



## EARTH PEOPLE TECHNOLOGY, Inc

# FPGA DEVELOPMENT SYSTEM FOR THE ARDUINO DUE User Manual

The DueProLogic (DPL) and its integrated development and distinctive runtime environment has been specifically designed for Electrical Engineering students, hobbyists, and entrepreneurs prototyping/developing/running projects involving logic, with the added opportunity, of readily mating with a widely used microprocessor board, the Arduino Due, and other ARM Cortex compatibles . The combination of FPGA programmable logic and a microcontroller is unbeatable in an educational student learning setting and in many other projects where each can bring its strength.

The DPL FPGA development system provides a convenient, user-friendly work flow by connecting seamlessly with Altera's Quartus II software. The user will develop the code in the Quartus environment on a Windows Personal Computer. The programmable logic code is loaded into the FPGA using only the Quartus Programmer tool and a standard USB cable. The Active Host SDK provides a highly configurable bi-directional communications interface between Arduino and host. It connects transparently with the Active Transfer Library in the FPGA code. This Active Host/Active Transfer combination eliminates the complexity of designing a USB communication system. No scheduling USB transfers, USB driver interface or inf file changes are needed. The EPT FPGA development system is a unique combination of hardware and software.

Circuit designs, software and documentation are copyright © 2015, Earth People Technology, Inc

Microsoft and Windows are both registered trademarks of Microsoft Corporation.  
Altera is a trademark of the Altera Corporation. All other trademarks referenced herein are the property of their respective owners and no trademark rights to the same are claimed.

<http://www.earthpeopletechnology.com/>



## Table of Contents

1	Introduction and General Description .....	4
1.1	Test Driving the Active Host Test Application .....	5
1.2	EPT-4CE6-AF-D1 .....	9
1.2.1	Inputs and Outputs .....	11
1.2.2	FPGA Configuration .....	11
1.2.3	FT2232H Dual Channel USB to Serial Chip .....	11
1.2.4	SD Card Interface .....	12
1.3	FPGA Active Host Development .....	12
1.4	Active Host EndTerms .....	12
1.5	Active Transfer EndTerms .....	15
2	EPT Drivers .....	16
2.1	USB Driver .....	16
2.2	JTAG DLL Insert to Quartus II .....	22
2.2.1	Installing Quartus .....	22
2.2.2	Downloading Quartus .....	23
2.2.3	Quartus Installer .....	26
2.2.4	Adding the EPT_Blaster to Quartus II .....	30
2.3	Active Host Application DLL .....	32
		FPGA .....
		36 .....
3	Active Transfer Library .....	36
3.1	EPT Active Transfer System Overview .....	37
3.2	Active Transfer Library .....	38
3.2.1	Active Trigger EndTerm .....	40
3.2.2	Active Transfer EndTerm .....	44
3.2.3	Active Block EndTerm .....	46
3.3	Timing Diagram for Active Transfer EndTerms .....	49
3.3.1	Active Trigger EndTerm Timing .....	49
3.3.2	Active Transfer EndTerm Timing .....	49
3.3.3	Active Block EndTerm Timing .....	50
4	Compiling, Synthesizing, and Programming FPGA .....	50
4.1	Setting up the Project and Compiling .....	51
4.1.1	Selecting Pins and Synthesizing .....	57
4.1.2	Configuring the FPGA .....	65
5	Active Host Application .....	72
5.1	Trigger EndTerm .....	72
5.2	Transfer(Byte) EndTerm .....	73
5.3	Block EndTerm .....	73
5.4	Active Host DLL .....	73
5.4.1	Active Host Open Device .....	75
5.4.2	Active Host Read Callback Function .....	77



## FPGA Development System User Manual

5.4.3	Active Host Triggers .....	78
5.4.4	Active Host Byte Transfers .....	80
5.4.5	Active Host Block Transfers .....	82
6	Assembling, Building, and Executing a .NET Project on the PC .....	85
6.1	Creating a Project .....	86
6.1.1	Setting up the C# Express Environment x64 bit .....	87
6.2	Assembling Files into the Project.....	94
6.2.1	Changing Project Name .....	94
6.2.2	Add Files to Project.....	96
6.2.3	Adding Controls to the Project.....	98
6.2.4	Adding the DLL's to the Project.....	102
6.2.5	Building the Project .....	103
6.2.6	Testing the Project.....	104
	.....FPGA .....	108
7	Development Process .....	108
7.1	Designing a Simple Data Collection Sampler .....	108
7.1.1	The Arduino Microcontroller Board .....	109
Create Data Generator .....	109	
7.1.2	.....	109
7.1.3	Select I/O's for Fast Throughput on Arduino .....	110
7.1.4	Coding the Arduino Data Sampler.....	111
7.1.5	Building Arduino Project .....	114
7.1.6	Programming the Arduino.....	117
7.1.7	FPGA Active Transfer EndTerm Coding and Initiation .....	120
7.1.8	FPGA: Define the User Design.....	120
7.1.9	FPGA: Compile/Synthesize the Project .....	138
7.1.10	FPGA: Program the DPL Flash .....	142
7.1.11	PC: Design the Project .....	145
7.1.12	PC: Coding the Project.....	146
7.1.13	PC: Compiling the Active Host Application.....	157
7.1.14	Adding the DLL's to the Project.....	158
7.1.15	Connecting the Project Together.....	159
7.1.16	Testing the Project.....	163



## 1 Introduction and General Description

The **DPL** gives learners the opportunity to have an appropriate hands-on approach when learning logic, exploring different iterations of schematic/code designs with simple uploads of the design, and the operation of those circuits with relatively easy runtime passing of project parameters and data, and an abundance of headers that can interface to external components, without having to spend inordinate amounts of time reading datasheets, designing the right combinations of gates on multi-gate chips, and building/revising/debugging/revising repeatedly... spaghetti bowls of wires and chips on multiple breadboards to connect to those same external components.

With the **DPL**'s FPGA, projects can also more easily be attempted which rely on asynchronous, exceedingly fast, and even multiple separate concurrent logic structures operating in parallel which would have traditionally required a plethora of chip gates or multiple high speed microprocessors to implement parallel processes. Logic circuits are implemented within the FPGA at few-nanosecond gate speeds and highly parallel in operation, effectively a few hundred MHz; Microprocessors often rely on inherently slower single threaded program loops with interrupt servicing, which is typically much slower. Programmable logic is today's technology for logic learners and implementers, replacing discrete logic chips. The **DPL** allows the learner to be more productive and better focus on the underlying logic and integration with the non-logic aspects of non-trivial projects.

The Earth People Technology FPGA development system hardware consists of a High Speed (480 Mb/s) USB to parallel (8 bit) bus chip and an FPGA. The USB interface provides both Configuration of the FPGA and a High Speed transfer path. The software consists of the Active Host SDK for the PC. The firmware includes the Active Transfer Library which is used in the FPGA to provide advanced functions for control and data transfer to/from the Arduino.



## FPGA Development System User Manual

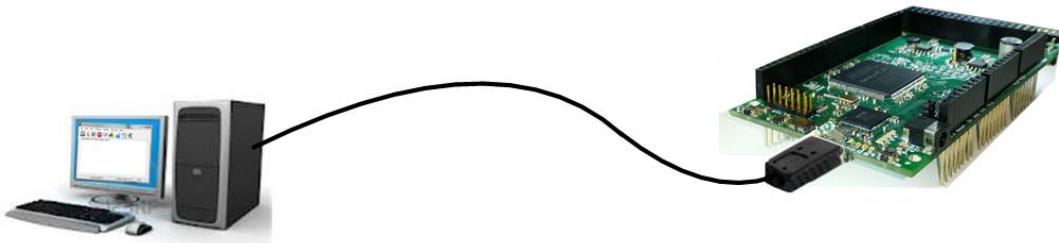


The EPT FPGA Development System allows users to write HDL code (either Verilog or VHDL) that will implement any digital logic circuit. The user's HDL code is compiled and synthesized and packaged into a programming file. The programming file is programmed into the Configuration Flash using one channel of the USB to Serial chip, the FT2232H. The Active Host SDK contains a dll which maintains device connection, polling, writes and includes a unique receive mechanism that automatically transfers data from DPL when data is ready. It also alerts the user code when the dll has stored the transfer and the data is available to the software GUI (graphical user interface). Users do not need to interface with the USB Host Driver or any Windows drivers. They need only to include the Active Host dll in their projects. The Active Transfer Libraries must be included in the FPGA project to take advantage of the configurability of the Active Host SDK. All of the drivers, libraries, and project source code are available at [www.earthpeopletechnology.com](http://www.earthpeopletechnology.com).

### ***1.1 Test Driving the Active Host Test Application***

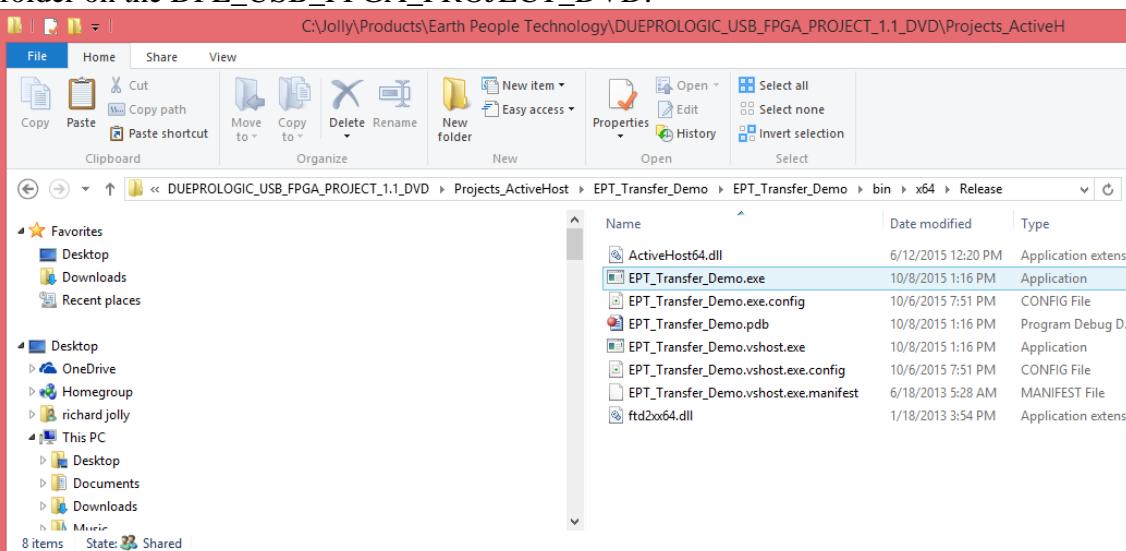
The EPT-4CE6-AF-D1 board comes pre-loaded with the EPT\_Transfer\_Demo HDL project in the FPGA. This project allows the user to test out the functions of the Active Host API and the board hardware.

To test drive the application, connect the DPL to the Windows PC using a Micro B USB cable. Load the driver for the board. See the section “EPT Drivers” for instructions on loading the DPL driver. If the USB driver fails to load, the Windows OS will indicate that no driver was loaded for the device. In the case of the failed USB driver, try rebooting the PC and following the steps in the EPT Drivers section of this User Manual.

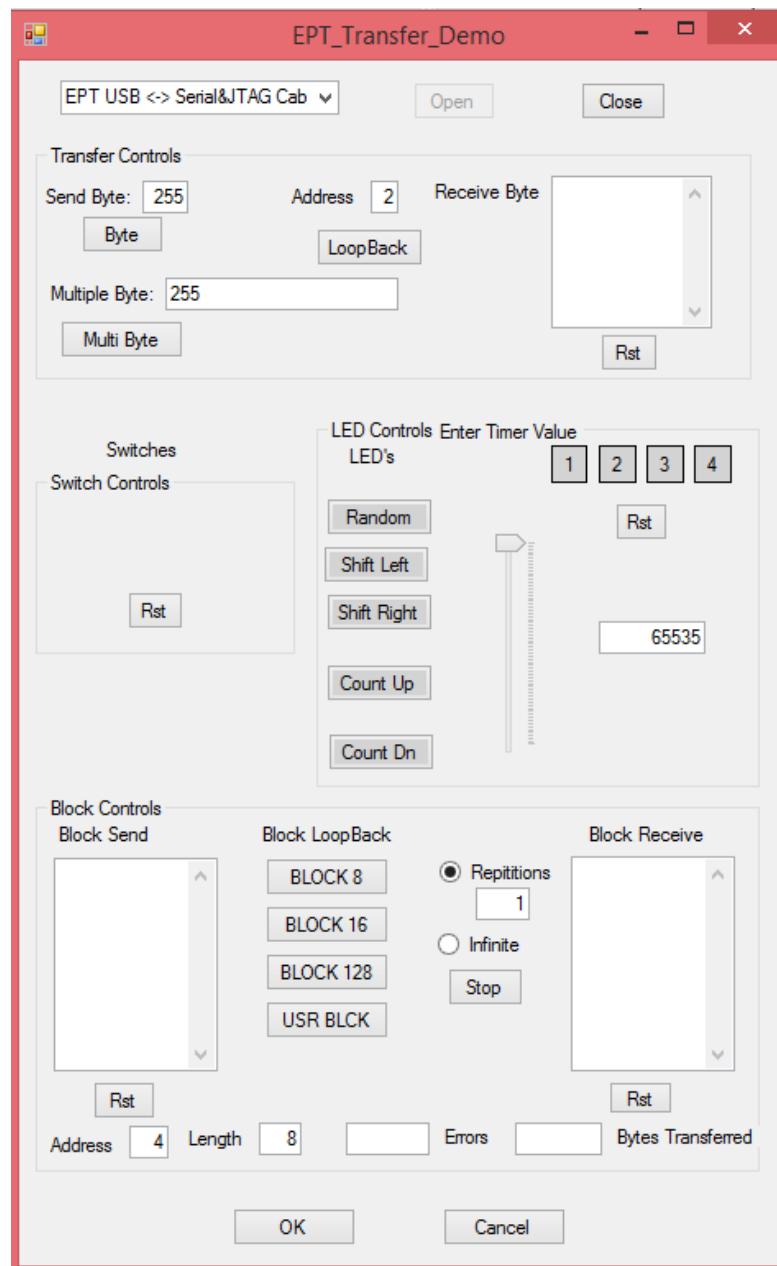


Next, open a Windows Explorer browser.

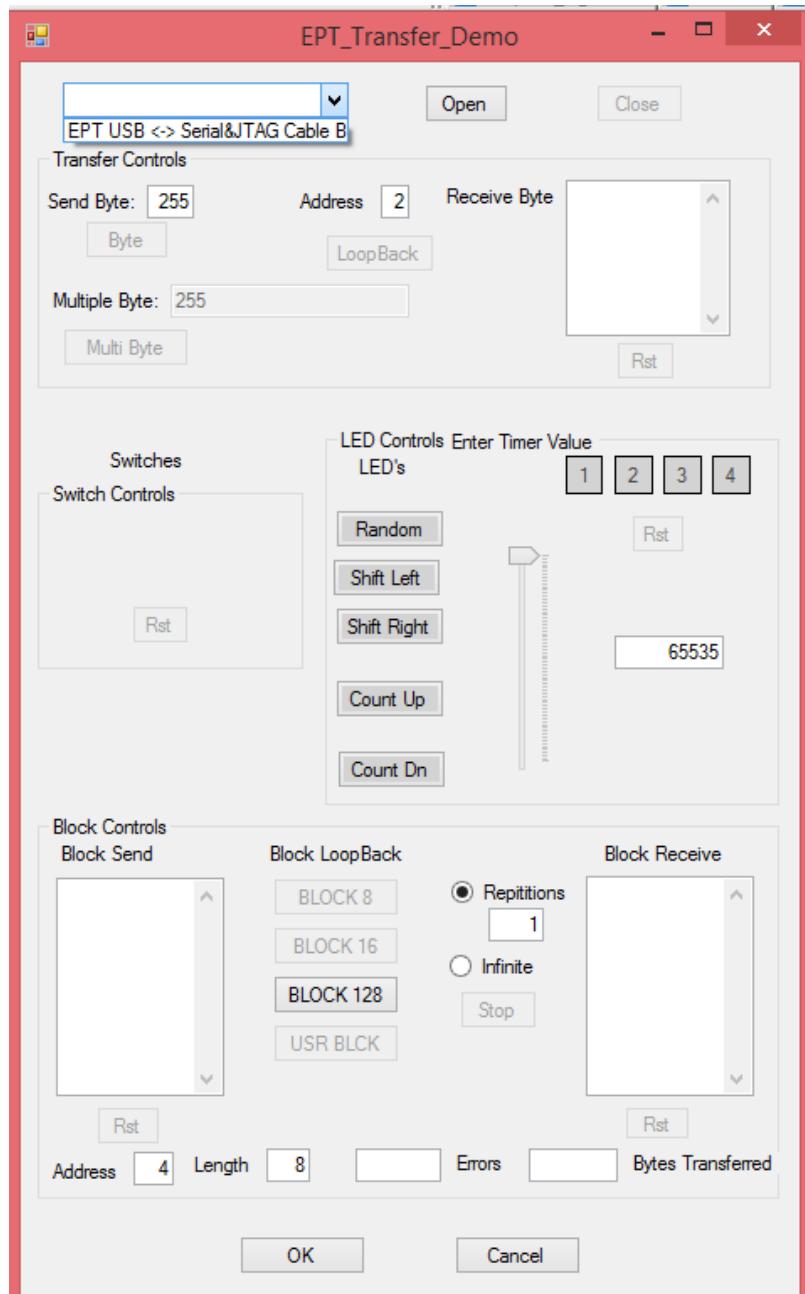
Browse to the  
Projects\_ActiveHost\EPT\_Transfer\_Demo\EPT\_Transfer\_Demo\bin\x64\Release\  
folder on the DPL\_USB\_FPGA\_PROJECT\_DVD.



Double click on the EPT\_Transfer\_Demo.exe. The application should load with a Windows form.



With the application loaded, select the FPGA board from the dropdown combo box and click on the “Open” button.



Leave the Address set at 2 for the Transfer Controls Group. And, leave the Address set at 4 for the Block Controls Group.

Click on one of the LED buttons in the middle of the window. The corresponding LED on the EPT-4CE6-AF-D1 board should light up.



## FPGA Development System User Manual

To exercise the Single Byte Transfer EndTerm, click the “LoopBack” button in the Transfer Controls group. Type in several numbers separated by a space and less 256 into the Multiple Byte textbox. Then hit the Multi Byte button. The numbers appear in the Receive Byte textbox.

To exercise the Block Transfer EndTerm, click the “BLOCK 8”, “BLOCK 16” or “BLOCK 128” button in the Block Controls group. A pre-selected group of numbers appear in the Block Receive textbox.

Press the PCB switches on the EPT-4CE6-AF-D1 to view the Switch Controls in action.



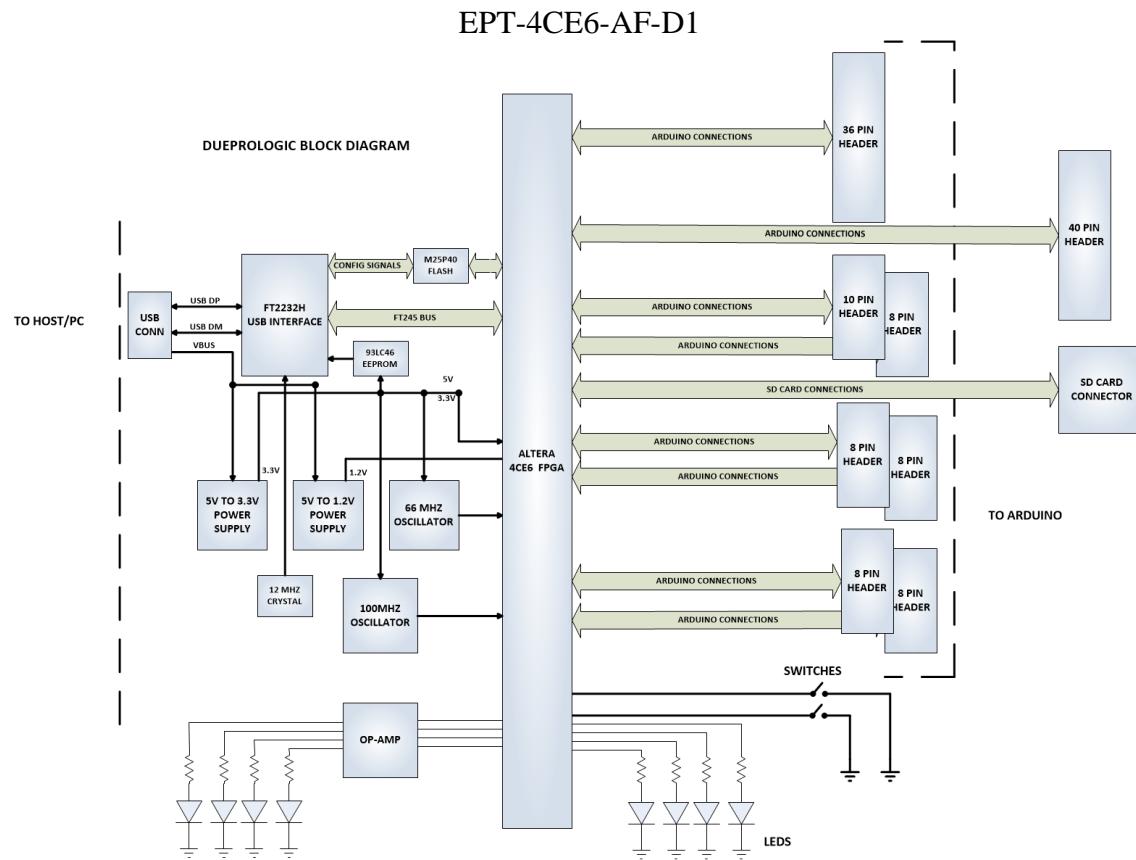
### 1.2 EPT-4CE6-AF-D1

The EPT-5M57-AP-U2 board is equipped with an Altera EP4CE6E22C8 FPGA; which is programmed using the Altera Quartus II software. The FPGA has 6672 Logic Elements and 276480 Total RAM Bits. An on board 66 MHz oscillator is used by the EPT Active Transfer Library to provide data transfer rates of up to 8 Mega Bytes per second. Fifty Four I/O's from the FPGA are attached to eight separate user connectors. I/O's. The user connectors are organized to fit the Due Arduino platform. There is a separate 40 pin dual row connector at the rear of the board arranged to mate with



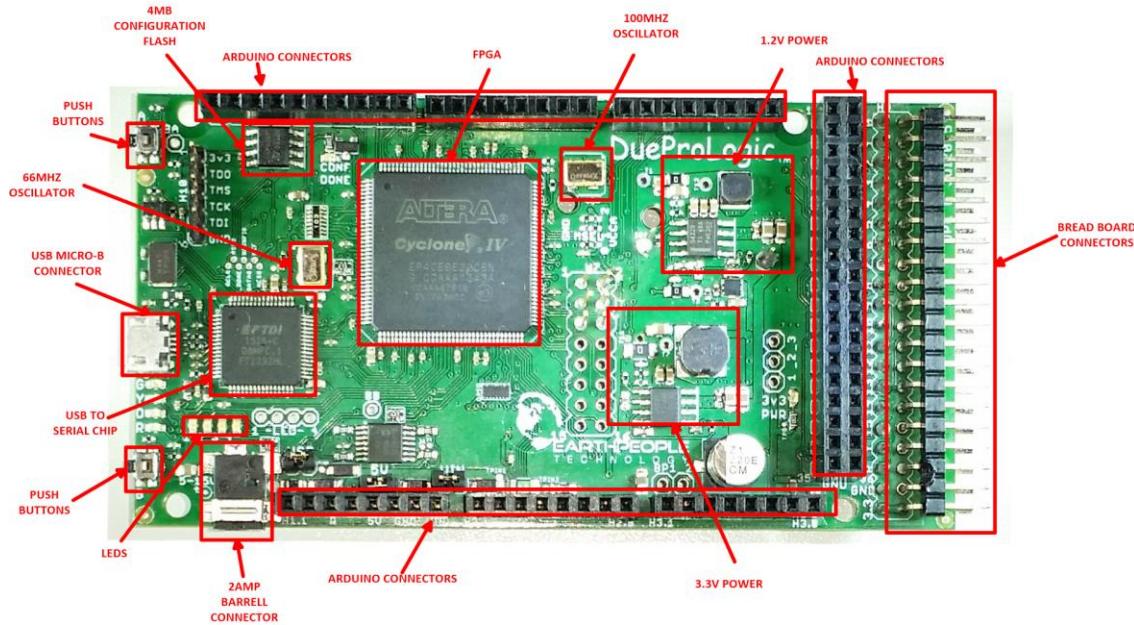
standard Bread boards. There are four green User LED's, four multicolored System LED's and two Push Buttons that are controllable by the user code. The hardware features are as follows.

- Altera EP4CE6 FPGA with 6272 Logic Cells
- Dual Channel High Speed USB FT2232H
- 66 MHz oscillator for driving USB data transfers and users code
- 100MHz oscillator for scaling up/down for users needs
- Standard SD Card interface for memory expansion
- 54 user Input/Outputs (+3.3V only)
- Four Green User LED's accessible by the user
- Four Multi-colored System LED's accessible by the user
- Two PCB switches accessible by the user
- I/O connectors stack into the Arduino Due





### EPT-EPT-4CE6-AF-D1



#### 1.2.1 Inputs and Outputs

There are 54 Inputs/Outputs which are +3.3Volt only. Do not connect a 5Volt device to the DPL. The FPGA I/O's are organized as separate pins and connect to the Arduino connectors. Several of the same I/O's are repeated on the read edge bread board connector. Each I/O must be defined as input or output in the user code. Each pin of the Arduino and bread board connectors can behave as either input or output.

#### 1.2.2 FPGA Configuration

**Note: DueProLogic supports Quartus II 13.1 Only. Please use the Quartus version included on the EPT\_DUE\_FPGA\_DEV\_SYS\_DVD.**

The EP4CE6 FPGA is configured for operation when the power is applied to the board. A dedicated Configuration Flash chip is included on the EPT-4CE6-AF-D1 for the purpose of configuring the FPGA as power up. The DPL uses the second channel of the FT2232H chip as a dedicated Flash programming port..The Configuration Flash can be programmed directly from Quartus II by using the EPT-Blaster driver. Follow the instructions in the “EPT Drivers” section of this manual.

#### 1.2.3 FT2232H Dual Channel USB to Serial Chip

The DueProLogic contains an FTDI 2232H dual channel high speed (480 Mb/s) USB to FIFO (first in-first out) integrated circuit to interface between the Host PC and the

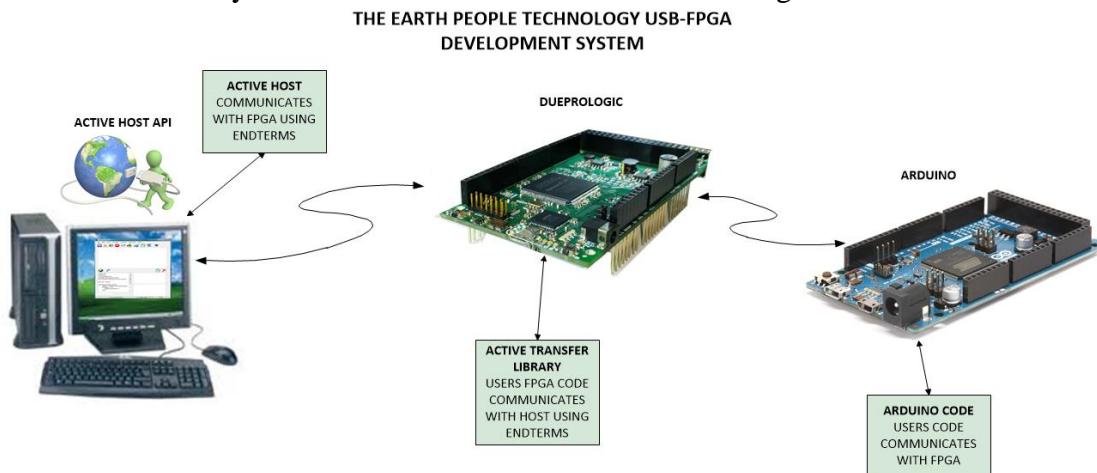
FPGA. The FT2232H provides a means of data conversion from USB to serial/ parallel data and serial/parallel to USB for data being sent from the FPGA to the PC. Channel A is configured as a Flash Configuration bus and Channel B is configured as an 8 bit parallel bus. FPGA Programming commands are transmitted via the channel A interface. Channel B has one dual port 4Kbyte FIFO for transmission from Host PC to the FPGA, it also has one dual port 4Kbyte FIFO for receiving data from the FPGA to the Host PC. The module uses the +5Vbus from the Host USB for self power.

### 1.2.4 SD Card Interface

The DPL includes a standard SD Card Interface. The SD connector is on the bottom of the DPL. This interface allows the user to add expansion memory or the standard SD interface.

## 1.3 FPGA Active Host Development

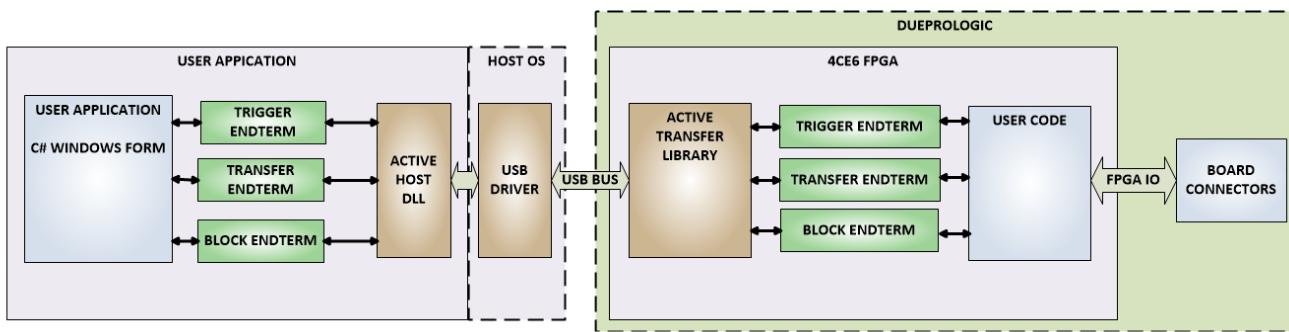
The DueProLogic comes complete with step by step instructions on building an entire communications system from FPGA to Windows Host. Using the EPT Active Host dll



provides an easy to use programming interface. The tools required are the free Visual Studio IDE and the free Quartus II.

### 1.4 Active Host EndTerms

The Active Host SDK is provided as a dll which easily interfaces to application software written in C#, C++ or C. It runs on the PC and provides transparent connection



from PC application code through the USB driver to the user FPGA code. The user code connects to “Endterms” in the Active Host dll. These Host “Endterms” have complementary HDL “Endterms” in the Active Transfer Library. Users have seamless bi-directional communications at their disposal in the form of:

- Trigger Endterm
- Transfer Endterm
- Block Endterm

User code writes to the Endterms as function calls. Just include the address of the individual module (there are eight individually addressable modules of each Endterm). Immediately after writing to the selected Endterm, the value is received at the HDL Endterm in the FPGA.



The screenshot shows a Microsoft Visual Studio interface with the title bar "EPT\_Transfer\_Demo". The tabs at the top include "EPT\_Transfer\_Demo", "BlockTransferPayload.cs", "Form1.Designer.cs", "Form1.cs", and "Form1.cs [Design]". The "Form1.cs" tab is active. The code editor displays the following C# code:

```
509     }
510
511     private void btnCountUp_Click(object sender, EventArgs e)
512     {
513         LoadTimerValue();
514
515         //Send the Initial value to the Active Transfer Module 2
516         //for loading into the Shift Left LED
517         EPT_AH_SendByte(0x01, (char)LEDStatus);
518
519         //Send Trigger 1 to latch the Shift-Count Value
520         EPT_AH_SendTrigger((byte)0x02);
521
522         //Send the start Shift Right to the Control Register
523         EPT_AH_SendTransferControlByte((char)2, (char)0x40);
524
525         //Set the Button State so User can know which function is selected
526         SetLEDControlButtons(SELECT_COUNT_UP);
527     }
528
529     private void btnCountDown_Click(object sender, EventArgs e)
530     {
```

The above is a code sample from a C# Windows Form. You can see functions that write a byte to the FPGA (EPT\_AH\_SendByte(0x01, (char)LEDStatus)) and write a trigger bit to the FPGA (EPT\_AH\_SendTrigger((byte)0x02)).



```
953      */
954
955      active_trigger          ACTIVE_TRIGGER_INST
956      (
957          .uc_clk                (CLK_66),
958          .uc_reset              (RST),
959          .uc_in                 (UC_IN),
960          .uc_out                (uc_out_m[ 0*22 +: 22 ]),
961
962          .trigger_to_host        (trigger_out),
963          .trigger_to_device      (trigger_in_byte)
964      );
965
966
967      active_transfer          ACTIVE_TRANSFER_INST_1
968      (
969          .uc_clk                (CLK_66),
970          .uc_reset              (RST),
971          .uc_in                 (UC_IN),
972          .uc_out                (uc_out_m[ 1*22 +: 22 ]),
973
974          .start_transfer         (led_start_transfer),
975          .transfer_received      (led_transfer_in_received),
976
977          .transfer_busy          (),
978
979          .uc_addr                (3'h1),
980
981          .transfer_to_host        (led_host_transfer_byte),
982          .transfer_to_device      (led_device_transfer_byte)
983      );
984
```

The above code is the interface Verilog which resides in the FPGA. When the C# Windows form sends a byte or trigger, the signals in the FPGA code react and allow the user code to receive the byte and trigger and perform some function with the information. In the case of this example, the LEDs will change state.

Receiving data from the FPGA is made simple by Active Host. Active Host transfers data from the FPGA as soon as it is available. It stores the transferred data into circular buffer. When the transfer is complete, Active Host invokes a callback function which is registered in the users application. This callback function provides a mechanism to transparently receive data from the FPGA. The user application does not need to schedule a read from the USB or call any blocking threads.

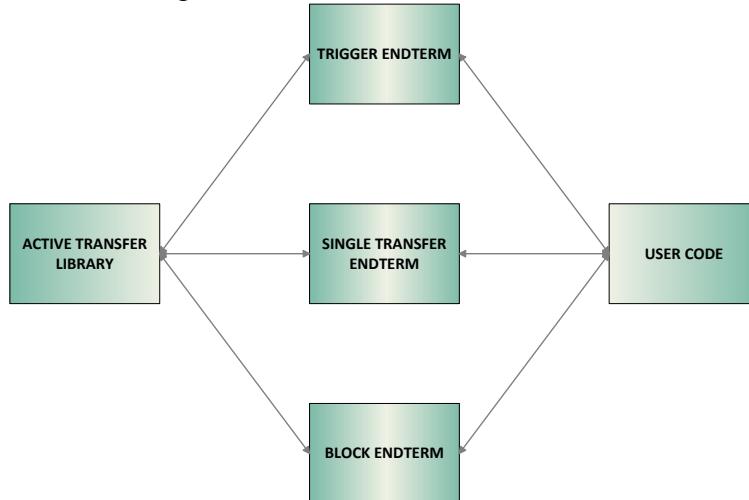
## 1.5 Active Transfer EndTerms

The Active Transfer Library is a portfolio of HDL modules that provides an easy to use yet powerful USB transfer mechanism. The user HDL code communicates with EndTerms in the form of modules. These EndTerm modules are commensurate with the

Active Host EndTerms. There are three types of EndTerms in the Active Transfer Library:

- Trigger Endterm
- Transfer Endterm
- Block Endterm

They each have a simple interface that the user HDL code can use to send or receive data across the USB. Writing to an EndTerm will cause the data to immediately arrive



at the commensurate EndTerm in the Active Host/user application. The transfer through the USB is transparent. User HDL code doesn't need to set up Endpoints or respond to Host initiated data requests. The whole process is easy yet powerful.

## 2 EPT Drivers

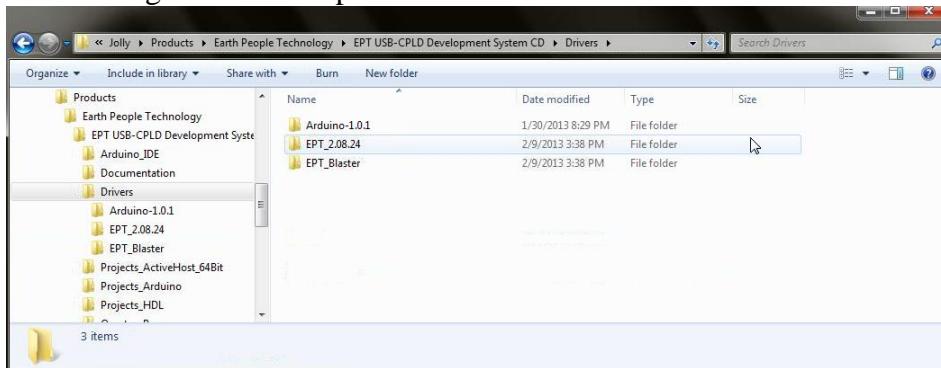
The EPT FPGA Development system requires drivers for any interaction between PC and the EPT-4CE6-AF-D1. The communication between the two consists of programming the FPGA and data transfer. In both cases, the USB Driver is required. This will allow Windows to recognize the USB Chip and setup a pathway for Windows to communicate with the USB hardware.

### 2.1 USB Driver

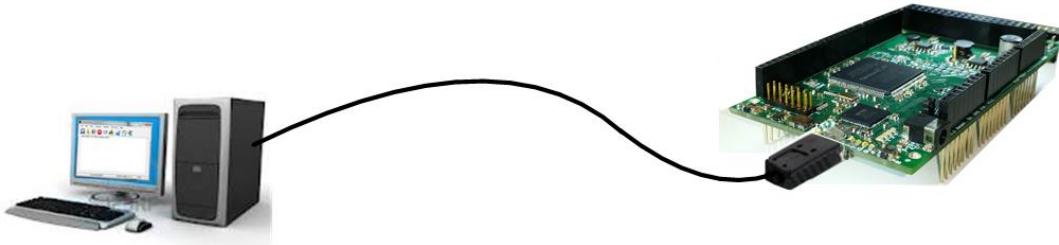
The EPT-4CE6-AF-D1 uses an FTDI FT2232H USB to Serial chip. This chip provides the USB interface to the PC and the serial/FIFO interface to the FPGA. The FT2232H requires the use of the EPT USB driver. To install the driver onto your PC, use the EPT\_2.08.24 Folder. The installation of the EPT\_2.08.24 driver is easily accomplished using the “Update Driver Software” utility in Device Manager.

## FPGA Development System User Manual

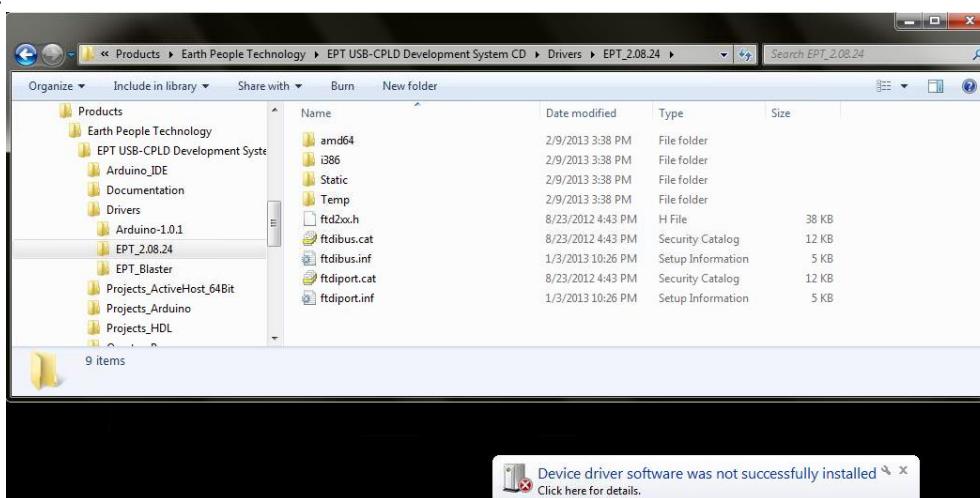
Locate the EPT\_2.08.24 folder in the Drivers folder of the EPT FPGA Development System DVD using Windows Explorer.



Plug in the EPT-4CE6-AF-D1 device into an available USB port.



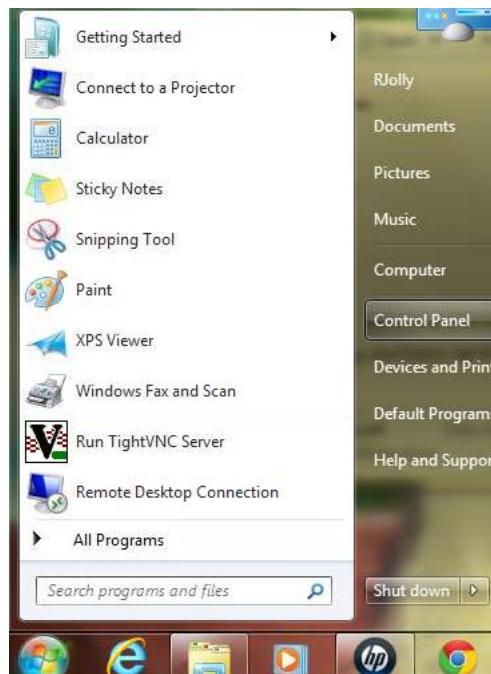
Windows will attempt to locate a driver for the USB device. When it does not find one, it will report an error, “Device driver software was not successfully installed”. Ignore this error.



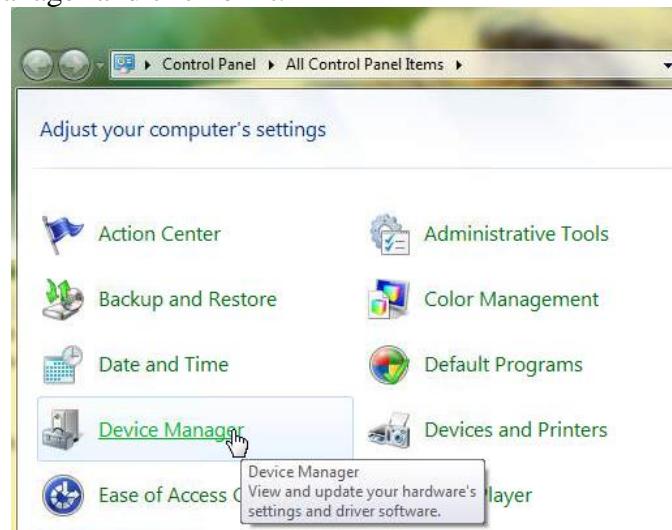
Go to Start->Control Panel



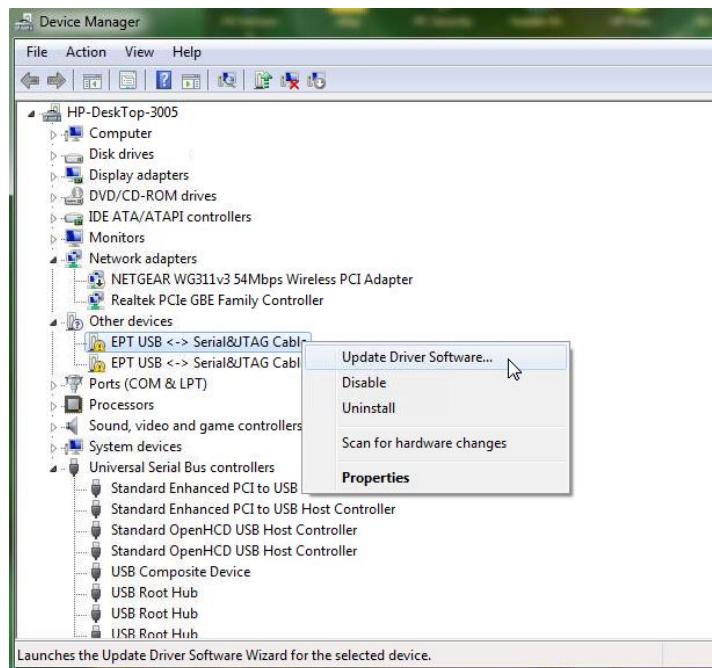
## FPGA Development System User Manual



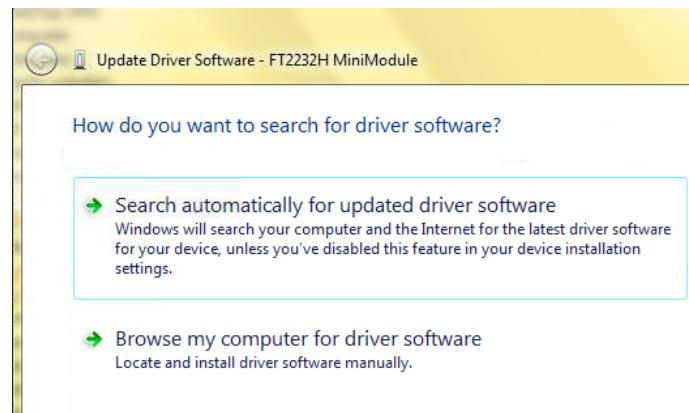
Locate Device Manager and click on it.



Locate the entry under “Other devices”. Right click “EPT USB <->Serial&JTAG Cable” and select “Update Driver Software...”.



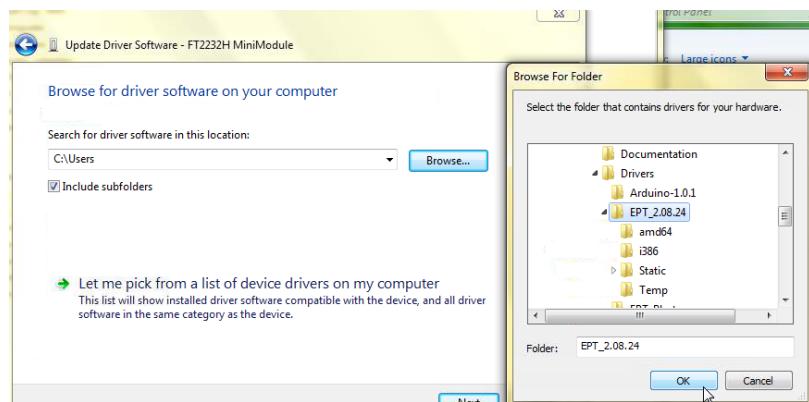
At the Update Driver Software Window, select “Browse my computer for driver software”.



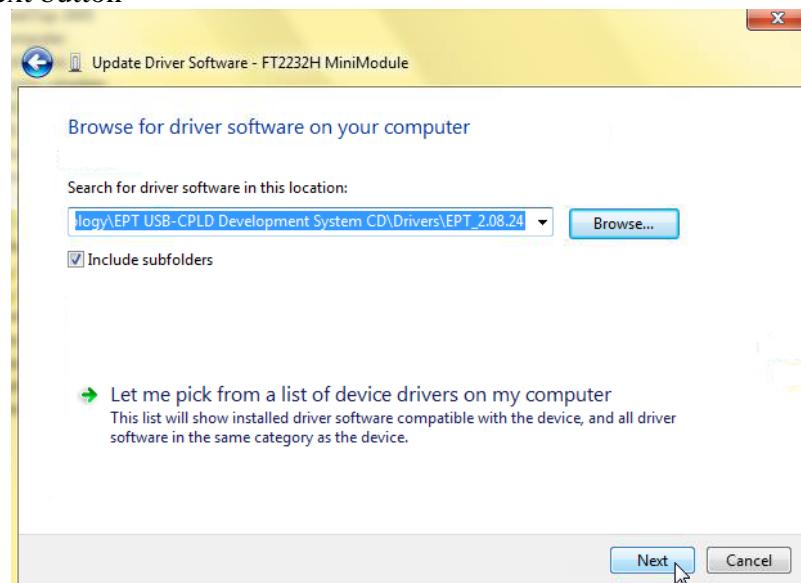
Click the Browse button and browse to the \Drivers\EPT\_2.08.24 folder of the EPT FPGA Development System DVD. Click the Ok button.



