license agreement included in the file
##
##    <install>/license/license.txt
##
## where <install> is the Cypress software
## installation root directory path.
##
## ===========================
*/

#ifndef _INCLUDED_CYCX3_UVC_H_
```

Figure 4-21: Utility generated CX3 Header file

4.4.4.4 *Adding CX3 UVC application logic*

Move to the “cycx3_uvc.c” tab to view the generated UVC application source code. This file can be added to the related CX3 project by pressing the **Save** button located at the upper right corner of window.

After saving to the project, the contents of this file can then be edited to add custom handling for any UVC Video Control requests.

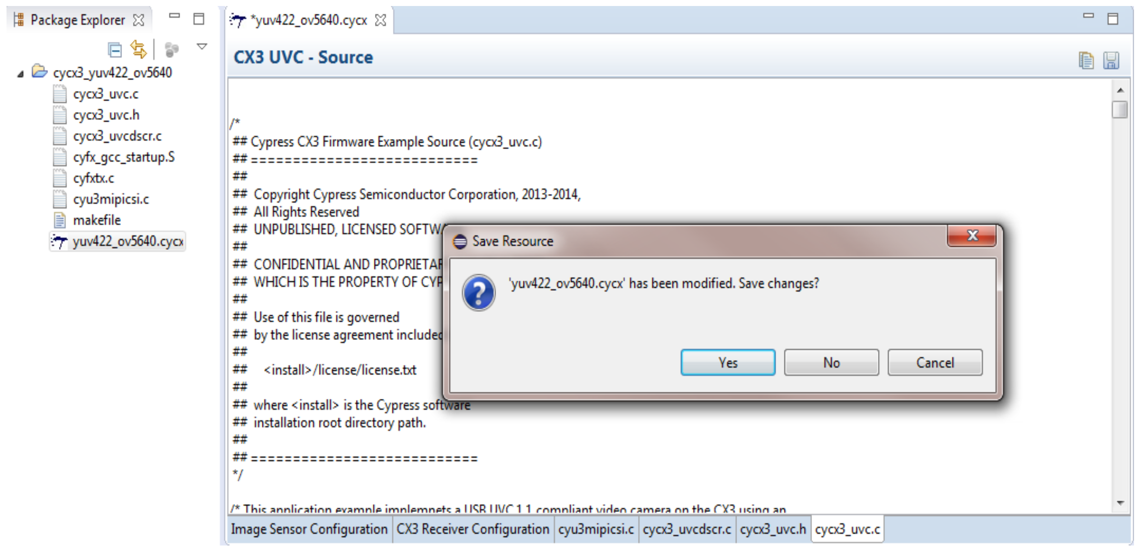


```
/*
## Cypress CX3 Firmware Example Source (cycx3_uvc.c)
## ===========================
##
## Copyright Cypress Semiconductor Corporation, 2013-2014,
## All Rights Reserved
## UNPUBLISHED, LICENSED SOFTWARE.
##
## CONFIDENTIAL AND PROPRIETARY INFORMATION
## WHICH IS THE PROPERTY OF CYPRESS.
##
## Use of this file is governed
## by the license agreement included in the file
##
##    <install>/license/license.txt
##
## where <install> is the Cypress software
## installation root directory path.
##
## ===========================
*/

/* This application example implemnets a USB UVC 1.1 compliant video camera on the CX3 using an
```

Figure 4-22: Utility generated CX3 UVC Source file

4.4.5 Locking the CX3 Project Configuration

Click the Close (X) Button to exit from the configuration utility once the project configuration is completed successfully and the code has been saved. The utility will prompt for saving any changes.



```
/*
## Cypress CX3 Firmware Example Source (cycx3_uvc.c)
## ===========================
##
## Copyright Cypress Semiconductor Corporation, 2013-2014,
## All Rights Reserved
## UNPUBLISHED, LICENSED SOFTW
##
## CONFIDENTIAL AND PROPRIETAR
## WHICH IS THE PROPERTY OF CYP
##
## Use of this file is governed
## by the license agreement include
##
##    <install>/license/license.txt
##
## where <install> is the Cypress software
## installation root directory path.
##
## ===========================
*/