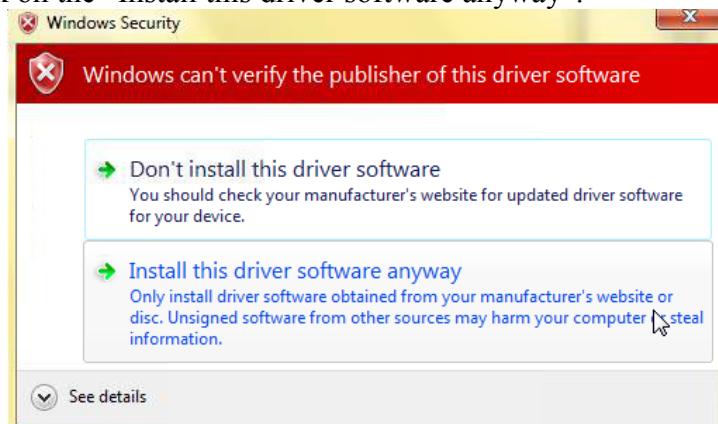
## FPGA Development System User Manual



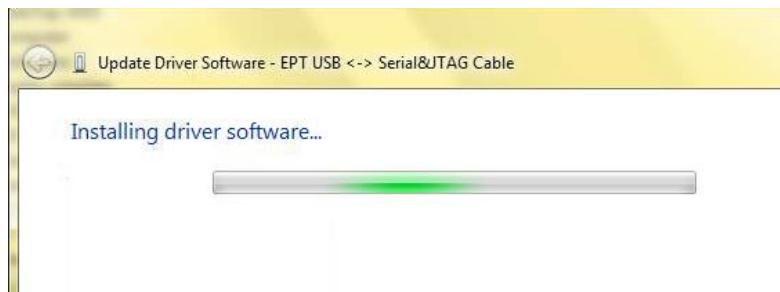
Click the Next button



The next window is the Windows Security notice. The EPT driver is not signed by Windows. Click on the "Install this driver software anyway".



Windows will add the EPT\_2.08.24 driver to the System Registry.

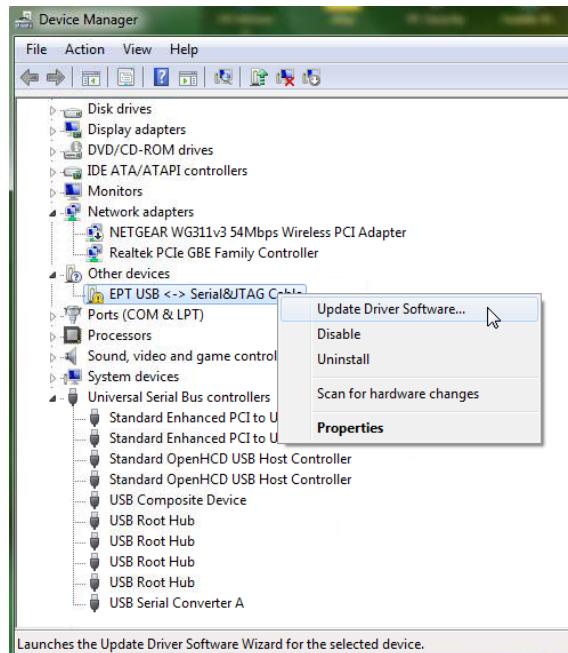


When Windows has completed the update driver the following screen will be displayed.



Channel A of the EPT-4CE6-AF-D1 is ready for use.

Next, repeat the process for Channel B.



The driver files will automatically install in the System Registry.



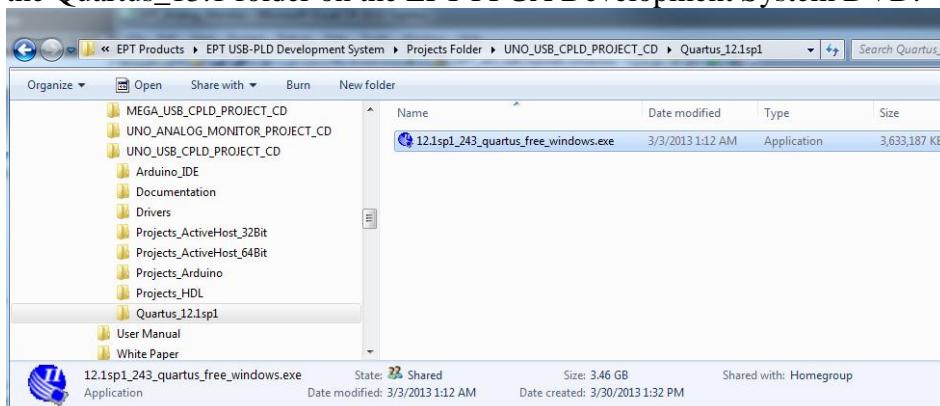
When this is complete, the drivers are installed and the EPT-4CE6-AF-D1 can be used with for programming and USB data transfers.

## 2.2 JTAG DLL Insert to Quartus II

The JTAG DLL Insert to Quartus II allows the Programmer Tool under Quartus to recognize the EPT-4CE6-AF-D1. The EPT-4CE6-AF-D1 can then be selected and perform programming of the FPGA. The file, `jtag_hw_mbftdi_blaster.dll` must be placed into the folder that hosts the `jtag_server` for Quartus. This dll is available for Windows XP 32-bit, Windows 7 32-bit and Windows 7 64-bit.

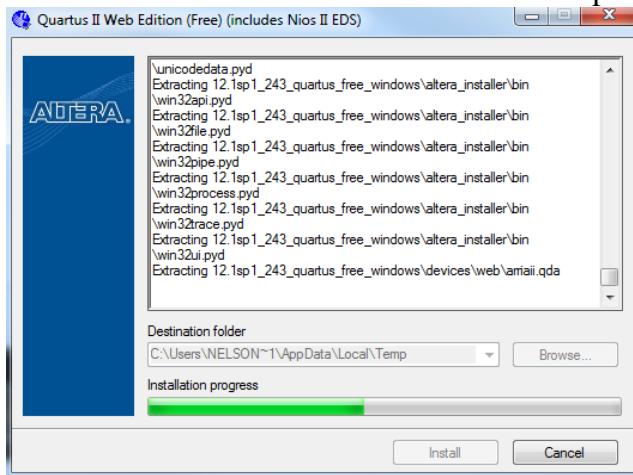
### 2.2.1 Installing Quartus

Locate the Quartus\_13.1 folder on the EPT FPGA Development System DVD.



If you don't have the EPT FPGA Development System DVD, you can download the Quartus II by following the directions in the Section Downloading Quartus.

If you don't need to download Quartus, double click on the 13.1\_xxx\_quartus\_free\_windows.exe (the xxx is the build number of the file, it is subject to change). The Quartus II Web Edition will start the installation process.



When the install shield window pops up click “Yes” or if needed, enter the administrator password for the users PC. Click “Ok”

Next, skip down to the Quartus Installer section to complete the Quartus installation.

### 2.2.2 Downloading Quartus

The first thing to do in order build a project in Quartus is to download and install the application. You can find the latest version of Quartus at:

<https://www.altera.com/download/dnl-index.jsp>

Click on the Download Windows Version.



## FPGA Development System User Manual

The screenshot shows the Altera Download Center. At the top, there's a navigation bar with links for Devices, Design Tools & Services, End Markets, Technology, Training, Support, About, and Buy. A search bar and a 'myAltera / Logout' link are also present. The main content area is titled 'Download Center' and features sections for QUARTUS® II, ModelSim, Nios® II, and DSP Builder. It lists various software packages for download, including Windows and Linux versions. Buttons for 'Install Subscription Edition Service Pack 1' and 'Install Web Edition Service Pack 1' are visible.

The next page will require you to sign into your “myAltera” account. If you do not have one, follow the directions under the box, “Don’t have an account?”

The screenshot shows the myAltera Account Sign In page. It has a navigation bar with links for Devices, Design Tools & Services, End Markets, Technology, Training, Support, About, and Buy. A search bar and a 'myAltera Account' link are at the top right. The main form has fields for 'User Name' and 'Password', with a 'Forgot Your User Name or Password?' link and a 'Remember me' checkbox. Below the form is a 'Don't have an account?' section with a radio button for 'Create Your myAltera Account'. It explains that a myAltera account allows users to file service requests, register for classes, and download software. There's a field for 'Enter your email address' with a note about retrieving information if it exists, and a 'Create Account' button. At the bottom is a 'Rate This Page' link.

Once you have created your myAltera account, enter the User Name and Password. The next window will ask you to allow pop ups so that the file download can proceed.



## FPGA Development System User Manual

The screenshot shows the Altera website's download center. A 'File Download - Security Warning' dialog box is open, asking if the user wants to run or save the file 'software\_name.exe'. The file is described as an application from altera.com. Below the dialog, there are 'Download Troubleshooting Steps' and a note about pop-up blockers.

Click on the “Allow Once” button. The next window will appear. It is the Download Manager.

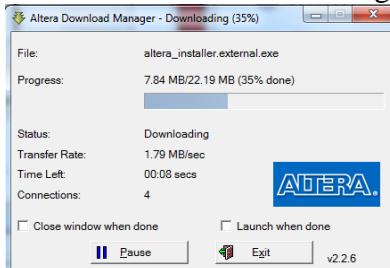
The screenshot shows an 'Internet Explorer Security' dialog box. It asks if the user wants to allow a program from 'Akamai Technologies Inc.' to run. The publisher is listed as 'Akamai Technologies Inc.'. There are 'Allow' and 'Don't allow' buttons at the bottom.

Click on the "Allow" button. This will bring up the Save As dialog box. Save the altera\_installer.external.exe to a download file.

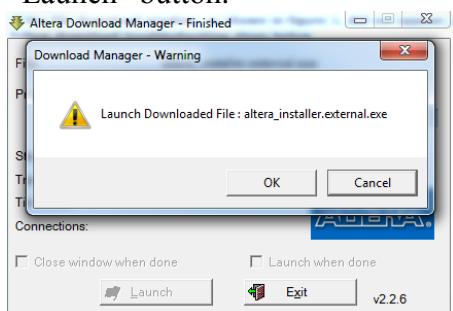
The screenshot shows a 'Save As' dialog box. The file name is set to 'altera\_installer.external.exe'. The 'Save as type:' dropdown is set to '.exe'. The 'Save' and 'Cancel' buttons are visible at the bottom.



Click the Save button. This will start the Download Manager.



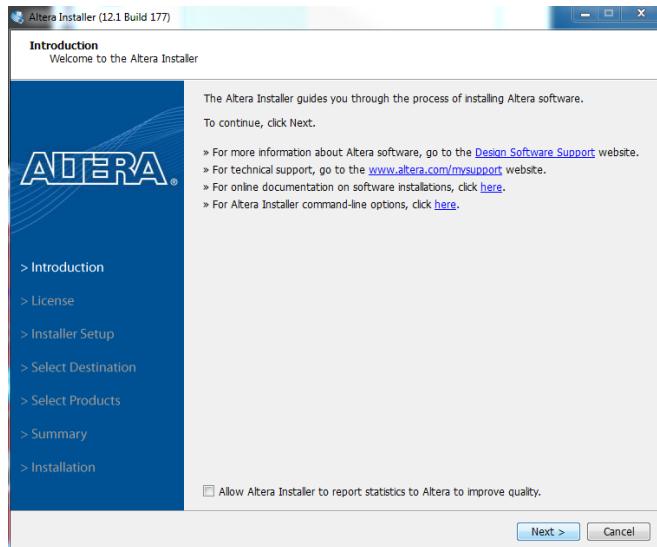
When it finishes, click the “Launch” button.



Click “Ok” and “Yes” to the following screen.

### 2.2.3 Quartus Installer

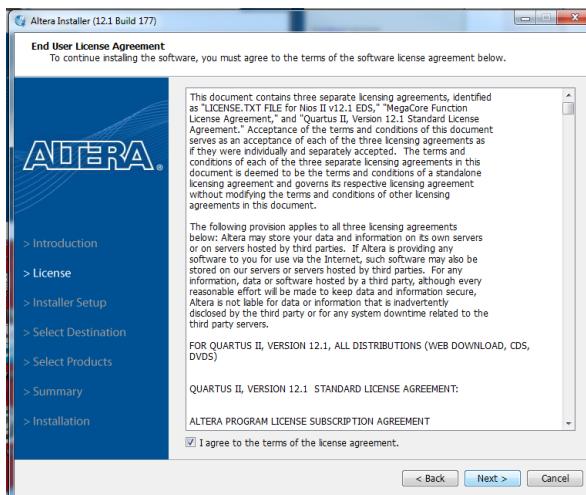
Click “Next” on the Introduction Window.



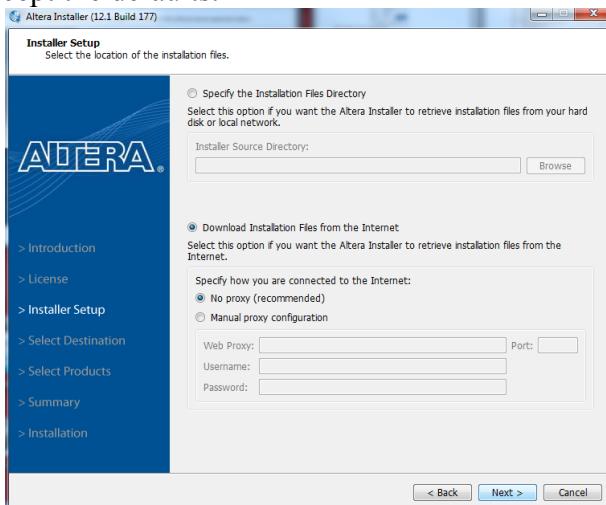
Click the checkbox to agree to the license terms. Then click “Next”.



## FPGA Development System User Manual



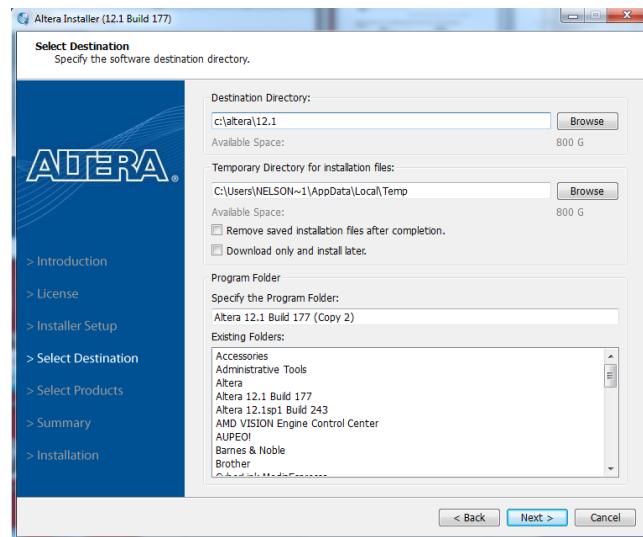
Click “Next” and accept the defaults.



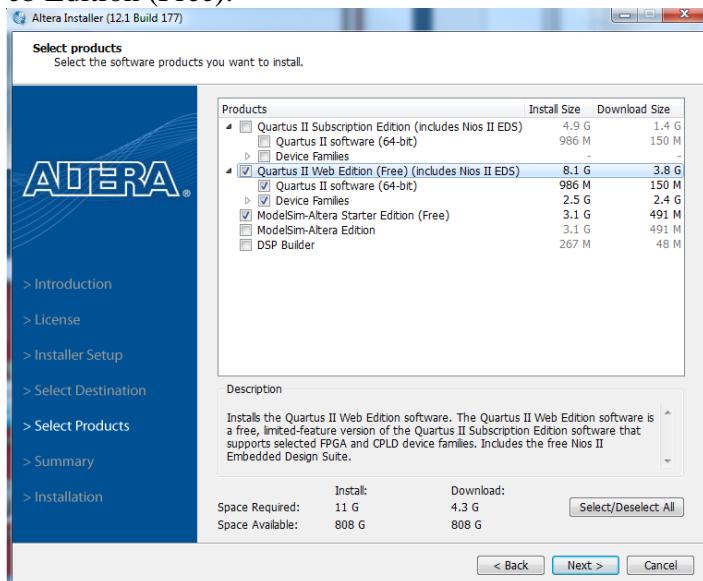
Click “Next” to accept the defaults



## FPGA Development System User Manual



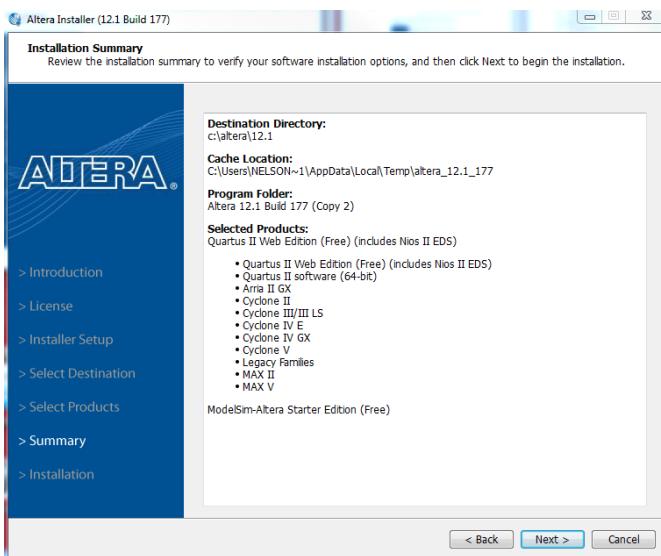
At the Select Products Window, de-select the Quartus II Subscription Edition by clicking on its check box so that the box is not checked. Then click on the check box by the Quartus II Web Edition (Free).



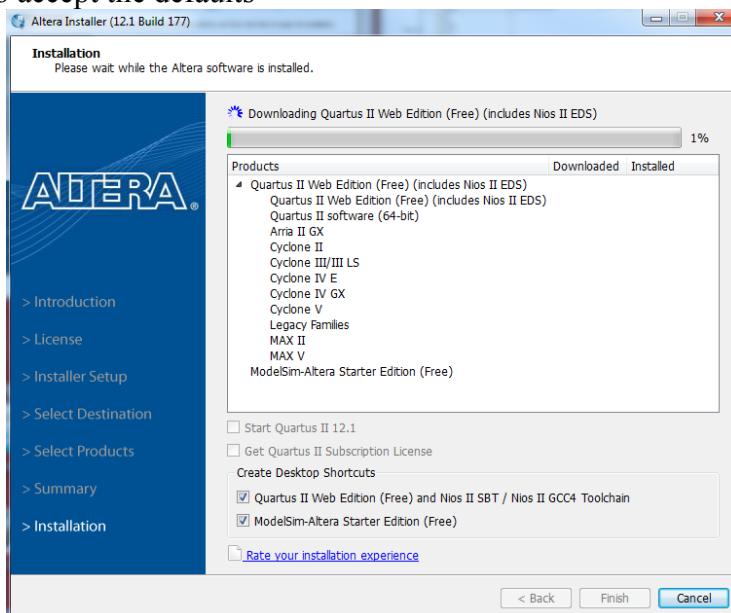
Click "Next" to accept the defaults



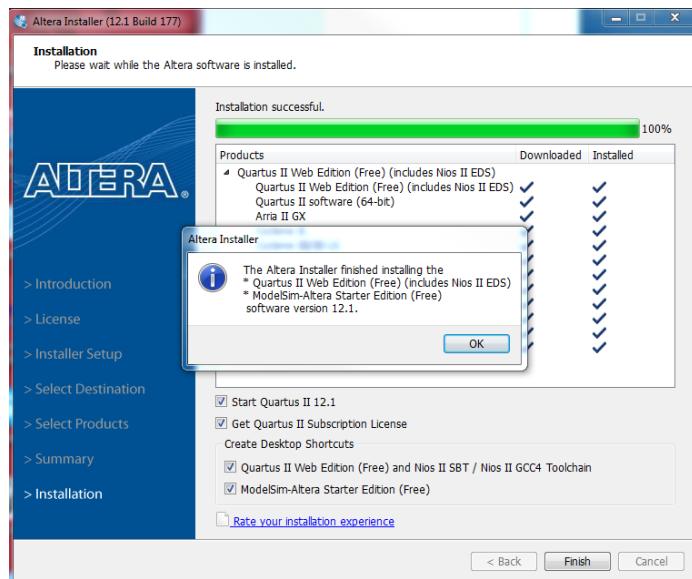
## FPGA Development System User Manual



Click “Next” to accept the defaults



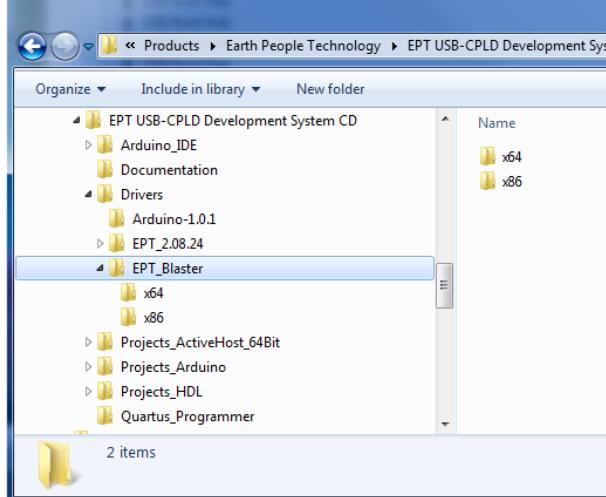
Wait for the download to complete. The file is 3.5 GB, so this could take a couple of hours depending on your internet connection. When installation is complete, the following window appears.



Click “Ok”, then click “Finish”. The Quartus II is now installed and ready to be used.

#### 2.2.4 Adding the EPT\_Blaster to Quartus II

Close out the Quartus II application. Locate the \Drivers\EPT\_Blaster folder on the EPT FPGA Development System DVD.

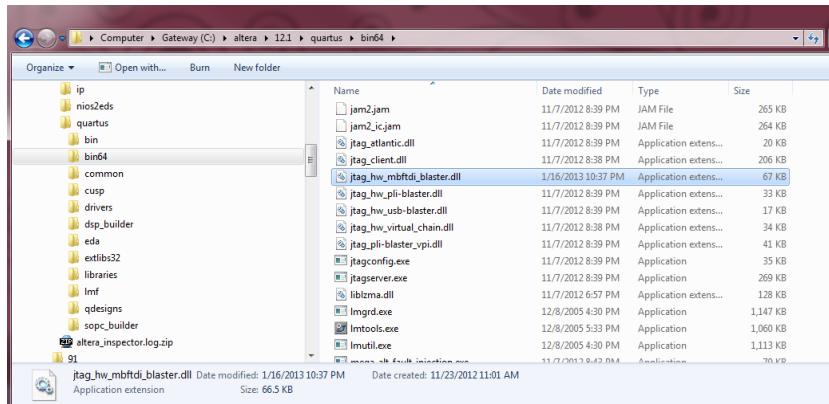


If your system is Windows 64 bit follow these directions:

Windows 64 bit

1. Open the C:\EPT FPGA Development System DVD\Drivers\EPT\_Blaster\x64 folder.
2. Select the file “jtag\_hw\_mbftdi\_blaster.dll” and copy it.
3. Browse over to C:\altera\12.1\quartus\bin64.

4. Right click in the folder and select Paste
5. Click Ok.
6. Open the Quartus II application.



The DLL is installed and the JTAG server should recognize it. Go to the section “Programming the FPGA” of this manual for testing of the programming. If the driver is not found in the Programmer Tool->Hardware Setup box, see the JTAG DLL Insert to Quartus II Troubleshooting Guide.

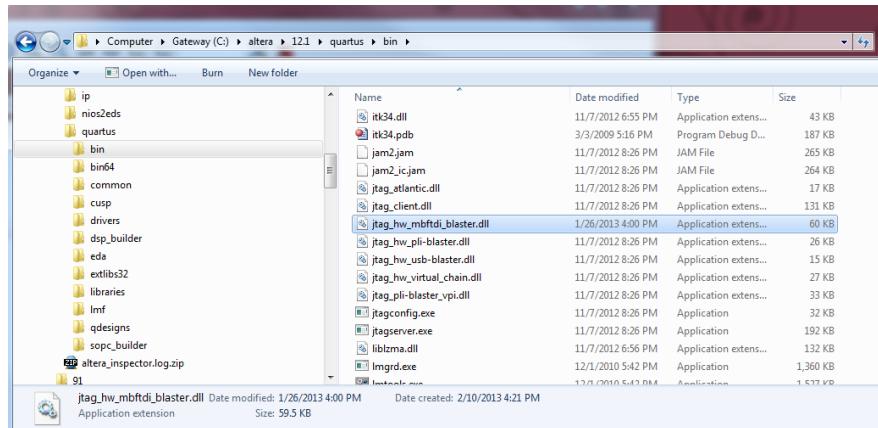
If your system is Windows 32 bit follow these directions:

#### Windows 32 bit

1. Open the C:\ EPT FPGA Development System DVD\Drivers\EPT\_Blaster\x86 folder.
2. Select the file “jtag\_hw\_mbftdi\_blaster.dll” and copy it
3. Browse over to C:\altera\13.1\quartus\bin.
4. Right click in the folder and select Paste
5. Click Ok.
6. Open the Quartus II application.



## FPGA Development System User Manual



The DLL is installed and the JTAG server should recognize it. Go to the section “Programming the FPGA” of this manual for testing of the programming. If the driver is not found in the Programmer Tool->Hardware Setup box, see the JTAG DLL Insert to Quartus II Troubleshooting Guide.

### 2.3 Active Host Application DLL

Download the latest version of Microsoft Visual C# 2010 Express environment from Microsoft. It's a free download.

<http://www.microsoft.com/visualstudio/eng/downloads#d-2010-express>

Go to the website and click on the “+” icon next to the Visual C# 2010 Express.

Visual Studio

Products    ALM    Download    Buy   

DOWNLOAD

2012 editions    2012 Express    2010 Express    Additional software  
Prerelease software    Readme

Visual Studio 2010 Express

+ Visual C++ 2010 Express

- Visual C# 2010 Express

Visual C# 2010 Express

Microsoft Captions Language Interface Pack (CLIP)

The Microsoft Captions Language Interface Pack (CLIP) uses tooltip captions to display translations for common user interface elements in the Visual Studio integrated development environment (IDE). Use CLIP as a language aid, to see translations in your own dialect, update resources in your own native tongue or as a learning tool.

Download language: English

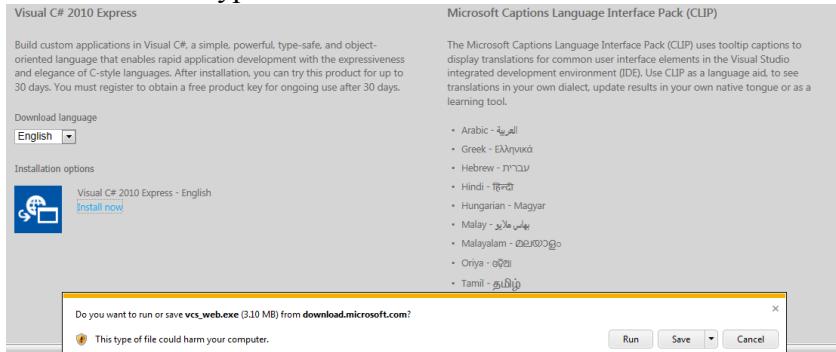
Installation options: Visual C# 2010 Express - English

Choose Language: Arabic - العربية, Greek - Ελληνικά, Hebrew - עברית, Hindi - हिन्दी, Hungarian - Magyar, Malay - ملايو, Malayalam - മലയാളം, Oriya - ଓଡ଼ିଆ, Tamil - தமிழ், Thai - ທາລີ



## FPGA Development System User Manual

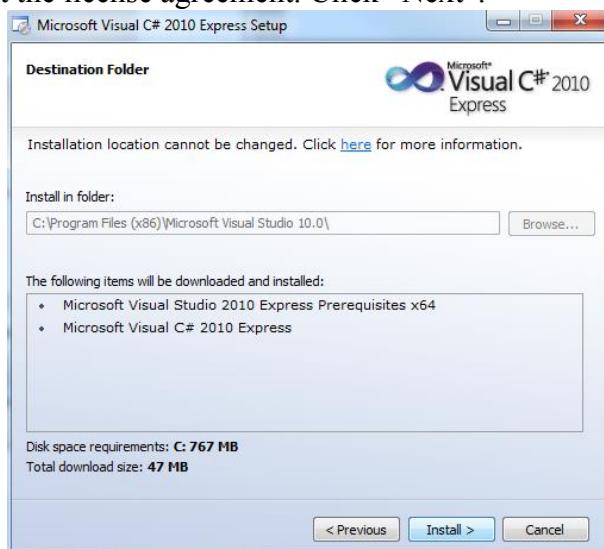
Click on the “Install now” hypertext.



Click the “Run” button.



Click “Next”, accept the license agreement. Click “Next”.





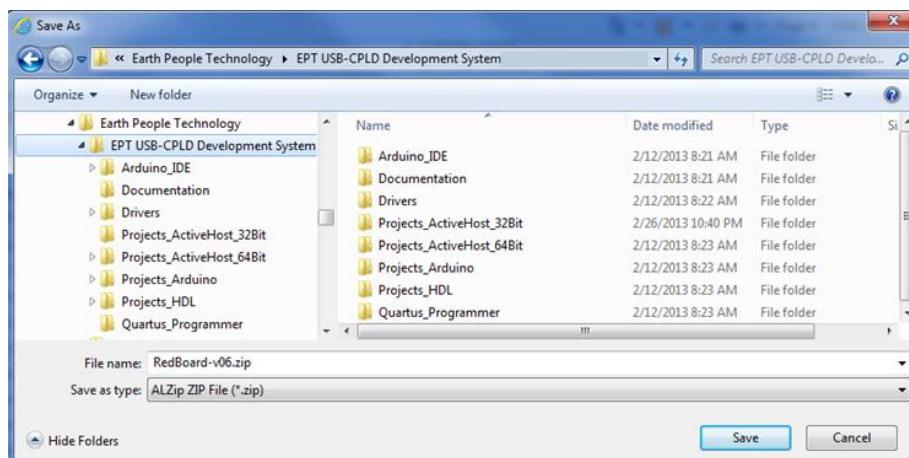
## FPGA Development System User Manual

Visual C# 2010 Express will install. This may take up to twenty minutes depending on your internet connection.



The installed successfully window will be displayed when Visual C# Express is ready to use.

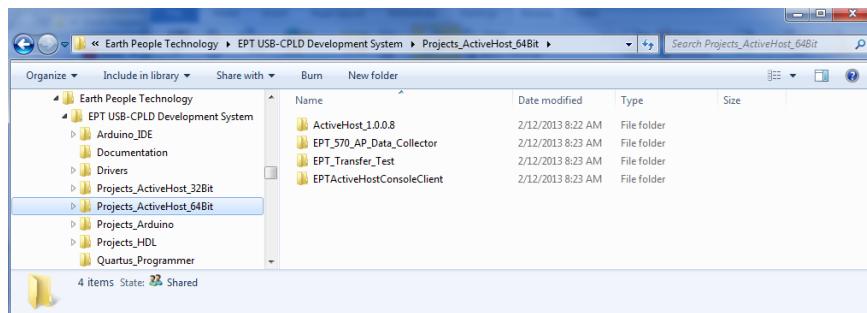
To use the Active Host Application Software, the Active Host DLL and the ftd2xx DLL must be included in the Microsoft Visual project. The Active Host Application Software will allow the user to create a custom applications on the PC using the EndTerms to perform Triggers and Data Transfer to/from the EPT-4CE6-AF-D1. The methods and parameters of the Active Host DLL are explained in the Active Host Application section. Locate the \Projects\_ActiveHost\_64Bit and \Projects\_ActiveHost\_32Bit folders on the EPT FPGA Development System DVD.



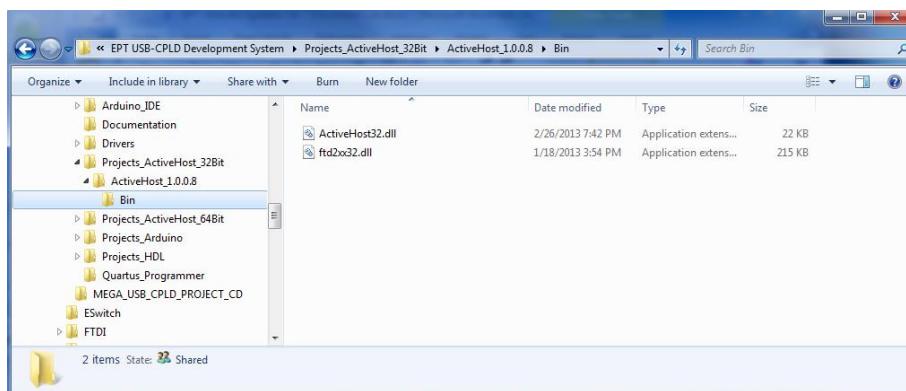
Locate the Projects\_ActiveHost\_64Bit and \Projects\_ActiveHost\_32Bit folders in the EPT FPGA Development System using Windows Explorer.



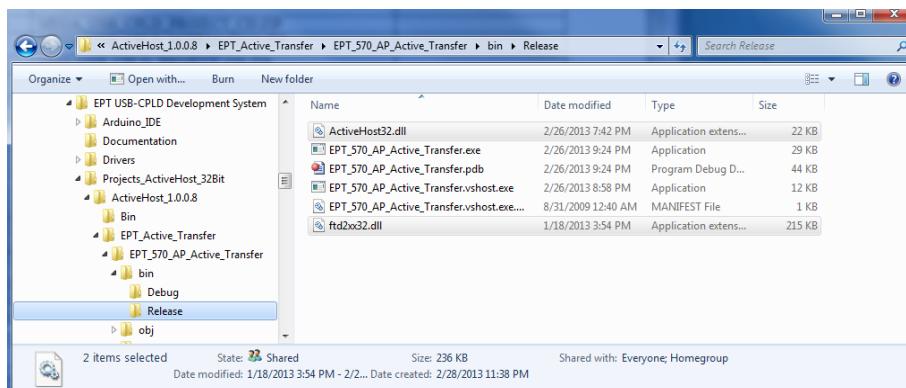
## FPGA Development System User Manual



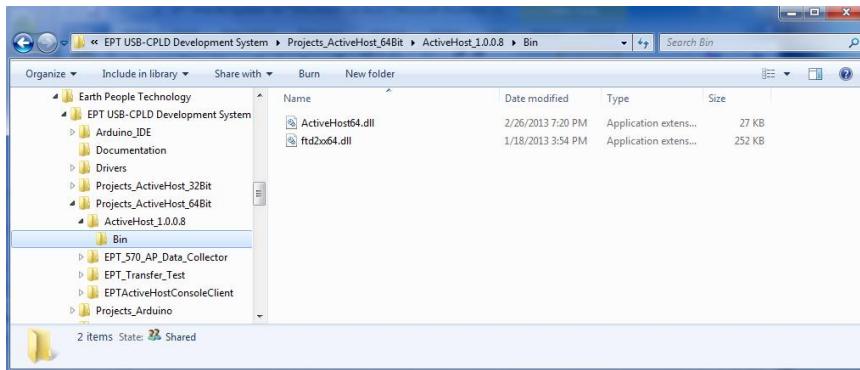
Locate the Projects\_ActiveHost\_32Bit \ActiveHost\_1.0.0.11\Bin folder and copy the ActiveHost32.dll and the ftd2xx32.dll.



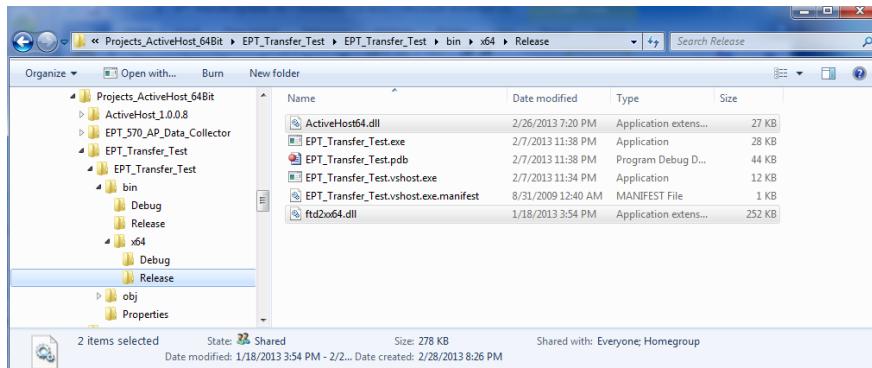
Save the DLL's in the \bin\Release folder for the 32 bit Windows 7 OS of the user project under the Microsoft C# Express project. See the Active Host Application for instructions on how to add the dll to the Microsoft C# Express project.



Locate the Projects\_ActiveHost\_64Bit\ActiveHost\_1.0.0.11\Bin folder and copy the ActiveHost64.dll and the ftd2xx64.dll.

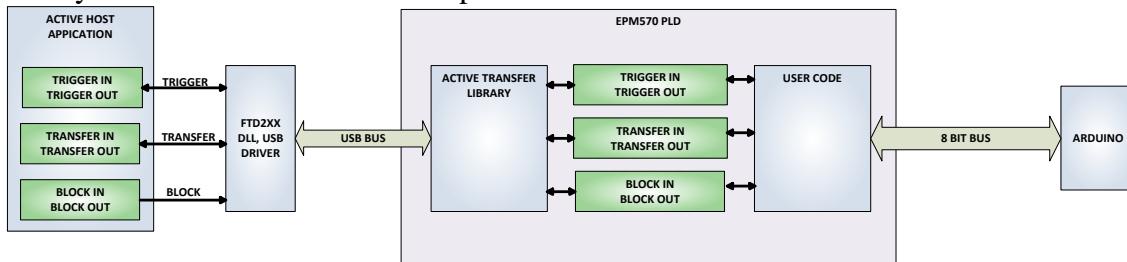


Save the DLL's in the bin\x64\Release folder of the user project under the Microsoft C# Express project. See the Active Host Application section of the EPT FPGA Development System User Manuals for instructions on how to add the dll to the Microsoft C# Express project.



### 3 FPGA Active Transfer Library

The Active Transfer Library is an HDL library designed to transfer data to and from the EPT-4CE6-AF-D1 via High Speed (480 MB/s) USB. It is a set of pre-compiled HDL files that the user will add to their project before building it. The description of what the library does and how to use its components are described in this manual.



### 3.1 EPT Active Transfer System Overview

The Active Transfer System components consist of the following:

- active\_transfer\_library.v
- ft\_245\_state\_machine.v
- endpoint\_registers.vqm
- active\_trigger.vqm
- active\_transfer.vqm
- active\_block.vqm

The Active\_Transfer\_Library provides the communication to the USB hardware. While separate Input and Output buses provide bi-directional communications with the plug in modules. See Figure 6 for an overview of the EPT Active\_Transfer system.

Figure 6 EPT Active Transfer Library Overview

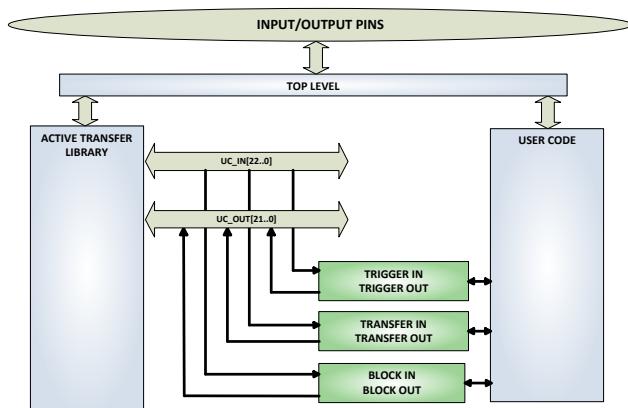
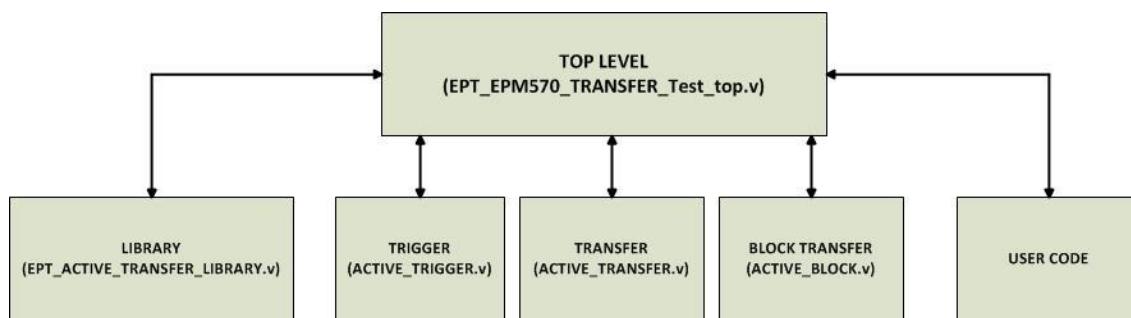


Figure 6 shows how the modules of the EPT Active Transfer Library attach to the overall user project. The EPT Active\_Transfer\_Library.vqm, Active\_Trigger.vqm, Active\_Transfer.vqm and Active\_Block.vqm modules are instantiated in the top level of the user project. The User\_Code.v module is also instantiated in the top level. The Active\_Transfer modules communicate with the User\_Code through module parameters. Each module is a bi-directional component that facilitates data transfer from PC to FPGA. The user code can send a transfer to the Host, and the Host can send a transfer to the user code. This provides significant control for both data transfers and signaling from the user code to PC. The Triggers are used to send momentary signals that can turn on (or off) functions in user code or PC. The Active Transfer is used to send a single byte. And the Active Block is used to send a block of data. The Active\_Transfer and Active\_Block modules have addressing built into them. This means the user can declare up to 8 individual instantiations of Active\_Transfer or Active\_Block, and send/receive data to each module separately.

### 3.2 Active Transfer Library

The Active Transfer Library contains the command, control, and data transfer mechanism that allows users to quickly build powerful communication schemes in the FPGA. Coupled with the Active Host application on the PC, this tools allows users to focus on creating programmable logic applications and not have to become distracted by USB Host drivers and timing issues. The Active Transfer Library is pre-compiled file that the user will include in the project files.





```
*****  
## Copyright Earth People Technology Inc. 2015  
##  
## File Name: EPT_4CE6_AF_D1_Top.v  
## Author: Earth People Technology  
## Date: October 6, 2015  
## Revision: A  
##  
## Development: EPT Data Collection Project  
## Application: Altera Cyclone IV FPGA  
## Description: This file contains verilog code which will allow access  
## to Active Transfer Library.  
##  
##  
##*****  
##  
## Revision History:  
## DATE VERSION DETAILS  
## 10/6/15 1 Created  
##  
##*****
```

```

//*****
//* Module Declaration
//*****

module EPT_4CE6_AF_D1_Top (
  input wire [1:0]          aa,
  input wire [1:0]          bc_in,
  output wire [2:0]         bc_out,
  inout wire [7:0]          bd inout,
  output wire [7:0]          XIO_1,      //XIO -- D2-D9
  output wire [2:0]          XIO_2,      //XIO -- D10-D12
  input wire [4:0]           XIO_2_IN,   //XIO -- D14-D18
  output wire [7:0]           XIO_3,      //XIO -- D22-D29
  output wire [7:0]           XIO_4,      //XIO -- D30-D37
  output wire [7:0]           XIO_5,      //XIO -- D38-D45
  output wire [7:0]           XIO_6,      //XIO -- D46-D53
  input wire [5:0]            XIO_7,      //XIO -- D69,D70,D71,D74,D75,D76
)

```

The interface from the library to the user code is two uni directional buses, UC\_IN[22:0] and UC\_OUT[20:0]. The UC\_IN[22:0] bus is an output bus (from the library, input bus to the Active Modules) that is used channel data, address, length and control information to the Active Modules. The UC\_OUT[21:0] bus is an input bus (to the library, output bus from the Active Modules) that is used to communicate data, address, length, and control information to the Active Modules.

The control buses, aa[1:0], bc\_in[1:0], bc\_out[2:0], and bd inout[7:0] are used to channel data, and control signals to the USB interface chip. These signals are connected directly to input and output pins of the FPGA.

### 3.2.1 Active Trigger EndTerm

The Active Trigger has eight individual self resetting, active high, signals. These signals are used to send a momentary turn on/off command to Host/User code. The Active Trigger is not addressable so the module will be instantiated only once in the top level.



```
743     wire [22*3-1:0]  uc_out_m;
744     eptWireOR #( .N(3) ) wireOR (UC_OUT, uc_out_m);
745     active_trigger           ACTIVE_TRIGGER_INST
746     (
747       .uc_clk                (CLK_66),
748       .uc_reset              (RST),
749       .uc_in                 (UC_IN),
750       .uc_out                (uc_out_m[ 0*22 +: 22 ]),
751
752       .trigger_to_host        (trigger_to_host),
753       .trigger_to_device      (trigger_in_byte)
754     );
755   
```

To send a trigger, decide which bit (or multiple bits) of the eight bits you want to send the trigger on. Then, set that bit (or bits) high. The Active Transfer Library will send a high on that trigger bit for one clock cycle (66 MHz), then reset itself to zero. The bit can stay high on the user code and does not need to be reset to zero. However, if the user sends another trigger using the trigger byte, then any bit that is set high will cause a trigger to occur on the Host side.



```
277      //-----
278      // Detect Trigger Out to Host
279      //-----
280      always @(TRIGGER_OUT or trigger_in_reset or reset)
281      begin
282          if(!reset)
283              trigger_to_host = 8'h0;
284          else if (trigger_in_reset)
285              trigger_to_host = 8'h0;
286          else if (TRIGGER_OUT > 8'h0)
287              trigger_to_host = TRIGGER_OUT;
288      end
289
290      //-----
291      // Reset Trigger Out to Host
292      //-----
293      always @(posedge CLK_66 or negedge reset)
294      begin
295          if(!reset)
296              begin
297                  trigger_in_reset <= 0;
298              end
299          else
300              begin
301                  if (trigger_to_host > 0)
302                      trigger_in_reset <= 1'b1;
303                  else
304                      trigger_in_reset <= 0;
305              end
306      end
```

So, care should be used if the user code uses byte masks to send triggers. It is best to set only the trigger bits needed for a given time when sending triggers.

The user code must be setup to receive triggers from the Host. This can be done by using an asynchronous always block. Whenever a change occurs on a particular trigger bit (or bits), a conditional branch can detect if the trigger bit is for that block of code. Then, execute some code based on that trigger.



```
308      //-----
309      // Detect Trigger In
310      //-----
311      always @(trigger_in_byte or trigger_in_reset or reset)
312      begin
313          if(!reset)
314          begin
315              trigger_in_detect = 1'b0;
316          end
317          else if (trigger_in_reset)
318          begin
319              trigger_in_detect = 1'b0;
320          end
321          else if (trigger_in_byte > 8'h0)
322          begin
323              trigger_in_detect = 1'b1;
324          end
325      end
326
327      //-----
328      // Store the value of Trigger In
329      //-----
330      always @ (posedge CLK_66 or negedge reset)
331      begin
332          if(!reset)
333          begin
334              trigger_in_store <= 8'h0f;
335              trigger_in_reg <= 1'b0;
336              trigger_in_reset <= 1'b0;
337          end
338          else if (trigger_in_detect & !trigger_in_reg)
339          begin
340              if(trigger_in_byte != 0)
341                  trigger_in_store[7:0] <= trigger_in_byte[7:0];
342                  trigger_in_reg <= 1'b1;
343          end
344          else if (trigger_in_reg)
345          begin
346              trigger_in_reg <= 1'b0;
347              trigger_in_reset <= 1'b1;
348          end
349          else if (!trigger_in_detect)
350          begin
351              trigger_in_reg <= 1'b0;
352              trigger_in_reset <= 1'b0;
353          end
354      end
```

### 3.2.2 Active Transfer EndTerm

The Active Transfer module is used to send or receive a byte to/from the Host. This is useful when the user's microcontroller needs to send a byte from a measurement to the Host for display or processing. The Active Transfer module is addressable, so up to eight individual modules can be instantiated and separately addressed.

```
757      active_transfer          ACTIVE_TRANSFER_INST
758      (
759          .uc_clk              (CLK_66),
760          .uc_reset            (reset),
761          .uc_in               (UC_IN),
762          .uc_out              (uc_out_m[ 1*22 +: 22 ]),
763
764          .start_transfer       (transfer_out_reg),
765          .transfer_received    (transfer_in_received),
766
767          .uc_addr              (3'h2),
768
769          .transfer_to_host     (transfer_out_byte),
770          .transfer_to_device   (transfer_in_byte)
771      );
772
```

To send a byte to the Host, select the appropriate address that corresponds to an address on Host side. Place the byte in the “transfer\_to\_host” parameter, then strobe the “start\_transfer” bit. Setting the “start\_transfer” bit to high will send one byte from the “transfer\_to\_host” byte to the Host on the next clock high signal (66 MHz). The “start\_transfer” bit can stay high for the duration of the operation of the device, the Active Transfer module will not send another byte. In order to send another byte, the user must cycle the “start\_transfer” bit to low for a minimum of one clock cycle (66 MHz). After the “start\_transfer” bit has been cycled low, the rising edge of the bit will cause the byte on the “transfer\_to\_host” parameter to transfer to the host.



```
181 //-----
182 // Transfer byte to Device
183 //-----
184 always @(TRANSFER_OUT_EN or reset)
185 begin
186   if(!reset)
187     begin
188       transfer_out_detect = 1'b0;
189     end
190   else
191     begin
192       if(transfer_to_device_reset)
193         transfer_out_detect = 1'b0;
194       else if(TFERNER_OUT_EN)
195         begin
196           transfer_out_byte = TRANSFER_OUT_BYTE;
197           transfer_out_detect = 1'b1;
198         end
199       end
200     end
201
202 //-----
203 // Reset transfer_to_device_reset
204 //-----
205 always @(posedge CLK_66 or negedge reset)
206 begin
207   if (!reset)
208     begin
209       transfer_to_device_reset <= 1'b0;
210     end
211   else
212     begin
213       if(transfer_out_detect)
214         transfer_to_device_reset <= 1'b1;
215       else
216         transfer_to_device_reset <= 1'b0;
217     end
218 end
```

To receive a byte, the Active Host will send a byte using it's dll. The user code must monitor the transfer\_received port. The transfer\_received port will assert high for one clock cycle (66 MHz) when a byte is ready for reading on the transfer\_to\_device port. User code should use an asynchronous always block to detect when the



transfer\_received port is asserted. Upon assertion, the user code should read the byte from the transfer\_to\_device port into a local register.

```
220 //-----  
221 // Transfer to Host  
222 //-----  
223 always @(posedge CLK_66 or negedge reset)  
224 begin  
225 if (!reset)  
226 begin  
227 transfer_out <= 1'b0;  
228 transfer_out_reg <= 1'b0;  
229 transfer_out_byte <= 8'h0;  
230 end  
231 else  
232 begin  
233 if(start_transfer_byte & !transfer_out)  
234 begin  
235 transfer_out_byte <= TRANSFER_HOST_BYTE;  
236 transfer_out_reg <= 1'b1;  
237 transfer_out <= 1'b1;  
238 end  
239 else if(start_transfer_byte & transfer_out)  
240 begin  
241 transfer_out_reg <= 1'b0;  
242 transfer_out <= 1'b1;  
243 end  
244 else if(!start_transfer_byte & transfer_out)  
245 begin  
246 transfer_out_reg <= 1'b0;  
247 transfer_out <= 1'b0;  
248 end  
249 end  
250 end
```

### 3.2.3 Active Block EndTerm

The Active Block module is designed to transfer blocks of data between Host and User Code and vice versa. This allows buffers of data to be transferred with a minimal amount of code. The Active Block module is addressable, so up to eight individual modules can be instantiated and separately addressed. The length of the block to be transferred must also be specified in the uc\_length port.



```
811      active_block          BLOCK_TRANSFER_INST
812      (
813          .uc_clk            (CLK_66) ,
814          .uc_reset          (RST) ,
815          .uc_in             (UC_IN) ,
816          .uc_out            (uc_out_m[ 2*22 +: 22 ]) ,
817
818          .start_transfer    (block_out_reg) ,
819          .transfer_received (block_in_rcv) ,
820
821          .transfer_ready    (block_byte_ready) ,
822
823          .uc_addr           (3'h4) ,
824          .uc_length          (BLOCK_COUNT_8) ,
825
826          .transfer_to_host   (block_out_byte) ,
827          .transfer_to_device (block_in_data) ,
828
829          .STATE_OUT          (block_state_out) ,
830          .TEST_BUS           (block_out_test_bus)
831
832      );
833
```

To send a block, it's best to have buffer filled in a previous transaction. Then assert the start\_transfer bit. This method is opposed to collecting and processing data bytes after the start\_transfer bit has been asserted and data is being sent to the Host.

Once the buffer to send is filled with the requisite amount of data, the address and buffer length should be written to the uc\_addr and uc\_length ports. Set the start\_transfer bit high, the user code should monitor the transfer\_ready port. At the rising edge of the transfer\_ready port, the byte at transfer\_to\_host port is transferred to the USB chip. Once this occurs, the user code should copy the next byte in the buffer to transfer\_to\_host port. On the next rising edge of transfer-ready, the byte at transfer\_to\_host will be transferred to the USB chip. This process continues until the number of bytes described by the uc\_length have been transferred into the USB chip.



```
542      //-
543      // Registers to start Block Transfer Out
544      //-
545      always @(posedge CLK_66 or negedge RST)
546      begin
547          if(!RST)
548          begin
549              block_out_reg <= 1'b0;
550              start_block_transfer_reg <= 1'b0;
551          end
552          else
553          begin
554              if(start_block_transfer & !start_block_transfer_reg)
555                  start_block_transfer_reg <= 1'b1;
556              else if(start_block_transfer_reg & !block_out_reg)
557              begin
558                  block_out_reg <= 1'b1;
559              end
560              else if(block_out_counter >= BLOCK_COUNT_8)
561              begin
562                  block_out_reg <= 1'b0;
563                  start_block_transfer_reg <= 1'b0;
564              end
565          end
566      end
567
568      //-
569      // Data for Block Transfer Out
570      //-
571      always @(posedge CLK_66 or negedge RST)
572      begin
573          if(!RST)
574          begin
575              block_out_counter <= 0;
576          end
577          else
578          begin
579              if(block_byte_ready)
580              begin
581                  block_out_counter <= block_out_counter + 1'd1;
582              end
583              else if(block_out_counter >= BLOCK_COUNT_8 )
584              begin
585                  block_out_counter <= 0;
586              end
587          end
588      end
```

To receive a buffer from the Host, the user code should monitor the transfer\_received port for assertion. When the bit is asserted, the next rising edge of transfer\_ready will indicate that the byte at transfer\_to\_device is ready for the user code to read.

[Add code snippet showing Active Block Module bytes received by the user code]

### 3.3 Timing Diagram for Active Transfer EndTerms

The Active Transfer Library uses the 66 MHz clock to organize the transfers to Host and transfer to Device. The timing of the transfers depends on this clock and the specifications of the USB chip. Users should use the timing diagrams to ensure proper operation of user code in data transfer.

#### 3.3.1 Active Trigger EndTerm Timing

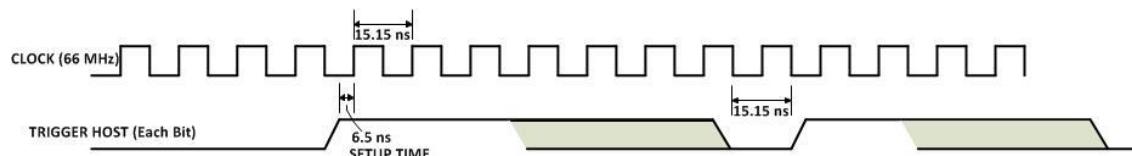


Figure xx Active Trigger to Host Timing

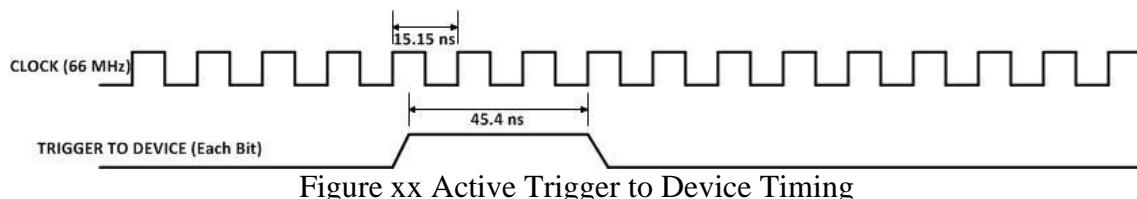


Figure xx Active Trigger to Device Timing

#### 3.3.2 Active Transfer EndTerm Timing

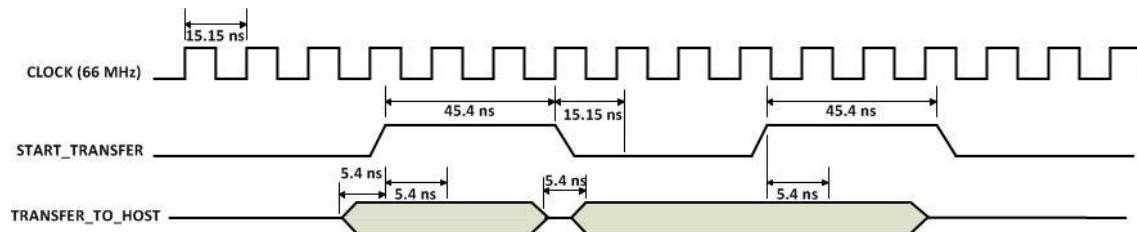


Figure xx Active Transfer To Host Timing

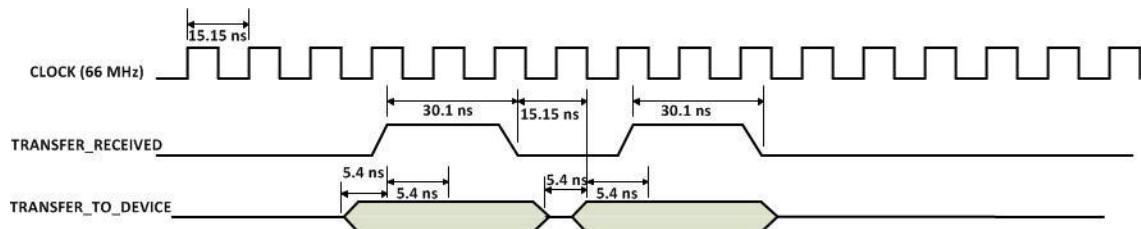


Figure xx Active Transfer To Device Timing

### 3.3.3 Active Block EndTerm Timing

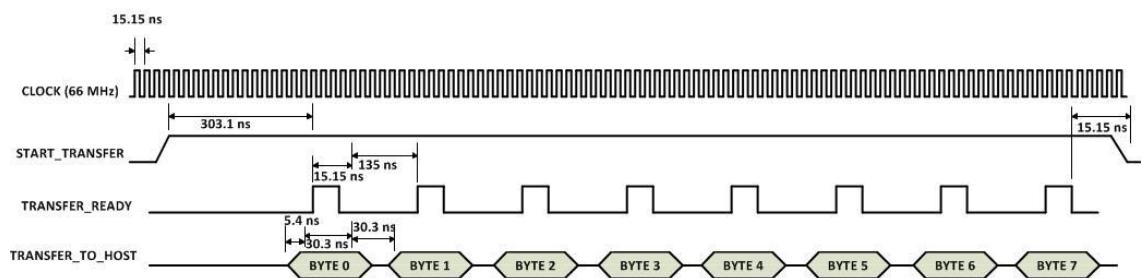


Figure xx Active Block To Host Timing

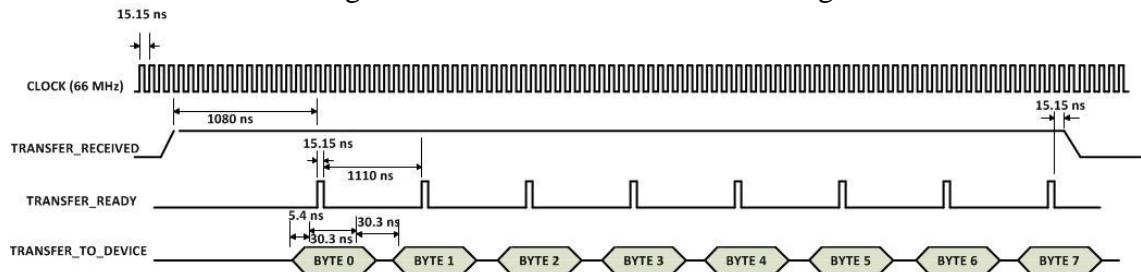


Figure xx Active Block To Device Timing

## 4 Compiling, Synthesizing, and Programming FPGA

**ALTERA**®

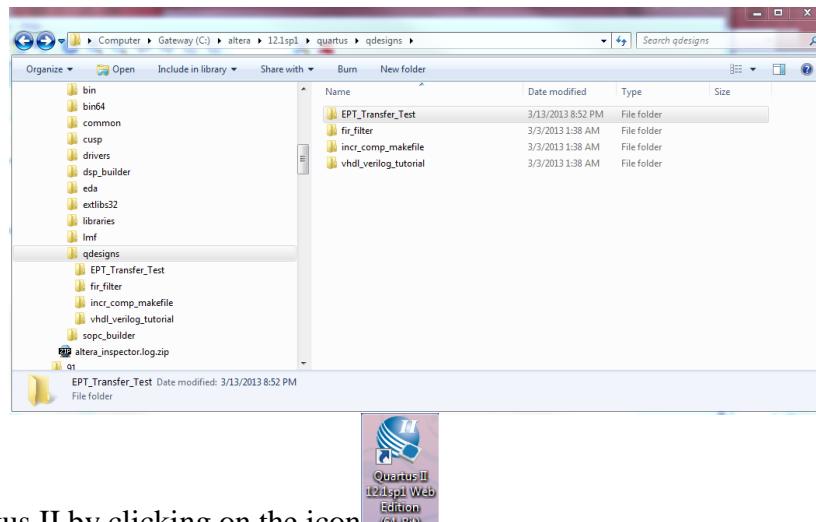


The FPGA on the EPT-4CE6-AF-D1 can be programmed with the Active Transfer Library and custom HDL code created by the user. Programming the FPGA requires the

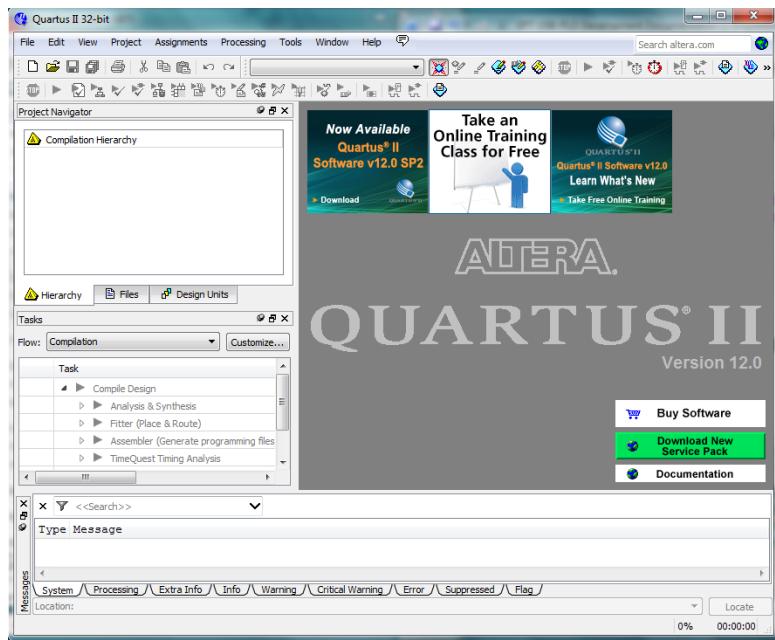
use of the Quartus II software and a standard USB cable. There are no extra parts to buy, just plug in the USB cable. Once the user HDL code is written according to the syntax rules of the language (Verilog and VHDL) it can be compiled and synthesized using the Quartus II software. This manual will not focus on HDL coding or proper coding techniques, instead it will use the example code to compile, synthesize and program the FPGA.

#### ***4.1 Setting up the Project and Compiling***

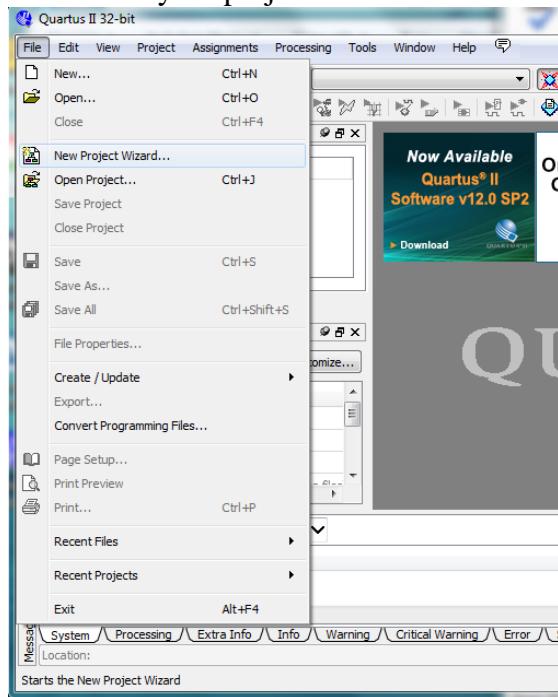
Once the HDL code (Verilog or VHDL) is written and verified using a simulator, a project can be created using Quartus II. Writing the HDL code and simulating it will be covered in later sections. Bring up Quartus II, then use Windows Explorer to browse to c:/altera/xxx/quartus/qdesigns create a new directory called: “EPT\_Transfer\_Demo”.



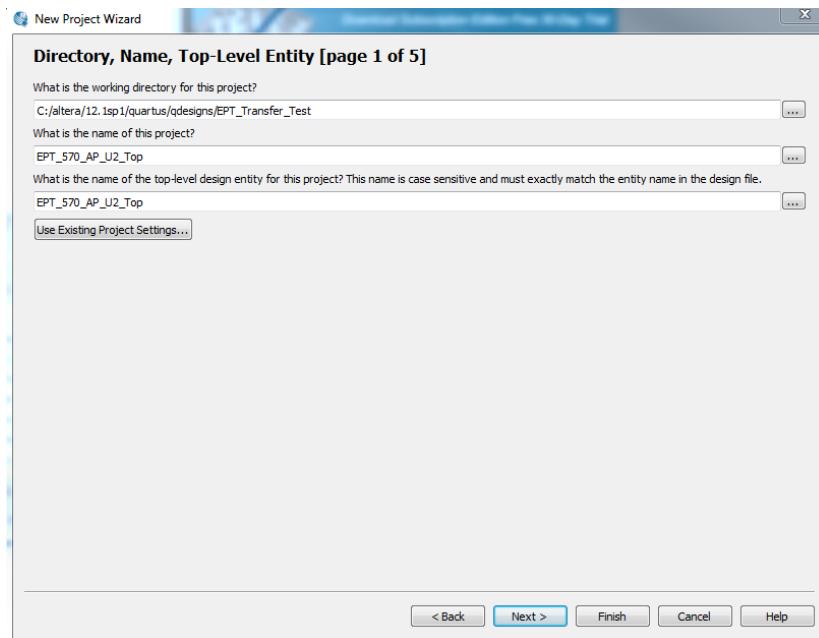
Open Quartus II by clicking on the icon



Under Quartus, Select File->New Project Wizard. The Wizard will walk you through setting up files and directories for your project.

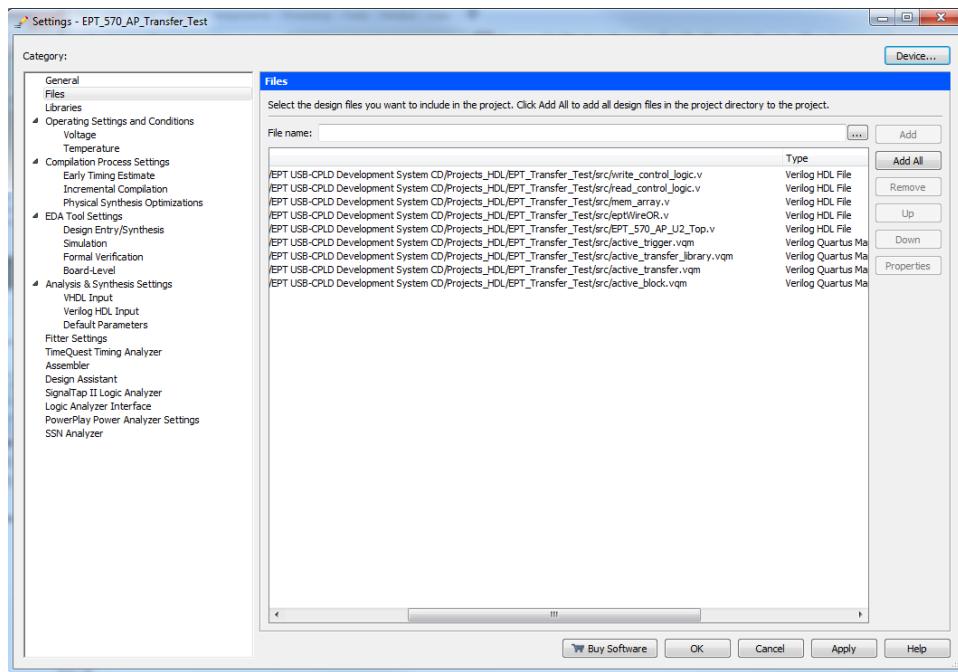


At the Top-Level Entity page, browse to the c:/altera/xxx/quartus/qdesigns directory to store your project. Type in a name for your project “EPT\_4CE6\_AF\_D1\_Top”.



Select Next. At the Add Files window: Browse to the  
\Projects\_HDL\EPT\_Transfer\_Demo\src folder of the EPT FPGA Development  
System DVD. Copy the files from the \src directory.

- Active\_block.vqm
- Active\_transfer.vqm
- Active\_trigger.vqm
- Active\_transfer\_library.v
- ft\_245\_state\_machine.v
- endpoint\_registers.vqm
- eptWireOr.v
- mem\_array.v
- read\_control\_logic.v
- write\_control\_logic.v
- EPT\_4CE6\_AF\_D1\_Top.v



Select Next, at the Device Family group, select Cyclone IV for Family. In the Available Devices group, browse down to EP4CE6E22C8 for Name.



New Project Wizard

**Family & Device Settings [page 3 of 5]**

Select the family and device you want to target for compilation.  
You can install additional device support with the Install Devices command on the Tools menu.

**Device family**

Family: Cyclone IV E  
Devices: All

**Target device**

Auto device selected by the Fitter  
 Specific device selected in 'Available devices' list  
 Other: n/a

Show in 'Available devices' list

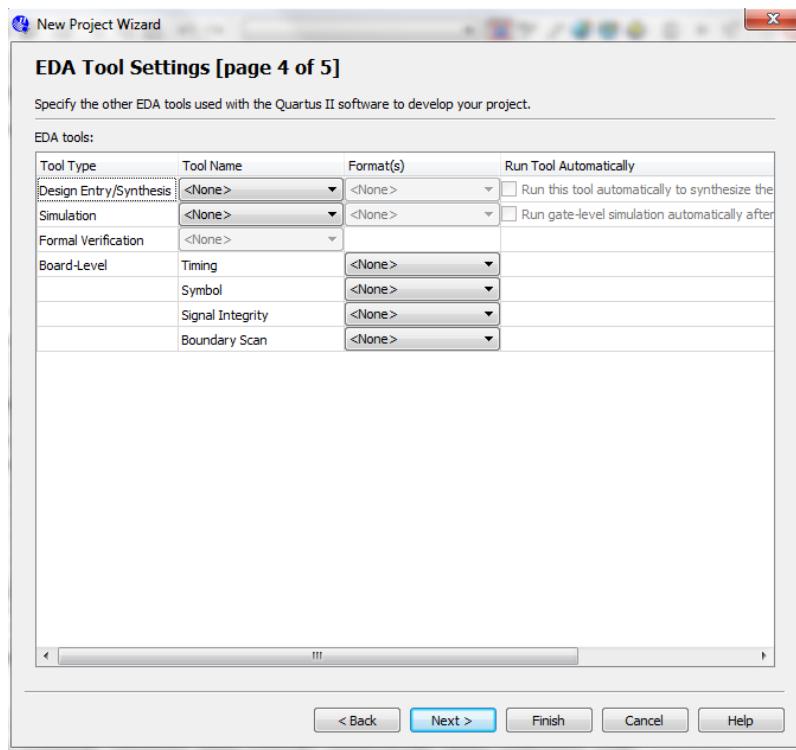
Package: Any  
Pin count: Any  
Speed grade: Any  
Name filter:   
 Show advanced devices

**Available devices:**

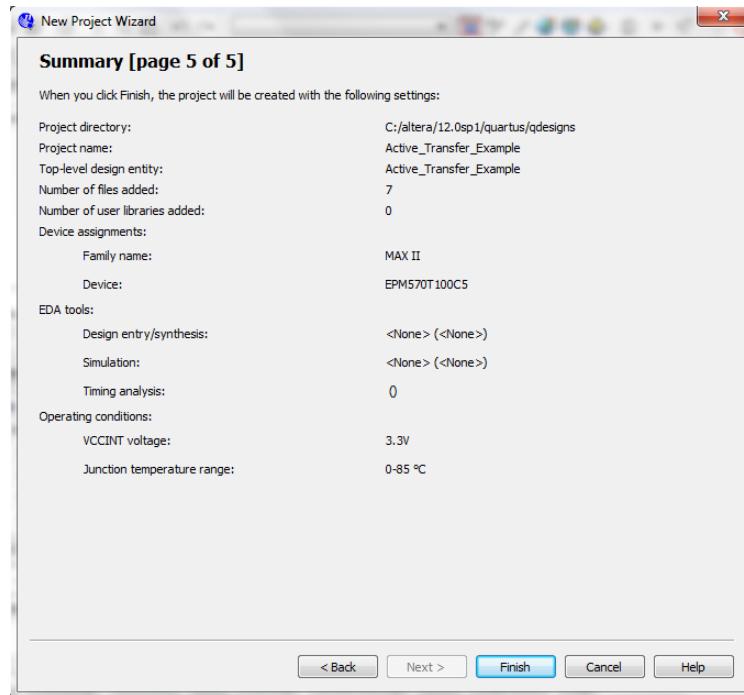
Name	Core Voltage	LEs	User I/Os	Memory Bits	Embedded multiplier 9-bit elements	PLL	Gl
EP4CE6E22A7	1.2V	6272	92	276480	30	2	10
EP4CE6E22C6	1.2V	6272	92	276480	30	2	10
EP4CE6E22C7	1.2V	6272	92	276480	30	2	10
EP4CE6E22C8	1.2V	6272	92	276480	30	2	10
EP4CE6E22C8L	1.0V	6272	92	276480	30	2	10
EP4CE6E22C9L	1.0V	6272	92	276480	30	2	10
EP4CE6E22I7	1.2V	6272	92	276480	30	2	10
EP4CE6E22I8L	1.0V	6272	92	276480	30	2	10
EP4CE6F17A7	1.2V	6272	180	276480	30	2	10
EP4CE6F17C6	1.2V	6272	180	276480	30	2	10

< Back  Finish Cancel Help

Select Next, leave defaults for the EDA Tool Settings.



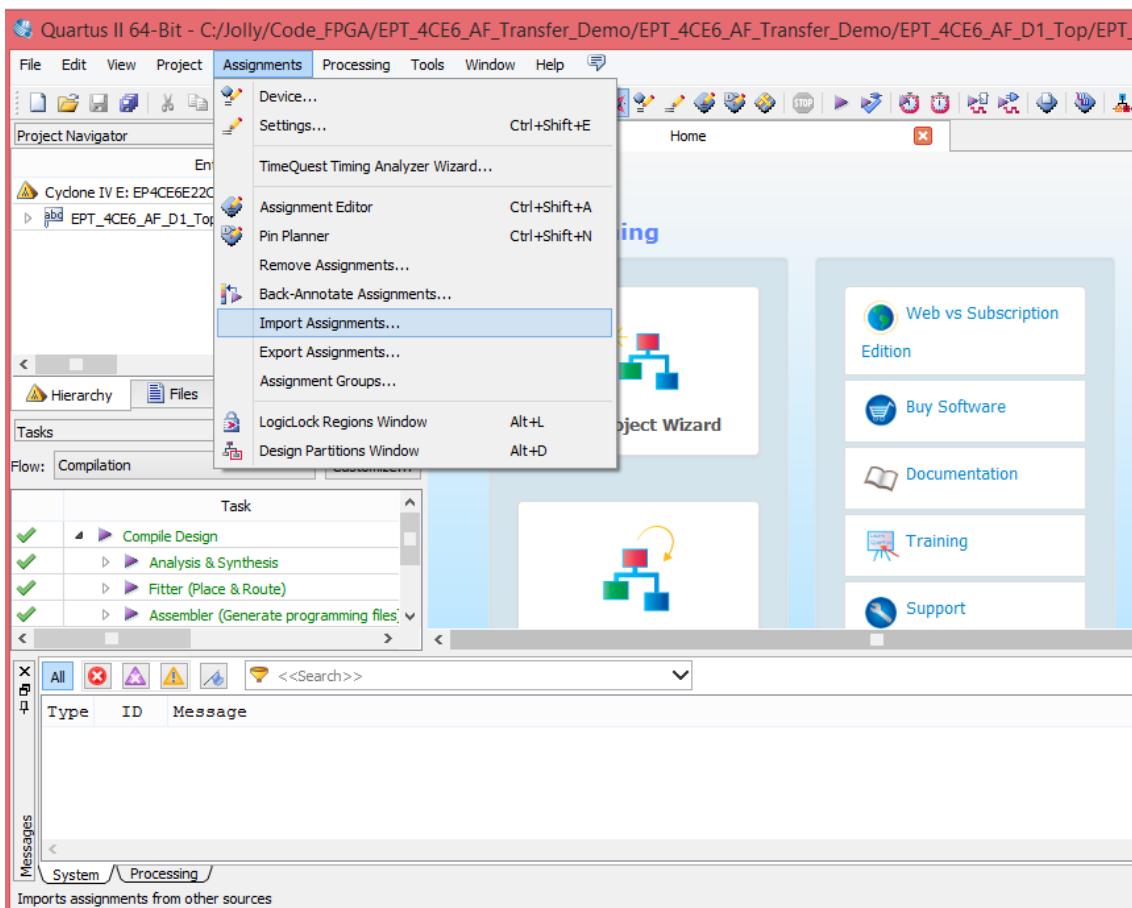
Select Next, then select Finish. You are done with the project level selections.



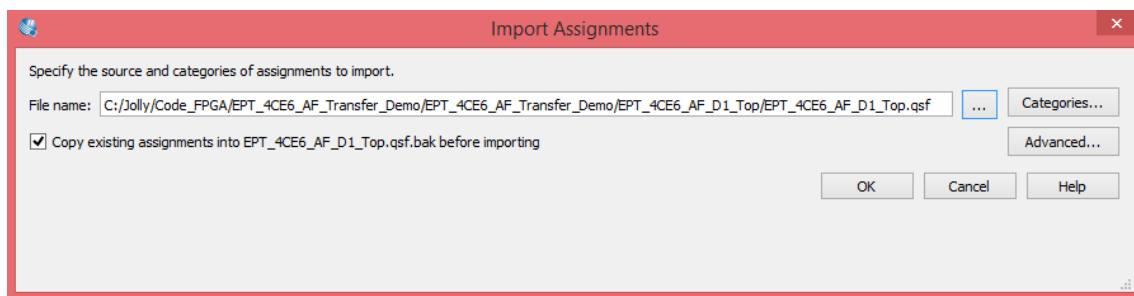
Next, we will select the pins and synthesize the project.

#### 4.1.1 Selecting Pins and Synthesizing

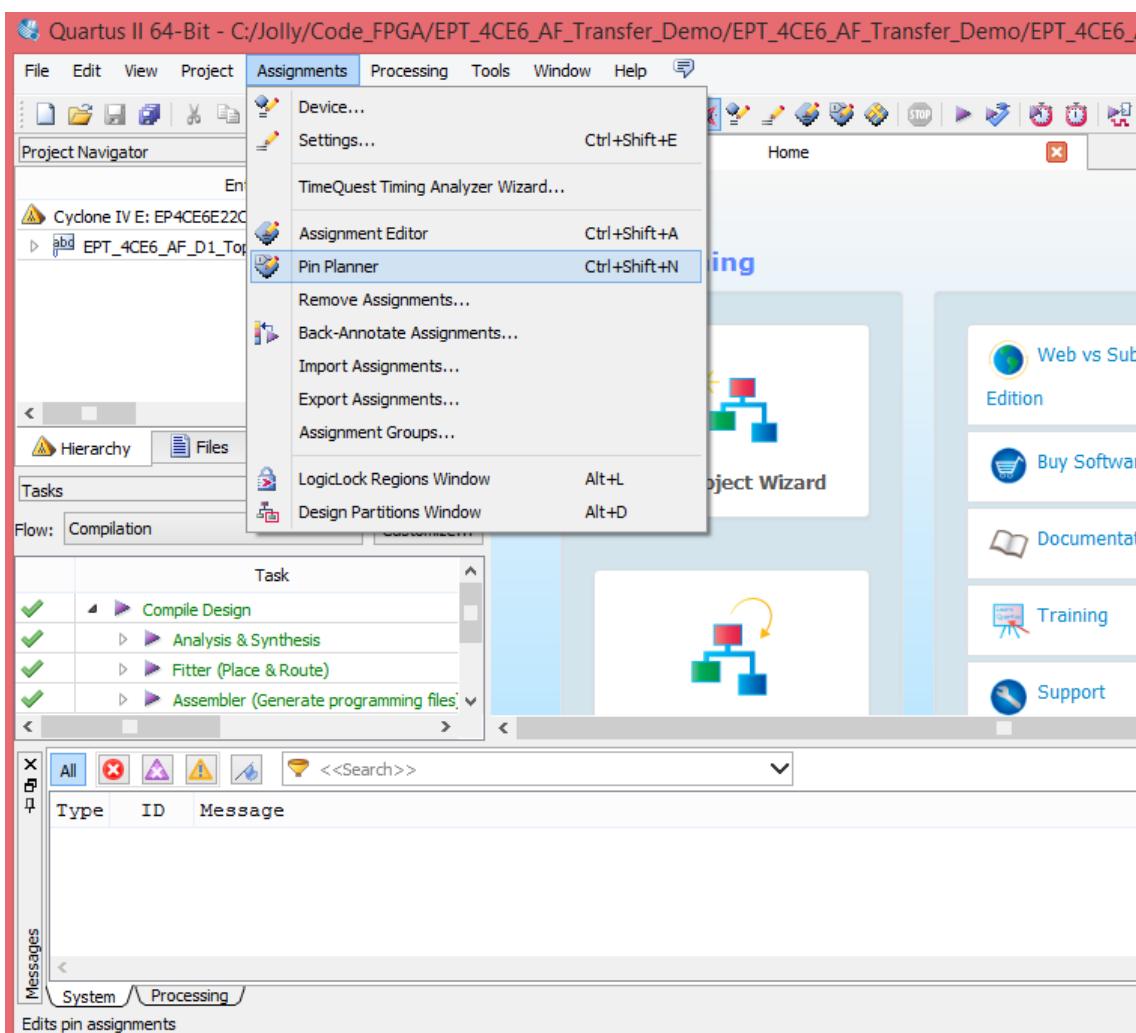
With the project created, we need to assign pins to the project. The signals defined in the top level file (in this case: EPT\_4CE6\_AF\_D1\_Top.v) will connect directly to pins on the FPGA. The Pin Planner Tool from Quartus II will add the pins and check to verify that our pin selections do not violate any restrictions of the device. In the case of this example we will import pin assignments that created at an earlier time. Under Assignments, Select Import Assignments.



At the Import Assignment dialog box, Browse to the  
\Projects\_HDL\EPT\_Transfer\_Demo \ EPT-4CE6-AF-D1\_TOP folder of the EPT  
FPGA Development System DVD. Select the “EPT-4CE6-AF-D1\_Top.qsf” file.



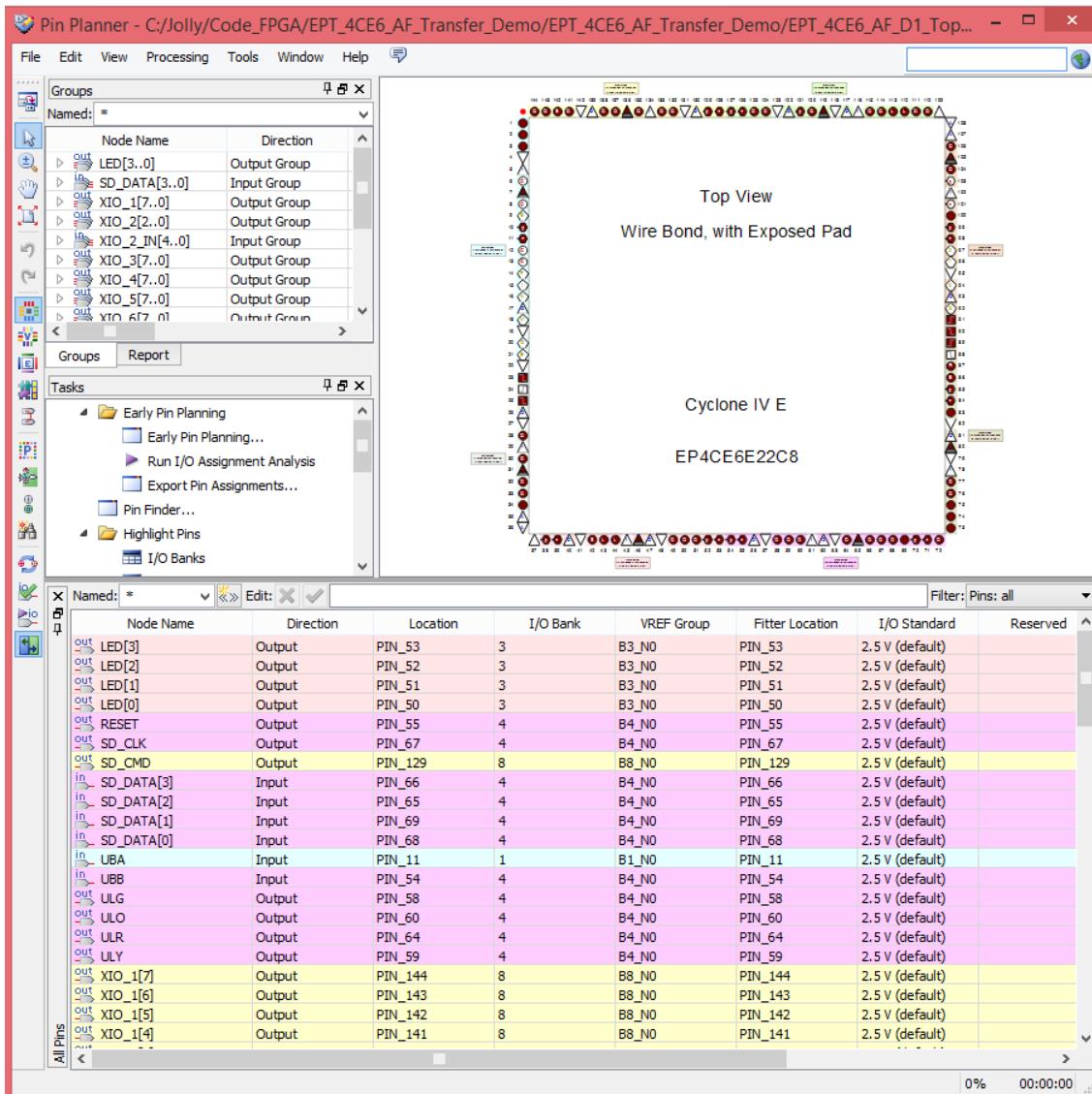
Click Ok. Under Assignments, Select Pin Planner. Verify the pins have been imported correctly.





## FPGA Development System User Manual

The pin locations should not need to be changed for EPT USB FPGA Development System. However, if you need to change any pin location, just click on the “location” column for the particular node you wish to change. Then, select the new pin location from the drop down box.



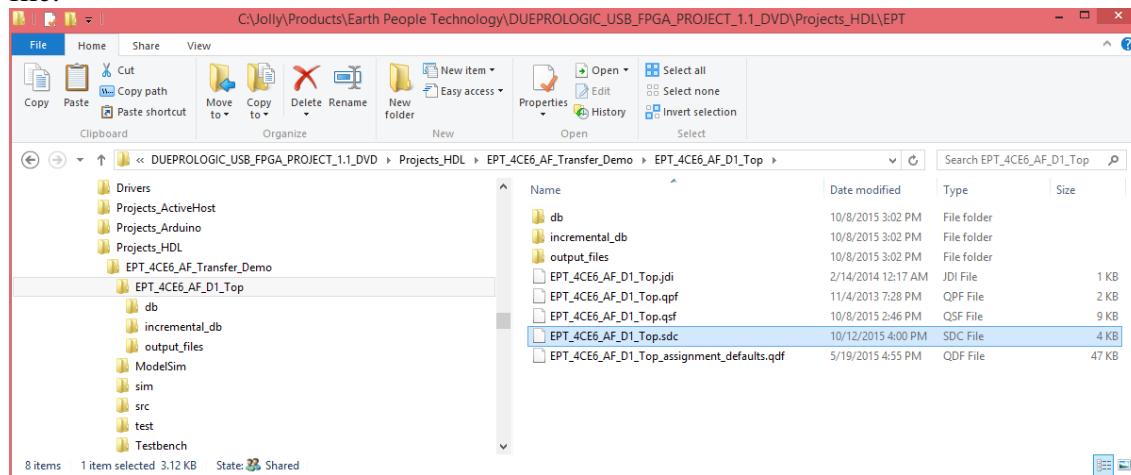
Exit the Pin Planner. Next, we need to add the Synopsys Design Constraint file. This file contains timing constraints which forces the built in tool called TimeQuest Timing Analyzer to analyze the path of the synthesized HDL code with setup and hold times of the internal registers. It takes note of any path that may be too long to appropriately meet the timing qualifications. For more information on TimeQuest Timing Analyzer, see

[http://www.altera.com/literature/hb/qts/qts\\_qii53018.pdf?GSA\\_pos=1&WT.oss\\_r=1&WT.oss=TimeQuest Timing Analyzer](http://www.altera.com/literature/hb/qts/qts_qii53018.pdf?GSA_pos=1&WT.oss_r=1&WT.oss=TimeQuest Timing Analyzer)

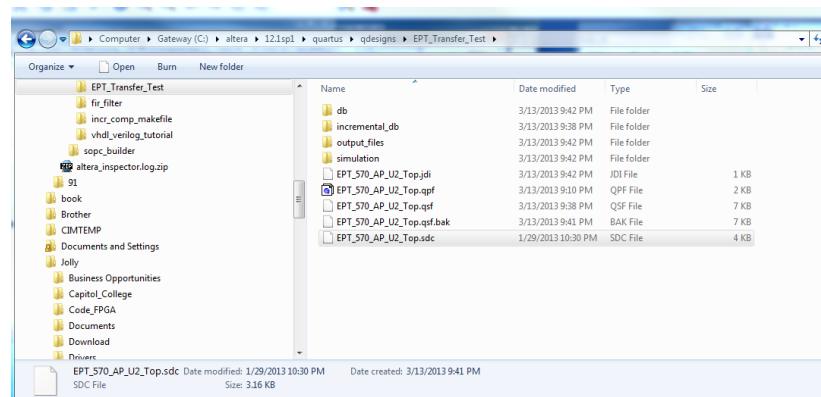


## FPGA Development System User Manual

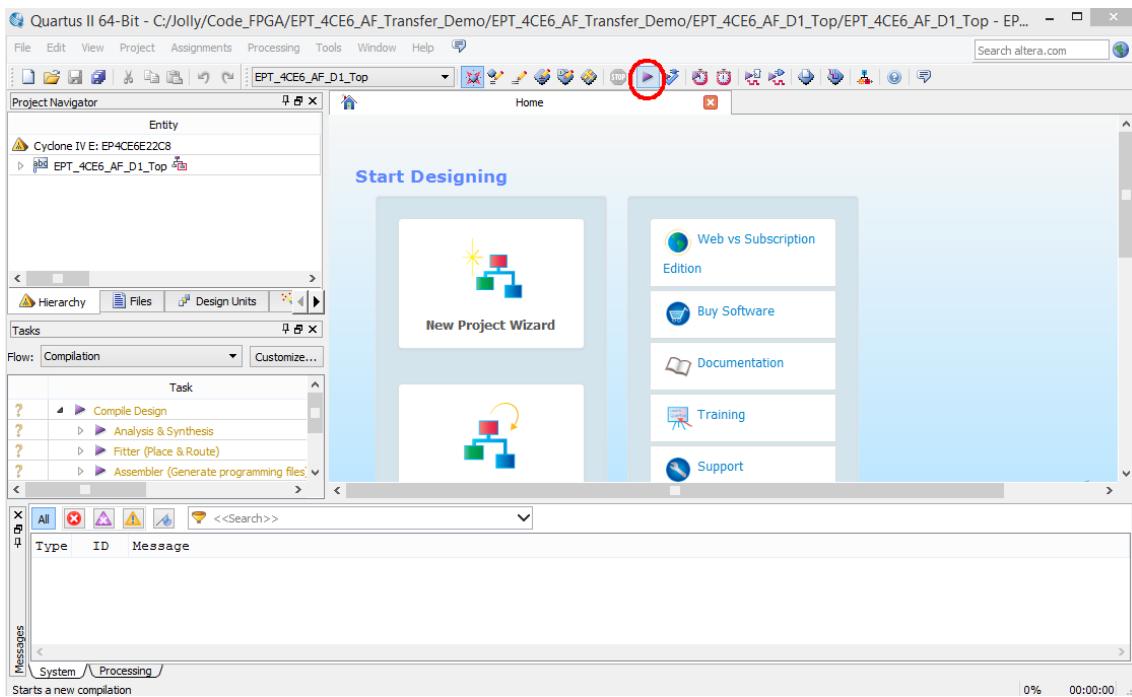
Browse to the \Projects\_HDL\EPT\_Transfer\_Demo \ EPT-4CE6-AF-D1\_TOP folder of the EPT FPGA Development System DVD. Select the “EPT-4CE6-AF-D1\_Top.sdc” file.