/* This application example implemnets a USB UVC 1.1 compliant video camera on the CX3 using an
```

Save Resource

'yuv422_ov5640.cycx' has been modified. Save changes?

Figure 4-23: CX3 Project Configuration Lock Window

Code for the USB descriptors, sensor and CSI-2 interface configuration structures and streaming application logic would have been generated by the tool and added to the project. The project can now be compiled and used like any standard FX3/CX3 firmware project.

4.5 GPIF-II interface on the CX3

The CX3 device makes use of the GPIF-II interface to connect the DMA and USB blocks to the MIPI CSI-2 interface. Since the GPIF-II Interface connectivity is fixed, a custom GPIF-II configuration is not useful. A fixed-function GPIF-II Configuration is used by the CX3 firmware to move the data from the CSI-2 interface block to the DMA block.

4.5.1 GPIF-II Waveform

The GPIF II state machine on the CX3 implements the state machine shown in Figure 4-24. The state machine makes the parallel data provided by the MIPI CSI-2 available for transfer over two GPIF-II sockets which can be connected to a Many to One DMA channel.

The functionality of this state machine is similar to the GPIF II state machine described in Application Note AN75779 - How to Implement an Image Sensor Interface with EZ-USB® FX3™ in a USB Video Class (UVC) Framework.

Detailed description of this state machine along with instructions on creating CX3 based application projects is provided in the Application Note AN90369 - How to interface a MIPI CSI-2 Image Sensor with EZ-USB® CX3.

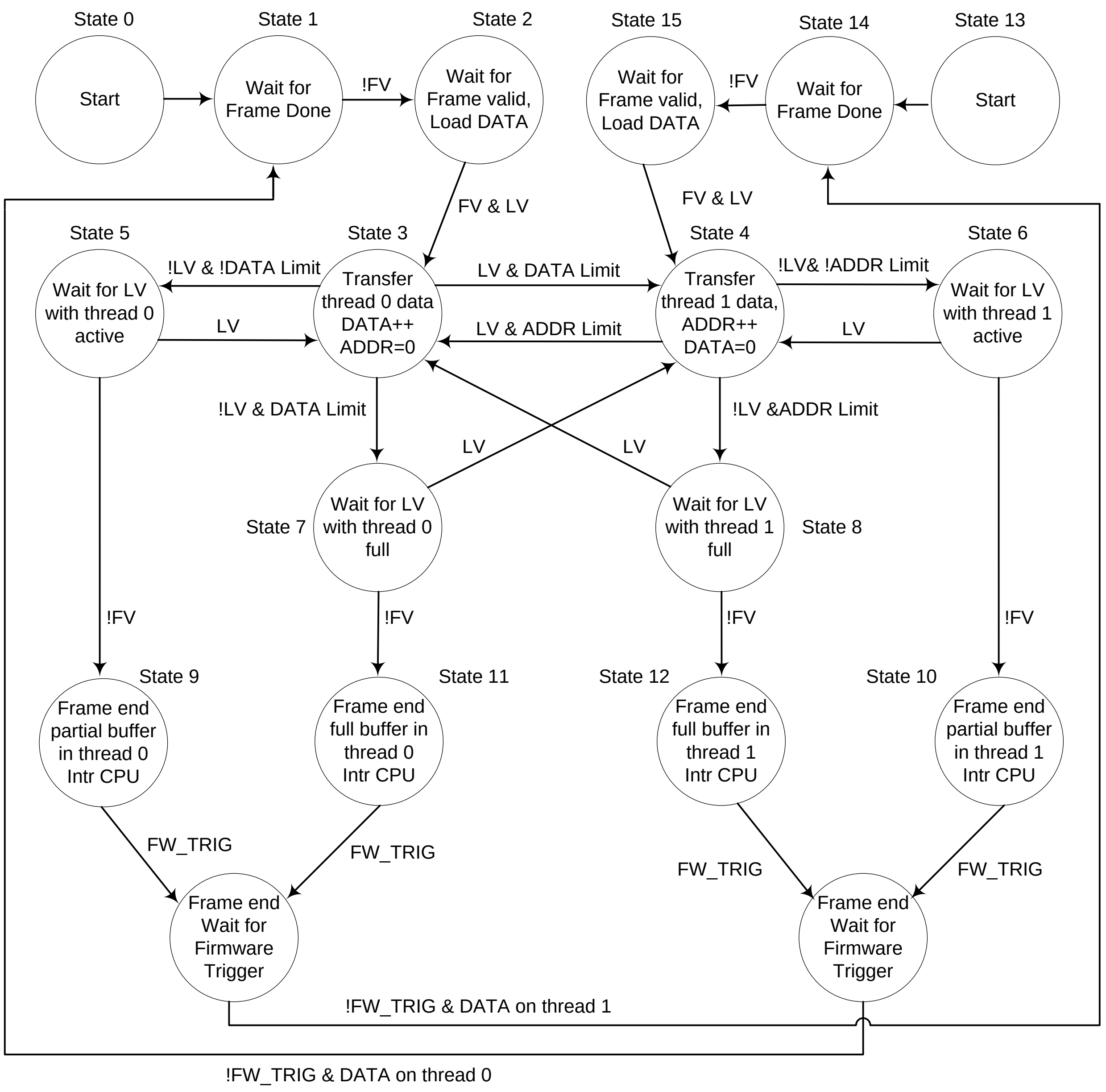

State 0
Start
State 1
Wait for Frame Done
!FV
State 2
Wait for Frame valid, Load DATA
State 15
Wait for Frame valid, Load DATA
!FV
State 14
Wait for Frame Done
State 13
Start
FV & LV
FV & LV
State 5
Wait for LV with thread 0 active
!LV & !DATA Limit
LV
State 3
Transfer thread 0 data DATA++ ADDR=0
LV & DATA Limit
LV & ADDR Limit
State 4
Transfer thread 1 data, ADDR++ DATA=0
!LV& !ADDR Limit
LV
State 6
Wait for LV with thread 1 active
!LV & DATA Limit
!LV &ADDR Limit
LV
LV
State 7
Wait for LV with thread 0 full
Wait for LV with thread 1 full
State 8
!FV
!FV
!FV
!FV
State 9
Frame end partial buffer in thread 0 Intr CPU
State 11
Frame end full buffer in thread 0 Intr CPU
State 12
Frame end full buffer in thread 1 Intr CPU
State 10
Frame end partial buffer in thread 1 Intr CPU
FW_TRIG
FW_TRIG
FW_TRIG
FW_TRIG
Frame end Wait for Firmware Trigger
Frame end Wait for Firmware Trigger
!FW_TRIG & DATA on thread 1
!FW_TRIG & DATA on thread 0

Figure 4-24: CX3 GPIFII State Machine