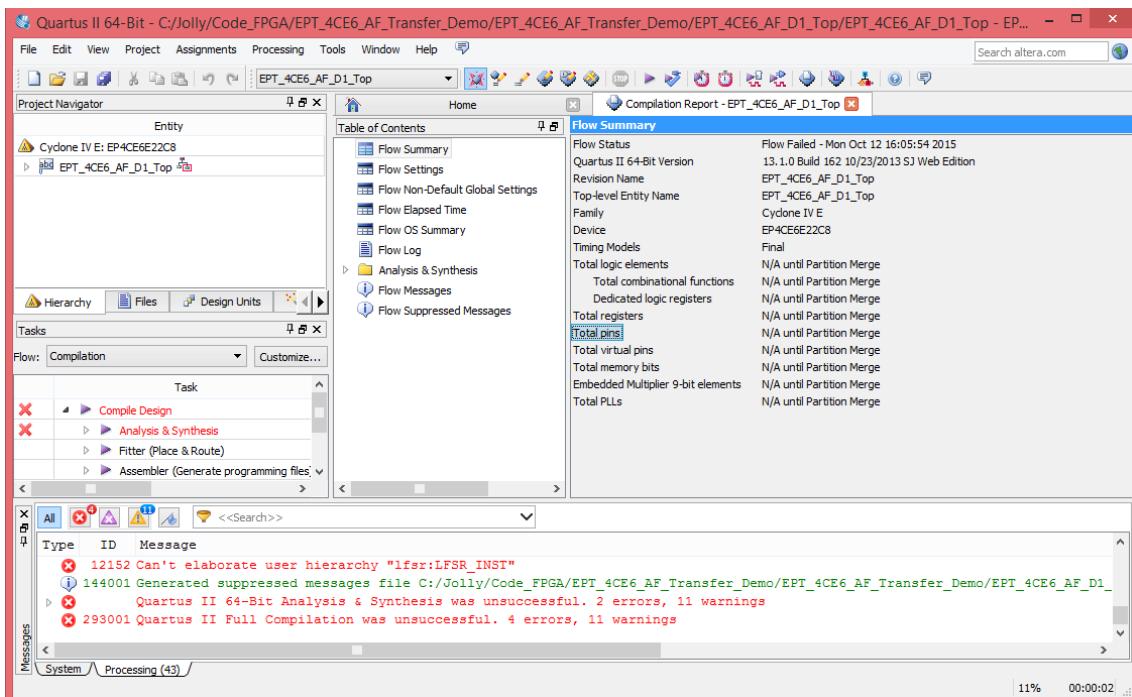
Copy the file and browse to c:\altera\xxx\quartus\qdesigns\EPT\_Transfer\_Demo directory. Paste the file.



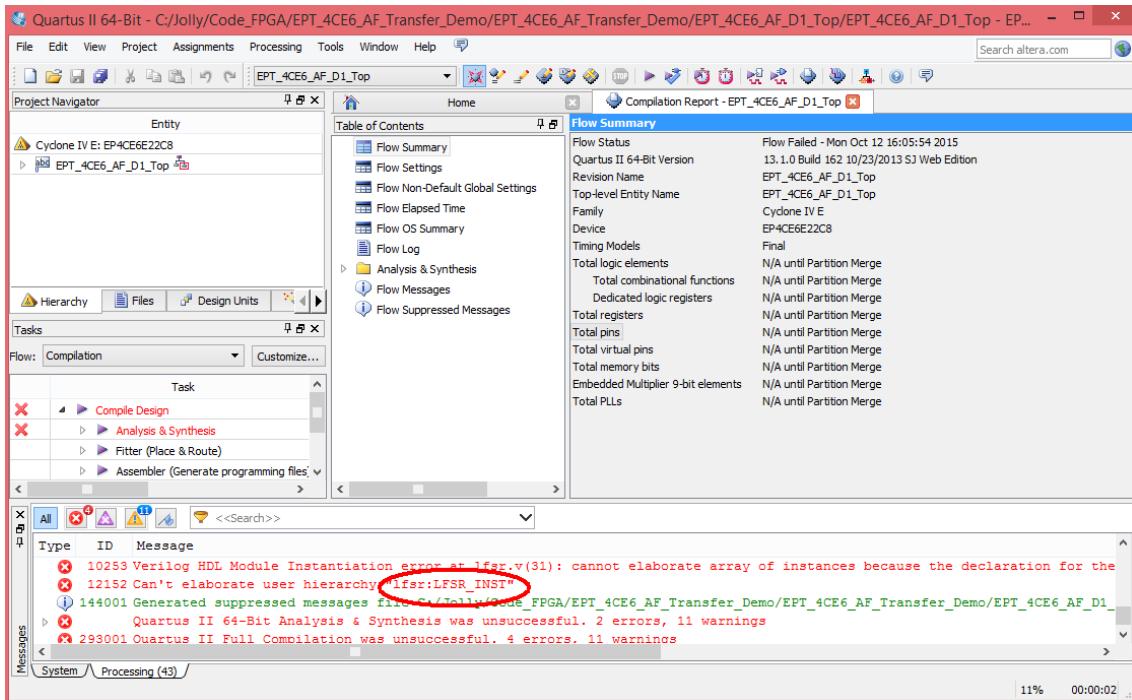
Select the Start Compilation button.



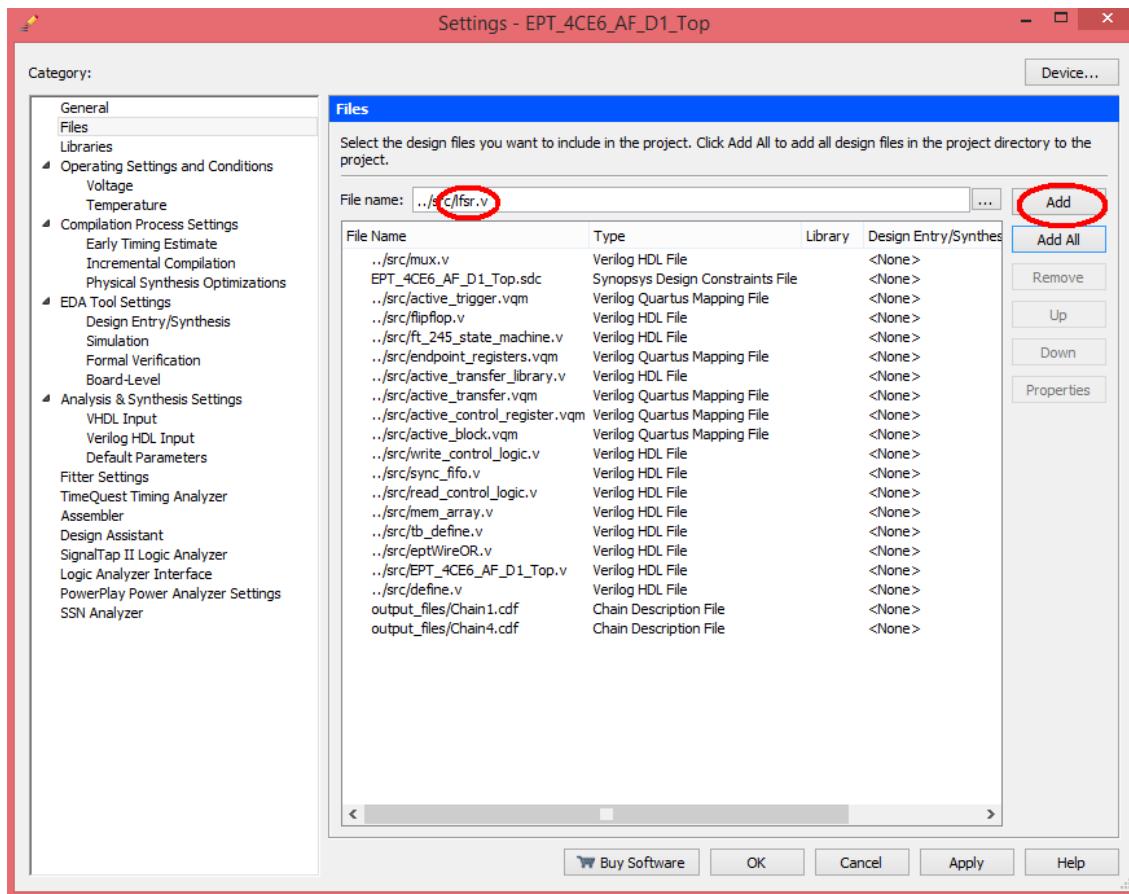
If you forget to include a file or some other error you should expect to see a screen similar to this:



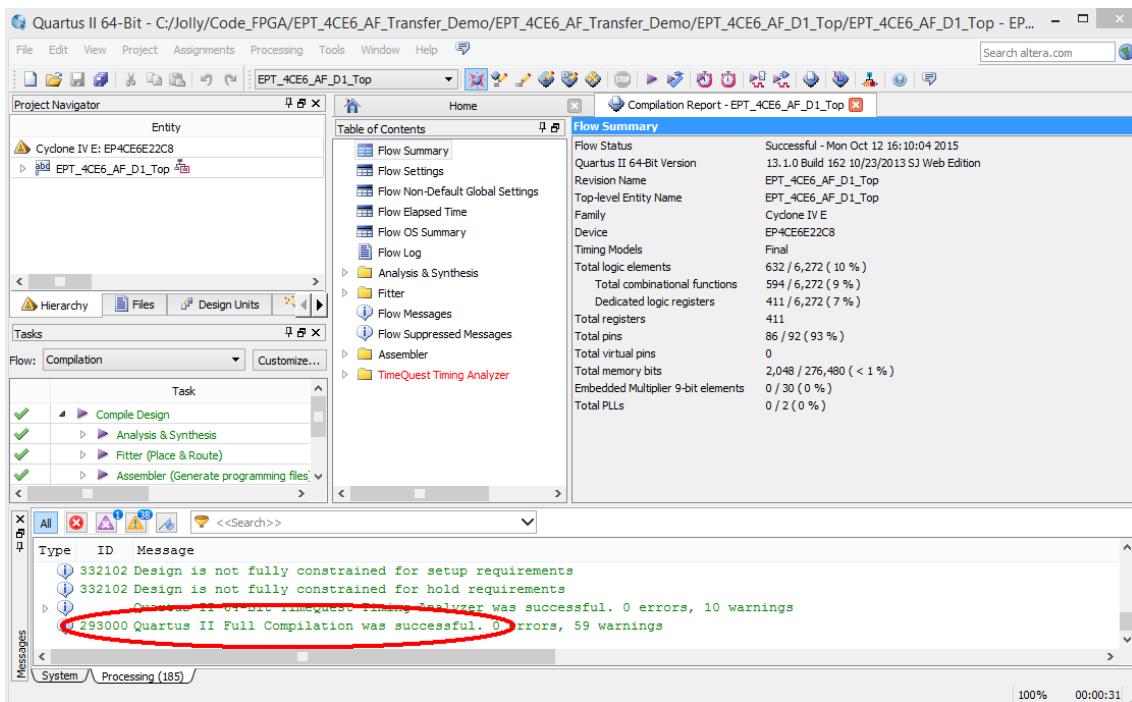
Click Ok, then select the “Error” tab to see the error.



The error in this case is the missing file “sync\_fifo”. Click on the Assignment menu, then select Settings, then select Files. Add the “sync\_fifo.v” file from the database.



Click Ok then re-run the Compile process. After successful completion, the screen should look like the following:

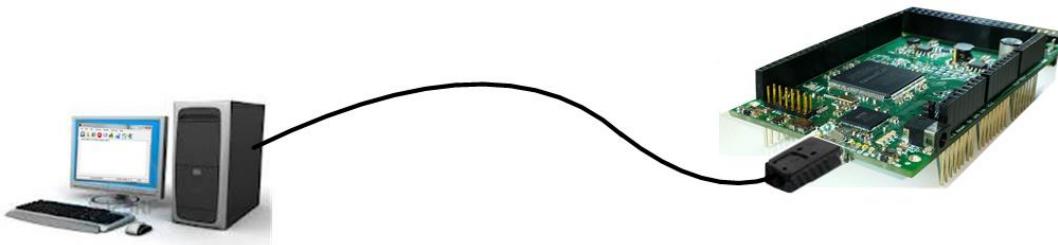


At this point the project has been successfully compiled, synthesized and a programming file has been produced. See the next section on how to program the FPGA.

#### 4.1.2 Configuring the FPGA

**Note: EPT Blaster Driver supports Quartus II 13.1 only! Please use the Quartus version supplied with the EPT FPGA DVD.**

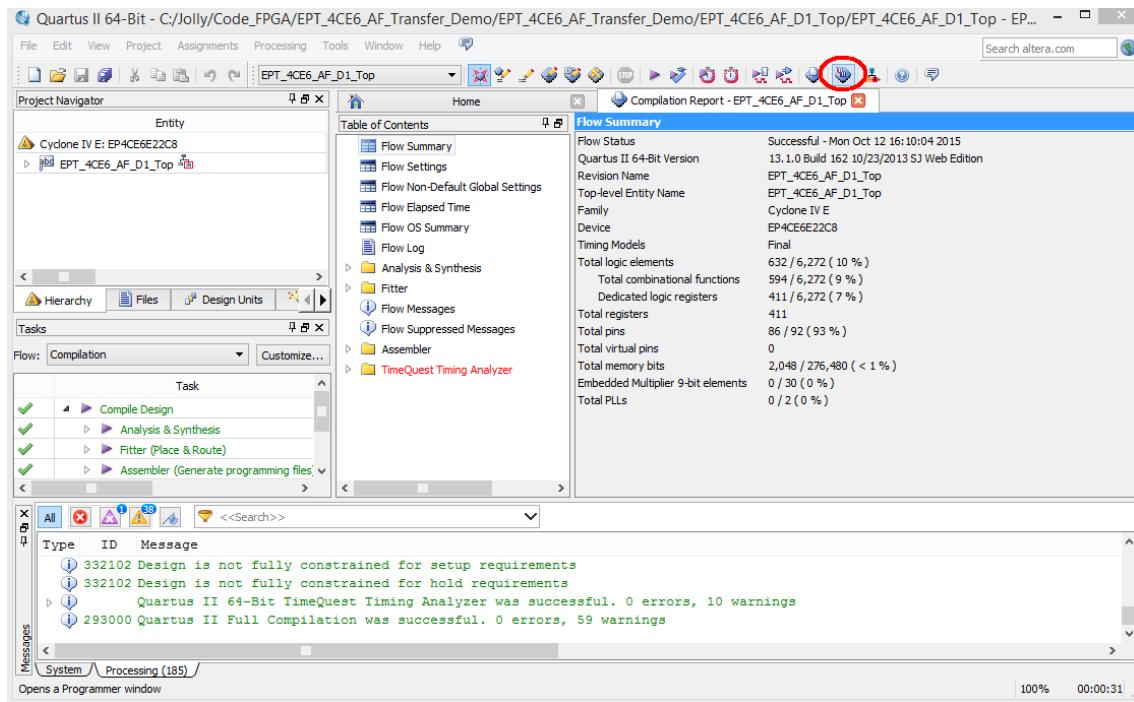
Configuring the FPGA is quick and easy. All that is required is a standard USB Micro B cable and the EPT\_Blaster Driver DLL. Connect the EPT-4CE6-AF-D1 to the PC, open up Quartus II, open the programmer tool, and click the Start button. To program the DPL Configuration Flash, follow the steps to install the USB Driver and the JTAG Driver Insert for Quartus II.



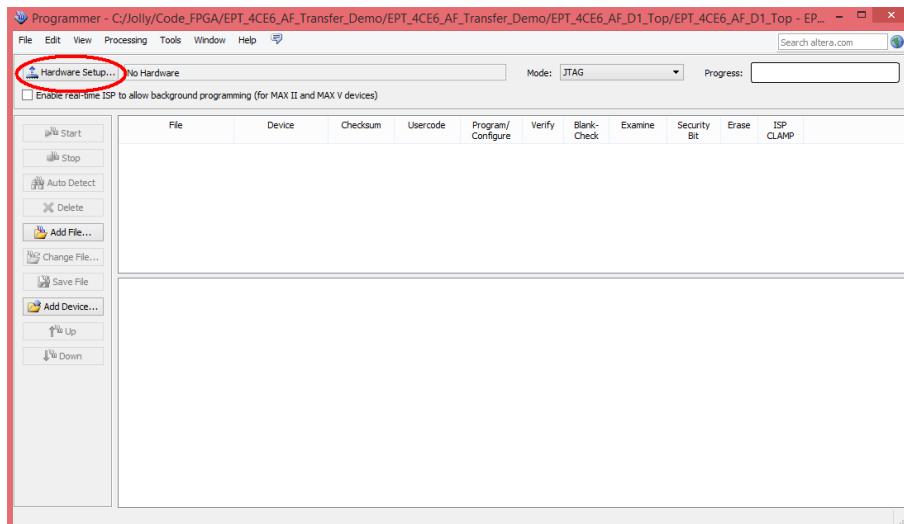
If the project created in the previous sections is not open, open it. Click on the Programmer button.



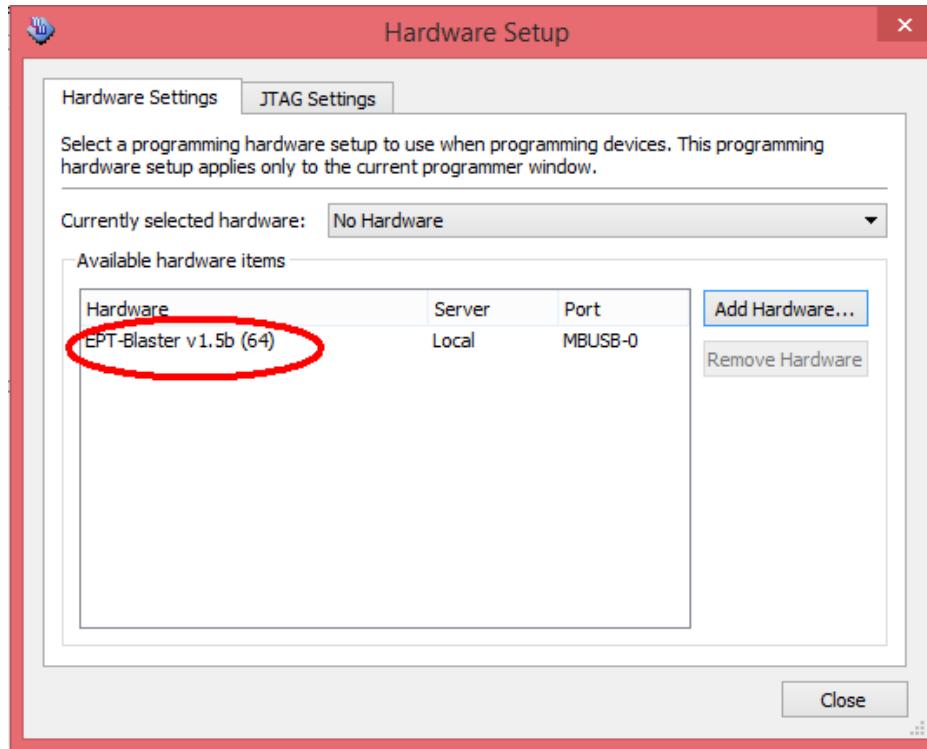
## FPGA Development System User Manual



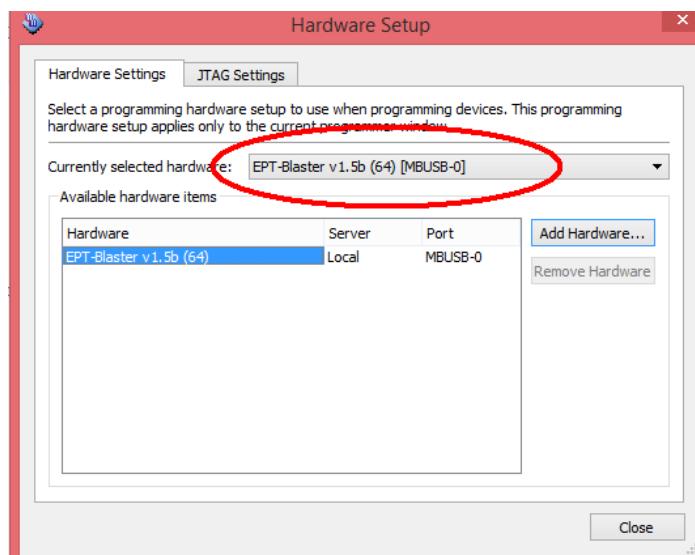
The Programmer Window will open up with the programming file selected. Click on the Hardware Setup button in the upper left corner.



The Hardware Setup Window will open. In the “Available hardware items”, double click on “EPT-Blaster v1.5b”.



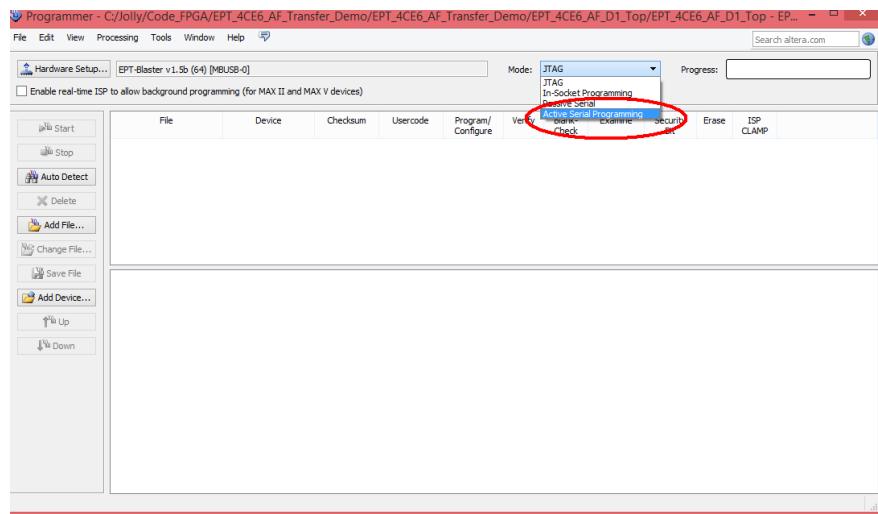
If you successfully double clicked, the “Currently selected hardware:” dropdown box will show the “EPT-Blaster v1.5b”.



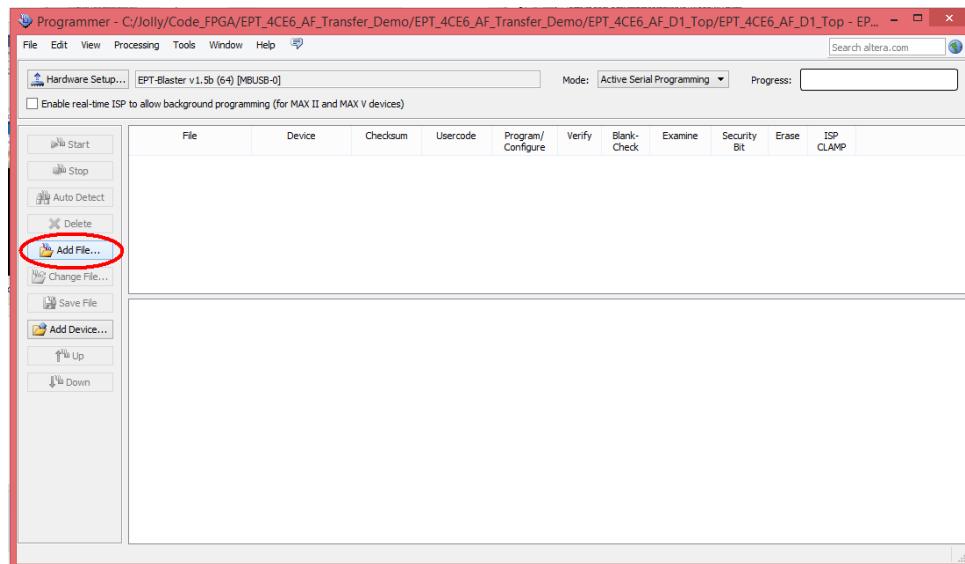
Click on the “Mode:” drop down box. Select the “Active Serial Programming” option.

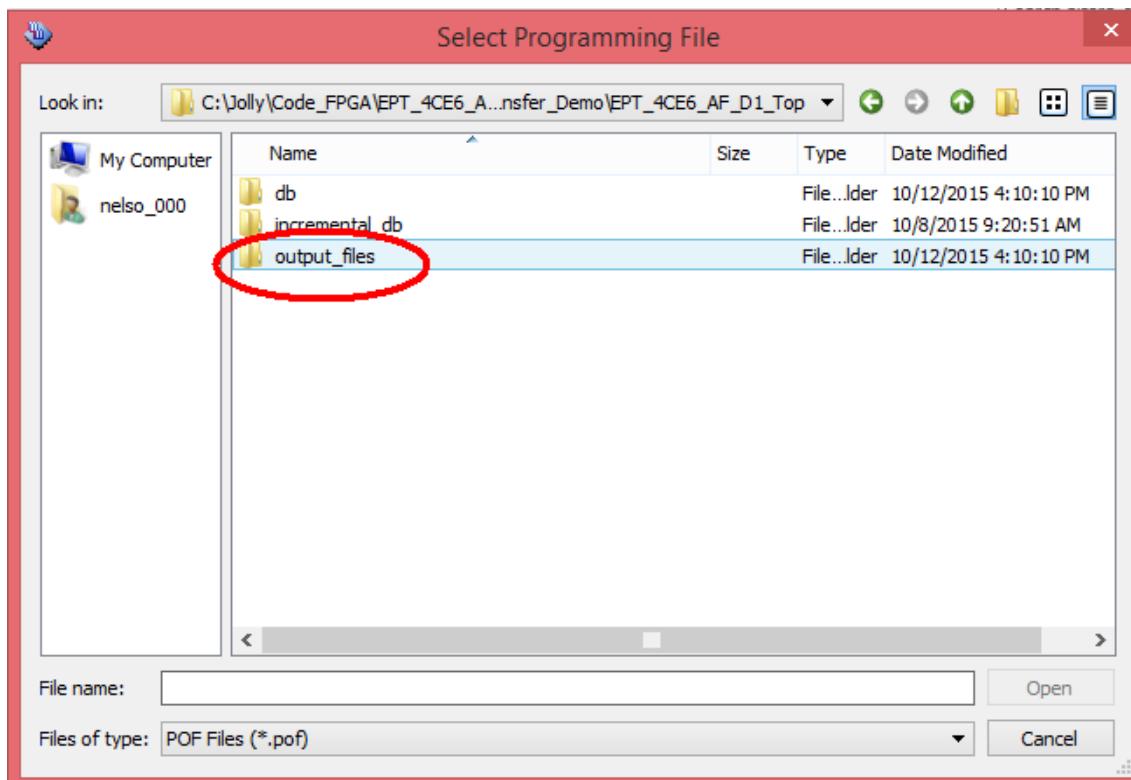


## FPGA Development System User Manual

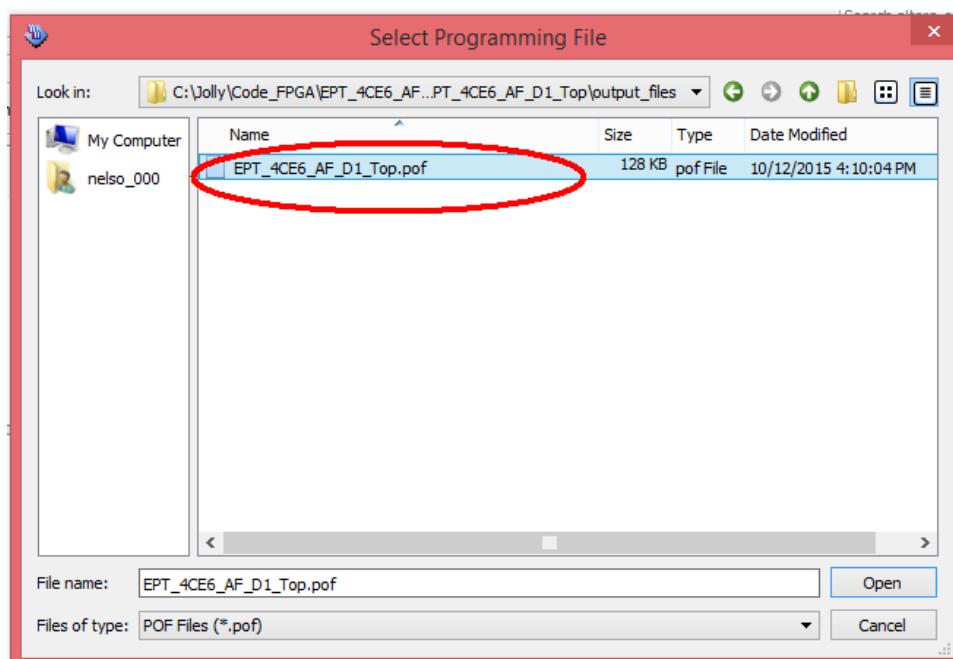


Click on the “Add File” button





At the Browse window, double click on the output files folder.

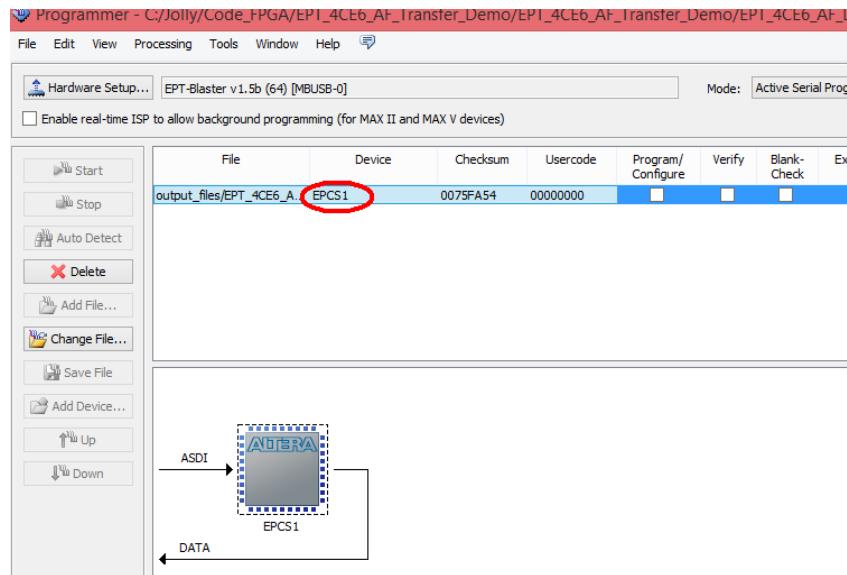




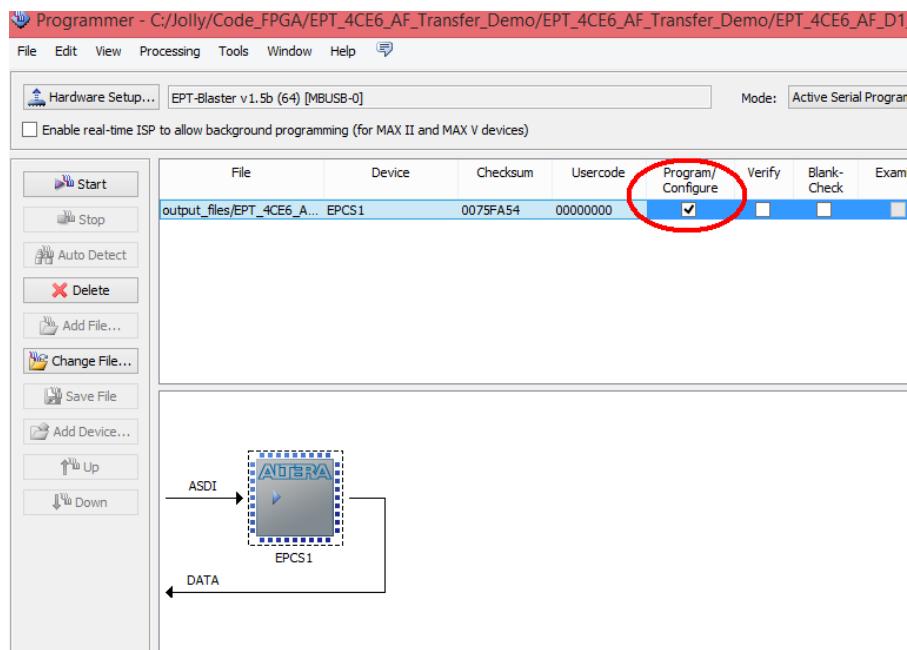
## FPGA Development System User Manual

Double click on the “EPT\_4CE6\_D1\_Top.pof” file. Click the Open button in the lower right corner.

Select the EPSC1 under “Device”.



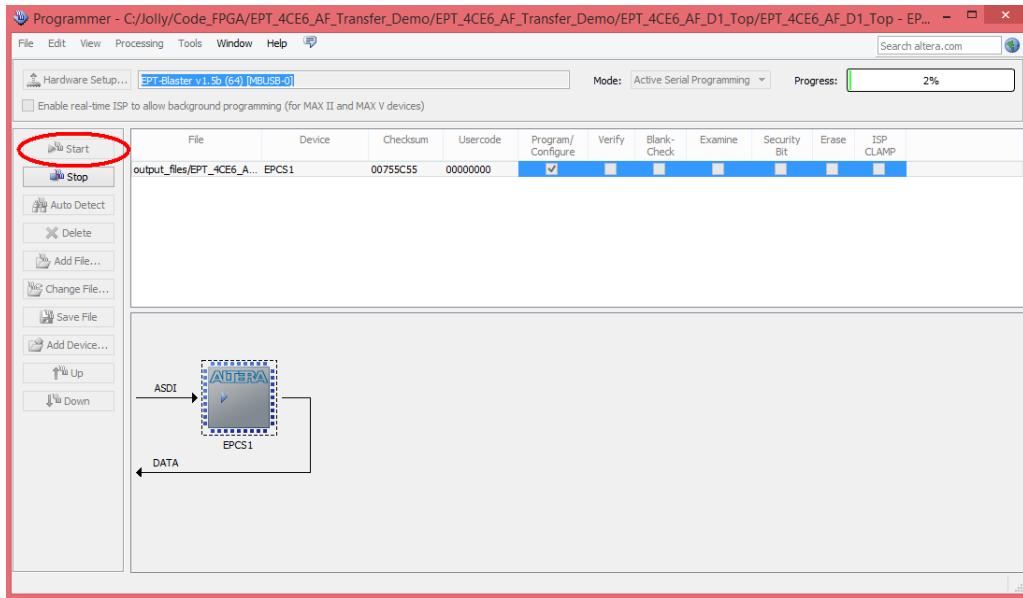
Next, select the checkbox under the “Program/Configure” of the Programmer Tool.



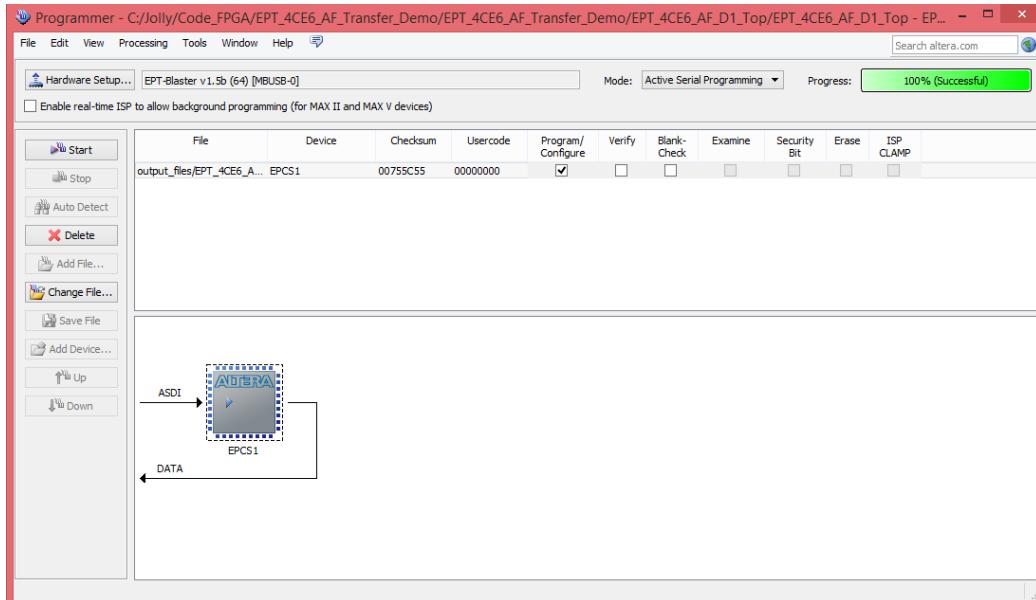


## FPGA Development System User Manual

Click on the Start button to to start programming the FPGA. The Progress bar will indicate the progress of programming.



When the programming is complete, the Progress bar will indicate success.



At this point, the EPT-4CE6-AF-D1 is programmed and ready for use. To test that the FPGA is properly programmed, bring up the Active Transfer Demo Tool. Click on one

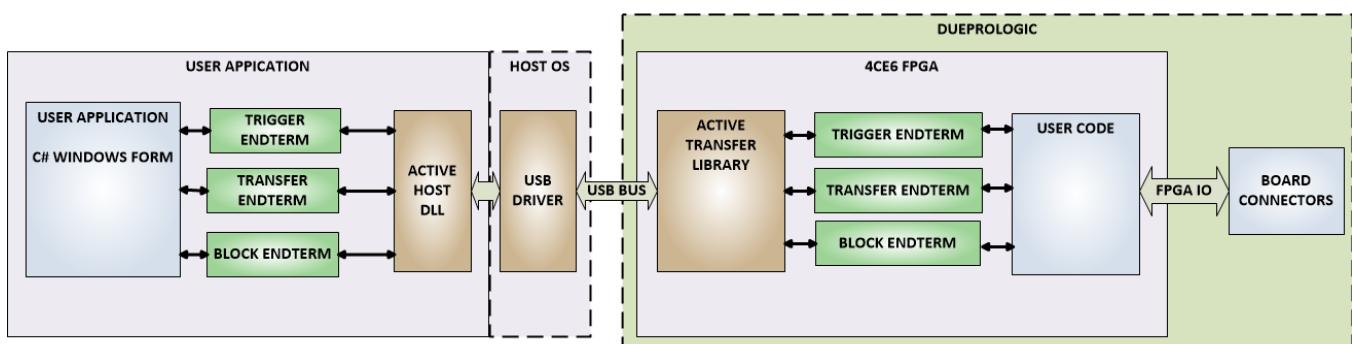
of the LED's and verify that the LED selected lights up. Press one of the switches on the board and ensure that the switch is captured on the Active Host Test Tool. Now you are ready to connect to the Arduino Due and write some code to transfer data between microcontroller and PC.

## 5 Active Host Application

The Active Host SDK is provided as a dll which easily interfaces to application software written in C#, C++ or C. It runs on the PC and provides transparent connection from PC application code through the USB driver to the user FPGA code. The user code connects to "Endterms" in the Active Host dll. These host "Endterms" have complementary HDL "Endterms" in the Active Transfer Library. Users have seamless bi-directional communications at their disposal in the form of:

- Trigger Endterm
- Transfer Endterm
- Block Endterm

User code writes to the Endterms as function calls. Just include the address of the individual module (there are eight individually addressable modules of each Endterm). Immediately after writing to the selected Endterm, the value is received at the HDL Endterm in the FPGA. The Trigger Endterms are used as "switches". The user code can set a Trigger bit in the FPGA and cause an event to occur. The Transfer Endterm sends one byte to the FPGA. The Block Endterm sends a block of bytes. By using one of the Active Host Endterms, the user can create a dynamic, bi-directional, and configurable data transfer design.



### 5.1 Trigger EndTerm

The Trigger EndTerm is a software component that provides a direct path from the users application to the commensurate Trigger EndTerm in the FPGA. The Trigger has eight bits and is intended to be used to provide a switch at the opposite EndTerm. They are fast acting and are not stored or buffered by memory. When the user code sets a Trigger, it is immediately passed through to the opposite EndTerm via the USB driver.



## FPGA Development System User Manual

When receiving Trigger, the user application is required to respond to a callback from the Active Host dll.

### **5.2 Transfer(Byte) EndTerm**

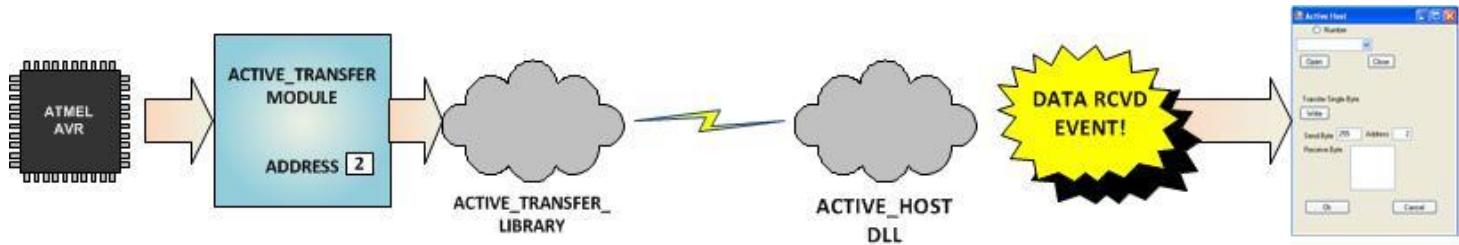
The Transfer EndTerm is a software component that provides a direct path from the users application to the commensurate Transfer EndTerm in the FPGA. It is used to transfer a byte to and from the FPGA. Eight separate Transfer EndTerm modules can be instantiated in the FPGA. Each module is addressed by the user application. Sending a byte is easy, just use the function call with the address and byte value. The byte is immediately sent to the corresponding EndTerm in the FPGA. Receiving a byte is just as easy, a callback function is registered at initialization. When the FPGA transmits a byte using its EndTerm, the callback function is called in the user application. The user code must store this byte in order to use it. The incoming Transfers are stored in a circular buffer in memory. This allows the user code to fetch the transfers with out losing bytes.

### **5.3 Block EndTerm**

The Block EndTerm is a software component that provides a direct path from the users application to the commensurate Block EndTerm in the FPGA. The Block EndTerm is used to transfer a complete block to the FPGA. Block size is limited to 1 to 256 bytes. Eight separate Block EndTerm modules can be instantiated in the FPGA. Each module is addressed by the user application. Sending a block is easy, just use the function call with the address, block length, byte array. The block is buffered into a circular buffer in memory then transmitted via the USB bus to the Block EndTerm in the FPGA. Receiving a block is just as easy, a callback function is registered at initialization. When the FPGA transmits a block using its EndTerm, the callback function is called in the user application. The incoming Transfers are stored in a circular buffer in memory. This allows the user code to fetch the transfers with out losing bytes.

### **5.4 Active Host DLL**

The Active\_Host DLL is designed to transfer data from the FPGA when it becomes available. The data will be stored into local memory of the PC, and an event will be triggered to inform the user code that data is available from the addressed module of the FPGA. This method of automatically moving data from the user code Endterm in the FPGA makes the data transfer transparent.



The data seamlessly appears in Host PC memory from the Arduino. The user code will direct the data to a control such as a textbox on a Windows Form. The transparent receive transfer path is made possible by a Callback mechanism in the Active Host dll. The dll calls a registered callback function in the user code. The user code callback can be designed to generate any number of events to handle the received data.

The user application will access the FPGA by use of functions contained in the Active Host dll. The functions to access the FPGA are:

- EPT\_AH\_GetName()
- EPT\_AH\_GetVersionString()
- EPT\_AH\_GetVersionControl()
- EPT\_AH\_GetInterfaceVersion()
- EPT\_AH\_CheckCompatibility()
- EPT\_AH\_Open()
- EPT\_AH\_Close()
- EPT\_AH\_Initialize()
- EPT\_AH\_Release()
- EPT\_AH\_QueryDevices()
- EPT\_AH\_SelectActiveDeviceByName()
- EPT\_AH\_SelectActiveDeviceByIndex()
- EPT\_AH\_GetDeviceName()
- EPT\_AH\_GetDeviceSerial()
- EPT\_AH\_OpenDeviceByIndex()
- EPT\_AH\_CloseDeviceByIndex()
- EPT\_AH\_CloseDeviceByName()
- EPT\_AH\_SendTrigger ()
- EPT\_AH\_SendByte ()
- EPT\_AH\_SendBlock ()
- EPT\_AH\_SendTransferControlByte()
- EPT\_AH\_RegisterReadCallback ()
- EPT\_AH\_GetLastError()
- EPT\_AH\_PerformSelfTest()
- EPT\_AH\_LEDBlinky()
- EPT\_AH\_SetDebugMode()



- EPT\_AH\_RegisterReadCallbackForChannel()
- EPT\_AH\_FlushDeviceChannelBuffer()
- EPT\_AH\_GetDeviceChannelFreeBufferBytes()
- EPT\_AH\_GetDeviceChannelPendingBufferBytes()
- EPT\_AH\_SetChannelConnectionFlag()
- EPT\_AH\_GetChannelConnectionFlag()

#### 5.4.1 Active Host Open Device

To use the library functions for data transfer and triggering, an Earth People Technology device must be opened. The first function called when the Windows Form loads up is the <project\_name>\_Load(). This function is called automatically upon the completion of the Windows Form, so there is no need to do anything to call it. Once this function is called, it in turn calls the ListDevices(). Use the function List Devices() to detect all EPT devices connected to the PC.

```
// Main object loader
private void EPT_Transfer_Demo_Load(object sender, System.EventArgs e)
{
    // Call the List Devices function
    ListDevices();

    //Active Host Debug
    //EPT_AH_SetDebugMode(1);

}
```

The ListDevices() function calls the EPT\_AH\_Open() function to load up the ActiveHost Dll. Next, it calls EPT\_AH\_QueryDevices() which searches through the registry files to determine the number of EPT devices attached to the PC. Next, EPT\_AH\_GetDeviceName() is called inside a for loop to return the ASCII name of each device attached to the PC. It will automatically populate the combo box, cmbDevList with all the EPT devices it finds.

```
// List Devices| function
private unsafe Int32 ListDevices ()
{
    Int32 result;
    Int32 num_devices;
    Int32 icurrentIndex;

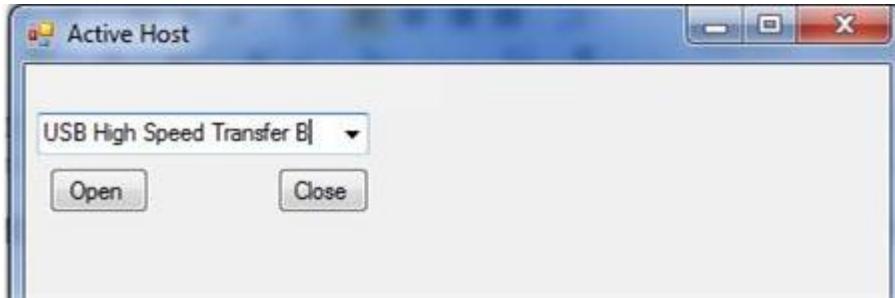
    // Open the DLL
    result = EPT_AH_Open(null, null, null);
    if (result != 0)
    {
        MessageBox.Show("Could not attach to the ActiveHost library");
        return 0;
    }

    // Query connected devices
    num_devices = EPT_AH_QueryDevices();

    //Prepare the Combo box for population
    icurrentIndex = cmbDevList.SelectedIndex;
    cmbDevList.Items.Clear();

    // Go through all available devices
    for (device_index = 0; device_index < num_devices; device_index++)
    {
        String str;
        str = Marshal.PtrToStringAnsi((IntPtr)EPT_AH_GetDeviceName(device_index));
        cmbDevList.Items.Add(str);
    }
    return 0;
}
```

The user will select the device from the drop down combo box. This can be seen when the Windows Form is opened and the cmbDevList combo box is populated with all the devices. The selected device will be stored as an index number in the variable device\_index.



In order to select the device, the user will click on the “Open” button which calls the Open\_Device() function. The device\_index is passed into the EPT\_AH\_OpenDeviceByIndex() function. If the function is successful, the device name is displayed in the label, labelDeviceCnt. Next, the device is made the active



## FPGA Development System User Manual

device and the callback function is registered. Finally, the Open button is grayed out and the Close button is made active.

```
// Open the device
public unsafe Int32 OpenDevice()
{
    device_index = (int)cmbDevList.SelectedIndex;
    if (EPT_AH_OpenDeviceByIndex(device_index) == 0)
    {
        String message = "Could not open device " +
            Marshal.PtrToStringAnsi((IntPtr)EPT_AH_GetDeviceName(device_index)) + ", " +
            Marshal.PtrToStringAnsi((IntPtr)EPT_AH_GetDeviceSerial(device_index));
        MessageBox.Show(message);
        return 0;
    }
    else
    {
        labelDeviceCnt.Text = "Connected to device " +
            Marshal.PtrToStringAnsi((IntPtr)EPT_AH_GetDeviceName(device_index)) + ", " +
            Marshal.PtrToStringAnsi((IntPtr)EPT_AH_GetDeviceSerial(device_index));
    }

    // Make the opened device the active device
    if (EPT_AH_SelectActiveDeviceByIndex(device_index) == 0)
    {
        String message = "Error selecting device: %s " +
            Marshal.PtrToStringAnsi((IntPtr)EPT_AH_GetLastError());
        MessageBox.Show(message);
        return 0;
    }

    // Register the read callback function
    RegisterCallBack();
    btnOpenDevice.Enabled = false;
    btnCloseDevice.Enabled = true;
    return 0;
}
```

### 5.4.2 Active Host Read Callback Function

The local callback function is populated. It resides in the active\_transfer.cs file. This function will be called from the Active Host dll. When the EPT Device has transferred data to the PC, the callback function will do something with the data and command.

```
// Actual callback function which will read messages coming from the EPT device
unsafe void EPTReadFunction (Int32 device_id, Int32 device_channel, byte command, byte payload,
{
    char*     message = (char *)data;

    // Select current device
    EPT_AH_SelectActiveDeviceByIndex(device_id);

    // Make sure we have data in the system
    if (message != null)
    {
        MessageBox.Show("Message " + Marshal.PtrToStringAnsi((IntPtr) message));
    }
}
```



## FPGA Development System User Manual

Because the callback function communicates directly with the dll and must pass pointers from the dll to the C# Windows Form, the Marshaling scheme must be used. Marshaling allows pointer variables created in the dll to be passed into the C#. It is an advanced topic and will not be covered in this manual.

### 5.4.3 Active Host Triggers

The user application can send a trigger to the FPGA by using the EPT\_AH\_SendTrigger() function. First, open the EPT device to be used with EPT\_AH\_OpenDeviceByIndex(). Call the function with the bit or bits to assert high on the trigger byte as the parameter. Then execute the function, the trigger bit or bits will momentarily assert high in the user code on the FPGA.

```
private void btnTrigger1_Click(object sender, EventArgs e)
{
    EPT_AH_SendTrigger((char) 1);
}
```

To detect a trigger from the FPGA, the user application must subscribe to the event created when the incoming trigger has arrived at the Read Callback function. The Read Callback must store the incoming trigger in a local variable. A switch statement is used to decode which event should be called to handle the incoming received data.

- TRIGGER\_IN
- TRANSFER\_IN
- BLOCK\_IN



```
unsafe void EPTReadFunction (Int32 device_id, Int32 device_channel, byte command, byte payload,
{
    byte* message = data;

    // Select current device
    EPT_AH_SelectActiveDeviceByIndex(device_id);

    //Add command and device_channel to the receive object
    EPTReceiveData.Command = ((command & 0x38) >> 3);
    EPTReceiveData.Address = device_channel;

    //Check if the command is Block Receive. If so,
    //use Marshalling to copy the buffer into the receive
    //object
    if (EPTReceiveData.Command == BLOCK_OUT_COMMAND)
    {
        EPTReceiveData.Length = data_size;
        EPTReceiveData.cBlockBuf = new Byte[data_size];

        Marshal.Copy(new IntPtr(message), EPTReceiveData.cBlockBuf, 0, data_size);
        //UpdateTextBlock();
    }
    else
    {
        EPTReceiveData.Payload = payload;
    }
    this.Invoke(new EventHandler(EPTParseReceive));
}

private void EPTParseReceive(object sender, System.EventArgs e)
{
    switch (EPTReceiveData.Command)
    {
        case TRIGGER_OUT_COMMAND:
            TriggerOutReceive();
            break;
        case TRANSFER_OUT_COMMAND:
            TransferOutReceive();
            break;
        case BLOCK_OUT_COMMAND:
            BlockOutReceive();
            break;
        default:
            break;
    }
}
```

The event handler function for the TRIGGER\_IN's uses a switch statement to determine which trigger was asserted and what to do with it.

```
public void Receive_Trigger_In(object sender, EventArgs e)
{
    switch (ept_data.Payload)
    {
        case 0x01:
            lLabelSwitch1.Text = "Switch 1\n Pressed";
            break;
        case 0x02:
            lLabelSwitch2.Text = "Switch 2\n Pressed";
            break;
        case 0x04:
            lLabelSwitch1.Text = "";
            lLabelSwitch2.Text = "";
            break;
    }
}
```

The receive callback method is complex, however, Earth People Technology has created several projects which implement callbacks. Any part of these sample projects can be copied and pasted into a user's project.

#### 5.4.4 Active Host Byte Transfers

The Active Host Byte Transfer EndTerm is designed to send/receive one byte to/from the EPT Device. To send a byte to the Device, the appropriate address must be selected for the Transfer module in the FPGA. Up to eight modules can be instantiated in the user code on the FPGA. Each module has its own address.

```
private void btnWriteByte_Click(object sender, EventArgs e)
{
    int ibyte, address_to_device;
    ibyte = Convert.ToInt32(tbNumBytes.Text);
    address_to_device = Convert.ToInt32(tbAddress.Text);
    EPT_AH_SendByte(address_to_device, (char)ibyte);
}
```

Use the function EPT\_AH\_SendByte() to send a byte to the selected module. First, open the EPT device to be used with EPT\_AH\_OpenDeviceByIndex(). Then add the address of the transfer module as the first parameter of the EPT\_AH\_SendByte() function. Enter the byte to be transferred in the second parameter. Then execute the function, the byte will appear in the ports of the Active Transfer module in the user code on the FPGA.

To transfer data from the FPGA Device, a polling technique is used. This polling technique is because the Bulk Transfer USB is a Host initiated bus. The Device will not transfer any bytes until the Host commands it to. If the Device has data to send to the Host in an asynchronous manner (meaning the Host did not command the Device to send data), the Host must periodically check the Device for data in its transmit FIFO. If data exists, the Host will command the Device to send its data. The received data is



## FPGA Development System User Manual

then stored into local memory and register bits are set that will indicate data has been received from a particular address.

To receive a byte transfer from the Active host dll, user code must subscribe to the event created when the incoming byte transfer has arrived at the Read Callback function. The Read Callback must store the incoming transfer payload and module address in a local memory block. A switch statement is used to decode which event should be called to handle the incoming received data. The event handler function will check for any bytes read for that address.

```
unsafe void EPTReadFunction (Int32 device_id, Int32 device_channel, byte command, byte payload,
{
    byte* message = data;

    // Select current device
    EPT_AH_SelectActiveDeviceByIndex(device_id);

    //Add command and device_channel to the receive object
    EPTReceiveData.Command = ((command & 0x38) >> 3);
    EPTReceiveData.Address = device_channel;

    //Check if the command is Block Receive. If so,
    //use Marshalling to copy the buffer into the receive
    //object
    if (EPTReceiveData.Command == BLOCK_OUT_COMMAND)
    {
        EPTReceiveData.Length = data_size;
        EPTReceiveData.cBlockBuf = new Byte[data_size];

        Marshal.Copy(new IntPtr(message), EPTReceiveData.cBlockBuf, 0, data_size);
        //UpdateTextBlock();
    }
    else
    {
        EPTReceiveData.Payload = payload;
    }
    this.Invoke(new EventHandler(EPTParseReceive));
}
```



## FPGA Development System User Manual

```
private void EPTParseReceive(object sender, System.EventArgs e)
{
    switch (EPTReceiveData.Command)
    {
        case TRIGGER_OUT_COMMAND:
            TriggerOutReceive();
            break;
        case TRANSFER_OUT_COMMAND:
            TransferOutReceive();
            break;
        case BLOCK_OUT_COMMAND:
            BlockOutReceive();
            break;
        default:
            break;
    }
}
```

The EventHandler function EPTParseReceive() is called by the Read Callback function. The EPTParseReceive() function will examine the command of the incoming byte transfer and determine which receive function to call.

```
public void TransferOutReceive()
{
    string WriteRcvChar = "";
    WriteRcvChar = String.Format("{0}", (int)EPTReceiveData.Payload);
    tbDataBytes.AppendText(WriteRcvChar + ' ');
    tbAddress.Text = String.Format("{0:x2}", (uint)System.Convert.ToInt32(EPTReceiveData.Address.ToString())
}
```

For our example project, the TransferOutReceive() function writes the Transfer byte received to a text block. The receive callback method is complex, however, Earth People Technology has created several projects which implement callbacks. Any part of these sample projects can copied and pasted into a user's project.

### 5.4.5 Active Host Block Transfers

The Active Host Block Transfer is designed to transfer blocks of data between Host and FPGA and vice versa through the Block EndTerm. This allows buffers of data to be transferred with a minimal amount of code. The Active Host Block module (in the User Code) is addressable, so up to eight individual modules can be instantiated and separately addressed. The length of the block to be transferred must also be specified. The Block EndTerm is limited to 1 to 256 bytes.

To send a block, first, open the EPT device to be used with EPT\_AH\_OpenDeviceByIndex(). Next, use the EPT\_AH\_SendBlock() function to send the block. Add the address of the transfer module as the first parameter. Next, place the pointer to the buffer in the second parameter of EPT\_AH\_SendBlock(). Add the length



of the buffer as the third parameter. Then execute the function, the entire buffer will be transferred to the USB chip. The data is available at the port of the Active Block module in the user code on the FPGA.

```
public unsafe void BlockCompare(object data)
{
    int BlockAddress = (int)data;
    byte[] cBuf = new Byte[device[BlockAddress].Length];

    if ((device[BlockAddress].Repititions > 0) &
        !device[BlockAddress].TransferPending & !BlockTransferStop)
    {
        device[BlockAddress].TransferPending = true;
        Buffer.BlockCopy(block_8_in_payload, 0, cBuf, 0,
                         device[BlockAddress].Length);
        fixed (byte* pBuf = cBuf)
        {
            EPT_AH_SendBlock(device[BlockAddress].Address,
                              (void*)pBuf, (uint)device[BlockAddress].Length);
        }
        Thread.Sleep(1);
        EPT_AH_SendTransferControlByte((char)2, (char)2);
        Thread.Sleep(1);
        EPT_AH_SendTrigger((char)128);
        Thread.Sleep(1);
        EPT_AH_SendTransferControlByte((char)2, (char)0);

        if (BlockTransferInfinite)
            device[BlockAddress].Repititions = 1;
        else
            device[BlockAddress].Repititions--;
    }
}
```

To receive a block transfer from the FPGA Device, a polling technique is used by the Active Host dll. This is because the Bulk Transfer USB is a Host initiated bus. The Device will not transfer any bytes until the Host commands it to. If the Device has data to send to the Host in an asynchronous manner (meaning the Host did not command the Device to send data), the Host must periodically check the Device for data in its transmit FIFO. If data exists, the Host will command the Device to send its data. The received data is then stored into local memory and register bits are set that will indicate data has been received from a particular address. The receive callback function is then called from the Active Host dll. This function start a thread to do something with the block data.

To receive a byte transfer from the callback function, user code must subscribe to the event created when the incoming byte transfer has arrived at the Read Callback function. The Read Callback must store the incoming transfer payload and module



address in a local memory block. A switch statement is used to decode which event should be called to handle the incoming received data. The event handler function will check for any bytes read for that address.

```
unsafe void EPTReadFunction (Int32 device_id, Int32 device_channel, byte command, byte payload,
{
    byte* message = data;

    // Select current device
    EPT_AH_SelectActiveDeviceByIndex(device_id);

    //Add command and device_channel to the receive object
    EPTReceiveData.Command = ((command & 0x38) >> 3);
    EPTReceiveData.Address = device_channel;

    //Check if the command is Block Receive. If so,
    //use Marshalling to copy the buffer into the receive
    //object
    if (EPTReceiveData.Command == BLOCK_OUT_COMMAND)
    {
        EPTReceiveData.Length = data_size;
        EPTReceiveData.cBlockBuf = new Byte[data_size];

        Marshal.Copy(new IntPtr(message), EPTReceiveData.cBlockBuf, 0, data_size);
        //UpdateTextBlock();
    }
    else
    {
        EPTReceiveData.Payload = payload;
    }
    this.Invoke(new EventHandler(EPTParseReceive));
}

private void EPTParseReceive(object sender, System.EventArgs e)
{
    switch (EPTReceiveData.Command)
    {
        case TRIGGER_OUT_COMMAND:
            TriggerOutReceive();
            break;
        case TRANSFER_OUT_COMMAND:
            TransferOutReceive();
            break;
        case BLOCK_OUT_COMMAND:
            BlockOutReceive();
            break;
        default:
            break;
    }
}
```

The EventHandler function EPTParseReceive() is called by the Read Callback function. The EPTParseReceive() function will examine the command of the incoming byte transfer and determine which receive function to call.

```
public void Receive_Block_In(object sender, EventArgs e)
{
    device[ept_data.Address].TransferPending = false;
    Thread.Sleep(5);
    if (device[ept_data.Address].ContinuosCountTest == false)
    {
        Thread t = new Thread(new ParameterizedThreadStart(BlockCompare));
        t.Start(ept_data.Address);
    }
    if (device[ept_data.Address].Repititions == 0)
    {
        Thread u = new Thread(new ParameterizedThreadStart(Display_Block_In));
        u.Start(BlockCount);
    }
    else if (BlockTransferInfinite | device[ept_data.Address].ContinuosCountTest)
    {
        if ((BlockCount % 100) == 0)
        {
            Thread u = new Thread(new ParameterizedThreadStart(Display_Block_In));
            u.Start(BlockCount);
        }
    }
}
```

For our example project, the Receive\_Block\_In() function writes the Transfer block received to a text block. The receive callback method is complex, however, Earth People Technology has created several projects which implement callbacks. Any part of these sample projects can copied and pasted into a user's project.

## 6 Assembling, Building, and Executing a .NET Project on the PC

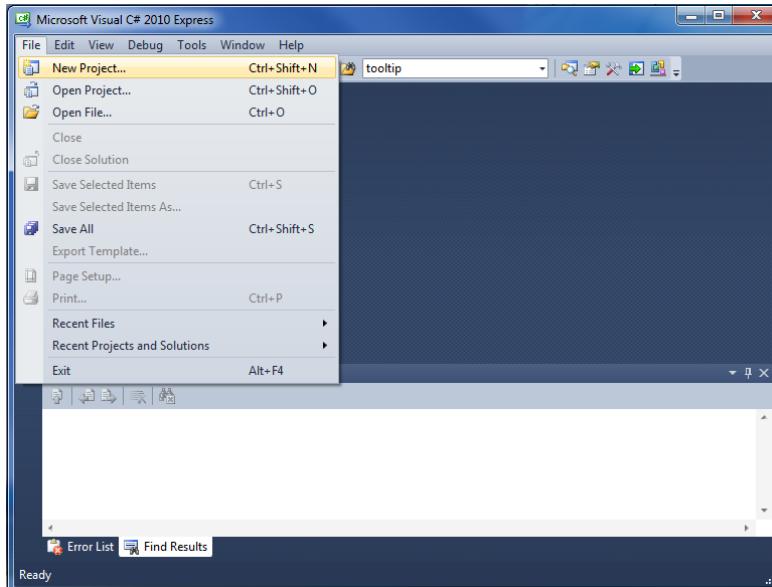
The Active Host Application DLL is used to build a custom standalone executable on the PC that can perform Triggers and Transfer data to/from the EPT-4CE6-AF-D1. A standalone project can be range from a simple program to display and send data from the user to/from the Arduino Due. Or it can more complex to include receiving data, processing it, and start or end a process on the Arduino. This section will outline the procedures to take an example project and Assemble it, Build it, and Execute it.

This guide will focus on writing a Windows Forms application using the C# language for the Microsoft Visual Studio with .NET Framework. This is due to the idea that beginners can write effective Windows applications with the C# .NET Framework. They can focus on a subset of the language which is very similar to the C language. Anything that deviates from the subset of the C language, presented as in the Arduino

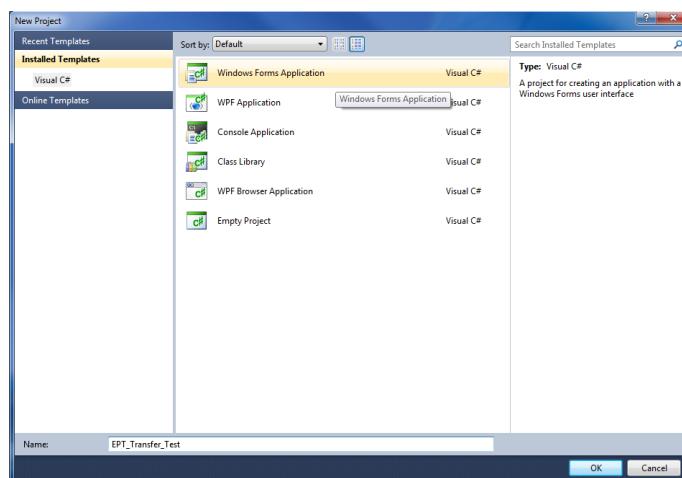
implication (such as events and controls), will be explained as the explanation progresses. Any language can be used with the Active Host Application DLL.

## 6.1 Creating a Project

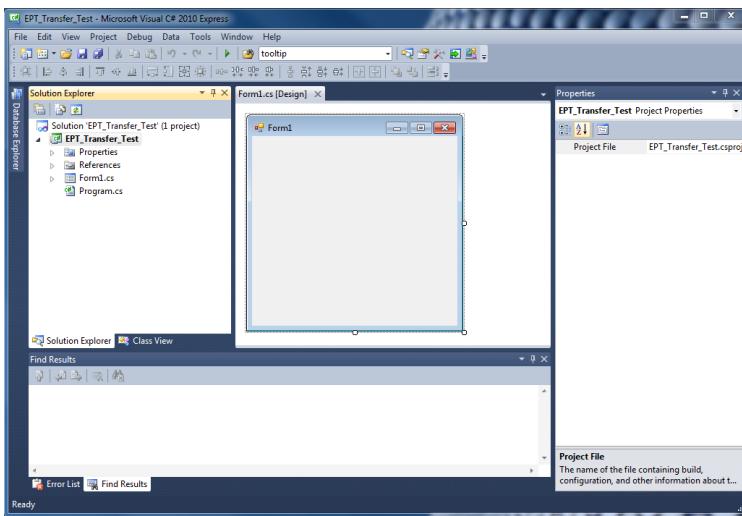
Once the application is installed, open it up. Click on File->New Project.



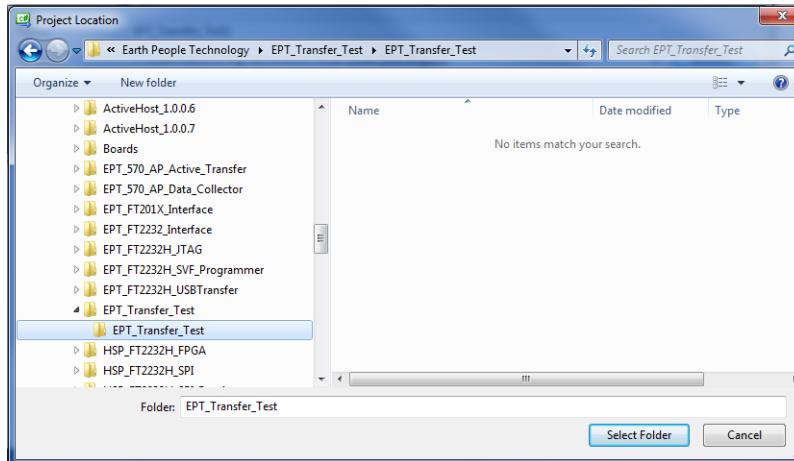
At the New Project window, select the Windows Forms Application. Then, at the Name: box, type in EPT\_Transfer\_Demo



The project creation is complete.



Save the project, go to File->Save as, browse to a folder to create EPT\_Transfer\_Demo folder. The default location is c:\Users\<Users Name>\documents\visual studio 2010\Projects.

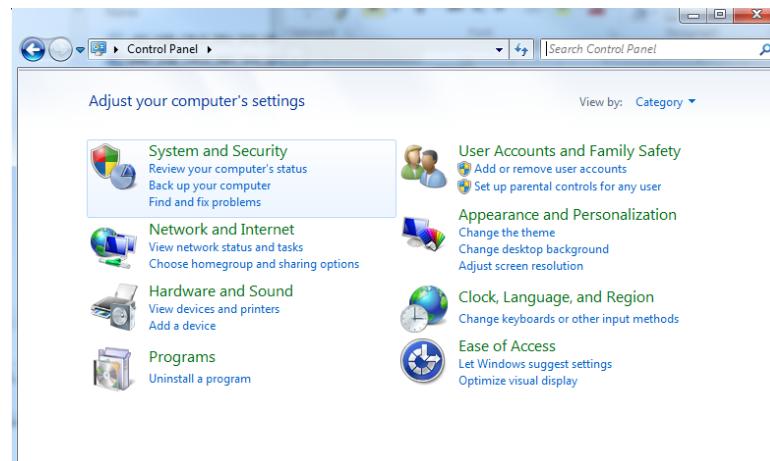


### 6.1.1 Setting up the C# Express Environment x64 bit

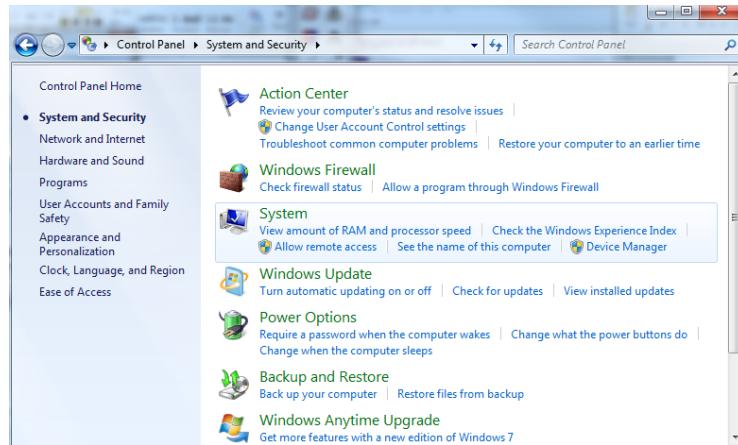
The project environment must be set up correctly in order to produce an application that runs correctly on the target platform. If your system supports 64 bit operation, perform the following steps. Otherwise if your system is 32 bit skip to the Section, Assembling Files into the Project. Visual C# Express defaults to 32 bit operation. If you are unsure if your system supports, you can check it by going to Start->Control Panel->System and Security->System



## FPGA Development System User Manual



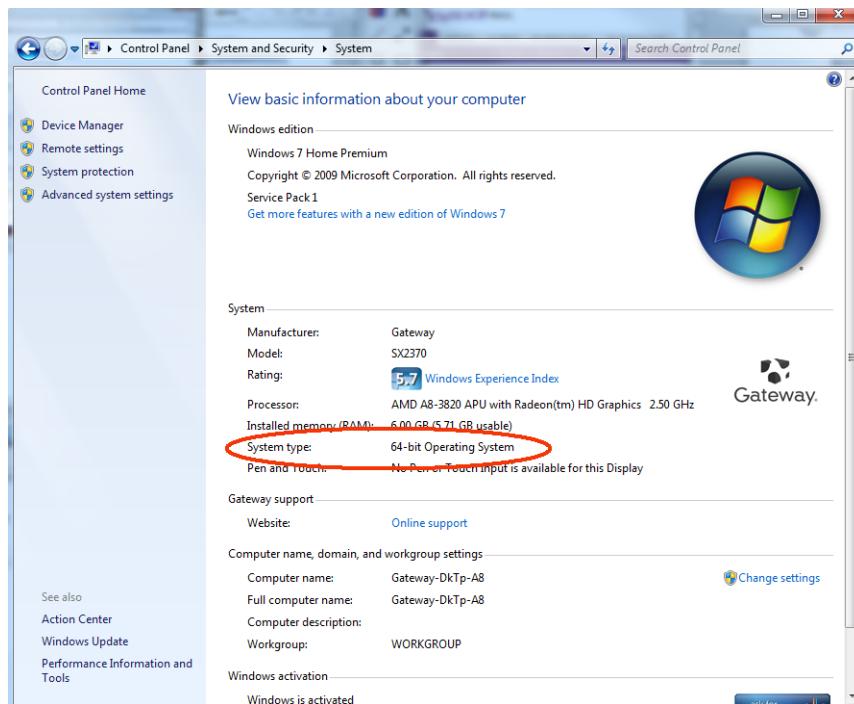
Click on System.



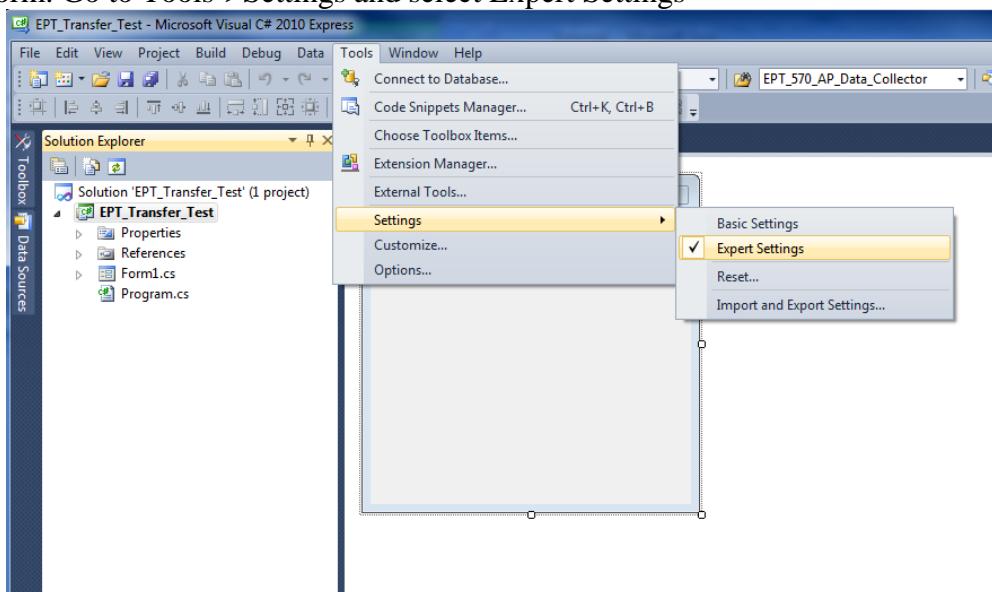
Check under System\System type:



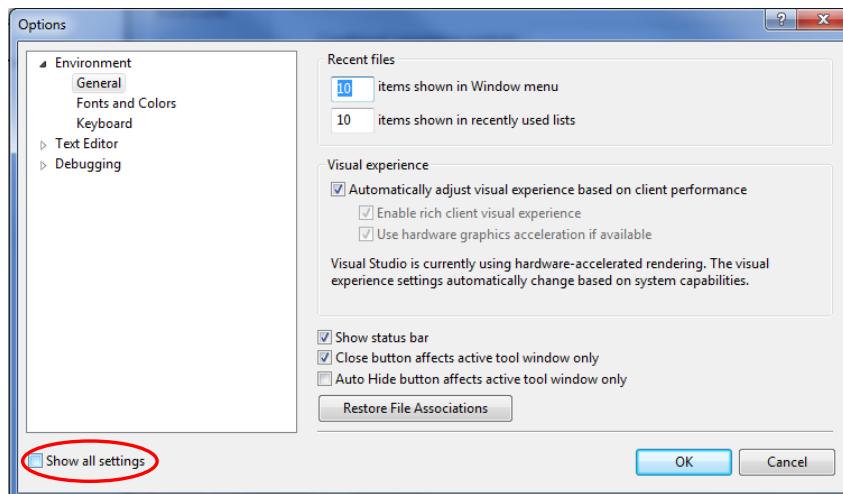
## FPGA Development System User Manual



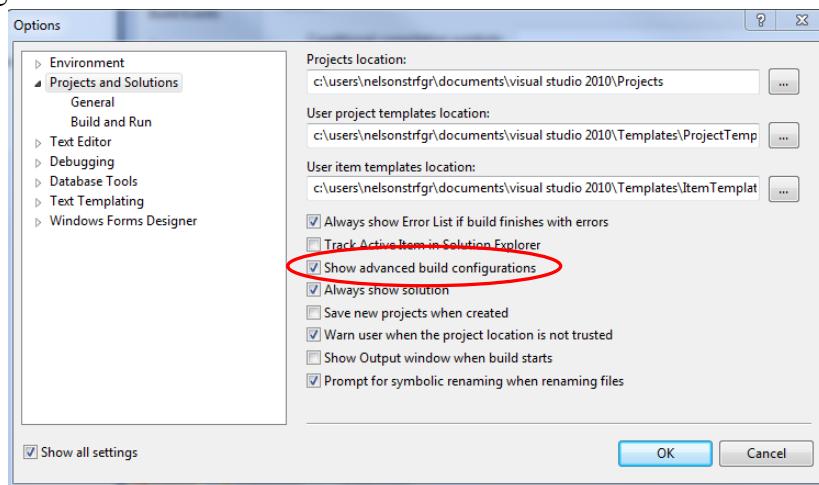
First, we need tell C# Express to produce 64 bit code if we are running on a x64 platform. Go to Tools->Settings and select Expert Settings



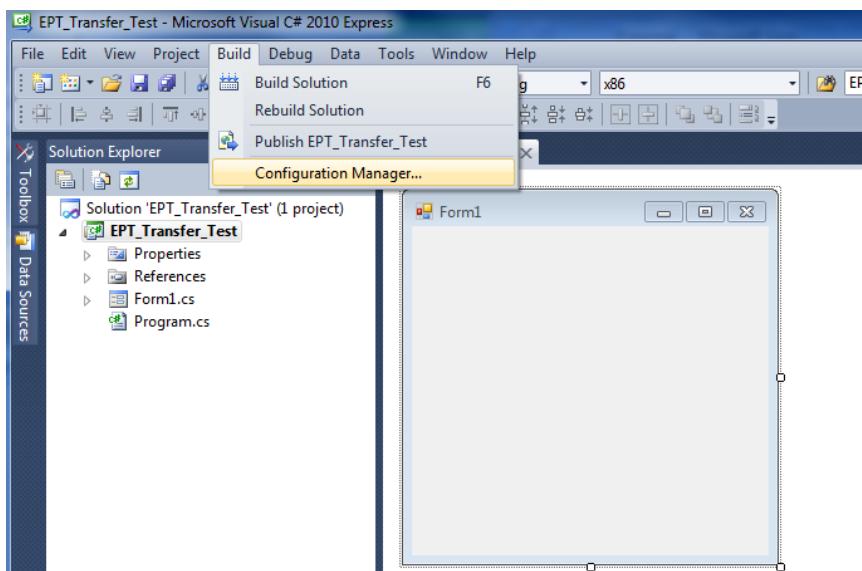
Go to Tools->Options, locate the “Show all settings” check box. Check the box.



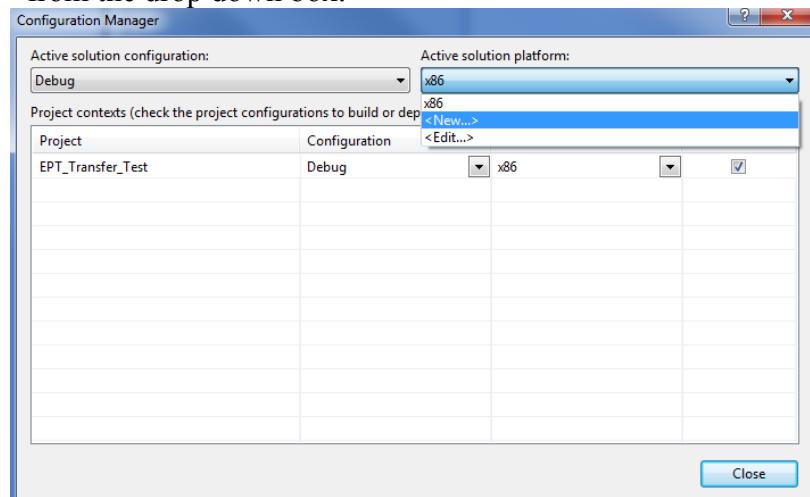
In the window on the left, go to “Projects and Solutions”. Locate the “Show advanced build configurations” check box. Check the box.



Go to Build->Configuration Manager.



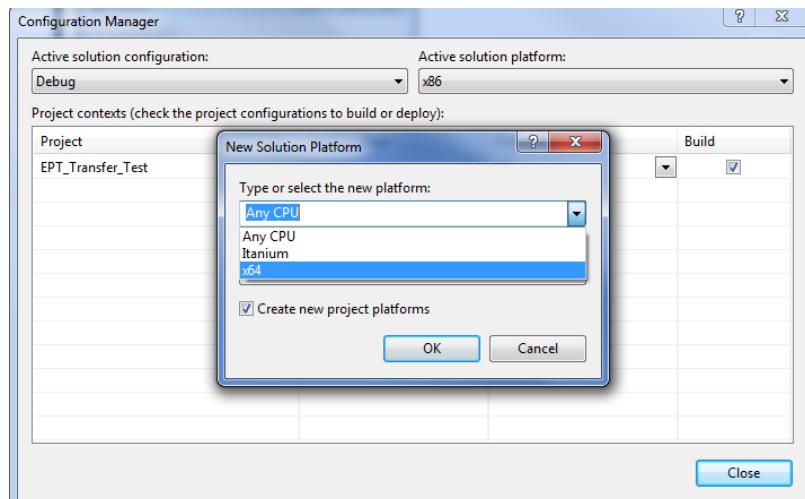
In the Configuration Manager window, locate the “Active solution platform:” label, select “New” from the drop down box.



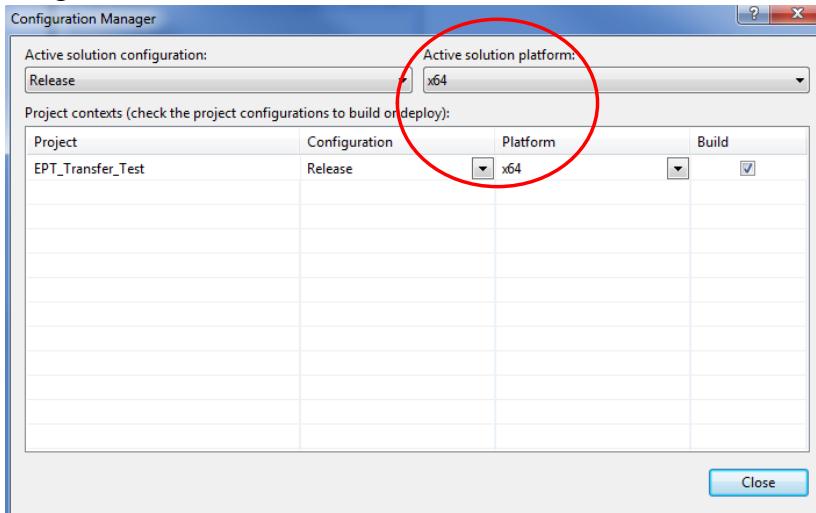
In the New Solution Platform window, click on the drop down box under “Type or select the new platform:”. Select “x64”.



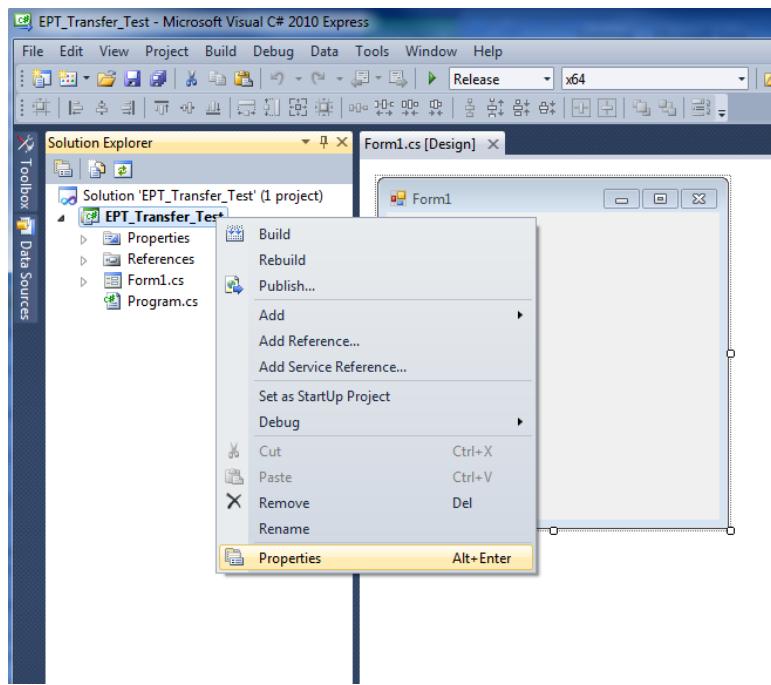
## FPGA Development System User Manual



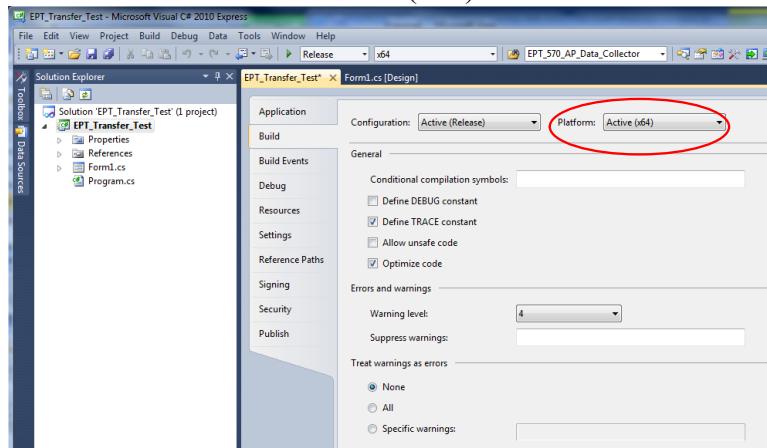
Click the Ok button. Verify that the “Active Solution Platform” and the “Platform” tab are both showing “x64”.



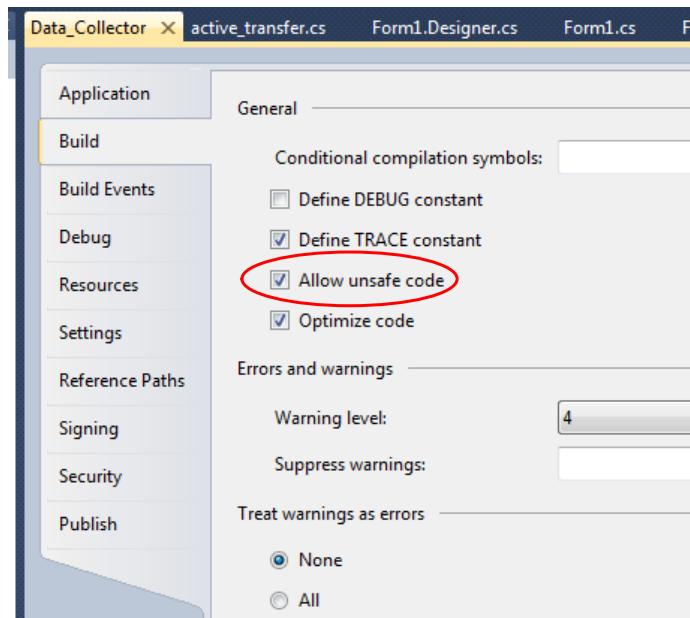
Also, select “Release” under “Active solution configuration”. Click Close.  
Then, using the Solution Explorer, you can right click on the project, select Properties and click on the Build tab on the right of the properties window.



Verify that the “Platform:” label has “Active (x64)” selected from the drop down box.



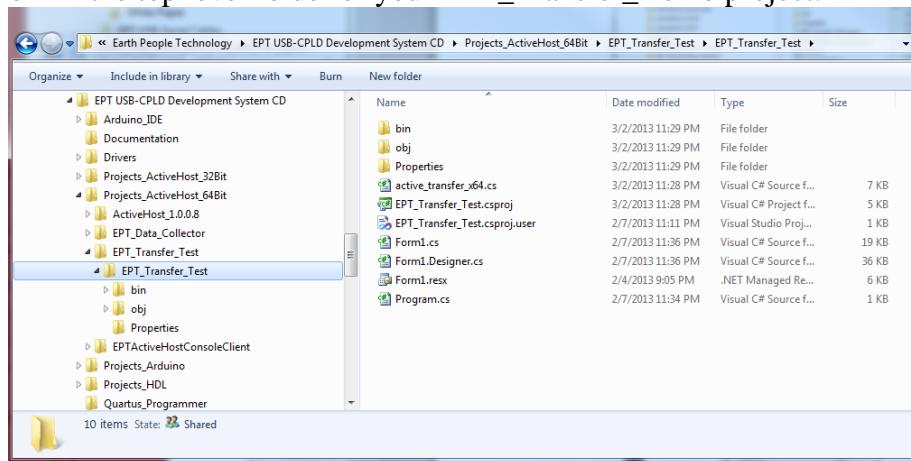
Next, unsafe code needs to be allowed so that C# can be passed pointer values from the Active Host. Click on the Build tab and locate the “Allow unsafe code” check box. Check the box



Click on the Save All button on the tool bar. The project environment is now setup and ready for the project files. Close the Project.

## 6.2 Assembling Files into the Project

Locate the EPT FPGA Development System DVD installed on your PC. Browse to the EPT\_Transfer\_Demo folder where the Project files reside (choose either the 32 bit or 64 bit version, depending on whether your OS is 32 or 64 bit), copy the \*.cs files, and install them in the top level folder of your EPT\_Transfer\_Demo project.

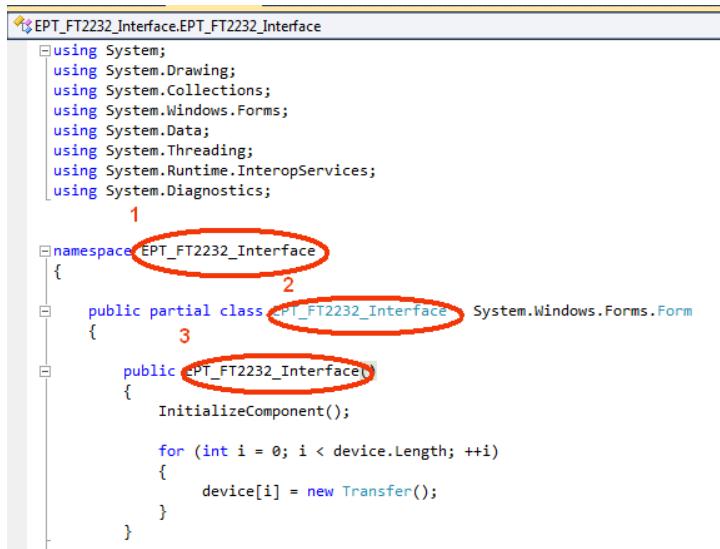


### 6.2.1 Changing Project Name

\*\*\*NOTE\*\*\*

If you named your project something other than EPT\_Transfer\_Demo, you will have to make changes to the \*.cs files above. This is because Visual C# Express links the project files and program files together. These changes can be made by modifying the following:

1. Change namespace of Form1.cs to new project name.
2. Change class of Form1.cs to new project name.
3. Change constructor of Form1.cs to new project name.



```

1 using System;
using System.Drawing;
using System.Collections;
using System.Windows.Forms;
using System.Data;
using System.Threading;
using System.Runtime.InteropServices;
using System.Diagnostics;

2 namespace EPT_FT2232_Interface
{
3     public partial class EPT_FT2232_Interface : System.Windows.Forms.Form
    {
        public EPT_FT2232_Interface()
        {
            InitializeComponent();

            for (int i = 0; i < device.Length; ++i)
            {
                device[i] = new Transfer();
            }
        }
    }
}

```

4. Change EPT\_Transfer\_Demo\_Load of Form1.cs to new <project name>\_Load

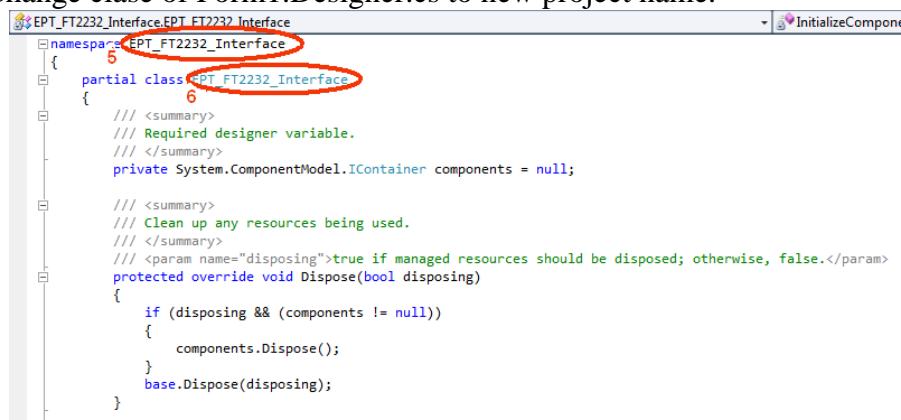
```

// Main object loader
private void EPT_FT2232_Interface_Load(object sender, System.EventArgs e)
{
    // Call the List Devices function
    ListDevices();
}

```

5. Change namespace of Form1.Designer.cs to new project name.

6. Change class of Form1.Designer.cs to new project name.



```

5 namespace EPT_FT2232_Interface
{
6     partial class EPT_FT2232_Interface : System.ComponentModel.IContainer
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }
    }
}

```

7. Change the this.Name and this.Text in Form1Designer.cs to new project name.
8. Change this.Load in Form1Designer.cs to include new project name.

```

this.Controls.Add(this.btnCloseDevice);
this.Controls.Add(this.btnOpenDevice);
this.Controls.Add(this.cmbDevList);
this.Controls.Add(this.LEDBox);
this.Controls.Add(this.gbTriggerOut);
this.Controls.Add(this.gbTransferControl);
this.Controls.Add(this.groupBox1);
this.Name = "EPT_FT2232_Interface";
this.Text = "EPT_FT2232_Interface";
this.Load += new System.EventHandler(this.EPT_FT2232_Interface_Load);
this.LEDBox.ResumeLayout(false);
this.LEDBox.PerformLayout();
this.gbTriggerOut.ResumeLayout(false);
this.gbTransferControl.ResumeLayout(false);
this.groupBox1.ResumeLayout(false);
this.groupBox1.PerformLayout();
this.ResumeLayout(false);
this.PerformLayout();

```

8

9. Change namespace in Program.cs to new project name
10. Change Application.Run() in Program.cs to new projectname.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;

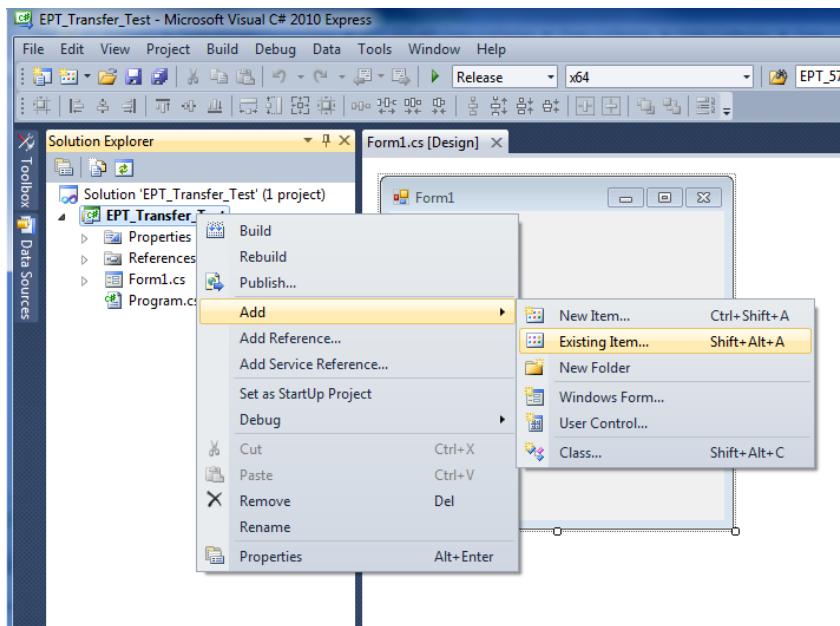
namespace EPT_FT2232_Interface
{
  static class Program
  {
    /// <summary>
    /// The main entry point for the application.
    /// </summary>
    [STAThread]
    static void Main()
    {
      Application.EnableVisualStyles();
      Application.SetCompatibleTextRenderingDefault(false);
      Application.Run(new EPT_FT2232_Interface());
    }
  }
}

```

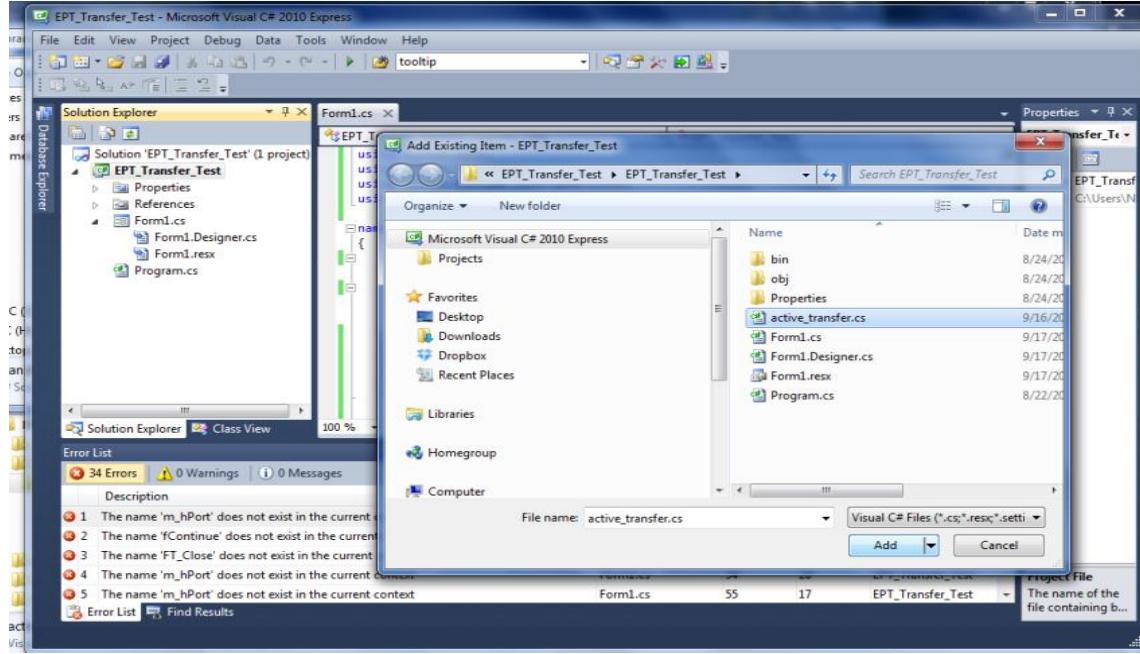
10

### 6.2.2 Add Files to Project

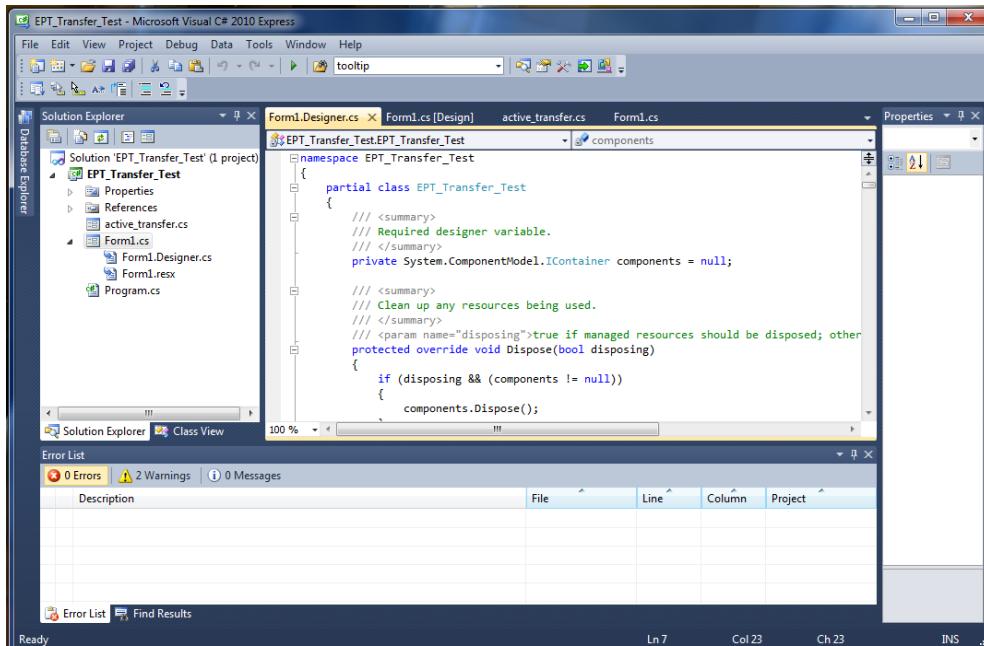
Open the EPT\_Transfer\_Demo project. Right click on the project in the Solutions Explorer. Select Add->Existing Item.



Browse to the EPT\_Transfer\_Demo project folder and select the active\_transfer\_xx.cs file (choose either the 32 bit or 64 bit version, depending on whether your OS is 32 or 64 bit). Click Add.



In the C# Express Solution Explorer, you should be able to browse the files by clicking on them. There should be no errors noted in the Error List box.

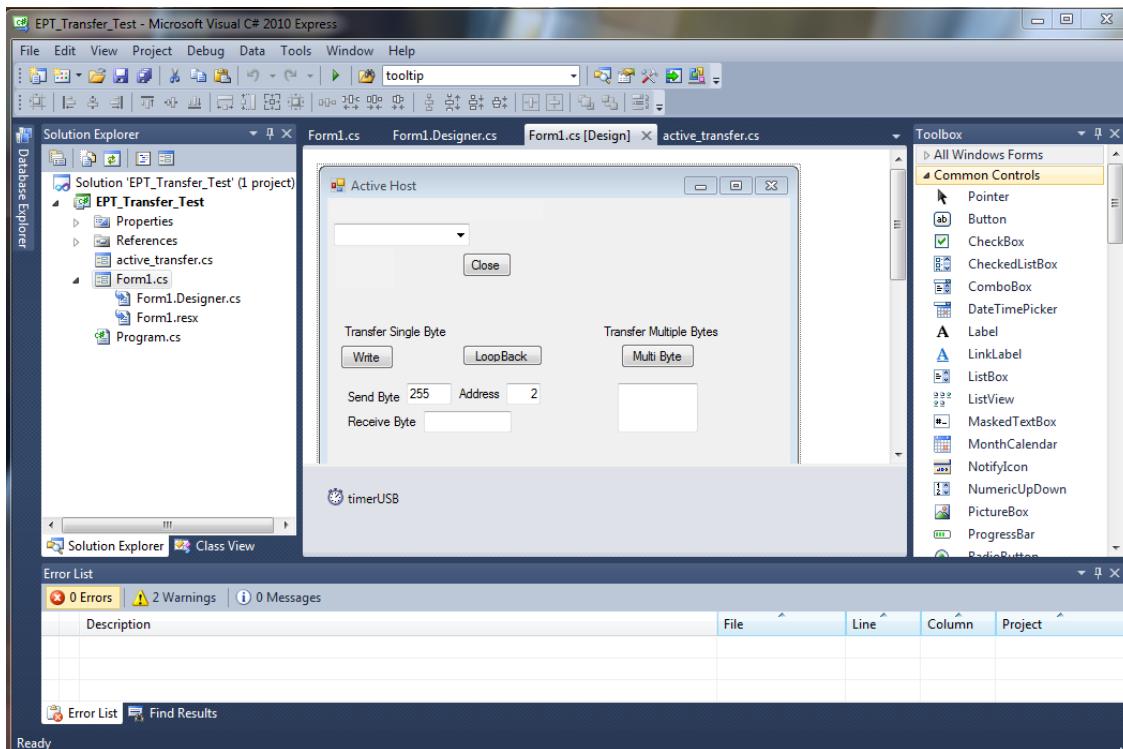


### 6.2.3 Adding Controls to the Project

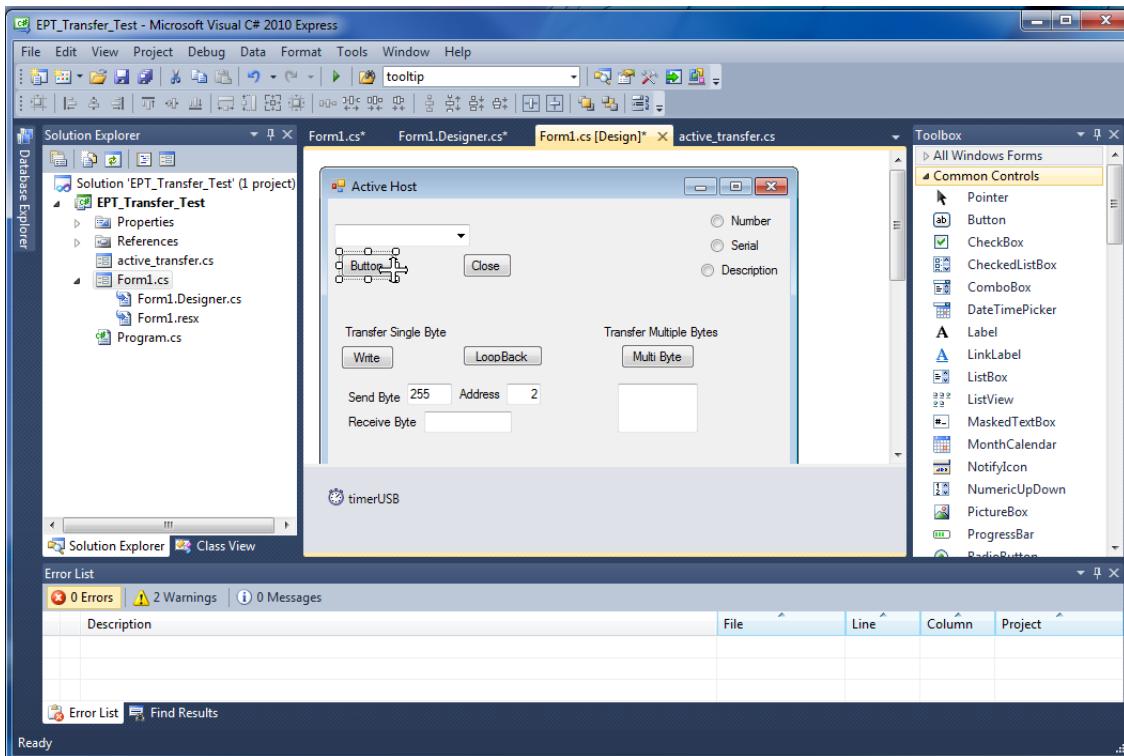
Although, the C# language is very similar to C Code, there are a few major differences. The first is C#.NET environment is event based. A second is C# utilizes classes. This guide will keep the details of these items hidden to keep things simple. However, a brief introduction to events and classes will allow the beginner to create effective programs.

Event based programming means the software responds to events created by the user, a timer event, external events such as serial communication into PC, internal events such as the OS, or other events. The events we are concerned with for our example program are user events and the timer event. The user events occur when the user clicks on a button on the Windows Form or selects a radio button. We will add a button to our example program to show how the button adds an event to the Windows Form and a function that gets executed when the event occurs.

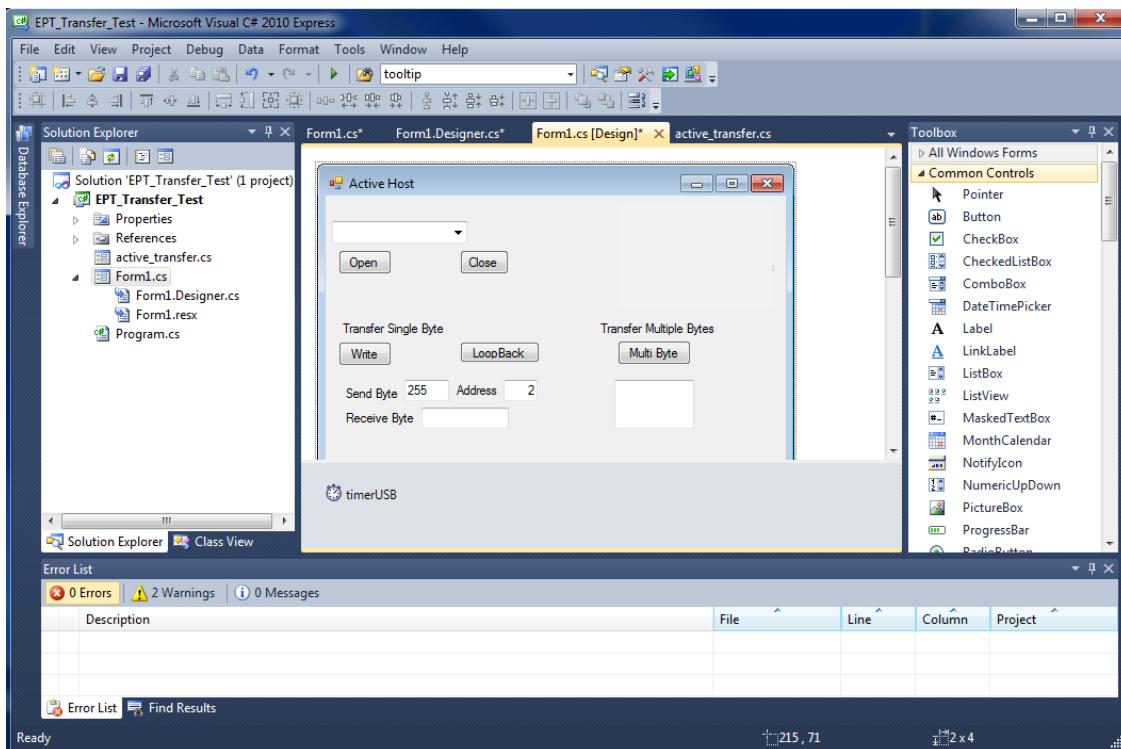
The easiest way to add a button to a form is to double click the Form1.cs in the Solution Explorer. Click on the  button to launch the Toolbox.



Locate the button on the Toolbox, grab and drag the button onto the Form1.cs [Design] and drop it near the top.



Go to the Properties box and locate the (Name) cell. Change the name to “btnOpenDevice”. Locate the Text cell, and change the name to Open.



Double click on the Open button. The C# Explorer will automatically switch to the Form1.cs code view. The callback function will be inserted with the name of the button along with “\_click” appended to it. The parameter list includes (object sender, System.EventArgs e). These two additions are required for the callback function to initiate when the “click” event occurs.

```
Private void btnOpenDevice_Click(object sender, System.EventArgs e)
```

There is one more addition to the project files. Double click on the Form1.Designer.cs file in the Solution Explorer. Locate the following section of code.

```
//  
// btnOpenDevice  
//  
this.btnOpenDevice.Location = new System.Drawing.Point(240, 13);  
this.btnOpenDevice.Name = "btnOpenDevice";  
this.btnOpenDevice.Size = new System.Drawing.Size(50, 23);  
this.btnOpenDevice.TabIndex = 2;  
this.btnOpenDevice.Text = "Open";  
this.btnOpenDevice.UseVisualStyleBackColor = true;  
this.btnOpenDevice.Click += new System.EventHandler(this.btnOpenDevice_Click);
```

This code sets up the button, size, placement, and text. It also declares the “System.EventHandler()”. This statement sets the click method (which is a member of



## FPGA Development System User Manual

the button class) of the btnOpenDevice button to call the EventHandler – btnOpenDevice\_Click. This is where the magic of the button click event happens.

```
private void btnOpenDevice_Click(object sender, EventArgs e)
{
    //Open the Device
    OpenDevice();
}

private void btnCloseDevice_Click(object sender, EventArgs e)
{
    if (EPT_AH_CloseDeviceByIndex(device_index) != 0)
    {
        btnBlkCompare8.Enabled = false;
        btnBlkCompare16.Enabled = false;
        btnTrigger1.Enabled = false;
        btnTrigger2.Enabled = false;
        btnTrigger3.Enabled = false;
        btnTrigger4.Enabled = false;
        btnLEDReset.Enabled = false;
    }
    btnOpenDevice.Enabled = true;
    btnCloseDevice.Enabled = false;
}
```

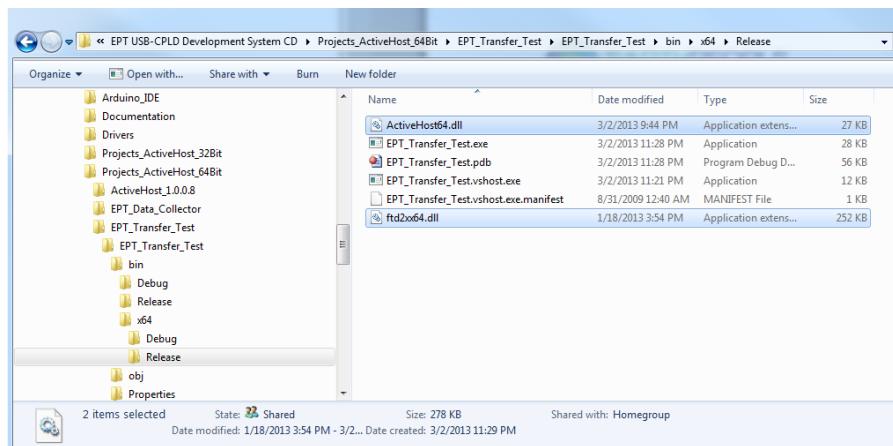
When btnOpenDevice\_Click is called, it calls the function “OpenDevice()”. This function is defined in the dll and will connect to the device selected in the combo box. This is a quick view of how to create, add files, and add controls to a C# project. The user is encouraged to spend some time reviewing the online tutorial at <http://www.homeandlearn.co.uk/csharp/csharp.html> to become intimately familiar with Visual C# .NET programming. In the meantime, follow the examples from the Earth People Technology to perform some simple reads and writes to the EPT USB-FPGA Development System.

### 6.2.4 Adding the DLL's to the Project

Locate the EPT FPGA Development System DVD installed on your PC. Browse to the Projects\_ActiveHost folder (choose either the 32 bit or 64 bit version, depending on whether your OS is 32 or 64 bit). Open the Bin folder, copy the following files:

- ActiveHostXX.dll
- ftd2xxXX.dll

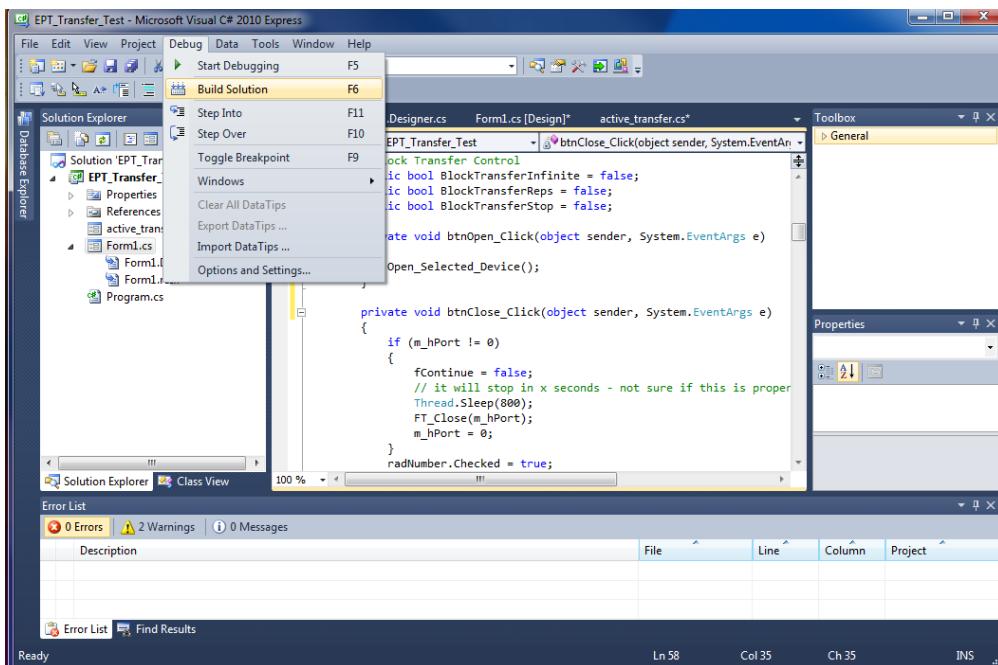
and install them in the bin\x64\x64 folder of your EPT\_Transfer\_Demo project.



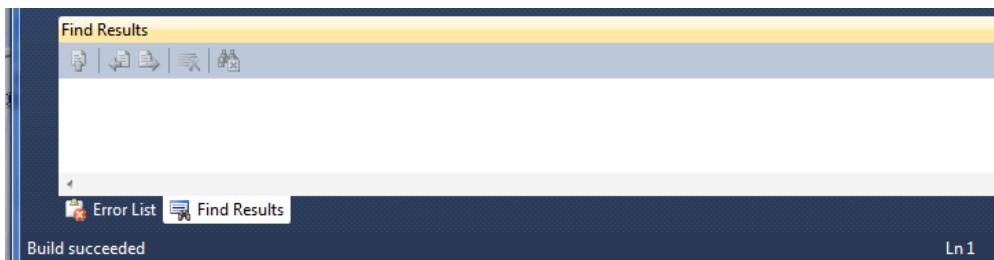
Save the project.

### 6.2.5 Building the Project

Building the EPT\_Transfer\_Demo project will compile the code in the project and produce an executable file. To build the project, go to Debug->Build Solution.

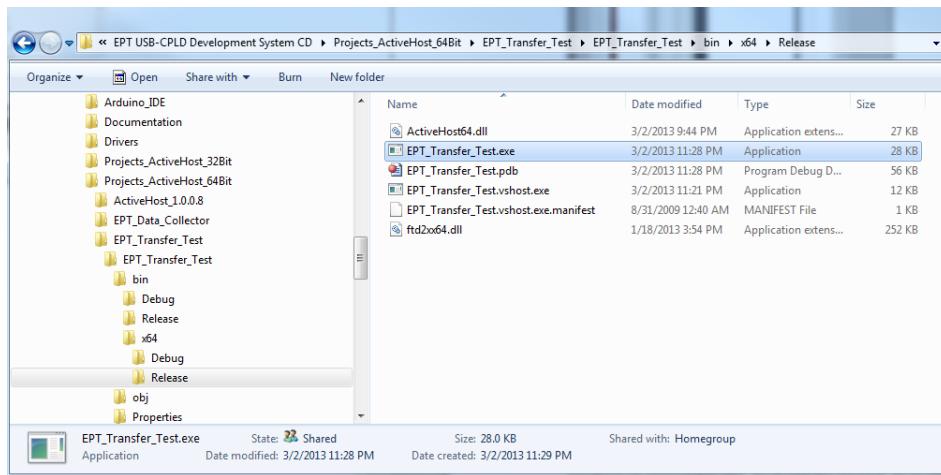


The C# Express compiler will start the building process. If there are no errors with code syntax, function usage, or linking, then the environment responds with “Build Succeeded”.



### 6.2.6 Testing the Project

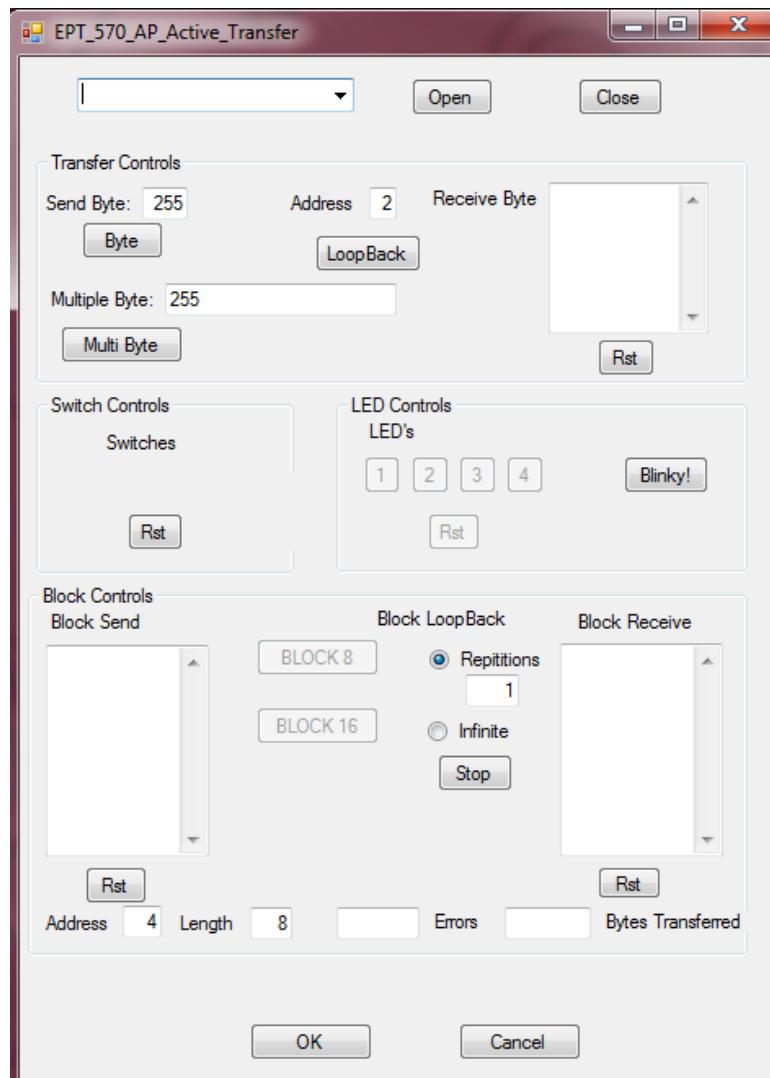
Once the project has been successfully built, it produces an \*.exe file. The file will be saved in the Release or Debug folders.



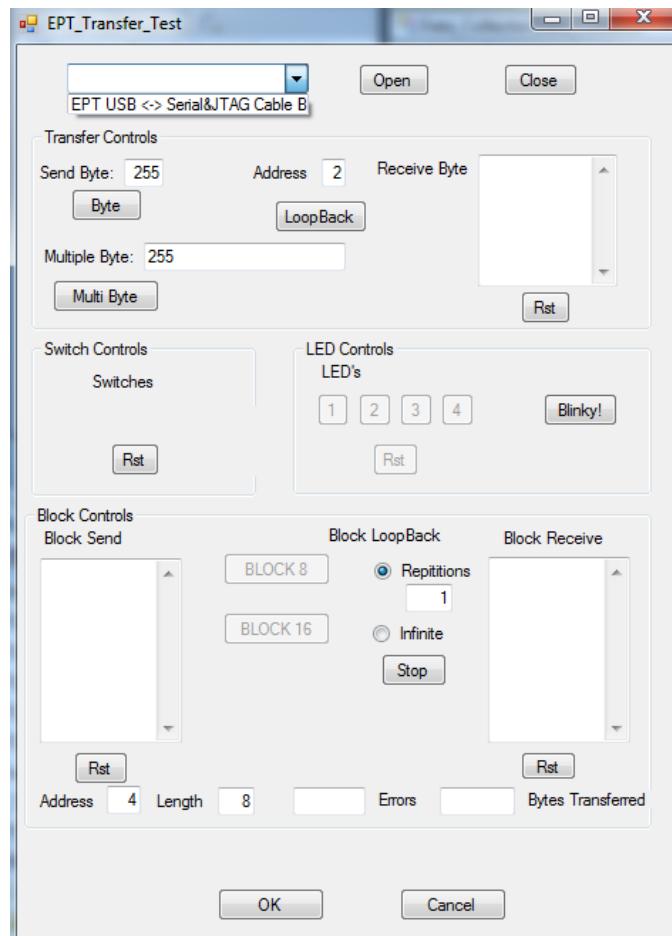
The EPT\_Transfer\_Text.exe file can now be tested using the EPT-4CE6-AF-D1 board. To test the file, connect the EPT-4CE6-AF-D1 to the Windows PC using Type A to Type Mini B USB cable. Make sure the driver for the board loads. If the USB driver fails to load, the Windows OS will indicate that no driver was loaded for the device. Go to the folder where the EPT\_Transfer\_Text.exe file resides, and double click on the file. The application should load with a Windows form.



## FPGA Development System User Manual



With the application loaded, select the USB-FPGA board from the dropdown combo box and click on the “Open” button.

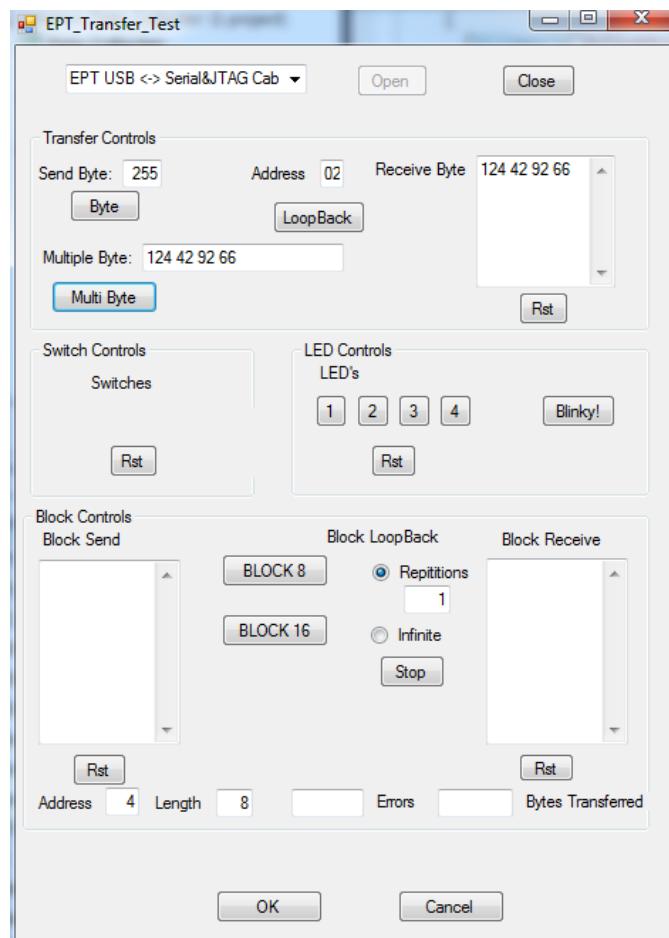


Click on one of the LED buttons in the middle of the window. The corresponding LED on the EPT-4CE6-AF-D1 board should light up.

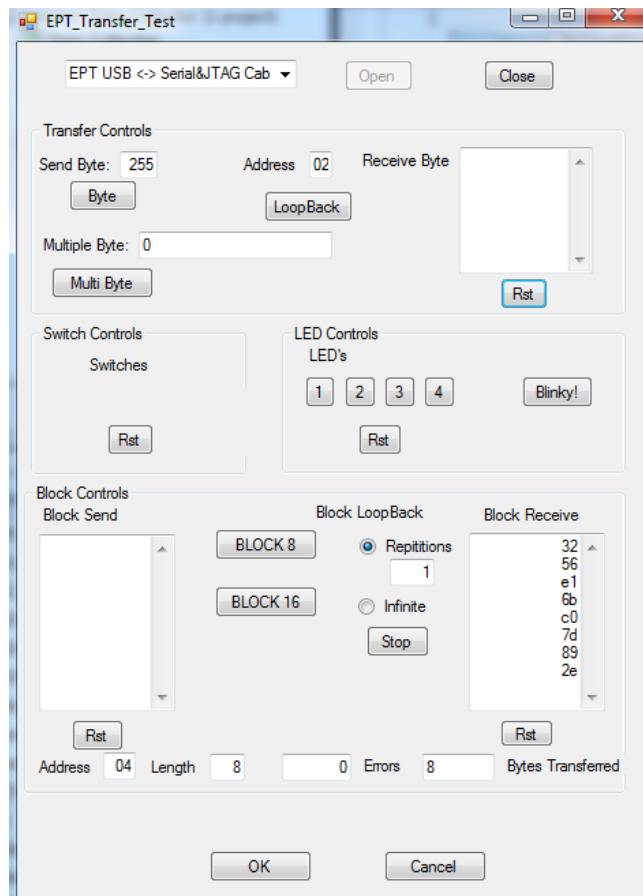
To exercise the Single Byte Transfer EndTerm, click the “LoopBack” button in the Transfer Controls group. Type in several numbers separated by a space and less 256 into the Multiple Byte textbox. Then hit the Multi Byte button. The numbers appear in the Receive Byte textbox.



## FPGA Development System User Manual



To exercise the Block Transfer EndTerm, click the “Block8” or “Block16” button in the Block Controls group. A pre-selected group of numbers appear in the Block Receive textbox.



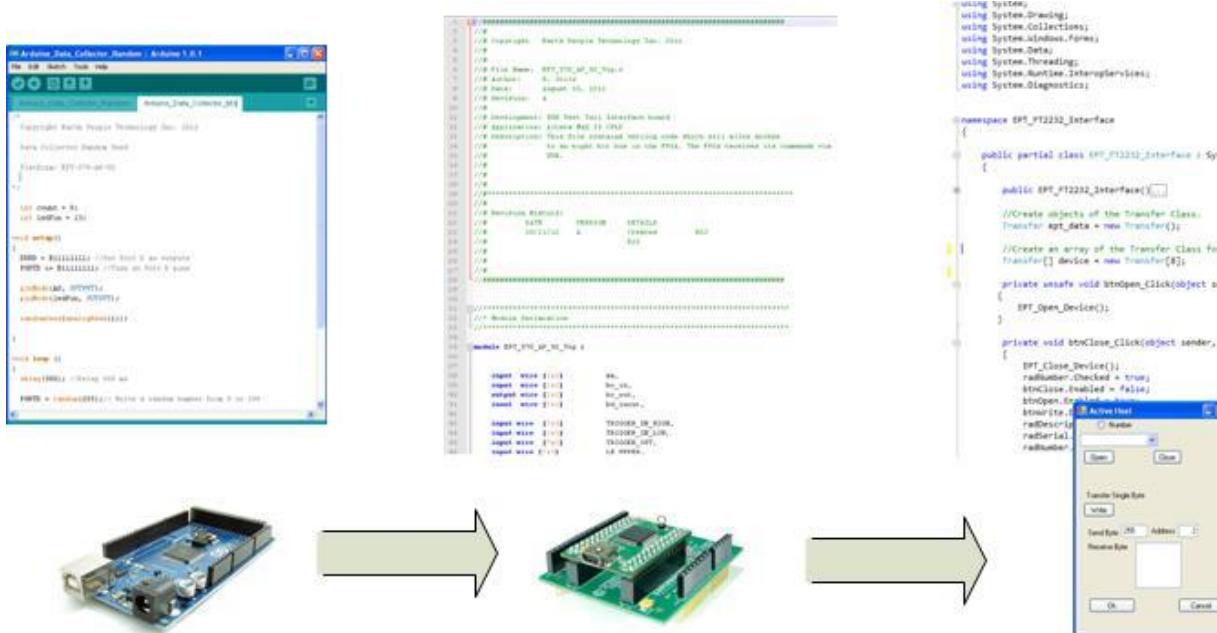
Press the PCB switches on the EPT-4CE6-AF-D1 to view the Switch Controls in action.

## 7 FPGA Development Process

There is no standard for developing embedded electronics. The best method is the one that works for the user. These methods can range from a top down approach where the design is written down first and all code is written, then compile, execute and test. Or a bottom up approach can be pursued where a small piece of the project is assembled and verified (i.e. I2C communication to a sensor). Then the next piece is assembled and verified (i.e. collect sensor data in a storage buffer) and connected to the first. And so on, until the whole design is complete. Or, you could use any combination of these two extremes.

### 7.1 Designing a Simple Data Collection Sampler

The Data Collection Sampler is a very simple introductory project that will guide the user in the creation of an overall design using the Arduino Programming Language, Verilog HDL, and C# Language. These elements will run on the Arduino Platform, EPT-4CE6-AF-D1 FPGA, and a Windows 7 PC respectively.



The first order of business is to layout the design. Start with the Arduino, and create a simple bit output using a random number generator. Next, use the EPT Active Transfer Library to create a byte transfer module to read the byte from the Arduino and send it to the Host PC. Finally, use EPT Active Host to accept the byte transfer from EPT Active Transfer, and display in a textbox. This is just the hierarchical system level design. In the following sections, we will fill in the above blocks.

### 7.1.1 The Arduino Microcontroller Board

Using the features and capabilities of the Arduino development system, the user will develop the source code using the “Wiring” programming language and download the resulting binary code from the Processing development environment to the Flash memory of the microcontroller.

### 7.1.2 Create Data Generator

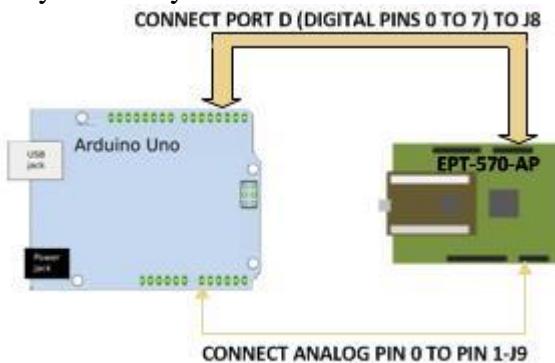
To keep the design simple, no external data source will be used. We will create a data source using the Arduino, then transmit this data to the EPT-4CE6-AF-D1 board. To create the data source, we will use the random() function. This function generates pseudo random numbers from a seed value. We will give the randomSeed() function a fairly random input using the value from the analogRead(). This will give different values every time the random() function is called. We will limit the random number output from the function to 8 bits. The random() function will be called once per iteration of the loop() function.

The randomSeed() function must be called during the setup() function. It takes as input parameter the output of the Analog Pin 1. The output of this Pin 1 will have a small amount of random noise on it. Because of this noise, the randomSeed() function will produce a different seed every time the sketch is initialized.

```
void setup()
{
  randomSeed(analogRead(1));
}
```

### 7.1.3 Select I/O's for Fast Throughput on Arduino

An 8 bit port is used to connect the 8 bit byte from the random function output to the input of the EPT-4CE6-AF-D1. There is also a one bit control line which will be used to inform the FPGA that a byte is ready to be written to the USB.



Each port is controlled by three registers, which are also defined variables in the Arduino language. The DDR register, determines whether the pin is an INPUT or OUTPUT. The PORT register controls whether the pin is HIGH or LOW, and the PIN register reads the state of INPUT pins set to input with pinMode(). The maps of the ATmega328 chips show the ports.

DDR and PORT registers may be both written to, and read. PIN registers correspond to the state of inputs and may only be read.

**PORTD** maps to Arduino digital pins 0 to 7

DDRD - The Port D Data Direction Register - read/write

PORTD - The Port D Data Register - read/write

PIND - The Port D Input Pins Register - read only



## FPGA Development System User Manual

The ports and pins for the Data Collection Sampler project must be initialized in the setup() function. The setup function will only run once, after each powerup or reset of the Arduino board.

```
int ledPin = 13;

void setup()
{
    DDRD = B11111111; //Set Port D as outputs
    PORTD &= B11111111; //Turn on Port D pins

    pinMode(A0, OUTPUT);

}
```

After the setup() function executes, the PORTD is ready to be assigned the results of our random() function. And the A0 pin will be used to latch the value on PORTD pins into the FPGA.

### 7.1.4 Coding the Arduino Data Sampler

Now that we have the data generator and the ports defined, we can add some delays in the loop() function and make a simulated data collector. Because Start and Stop buttons will be added to the C# Windows Form, the Data Collector code will need to monitor a single pin output from the EPT-4CE6-AF-D1. This output pin (from the EPT-4CE6-AF-D1) becomes an input to the Arduino and is used in conditional switch.

```
void loop ()
{
    //Sample the Start/Stop switch
    //from the EPT-570-AP
    startStopBit = digitalRead(inPin8);

    delay(500); //Delay 500 ms

    if(startStopBit)
    {
```

This code will sample the Start/Stop switch which is an output from the EPT-4CE6-AF-D1 on J10 PIN 1. On the Arduino, this is PIN 8 of the Digital pins. Each iteration of the loop() function, the startStopBit variable stores the state of DigitalPin8. Then, a delay of 500 milliseconds is added. The delay() function pauses the program for the amount of time (in milliseconds) specified as parameter. Next, the startStopBit is checked with a conditional switch. If the bit is set, the conditional branch is entered and the random number is sent to the EPT-4CE6-AF-D1. If the bit is not set, the end of the loop() function is reached and it branches to the top of the loop().



## FPGA Development System User Manual

We will also add an LED Pin that will blink so that we can have a visual indication that the project is working.

We want to add a delay so that the data from the generated displays on the Windows PC long enough for our eyes to verify that the data is updating correctly. This delay should be one second in total. So, the data will change then stay stable in the textbox for one second before changing again.

For the LED to blink correctly, it should turn on, delay for half a second then turn off and delay for half a second. If we don't use half second intervals for the LED blink, the LED will appear to not change at all. It will look like it stays on all the time or off all the time.

So, the code looks like this:



```
/*
Copyright Earth People Technology Inc. 2012

Data Collector Random Seed

Platform: EPT-570-AP-U2

*/
int startStopBit = 0;
int count = 0;
int ledPin = 13;
int inPin8 = 8;

void setup()
{
    DDRD = B1111111; //Set Port D as outputs
    PORTD &= B1111111; //Turn on Port D pins

    pinMode(A0, OUTPUT);
    pinMode(ledPin, OUTPUT);
    pinMode(inPin8, OUTPUT);

    randomSeed(analogRead(1));
}

void loop ()
{
    //Sample the Start/Stop switch
    //from the EPT-570-AP
    startStopBit = digitalRead(inPin8);

    delay(500); //Delay 500 ms

    if(startStopBit)
    {
        // Write a random number from 0 to 299
        //to the input of the EPT-570-AP
        PORTD = random(255);
        //Set the Write Enable Pin High
        digitalWrite(A0, HIGH);
    }
}
```

```

//Set the LED Pin High
digitalWrite(ledPin, HIGH);

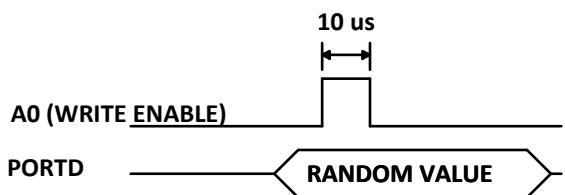
delay(500); //Delay 500 ms

//Set the LED Pin Low
digitalWrite(ledPin, LOW);
//Set teh Write Enable Pin Low
digitalWrite(A0, LOW);
}

}

```

Notice that PORTD equals the return of random(255). The parameter passed to the random() function is the maximum decimal value of the return value. In our case we want the maximum value to be an 8 bit value, B1111111 = 0xff = 255(decimal). Also, note that the A0 write enable signal for the FPGA has back to back instructions turning it on then off immedately. Because the ATMega328 chip takes approximately 160 clock cycles to execute the digitalWrite() function and affect the Pin at A0, this produces a write enable pulse of 10 microseconds.



The RANDOM VALUE will be stable before the A0(WRITE ENABLE) asserts thus guaranteeing a successful transfer of data from Arduino to FPGA.

### 7.1.5 Building Arduino Project

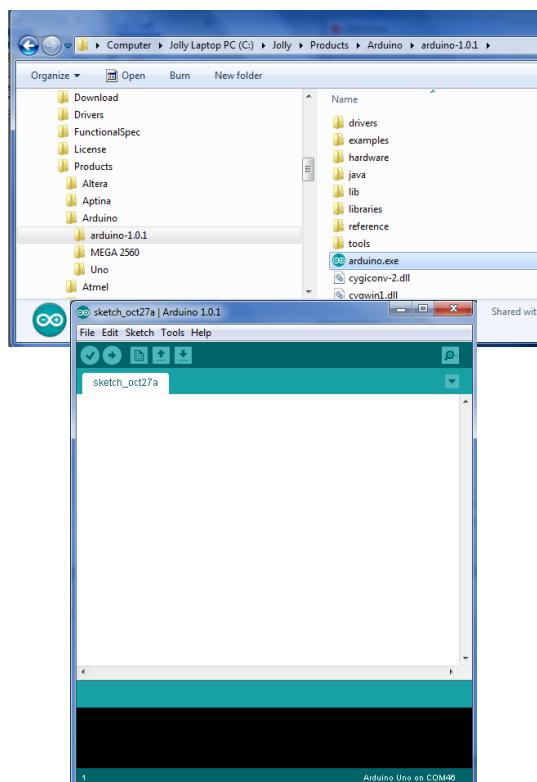
Building the Arduino project is the process of converting (compiling) the code you just wrote into machine level code that the processor can understand. The Arduino IDE is the software tool that does the compiling. The machine level code is a set of basic instructions that the processor uses to perform the functions the user code. Browse to the \Projects\_Arduino\Arduino\_Data\_Collector\_Code\ folder of the EPT FPGA Development System DVD. Copy Arduino\_Data\_Collector\_Code\_U2.ino .

To compile your code,

- Open up the Arduino IDE



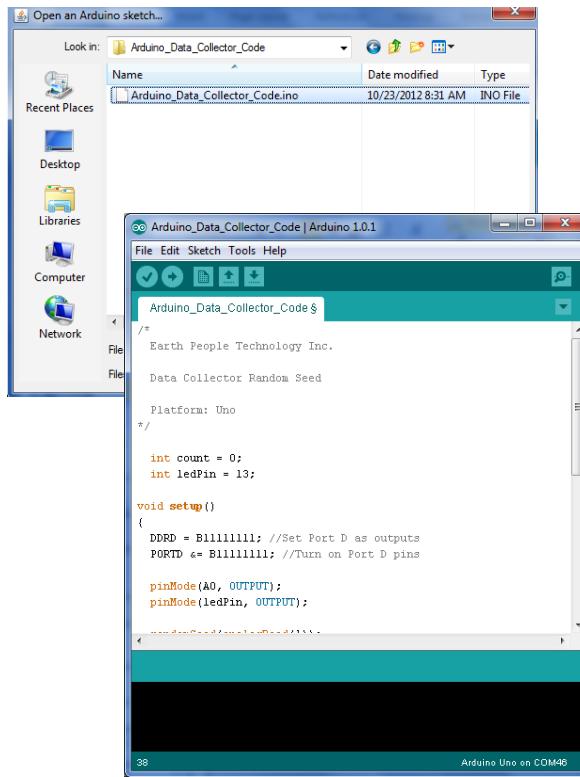
## FPGA Development System User Manual



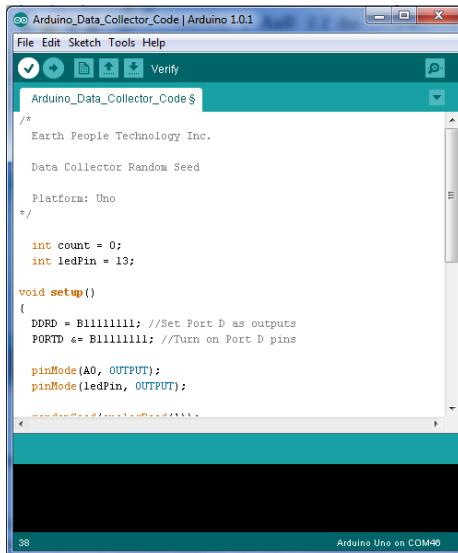
- Load your code into the Sketch.



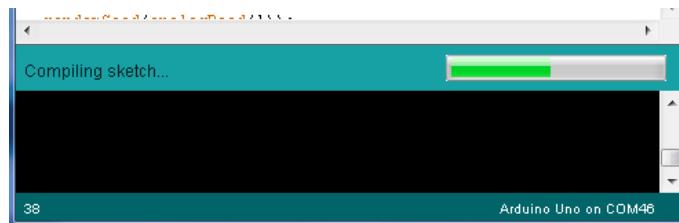
## FPGA Development System User Manual



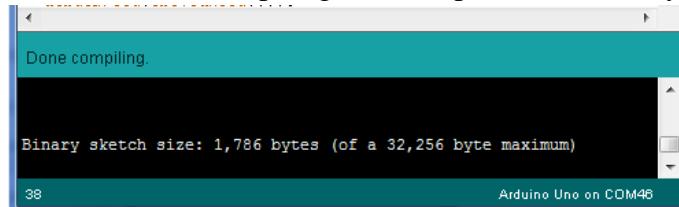
- Click the Verify button



- The sketch will compile



- If there are no errors, the compiling will complete successfully



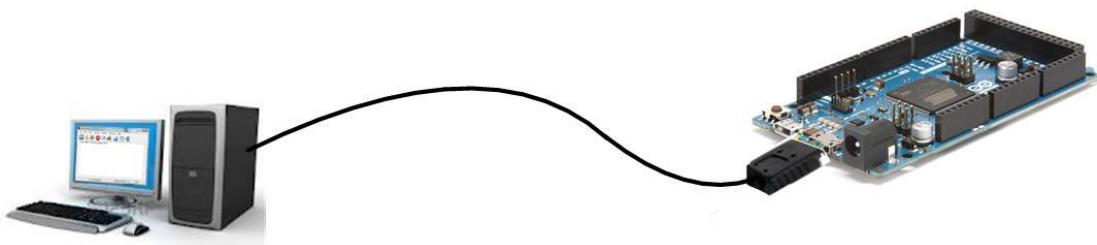
Now we are done with compiling and ready to program the Arduino

### 7.1.6 Programming the Arduino

Programming the Arduino is the process of downloading the user's compiled code into the Flash memory of the Atmel ATmega328 chip. Once the code is downloaded, the Arduino IDE resets the chip and the processor starts executing out of Flash memory.

To program the Arduino

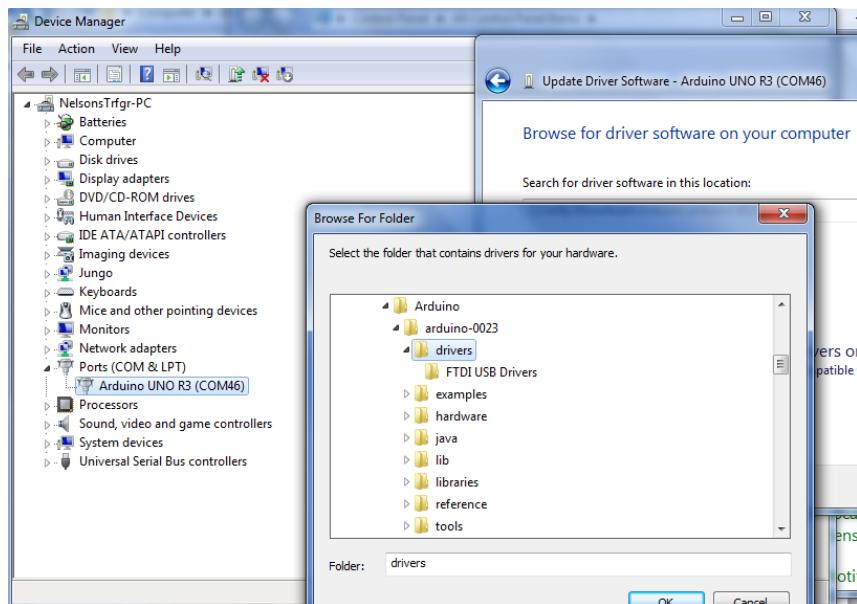
- Connect the USB cable from PC to Arduino



- Load the Arduino USB driver according to the manual
- Plug in your board and wait for Windows to begin its driver installation process. After a few moments, the process will fail, despite its best efforts
- Click on the Start Menu, and open up the Control Panel.
- While in the Control Panel, navigate to System and Security. Next, click on System. Once the System window is up, open the Device Manager.
- Look under Ports (COM & LPT). You should see an open port named "Arduino Due (COMxx)"

## FPGA Development System User Manual

- Right click on the "Arduino Due (COMxx)" port and choose the "Update Driver Software" option.
- Next, choose the "Browse my computer for Driver software" option.
- Finally, navigate to and select the Due's driver file, named "**ArduinoDUE.inf**", located in the "Drivers" folder of the Arduino Software download (not the "FTDI USB Drivers" sub-directory).
- Windows will finish up the driver installation from there.



- Once the driver is loaded, we can set the COM Port. Click on Tools and select Serial Port, then click the available port.



## FPGA Development System User Manual

The screenshot shows the Arduino IDE interface with the following details:

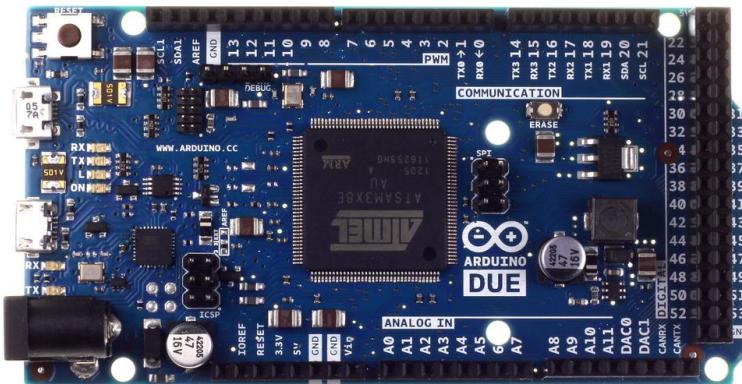
- Title Bar:** Arduino\_Data\_Collector\_Code | Arduino 1.0.1
- Tools Menu:** Auto Format (Ctrl+T), Archive Sketch, Fix Encoding & Reload, Serial Monitor (Ctrl+Shift+M), Board, Serial Port (selected: COM46), Programmer, Burn Bootloader.
- Code Editor:** The code for 'Arduino\_Data\_Collector\_Code' is displayed. It includes comments for Earth People Technology Inc. and the Data Collector Random Seed. It defines a Platform: Uno and initializes variables count and ledPin. The setup() function sets up Port D as outputs and turns on Port D pins A0 and ledPin. The loop() function contains a delay of 1000 ms.
- Output Window:** Shows the message "Done compiling." and the binary sketch size: 1,786 bytes (of a 32,256 byte maximum).
- Status Bar:** Shows the page number 38 and the connection: Arduino Uno on COM46.

- To load the code, click on the Upload button.

The screenshot shows the Arduino IDE interface during the upload process:

- Title Bar:** Arduino\_Data\_Collector\_Code | Arduino 1.0.1
- Tools Menu:** Auto Format (Ctrl+T), Archive Sketch, Fix Encoding & Reload, Serial Monitor (Ctrl+Shift+M), Board, Serial Port (selected: COM46), Programmer, Burn Bootloader.
- Code Editor:** The same code for 'Arduino\_Data\_Collector\_Code' is shown.
- Output Window:** Shows the message "Uploading..." followed by a progress bar, and the binary sketch size: 1,786 bytes (of a 32,256 byte maximum).
- Status Bar:** Shows the page number 38 and the connection: Arduino Uno on COM46.

When the code has completed loading, the Arduino IDE will automatically command the processor to start executing the code. The L LED will blink at one second intervals.

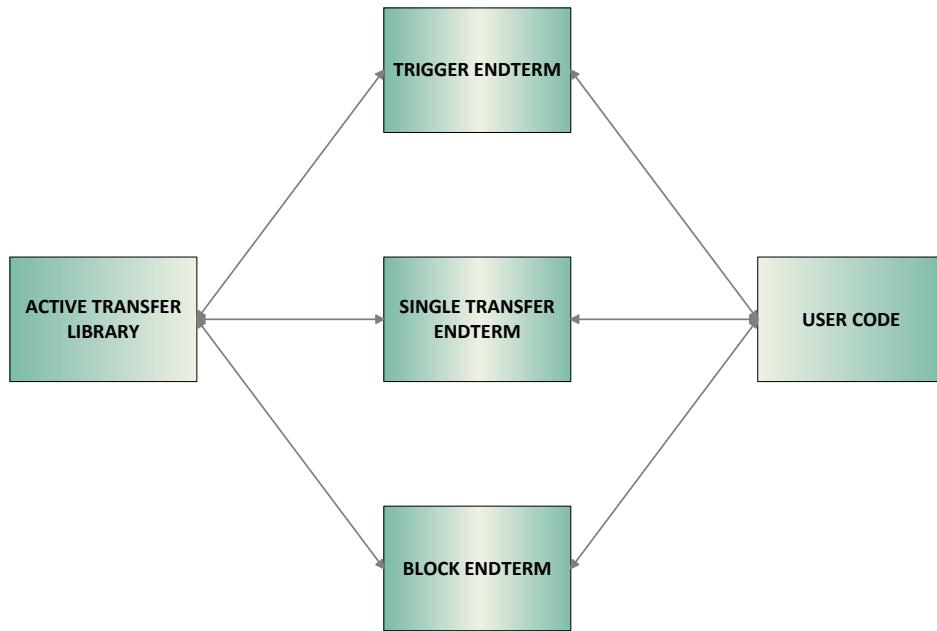


### 7.1.7 FPGA Active Transfer EndTerm Coding and Initiation

The EPT-4CE6-AF-D1 will accept the data collected by the Arduino and transfer it to the PC. It is designed to plug directly into the Arduino Due and there is no need for external wires to be added. The Active Transfer EndTerms are used to connect the Active Transfer Library to the user code. This makes it easy to transfer data to and from the PC via the USB. The user needs to create a state machine to control the transfer between the incoming data and the Active Transfer EndTerms. We will now go through exercise of creating the FPGA code for the Data Collector Sampler.

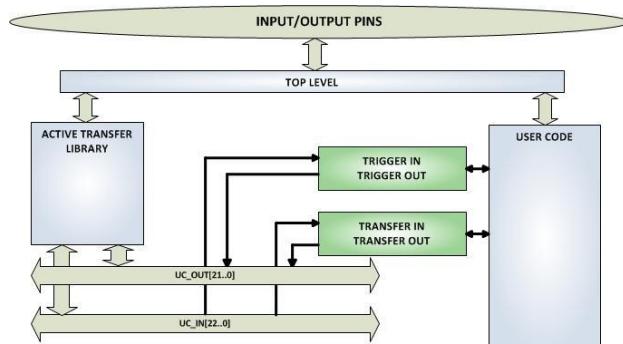
### 7.1.8 FPGA: Define the User Design.

In this step we will define the user's code and include EndTerms and the EPT Active Transfer Library. The Active Transfer Library contains a set of files with a ".vqm" name extension which select particular operations to perform (e.g., byte transfer, block transfer, trigger).. The active\_transfer\_library.vqm file must be included in the top level file of the project. The EndTerms will connect to the active\_transfer\_library and provide a path to connect user code to the library. All of these files are available on the Earth People Technology Project DVD.



We will build our FPGA project using Quartus II software from Altera. The primary file defining the user's FPGA project is named “EPT\_4CE6\_AF\_D1\_Top.v”. It defines the user code and connects the active\_transfer\_library and active\_transfer logic functions. In order to route the pins of the Arduino to the FPGA, the Pin Planner tool is used. This tool allows the user to match internal net names to the pins of the FPGA.

Our project needs to accept an 8 bit value on the J8 connector and a write enable on Pin 1 of J9. For this, we can use the active\_transfer.vqm module as the interface to the active\_transfer\_library. It accepts a single byte and latches it with a single enable net. Because the active\_transfer\_library runs at 66 MHz we will need to write some code ensure that the slower A0 (write enable) signal from the Arduino can latch the data into the active\_transfer module.



FPGA: Coding up the Design  
The first thing to do is to create a top level file for the project. The top level file will include the input and outputs for the FPGA. These are declared according to the Verilog syntax rules. We won't go through all the rules of Verilog here, but feel free to explore the language more thoroughly at

[www.asic-world.com/verilog/](http://www.asic-world.com/verilog/)

We need to add the inputs and outputs for active\_transfer\_library, user code, leds, and switches. Each port is described as input, output or inout. It is followed by the net type wire or reg. If it is a vector, the array description must be added.

```

module EPT_570_AP_U2_Top (
    input  wire [1:0]          aa,
    input  wire [1:0]          bc_in,
    output wire [2:0]          bc_out,
    inout  wire [7:0]          bd inout,
    input  wire [1:0]          TRIGGER_IN_HIGH, //XIOH -- J10
    input  wire [5:0]          TRIGGER_IN_LOW,  //AD   -- J9
    output reg [7:0]           LB_LOWER,        //XIOH -- J10
    input  wire [7:0]          LB_UPPER,        //XIOL -- J8

    //Transceiver Control Signals
    output reg                 TR_DIR_1,
    output reg                 TR_OE_1,
    output wire                TR_DIR_2,
    output wire                TR_OE_2,
    output wire                TR_DIR_3,
    output wire                TR_OE_3,
    input  wire                 SW_USER_1,
    input  wire                 SW_USER_2,
    output reg [2:0]            LED,
    output wire                LED3
);

```



## FPGA Development System User Manual

Next, the parameter's are defined. These are used as constants in the user code.

```
//-----
// Parameters
//-----

//Header Bytes for the Transfer Loopback detection
parameter          TRANSFER_CONTROL_BYTE1 = 8'h5A;
parameter          TRANSFER_CONTROL_BYTE2 = 8'hC3;
parameter          TRANSFER_CONTROL_BYTE3 = 8'h7E;

//State Machine Transfer Loopback detection
parameter          TRANSFER_CONTROL_IDLE = 0,
                    TRANSFER_CONTROL_HDR1 = 1,
                    TRANSFER_CONTROL_HDR2 = 2,
                    TRANSFER_DECODE_BYTE = 3,
                    TRANSFER_CONTROL_SET = 4;

parameter          GLOBAL_RESET_COUNT = 12'h09c8;
```



**EARTHPEOPLE**  
TECHNOLOGY

## FPGA Development System User Manual

Next is the Internal Signal and Register Declarations.



```
////////////////////////////////////////////////////////////////////////
/* Internal Signals and Registers Declarations
////////////////////////////////////////////////////////////////////////
    wire                      CLK_66;
    wire                      RST;

    wire [23:0]                UC_IN;
    wire [21:0]                UC_OUT;

    //Trigger Signals
    reg [7:0]                  trigger_out;
    wire [7:0]                  trigger_in_byte;
    reg [7:0]                  trigger_in_store;

    //LED registers
    reg                        led_reset;

    //Switch registers
    reg                        switch_reset;

    //Transfer registers
    wire                      transfer_out;
    reg                       transfer_out_reg;
    wire                      transfer_in_received;
    wire [7:0]                 transfer_in_byte;
    wire [7:0]                 transfer_out_byte;
    reg [3:0]                  transfer_to_host_counter;
    reg [3:0]                  transfer_to_host_state;

    //Transfer Control registers
    reg                       transfer_in_loop_back;
    reg                       transfer_in_received_reg;
    reg [3:0]                  transfer_control_state;
    reg [7:0]                  transfer_control_byte;

    //Transfer Write from Arduino
    reg                       transfer_write_reg;
    reg                       transfer_write;
    reg [7:0]                  transfer_write_byte;

    //Reset signals
    wire                      reset;
    reg [11:0]                 reset_counter;
    reg                       reset_signal_reg;

    //Input/Output Signals
    reg                       start_stop_cntrl;
```



Next, add the assignments. These assignments will set the direction of the bus transceivers that interface to the Arduino I/O's. The transceivers also include an output enable bit.

```
//*****
//**      Signal Assignments
//*****
assign          TR_DIR_2  = 1'b1; //1 = A to B; 0 = B to A
assign          TR_OE_2   = 1'b0;

assign          TR_DIR_3  = 1'b1; //1 = A to B; 0 = B to A
assign          TR_OE_3   = 1'b0;

//Clock and Reset
assign          CLK_66 = aa[1];
assign          RST = reset;
assign          reset = reset_signal_reg;

//Transfer registers
assign          transfer_out = transfer_out_reg | transfer_write;
assign          transfer_out_byte = transfer_write_byte;

//LED3 is used to signify to the user that the Start
//switch is enabled
assign          LED3 = ~start_stop_cntrl;
....
```

The reset signal is generated by a counter that starts counting upon power up. When the counter reaches GLOBAL\_RESET\_COUNT.



```
//*****
//*      Reset Signal
//*****
always @(posedge CLK_IN or negedge aa[0])
begin
    if(!aa[0])
        begin
            reset_signal_reg <= 1'b0;
            reset_counter <= 0;
        end
    else
        begin
            if( reset_counter < GLOBAL_RESET_COUNT )
                begin
                    reset_signal_reg <= 1'b0;
                    reset_counter <= reset_counter + 1'b1;
                end
            else
                begin
                    reset_signal_reg <= 1'b1;
                end
        end
end
```

The four LED's are set by the bottom four bits of the active\_trigger output register. These trigger outputs can be set by using a function in the Active\_Host DLL on the PC. The Data Collector project will use LED3 to indicate the state of the Start/Stop signal.

```
//LED3 is used to signify to the user that the Start
//switch is enabled
assign          LED3 = ~start_stop_cntrl;
```



## FPGA Development System User Manual

```
//-----
// LED Set
//-----
```

```
always @ (trigger_in_byte or led_reset or LED or RST)
begin
    if (!RST)
        LED[2:0] = 3'hz;
    else if (led_reset)
        LED[2:0] = 3'hz;
    else if (trigger_in_byte[3:0])
        begin
            case(trigger_in_byte[3:0])
                3'h1:
                    LED[0] = 1'b0;
                3'h2:
                    LED[1] = 1'b0;
                3'h4:
                    LED[2] = 1'b0;
                default:
                    LED[2:0] = LED[2:0];
            endcase
        end
    end
end
```

The two user switches are connected to the input trigger register. Pressing a switch will send a trigger to the PC to be decoded by the Active\_Host DLL.

```

//-----
// User Switch Trigger
//-----
always @(posedge CLK_IN or negedge RST)
begin
    if(!RST)
        begin
            trigger_out <= 8'h00;
        end
    else
        begin
            if(!SW_USER_1 )
                trigger_out <= 8'h01;
            else if(!SW_USER_2 )
                trigger_out <= 8'h02;
            else if(switch_reset)
                trigger_out <= 8'h04;
            else
                trigger_out <= 8'h00;
        end
end

```

Next, we will add the transfer detection signal from the Arduino. This block will require three registers.

- transfer\_write\_reg –This is a latch register to hold the state of the A0(Write Enable)
- transfer\_write –This register is used to start the active\_transfer single byte write to the PC.
- transfer\_write\_byte –This is an 8 bit register to hold the value of the Data Collection output.

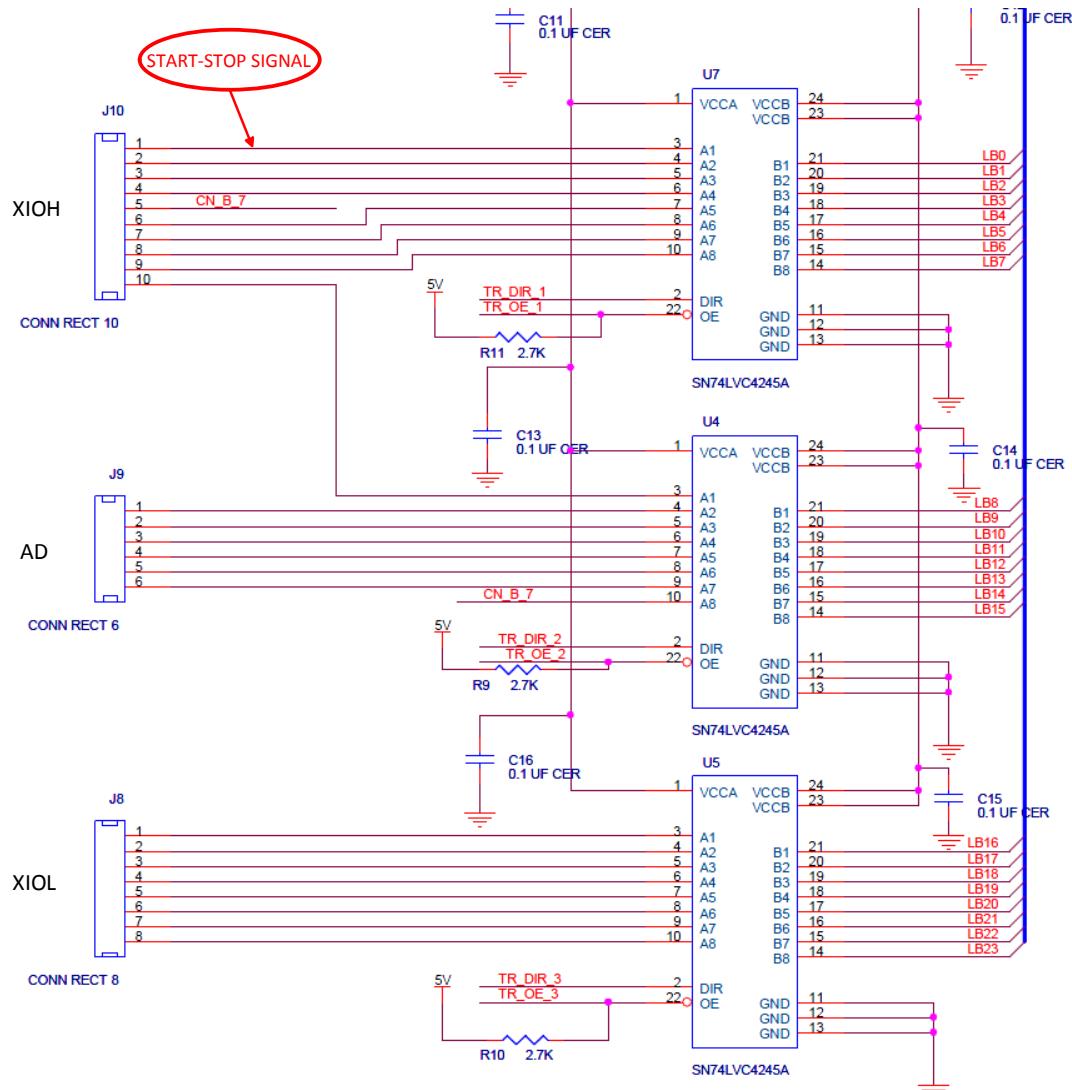
This block will compare the input signal on TRIGGER\_IN\_LOW[1] to a high. The TRIGGER\_IN\_LOW[1] pin is routed to Pin 1 of J9 which is routed to the A0(Write Enable) of the Arduino Data Collector. When this bit goes high, the priority encoder goes into statement 1 and sets transfer\_write\_reg and transfer\_write high and latches the value on the LB\_UPPER[7:0] pins to the transfer\_write\_byte register. By setting transfer\_write\_reg high, the priority encoder goes into statement 2 which will set transfer\_write register to low and stay in statement 2 of the priority encoder. The back to back high and low on the transfer\_write register will cause the active\_transfer module to latch the value of transfer\_write\_byte into the active\_transfer\_library module and sets up the byte transfer to the PC. When the TRIGGER\_IN\_LOW[1] -A0(Write Enable) pin goes low, the encoder will reset transfer\_write\_reg and transfer\_write to



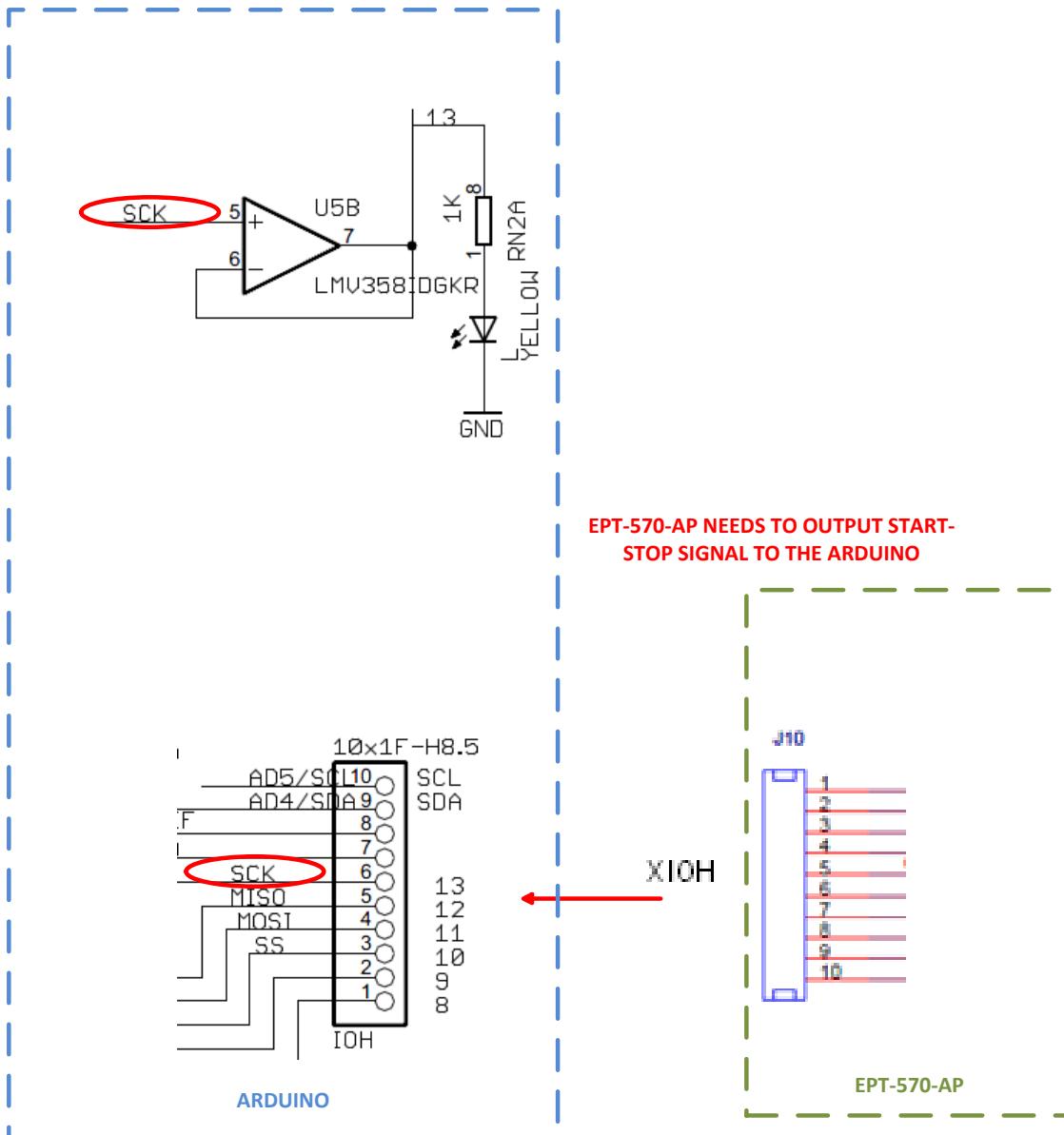
low. The encoder goes back to waiting for the TRIGGER\_IN\_LOW[1] -A0(Write Enable) to assert high.

```
//-----
// Detect Transfer From Arduino
//-----
always @(posedge CLK_IN or negedge RST)
begin
    if (!RST)
        begin
            transfer_write_reg <= 1'b0;
            transfer_write <= 1'b0;
            transfer_write_byte <= 0;
        end
    else
        begin
            if(TRIGGER_IN_LOW[1] & !transfer_write_reg)
                begin
                    transfer_write_reg <= 1'b1;
                    transfer_write <= 1'b1;
                    transfer_write_byte <= LB_UPPER;
                end
            else if(TRIGGER_IN_LOW[1] & transfer_write_reg)
                begin
                    transfer_write_reg <= 1'b1;
                    transfer_write <= 1'b0;
                end
            else if(!TRIGGER_IN_LOW[1] & transfer_write_reg)
                begin
                    transfer_write_reg <= 1'b0;
                    transfer_write <= 1'b0;
                    transfer_write_byte <= 0;
                end
        end
    end
end
```

This block of code takes care of reading the random word from the Arduino using the A0(Write Enable) Pin. However, because the Arduino is expecting a Start/Stop bit on Digital Pin8, the FPGA code has to provide this bit. This presents a problem, the EPT-4CE6-AF-D1 has 3 eight bit bi-directional ports. Which means each port is has a direction which is either input or ouput at a given time. However, the ports can be switched between input and output at any time. Two of the three ports must be used as inputs into the FPGA for the random word and the A0(Write Enable) Pin. So, the third port can be used as the output port.



This, however, causes another problem! The Arduino XIOH connector needs to output the Amber LED state. So, if one pin on the connector needs to be an output, the EPT-4CE6-AF-D1 port on J10 (XIOH) cannot be an output! This would interfere with the turning on and turning off of the LED.



So, we can fix this problem by noting that the 8 bit bi-directional ports on the EPT-4CE6-AF-D1 have Output Enables that allow the FPGA to “float” the signals of the port at any time. By floating the port, we can multiplex the signals of the port. When we need to drive the signals from the EPT-4CE6-AF-D1 port to the Arduino, we turn on the Output Enables of the port. And when we need to let the Arduino drive its signals, we turn off the Output Enables of the port.



```
]
  //-----
  // Detect Transfer From Arduino
  //-----
]
  always @ (posedge CLK_66 or negedge RST)
begin
  if (!RST)
    begin
      transfer_write_reg <= 1'b0;
      transfer_write <= 1'b0;
      transfer_write_byte <= 0;
      TR_DIR_1  <= 1'b0; //1 = A to B; 0 = B to A
      TR_OE_1   <= 1'b0;
      LB_LOWER[0] <= start_stop_cntrl;
      LB_LOWER[7:1] <= 7'b0100000;
    end
end
```

In the reset section of the synchronous block, we turn the Direction bit to “B to A”

TR\_DIR\_1 <= 1'b0;

and the Output Enable on.

TR\_OE\_1 <= 1'b0; (Output Enables are asserted with a zero)

The start\_stop\_cntrl signal is set by using the TRANSFER\_CONTROL state machine in the following section. So, if the start\_stop\_cntrl signal is set, the Output Enable is turned on and the signal will appear on DigitalPin8 on the Arduino XIOH connector. As the Data Collector code cycles through its loop() function, it will cause the if statement to branch into its conditional statement. The Data Collector code will assert the A0(Write Enable) Pin in its conditional statement. The A0(Write Enable) Pin will cause the FPGA code to enter into its first conditional statement. This first statement turns off the Output Enables of the Port J10. With the Port turned off, the Arduino can set the LED on when it executes its code. When the A0(Write Enable) Pin is de-asserted, the Output Enable of Port J10 is turned back on and the whole process can start over.

Next, we add a TRANSFER\_CONTROL state machine to read the Control Register from the Host PC using the active\_transfer EndTerm. This state machine will decode the 8 bit control register only after a sequence of three 8 bit bytes in the order of 0x5a, 0xc3, 0x7e. The operation of the state machine is as follows.

- The TRANSFER\_CONTROL state machine will stay in the idle state of the parallel encoder until a byte from the active\_transfer transfer\_to\_device register receives a 0x5a.
- This will cause the transfer\_control\_state to be changed to TRANSFER\_CONTROL\_HDR1.
- The state machine will stay in the TRANSFER\_CONTROL\_HDR1 state until the next byte is read from the active\_transfer.

- If the byte from transfer\_to\_device is a 0xc3, the transfer\_control\_state will be changed to TRANSFER\_CONTROL\_HDR2.
- If the byte from transfer\_to\_device is not a 0xc3, the transfer\_control\_state will go back to idle.
- In the TRANSFER\_CONTROL\_HDR2 state , the state machine will stay in this state until the next byte from the active\_transfer is received.
- If the byte from transfer\_to\_device is a 0x7e, the transfer\_control\_state will be changed to TRANSFER\_DECODE\_BYTE.
- If the byte from transfer\_to\_device is not a 0x7e, the transfer\_control\_state will go back to idle.
- In the TRANSFER\_DECODE\_BYTE state , the state machine will stay in this state until the next byte from the active\_transfer.
- The next byte transferred from active\_transfer will be decoded as the Control Register.

The bits of the Control Register are defined below.

<b>Register</b>	<b>Bits</b>	<b>Description</b>	<b>Assertion</b>
Control	0	Start Stop Cntrl	High
	1	Not Used	
	2	LED Reset	High
	3	Switch Reset	High
	4	Transfer In Loop Back	High
	5	Not Used	
	6	Not Used	
	7	Not Used	
	7	Not Used	



```
//-----
// State Machine: Control Register from Transfer In
//-----
always @ (posedge CLK_IN or negedge RST)
begin
    if (!RST)
    begin
        transfer_in_received_reg <= 1'b0;
        transfer_control_state <= TRANSFER_LOOPBACK_IDLE;
        transfer_in_loop_back <= 1'b0;
        led_reset <= 1'b0;
        switch_reset <= 1'b0;
    end
    else
    begin
        if(transfer_in_received & !transfer_in_received_reg)
        begin
            transfer_in_received_reg <= 1'b1;
            case(transfer_control_state)
                TRANSFER_CONTROL_IDLE:
                    if((transfer_in_byte == TRANSFER_CONTROL_BYTE1))
                        transfer_control_state <= TRANSFER_CONTROL_HDR1;
                    else if((transfer_in_byte != TRANSFER_CONTROL_BYTE1))
                        transfer_control_state <= TRANSFER_CONTROL_IDLE;
                    else
                        transfer_control_state <= TRANSFER_CONTROL_IDLE;
                TRANSFER_CONTROL_HDR1:
                    if((transfer_in_byte == TRANSFER_CONTROL_BYTE2))
                        transfer_control_state <= TRANSFER_CONTROL_HDR2;
                    else if((transfer_in_byte != TRANSFER_CONTROL_BYTE2))
```



```
        transfer_control_state <= TRANSFER_CONTROL_IDLE;
    else
        transfer_control_state <= TRANSFER_CONTROL_HDR1;
TRANSFER_CONTROL_HDR2:
    if((transfer_in_byte == TRANSFER_CONTROL_BYTE3))
        transfer_control_state <= TRANSFER_DECODE_BYTE;
    else if((transfer_in_byte != TRANSFER_CONTROL_BYTE3))
        transfer_control_state <= TRANSFER_CONTROL_IDLE;
    else
        transfer_control_state <= TRANSFER_CONTROL_HDR2;
TRANSFER_DECODE_BYTE:
begin
    transfer_in_loop_back <= transfer_in_byte[0];
    led_reset <= transfer_in_byte[2];
    switch_reset <= transfer_in_byte[3];
    transfer_loopback_state <= TRANSFER_LOOPBACK_SET;
end
TRANSFER_CONTROL_SET:
begin
    transfer_control_state <= TRANSFER_CONTROL_IDLE;
end
endcase
end
else if(!transfer_in_received & transfer_in_received_reg)
    transfer_in_received_reg <= 1'b0;
end
end
```

Next, up is the instantiation for the active\_transfer\_library. The ports include the input and output pins and the two buses that connect the active modules. These buses are the input UC\_IN[23:0] and output UC\_OUT[21:0].



## FPGA Development System User Manual

```
//-----
// Instantiate the EPT Active Transfer Library
//-----

active_transfer_library    ACTIVE_TRANSFER_LIBRARY_INST
(
    .aa                      (aa),
    .bc_in                   (bc_in),
    .bc_out                  (bc_out),
    .bd inout                (bd inout),

    .UC_IN                   (UC_IN),
    .UC_OUT                  (UC_OUT)

);
```

Finally, we instantiate the Active EndTerms. For the Data Collection project, we only need active\_transfer and active\_trigger EndTerms. The uc\_out port for both modules must be shared. Since they both drive this bus, a bus wide wired-or circuit is used so that they don't drive each other. The active\_transfer EndTerm has a port for the address (uc\_addr). This allows the PC to address up to 8 different modules. Just add a three bit address to this port and the PC must add this same address to communicate with this module.



```
//-----
// Instantiate the EPT Active Modules
//-----
wire [22*2-1:0] uc_out_m;
eptWireOR #( .N(2) ) wireOR (UC_OUT, uc_out_m);
active_trigger ACTIVE_TRIGGER_INST
(
    .uc_clk          (CLK_IN),
    .uc_reset        (RST),
    .uc_in           (UC_IN),
    // .uc_out          (UC_OUT),
    .uc_out          (uc_out_m[ 0*22 +: 22 ]),
    .trigger_to_host (trigger_out),
    .trigger_to_device (trigger_in_byte)
);

active_transfer ACTIVE_TRANSFER_INST
(
    .uc_clk          (CLK_IN),
    .uc_reset        (RST),
    .uc_in           (UC_IN),
    // .uc_out          (UC_OUT),
    .uc_out          (uc_out_m[ 1*22 +: 22 ]),
    .start_transfer  (transfer_out),
    .transfer_received (transfer_in_received),
    .uc_addr         (3'h2),
    .transfer_to_host (transfer_out_byte),
    .transfer_to_device (transfer_in_byte)
);
```

Next, we are ready to compile and synthesize.

### 7.1.9 FPGA: Compile/Synthesize the Project

The Quartus II application will compile/ synthesize the user code, active\_transfer\_library, and the active EndTerms. The result of this step is a file containing the FPGA code with “\*.pof”. First, we need to create a project in the Quartus II environment. Follow the directions in the section: “Compiling, Synthesizing, and Programming FPGA”.



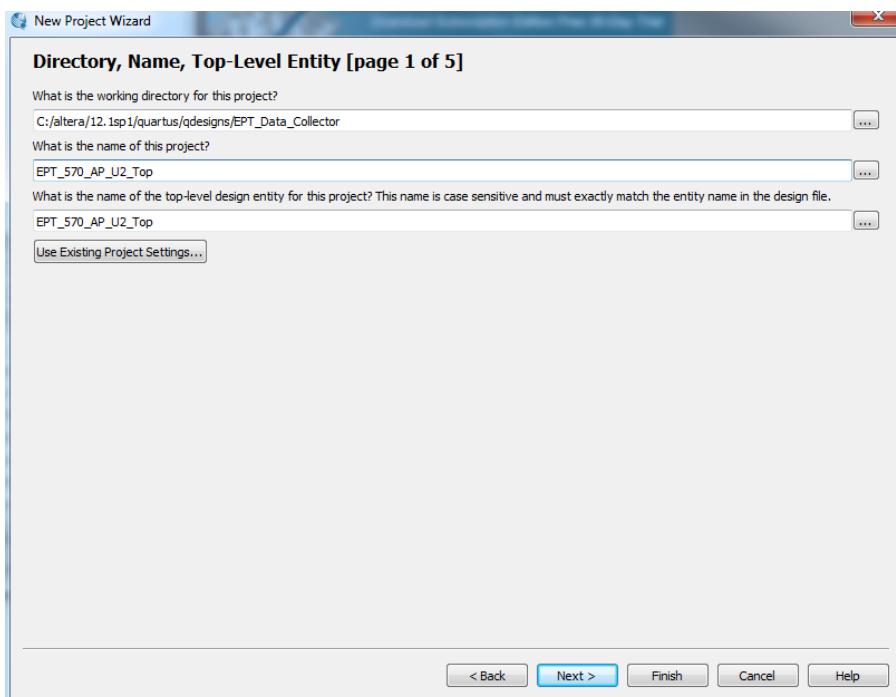
## FPGA Development System User Manual

Bring up Quartus II, then use Windows Explorer to browse to  
c:/altera/xxx/quartus/qdesigns create a new directory called: “EPT\_Data\_Collector”.



Open Quartus II by clicking on the icon .

Under Quartus, Select File->New Project Wizard. The Wizard will walk you through setting up files and directories for your project.



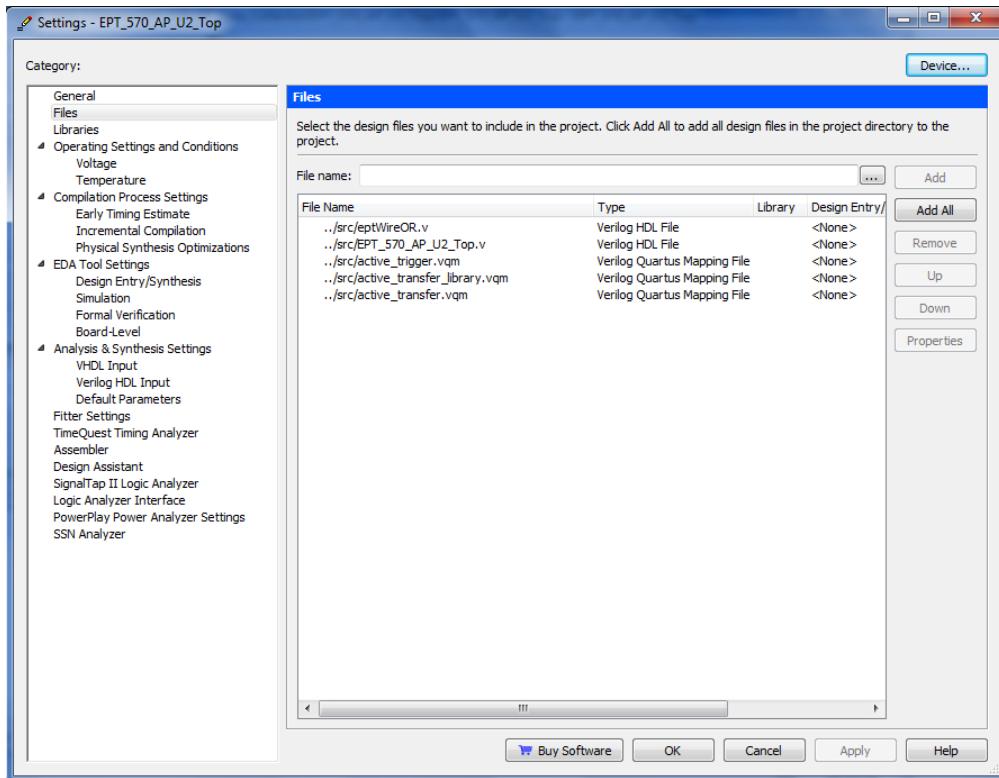
At the Top-Level Entity page, browse to the  
c:\altera\xxx\quartus\qdesigns\EPT\_Data\_Collector directory to store your project.  
Type in a name for your project EPT\_4CE6\_AF\_D1

Follow the steps up to Add Files. At the Add Files box, click on the Browse button and navigate to the project Data Collector install folder in the dialog box. Browse to the \Projects\_HDL\EPT\_Data\_Collector \ EPT\_4CE6\_AF\_ Top folder of the EPT FPGA Development System DVD. Copy the files from the \src directory.

- Active\_transfer.vqm
- Active\_trigger.vqm
- Active\_transfer\_library.vqm

- eptWireOr.v
- EPT\_4CE6\_AF\_Top.v

Add the files:



Continue following the instructions by adding a device and finishing the project instantiation. Then, add the Pins.

- Under Assignments, Select Import Assignments.
- At the Import Assignment dialog box, browse to the `\Projects_HDL\EPT_Data_Collector \ EPT_4CE6_AF_D1_Top` folder of the EPT FPGA Development System DVD. Select the Quartus Specification file, “`EPT_4CE6_AF_D1_Top.qsf`” .
- Click Ok. Under Assignments, Select Pin Planner. Verify the pins have been imported correctly.

Next, we need to add the Synopsys Design Constraint file. This file contains timing constraints which forces the built in tool called TimeQuest Timing Analyzer to analyze the path of the synthesized HDL code with setup and hold times of the internal registers.

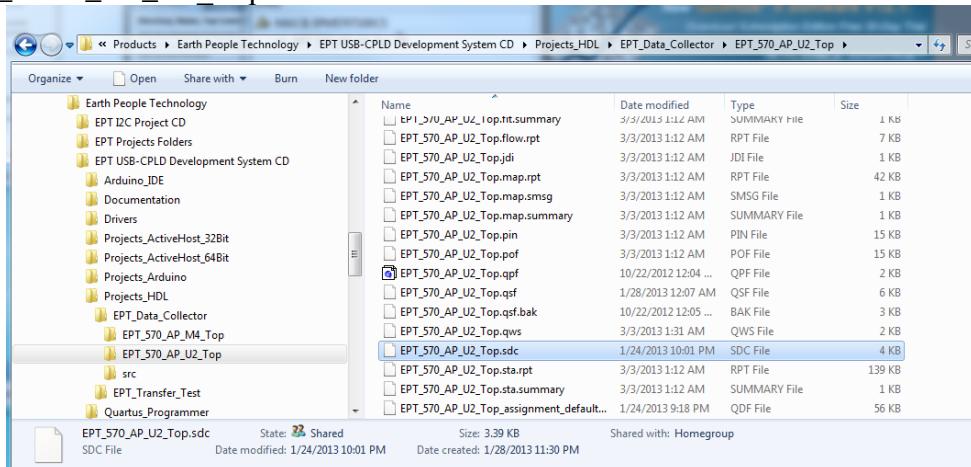


## FPGA Development System User Manual

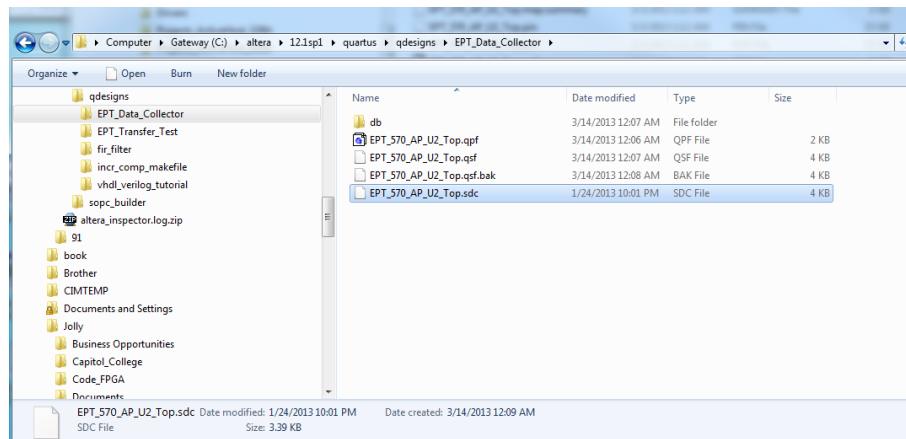
It takes note of any path that may be too long to appropriately meet the timing qualifications. For more information on TimeQuest Timing Analyzer, see

[http://www.altera.com/literature/hb/qts/qts\\_qii53018.pdf?GSA\\_pos=1&WT.oss\\_r=1&WT.oss=TimeQuest Timing Analyzer](http://www.altera.com/literature/hb/qts/qts_qii53018.pdf?GSA_pos=1&WT.oss_r=1&WT.oss=TimeQuest%20Timing%20Analyzer)

Browse to the \Projects\_HDL\EPT\_Data\_Collector\EPT\_EPT-4CE6-AF-D1\_TOP folder of the EPT FPGA Development System DVD. Select the “EPT\_4CE6\_AF\_D1\_Top.sdc” file.



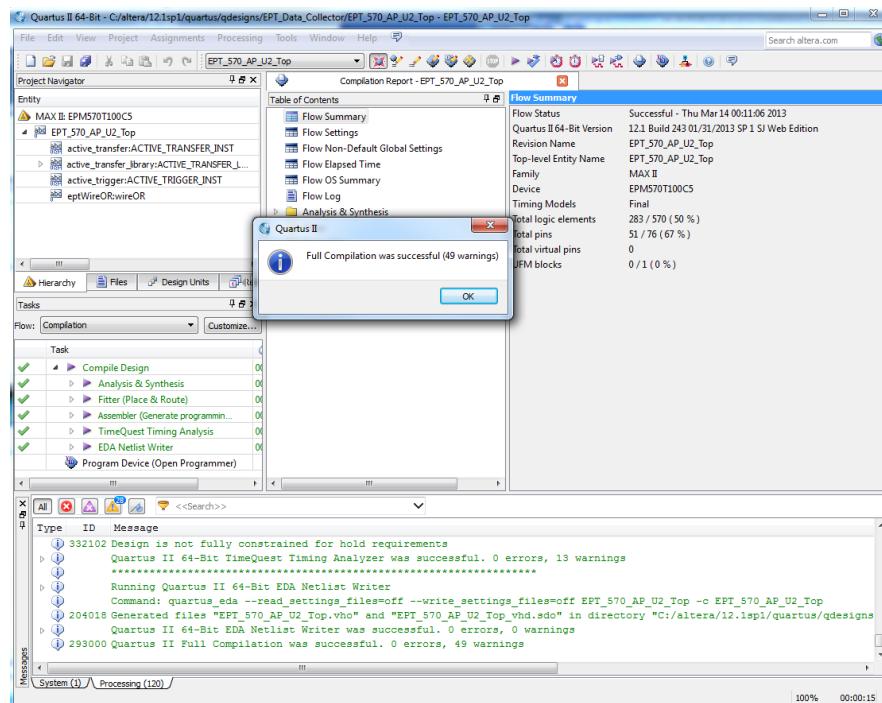
Copy the file and browse to c:\altera\xxx\quartus\qdesigns\EPT\_Data\_Collector directory. Paste the file.



and select the Start Compilation button.



This will cause the compile and synthesis process. After successful completion, the screen should look like the following:

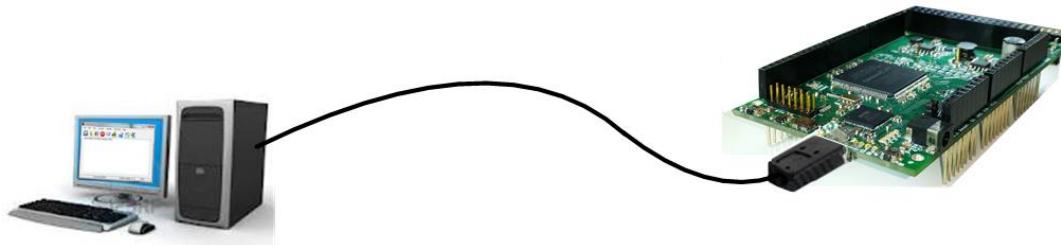


If the synthesis fails, you will see the failure message in the message window. Note that in addition to fatal errors, the compile process can produce “warnings” which do not necessarily prevent execution of the code but which should be corrected eventually.

At this point the project has been successfully compiled, synthesized and a programming file has been produced. See the next section on how to program the FPGA.

### 7.1.10 FPGA: Program the DPL Flash

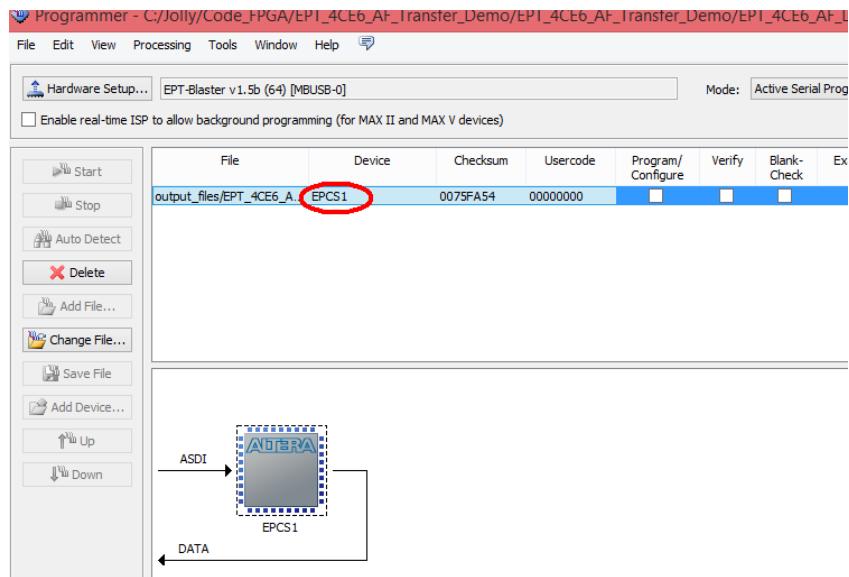
The final step is programming the “\*.pof” file into the FPGA. Follow the section: “Programming the FPGA”.



- Connect the EPT-4CE6-AF-D1 to the PC,
- Open up Quartus II,
- Open the programmer tool
- In the upper left corner of the Programmer Tool, there is a button labeled “Hardware Setup”. Verify that EPT-Blaster v1.5b” has been selected. If not, go to the section JTAG DLL Insert to Quartus II and follow the directions.
- Check the box under Program/Configure
- Click the Start button.

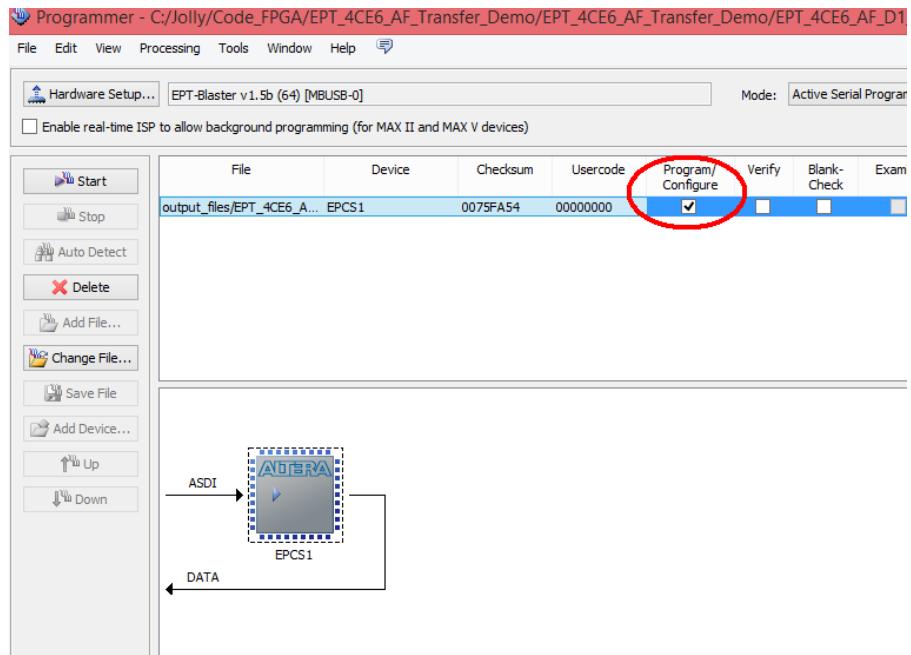
When the programming is complete, the Progress bar will indicate success.

Select the EPSC1 under “Device”.

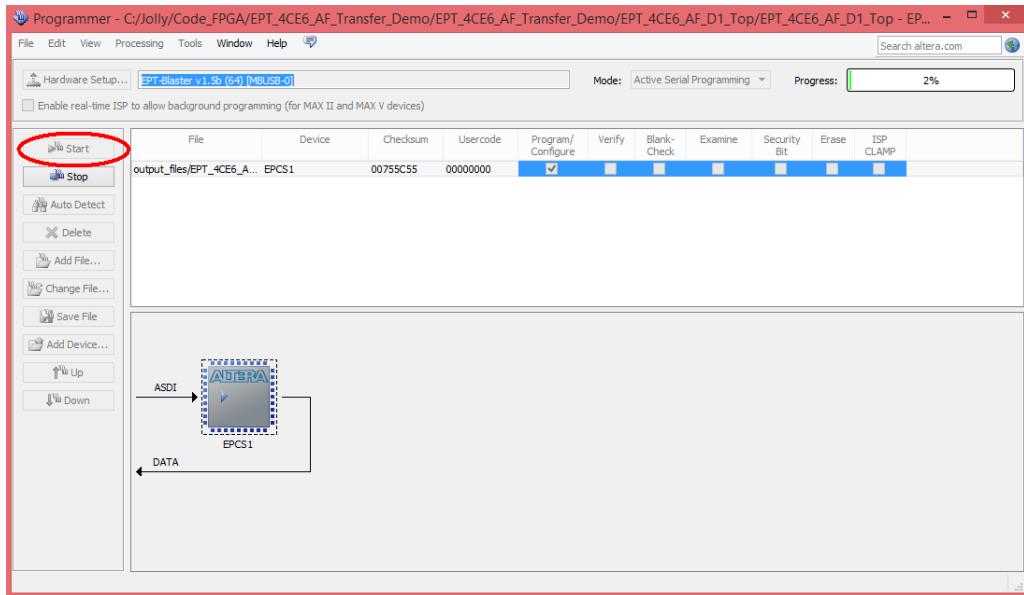


Next, select the checkbox under the “Program/Configure” of the Programmer Tool.

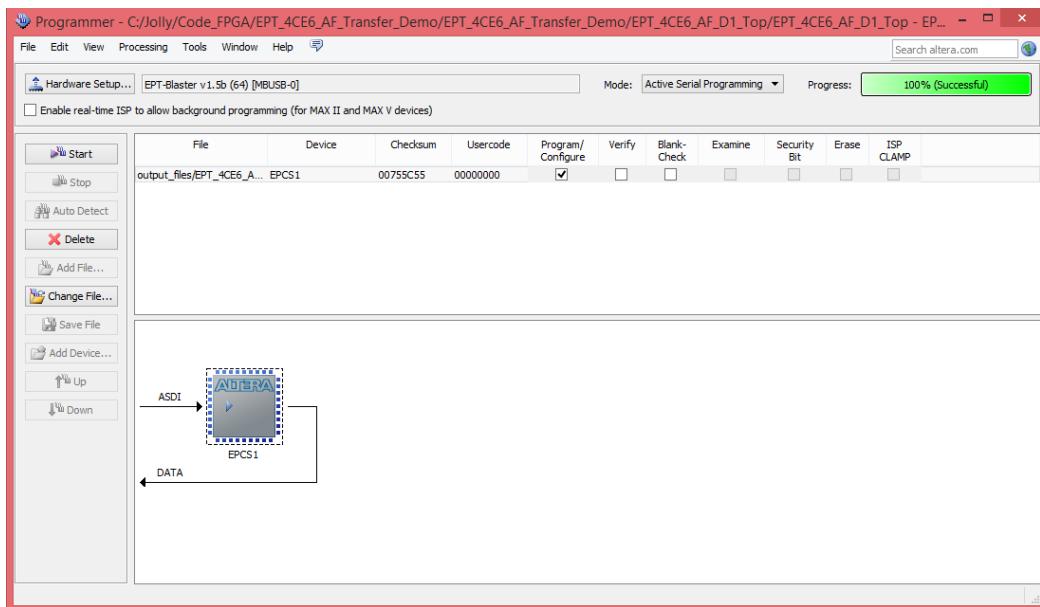
## FPGA Development System User Manual



Click on the Start button to to start programming the FPGA. The Progress bar will indicate the progress of programming.



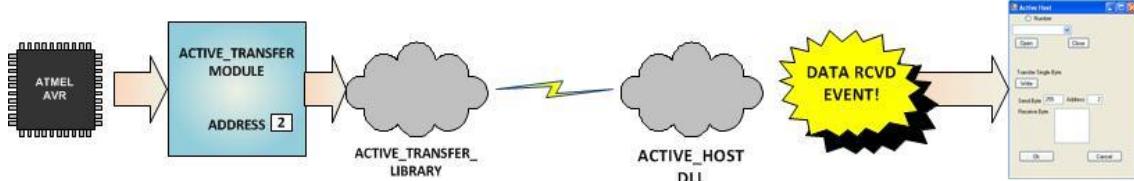
When the programming is complete, the Progress bar will indicate success.



At this point, the EPT-4CE6-AF-D1 is programmed and ready for use.

### 7.1.11 PC: Design the Project

The final piece of the Data Collection Sampler is the PC application. This application will fetch the data from the FPGA of the EPT-4CE6-AF-D1 and display it on the screen. It includes user code, windows form, and the Active\_Host DLL.



The Active\_Host DLL is designed to transfer data from the FPGA when it becomes available. The data will be stored into local memory of the PC, and an event will be triggered to inform the user code that data is available from the addressed module of the FPGA. This method, from the user code on the PC, makes the data transfer transparent. The data just appears in memory and the user code will direct the data to a textbox on the Windows Form.

The Data Collector project will perform the following functions.

- Find EPT-4CE6-AF-D1 Device.
- Open EPT-4CE6-AF-D1 Device.
- Start the Arduino data collection process.
- Wait for data from EPT-4CE6-AF-D1.

- Display data from EPT-4CE6-AF-D1 in textbox.

### 7.1.12 PC: Coding the Project

The user code is based on the .NET Framework and written in C#. The language is great for beginners as it is a subset of the C++ language. It has the look and feel of the familiar C language but adds the ease of use of classes, inheritance and method overloading. C# is an event based language which changes the method of writing code for this project. See the section “Assembling, Building, and Executing a .NET Project on the PC” for a better description of event based language programming.

To start the project, follow the section “Assembling, Building, and Executing a .NET Project on the PC”. Use the wizard to create project called “Data\_Collector”. When the wizard completes, the C# Express main window will look like the following.

```

using System;
using System.Drawing;
using System.Collections;
using System.Windows.Forms;
using System.Data;
using System.Threading;
using System.Runtime.InteropServices;
using System.Diagnostics;

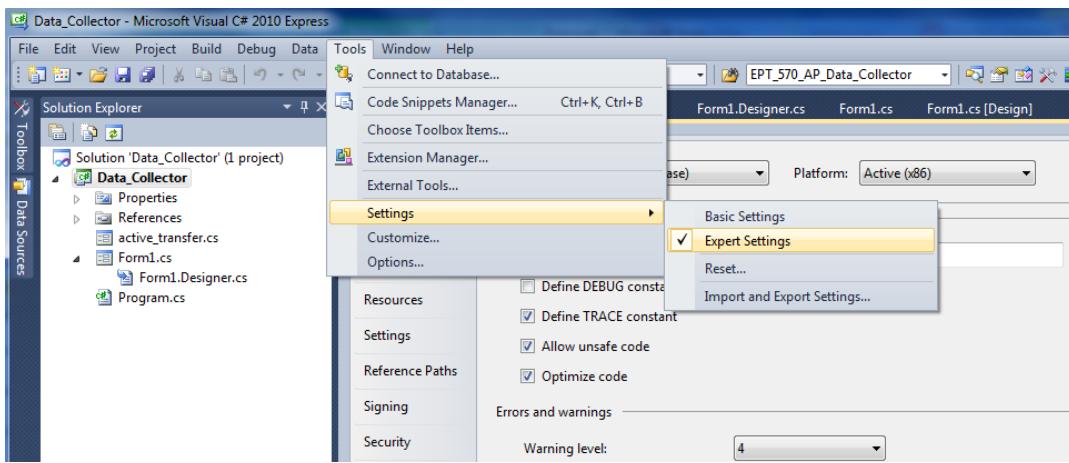
namespace Data_Collector
{
    public partial class Data_Collector : System.Windows.Forms.Form
    {
        public Data_Collector()...
    }
}
  
```

These statements setup the namespace and the class for the project. There are several other files that are created by the wizard such as Form1.Designer.cs, Program.cs, Form1.resx. We don't need to go into these support files, we will just focus on the Form1.cs as this is where all the user code goes.

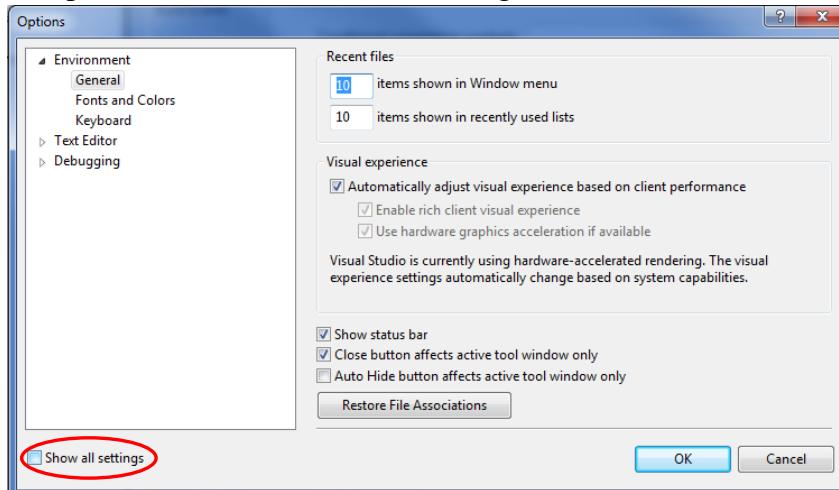
The project environment must be set up correctly in order to produce an application that runs correctly on the target platform. Visual C# Express defaults new projects to 32 bits. If you OS is a 64 bit platform, use the following directions to set up a 64 bit project. First, we need tell C# Express to produce 64 bit code if we are running on a x64 platform. Go to Tools->Settings and select Expert Settings



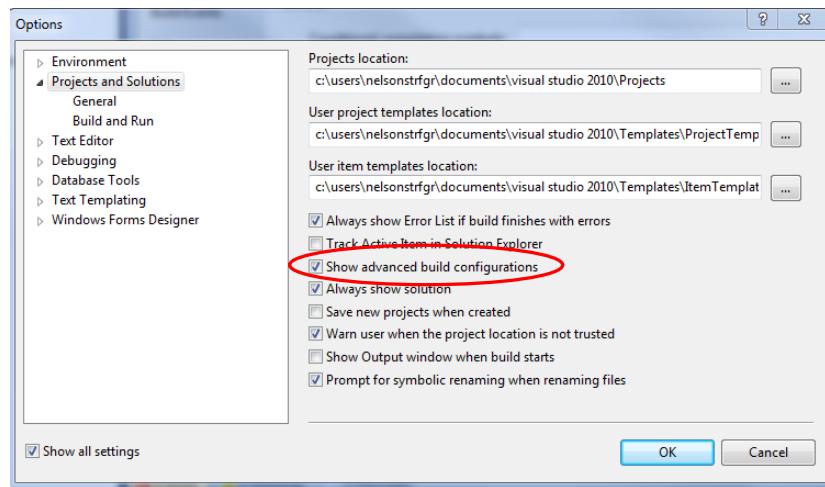
## FPGA Development System User Manual



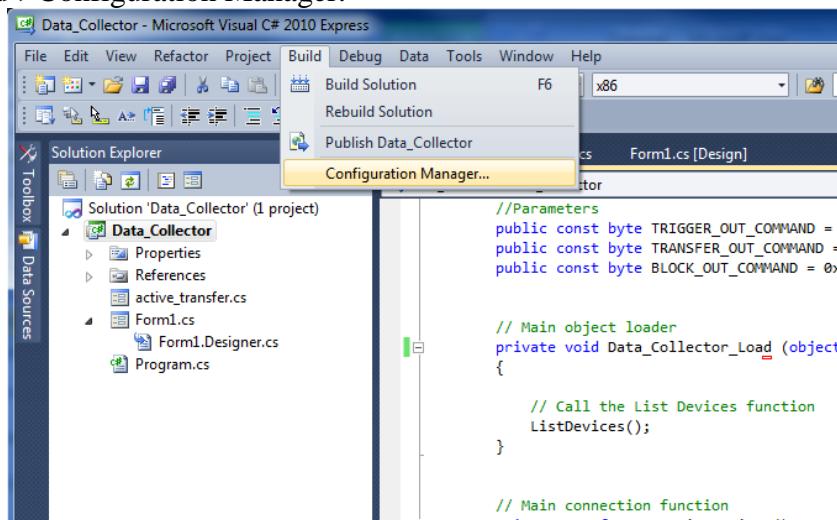
Go to Tools->Options, locate the “Show all settings” check box. Check the box.



In the window on the left, go to “Projects and Solutions”. Locate the “Show advanced build configurations” check box. Check the box.



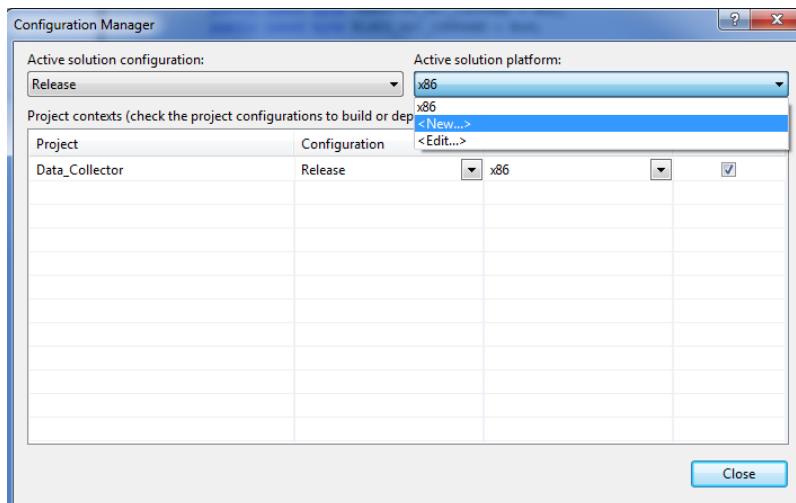
Go to Build->Configuration Manager.



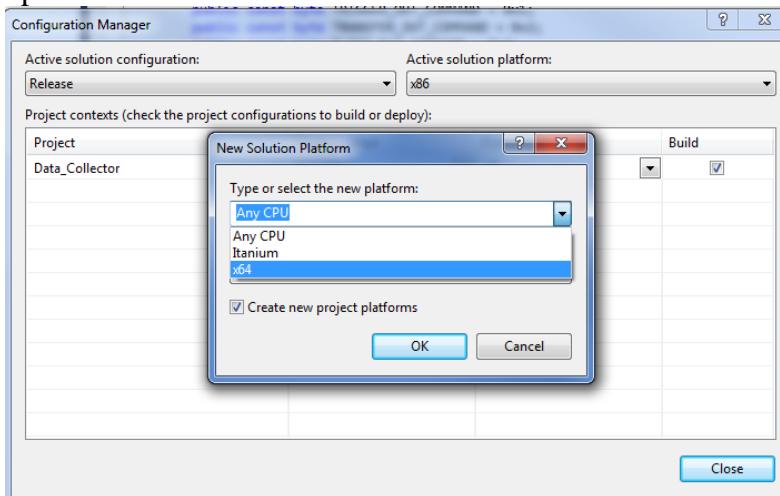
In the Configuration Manager window, locate the "Active solution platform:" label, select "New" from the drop down box.



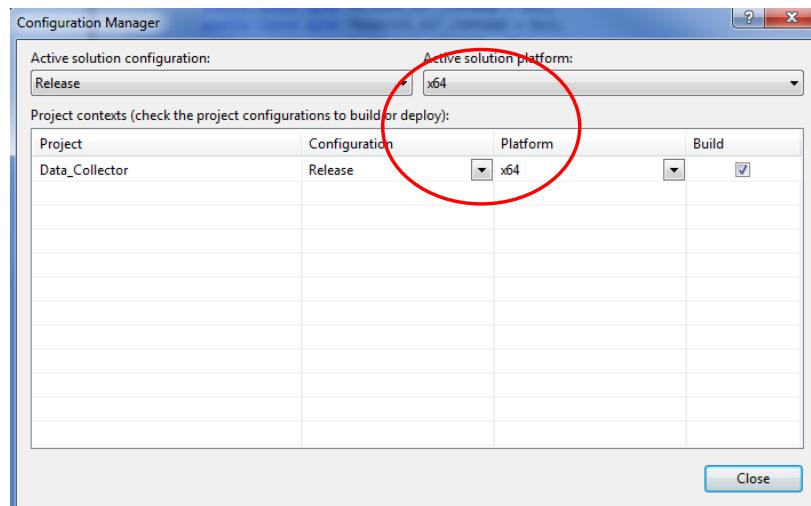
## FPGA Development System User Manual



In the New Solution Platform window, click on the drop down box under “Type or select the new platform:”. Select “x64”.

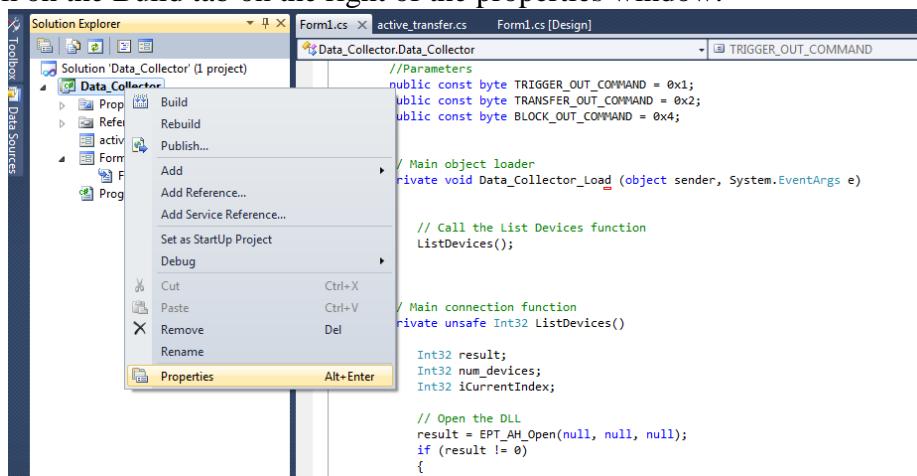


Click the Ok button. Verify that the “Active Solution Platform” and the “Platform” tab are both showing “x64”.

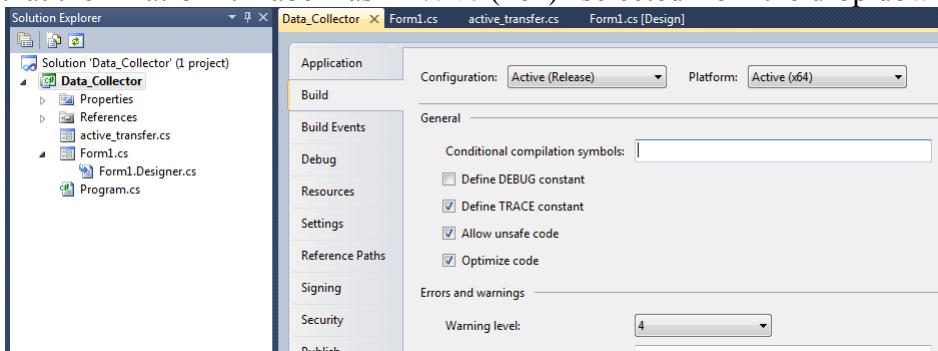


Click Close.

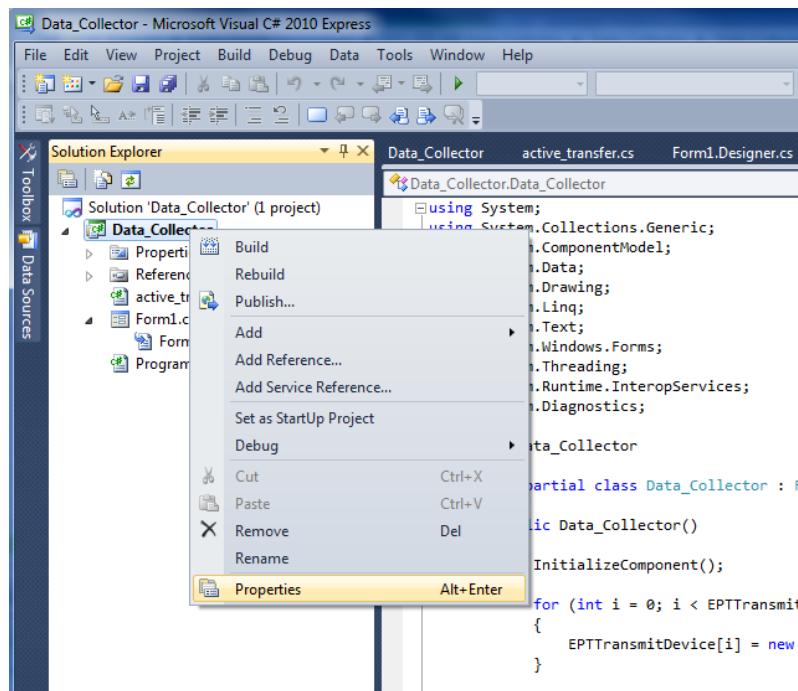
Then, using the Solution Explorer, you can right click on the project, select Properties and click on the Build tab on the right of the properties window.



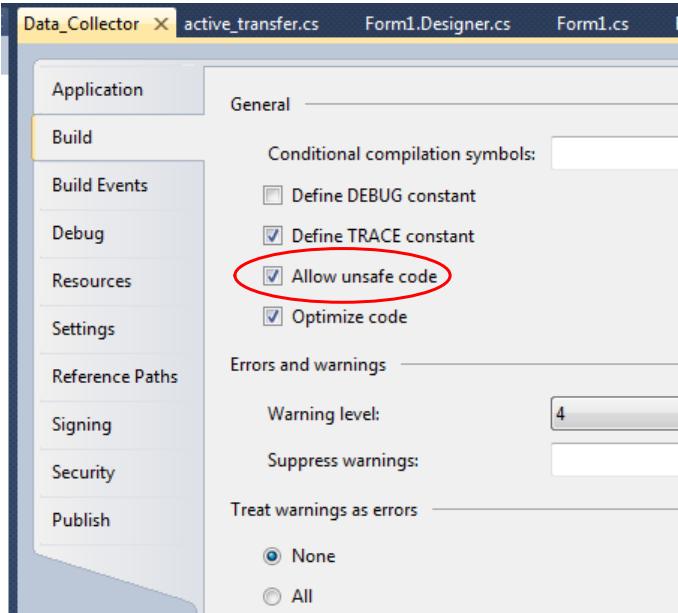
Verify that the "Platform:" label has "Active (x64)" selected from the drop down box.



Next, unsafe code needs to be allowed so that C# can be passed pointer values from the Active Host. Right click on the "Data Collector" project in the Solution Explorer. Select Properties.



Click on the Build tab and locate the “Allow unsafe code” check box. Check the box



Now we are ready to start coding.

Next, we add two classes for our device. One class stores the information useful for our device for Transmit to the EndTerms such as, address of module, length of transfer etc.



## FPGA Development System User Manual

```
//Create an array of the Transfer Class for device  
Transfer[] EPTTransmitDevice = new Transfer[8];
```

The next class is used to store parameters for receiving data from the device.

```
//Create a Receive object of the Transfer Class.  
Transfer EPTReceiveData = new Transfer();
```

EPT-4CE6-AF-D1

The first function called when the Windows Form loads up is the Data\_Collector\_Load(). This function is called automatically upon the completion of the Windows Form, so there is no need to do anything to call it. Once this function is called, it in turn calls the ListDevices().

```
// Main object loader  
private void Data_Collector_Load (object sender, System.EventArgs e)  
{  
// Call the List Devices function  
ListDevices();  
}
```

The ListDevices() function calls the EPT\_AH\_Open() function to load up the ActiveHost Dll. Next, it calls EPT\_AH\_QueryDevices() which searches through the registry files to determine the number of EPT devices attached to the PC. Next, EPT\_AH\_GetDeviceName() is called inside a for loop to return the ASCII name of each device attached to the PC. It will automatically populate the combo box, cmbDevList with all the EPT devices it finds.

```

// List Devices| function
private unsafe Int32 ListDevices ()
{
    Int32 result;
    Int32 num_devices;
    Int32 icurrentIndex;

    // Open the DLL
    result = EPT_AH_Open(null, null, null);
    if (result != 0)
    {
        MessageBox.Show("Could not attach to the ActiveHost library");
        return 0;
    }

    // Query connected devices
    num_devices = EPT_AH_QueryDevices();

    //Prepare the Combo box for population
    icurrentIndex = cmbDevList.SelectedIndex;
    cmbDevList.Items.Clear();

    // Go through all available devices
    for (device_index = 0; device_index < num_devices; device_index++)
    {
        String str;
        str = Marshal.PtrToStringAnsi((IntPtr)EPT_AH_GetDeviceName(device_index));
        cmbDevList.Items.Add(str);
    }
    return 0;
}

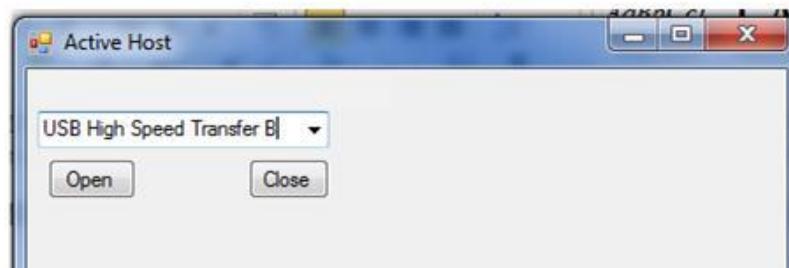
```

The user will select the device from the drop down combo box. This value can be sent to the OpenDevice() function using the button Click of the Open button.

```

// Open the device
if (EPT_AH_OpenDeviceByIndex(device_index) == false)
{
    printf("Could not open device %s\n", EPT_AH_GetDeviceName(device_index));
    exit(0);
}

```



The device\_index variable is used to store the index of the device selected from the combo box. This variable is passed into the EPT\_AH\_OpenDeviceByIndex(). This



## FPGA Development System User Manual

process is started by the user clicking on the “Open” button. If the function is successful, the device name is displayed in the label, labelDeviceCnt. Next, the device is made the active device and the call back function is registered using the RegisterCallBack() function. Finally, the Open button is grayed out and the Close button is made active.

```
// Open the device
public unsafe Int32 OpenDevice()
{
    device_index = (int)cmbDevList.SelectedIndex;
    if (EPT_AH_OpenDeviceByIndex(device_index) == 0)
    {
        String message = "Could not open device " +
            Marshal.PtrToStringAnsi((IntPtr)EPT_AH_GetDeviceName(device_index)) + ", " +
            Marshal.PtrToStringAnsi((IntPtr)EPT_AH_GetDeviceSerial(device_index));
        MessageBox.Show(message);
        return 0;
    }
    else
    {
        labelDeviceCnt.Text = "Connected to device " +
            Marshal.PtrToStringAnsi((IntPtr)EPT_AH_GetDeviceName(device_index)) + ", " +
            Marshal.PtrToStringAnsi((IntPtr)EPT_AH_GetDeviceSerial(device_index));
    }

    // Make the opened device the active device
    if (EPT_AH_SelectActiveDeviceByIndex(device_index) == 0)
    {
        String message = "Error selecting device: %s " +
            Marshal.PtrToStringAnsi((IntPtr)EPT_AH_GetLastError());
        MessageBox.Show(message);
        return 0;
    }

    // Register the read callback function
    RegisterCallBack();
    btnOpenDevice.Enabled = false;
    btnCloseDevice.Enabled = true;
    return 0;
}
```

Next, the callback function is populated. This function will be called from the Active Host dll. When the EPT Device has transferred data to the PC, the callback function will do something with the data and command.



## FPGA Development System User Manual

```
// Actual callback function which will read messages coming from the EPT device
unsafe void EPTReadFunction(Int32 device_id, Int32 device_channel, byte command, byte payload,
{
    byte* message = data;

    // Select current device
    EPT_AH_SelectActiveDeviceByIndex(device_id);

    //Add command and device_channel to the receive object
    EPTReceiveData.Command = ((command & COMMAND_DECODE) >> 3);
    EPTReceiveData.Address = device_channel;

    //Check if the command is Block Receive. If so,
    //use Marshalling to copy the buffer into the receive
    //object
    if (EPTReceiveData.Command == BLOCK_OUT_COMMAND)
    {
        EPTReceiveData.Length = data_size;
        EPTReceiveData.cBlockBuf = new Byte[data_size];

        Marshal.Copy(new IntPtr(message), EPTReceiveData.cBlockBuf, 0, data_size);
    }
    else
    {
        EPTReceiveData.Payload = payload;
    }
    EPTParseReceive();
}
```

Because the callback function communicates directly with the dll and must pass pointers from the dll to the C#, marshaling must be used. Marshaling is an advanced topic and will not be covered in this manual.

When EPTReadFunction() callback is called and passed parameters from the Active Host dll, it populates the EPTReceiveData object. It then calls EPTParseReceive() function. This function uses a case statement to call the TransferOutReceive() function.

```
private void EPTParseReceive()
{
    switch (EPTReceiveData.Command)
    {
        case TRANSFER_OUT_COMMAND:
            TransferOutReceive();
            break;
        default:
            break;
    }
}
```

TransferOutReceive() creates a string from the EPTReceiveData.Payload parameter. Then sends the string to the textbox, tbDataBytes.



## FPGA Development System User Manual

```
public void TransferOutReceive()
{
    string WriteRcvChar = "";
    WriteRcvChar = String.Format("{0}", (int)EPTReceiveData.Payload);
    tbDataBytes.AppendText(WriteRcvChar + " ");
}
```

Controls such as buttons are added to the Form1.cs[Design] window which allow turning on and off signals. These include

- btnWriteByte
- btnTransferReset
- btnOk
- btnClose
- btnResetBlock

Refer to section 1.6.4 Adding Controls to the Project for details about using the ToolBox to place controls on a design. The btnWriteByte click event calls the EPT\_AH\_SendTransferControlByte(). This function is used to turn on/off bits in the Control Register in the FPGA code. The btnWriteByte will set the start\_stop\_cntrl signal in the FPGA to one. This signal starts the Arduino Data Collector sending its random word to the FPGA.

```
private void btnWriteByte_Click(object sender, EventArgs e)
{
    int address_to_device;
    address_to_device = Convert.ToInt32(tbAddress.Text);
    EPT_AH_SendTransferControlByte((char)2, (char)1);
}
```

The btnTransferReset sets the start\_stop\_cntrl bit in the Control Register to zero. This action will cause the Arduino Data Collector to stop sending the random word to the FPGA.

```
private void btnTransferReset_Click(object sender, EventArgs e)
{
    int address_to_device;
    address_to_device = Convert.ToInt32(tbAddress.Text);
    EPT_AH_SendTransferControlByte((char)address_to_device, (char)0);
}
```

The btnResetBlock button will clear the tbDataBytes textblock. The Clear() method is inherited from the textbox class.

```
private void btnResetBlock_Click(object sender, EventArgs e)
{
    tbDataBytes.Clear();
}
```



## FPGA Development System User Manual

The btnOk and btnCancel buttons are used to end the application. It calls the function EPT\_AH\_CloseDeviceByIndex() to remove the device from the Active Host dll. The buttons btnOpen and btnCancel have their Enabled parameter set to true and false respectively. The Enabled parameter controls whether the button is allowed to launch an event or not. If it is not enabled, the button is grayed out. At the end of each click event, the Application.Exit() method is called. This exits the form.

```
private void btnOk_Click(object sender, EventArgs e)
{
    EPT_AH_CloseDeviceByIndex(device_index);
    btnOpenDevice.Enabled = true;
    btnCancelDevice.Enabled = false;

    lblDeviceConnected.Text = "";
    Application.Exit();
}

private void btnCancel_Click(object sender, EventArgs e)
{
    EPT_AH_CloseDeviceByIndex(device_index);
    btnOpenDevice.Enabled = true;
    btnCancelDevice.Enabled = false;

    lblDeviceConnected.Text = "";
    Application.Exit();
}
```

This is all that is needed for the Data Collector project. The Arduino will generate a random 8 bit word. It then transmits that word to the FPGA using the A0 (WRITE\_ENABLE) signal. The FPGA transmits the 8 bit word to the PC using the ACTIVE TRANSFER module of the Active\_Transfer Library. The dll reads the 8 bit word into local memory. It then calls the Callback function, EPTReadFunction. The 8 bit is finally displayed to screen using the MessageBox.Show().

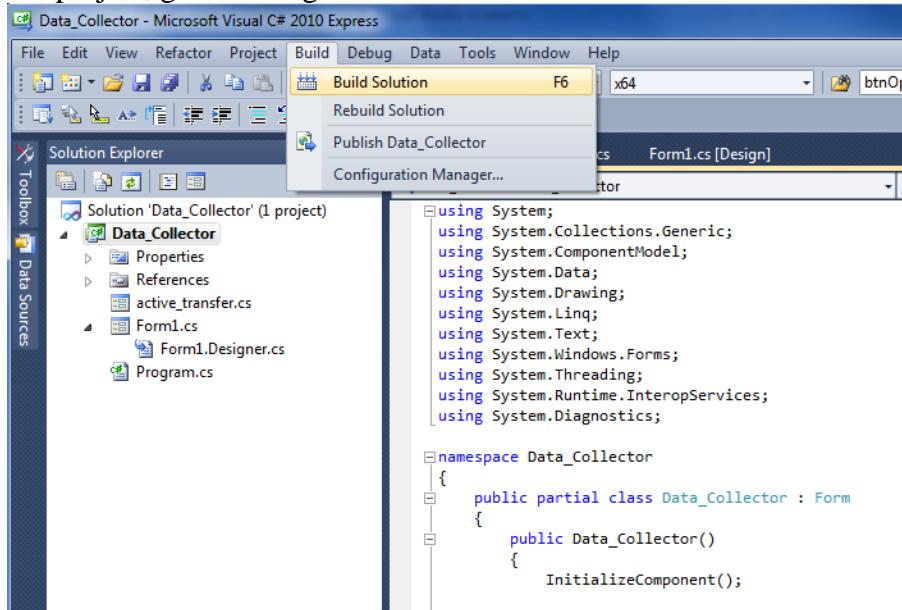
### 7.1.13 PC: Compiling the Active Host Application

Building the Data\_Collector project will compile the code in the project and produce an executable file. It will link all of the functions declared in the opening of the Data\_Collector Class with the Active Host dll. The project will also automatically link the FTD2XX.dll to the object code. Follow section: Assembling, Building, and Executing a .NET Project on the PC. Browse to the \Projects\_ActiveHost\_xxBit\EPT\_Data\_Collector\Data\_Collector\ folder of the EPT FPGA Development System DVD. Copy the following files into the project.

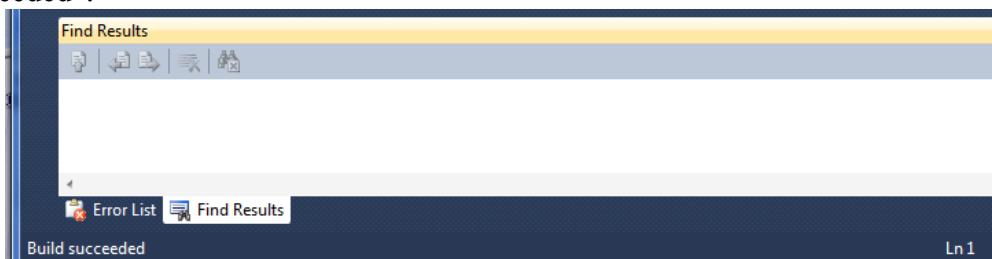
- Active\_transfer\_xxx.cs
- Data\_Collector.csproj

- Data\_Collector.csproj.user
- Form1.cs
- Form1.Designer.cs
- Program.cs

To build the project, go to Debug->Build Solution.



The C# Express compiler will start the building process. If there are no errors with code syntax, function usage, or linking, then the environment responds with “Build Succeeded”.



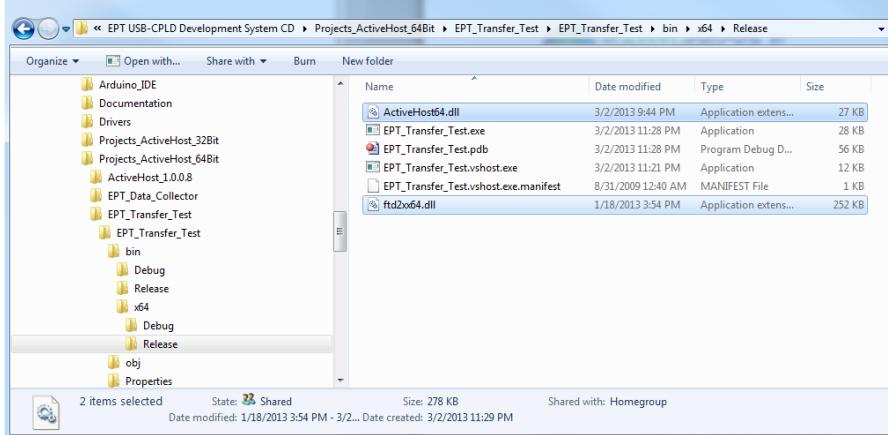
If the build fails, you will have to examine each error in the “Error List” and fix it accordingly. If you cannot fix the error using troubleshooting methods, post a topic in the Earth People Technology Forum. All topics will be answered by a member of the technical staff as soon as possible.

#### 7.1.14 Adding the DLL's to the Project

Locate the EPT FPGA Development System DVD installed on your PC. Browse to the Projects\_ActiveHost folder (choose either the 32 bit or 64 bit version, depending on whether your OS is 32 or 64 bit). Open the Bin folder, copy the following files:

- ActiveHostXX.dll

- ftd2xxXX.dll
- and install them in the bin\x64\x64 folder of your EPT\_Data\_Collector project.



Save the project.

At this point, the environment has produced an executable file and is ready for testing. Next, we will connect everything together and see it collect data and display it.

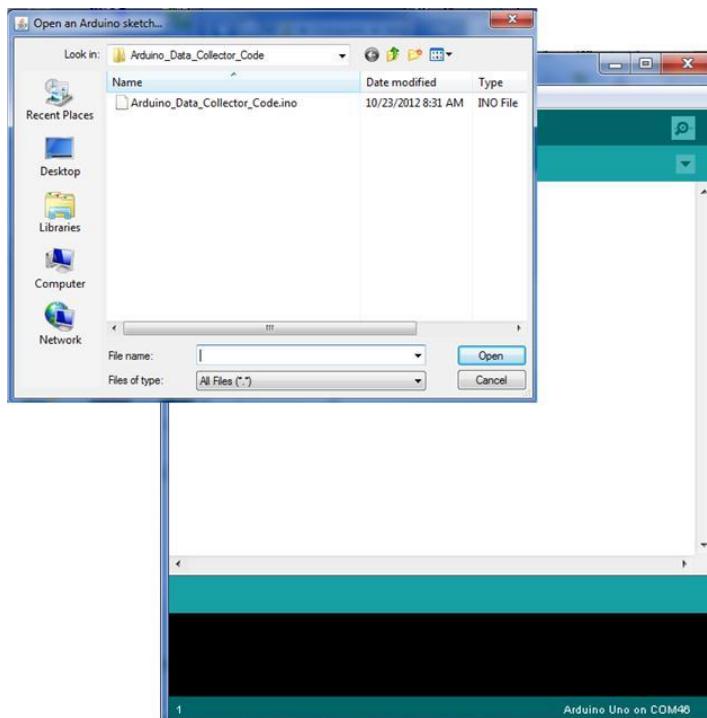
### 7.1.15 Connecting the Project Together

Now we will connect the Arduino, EPT-4CE6-AF-D1, and the PC to make a Data Collector. First, connect a USB cable from a USB port on the PC to the Arduino. Second, connect a USB cable from an open USB port on the PC to the EPT-4CE6-AF-D1.

Next, open the Arduino IDE and select File->Open and select your sketch created earlier, Arduino\_Data\_Collector\_Code\_U2.ino.



## FPGA Development System User Manual



Select the file and click Open. The sketch will now populate the Arduino IDE window. Compile and Download the sketch into the Arduino microcontroller using the Upload button.



## FPGA Development System User Manual

```
/*
Copyright Earth People Technology Inc. 2012

Data Collector Random Seed

Platform: EPT-570-AP-U2

*/
int count = 0;
int ledPin = 13;

void setup()
{
    DDRD = B1111111; //Set Port D as outputs
    PORTD &= B1111111; //Turn on Port D pins

    pinMode(A0, OUTPUT);
    pinMode(ledPin, OUTPUT);
}

void loop()
{
    count++;
    if(count > 100)
        count = 0;
    if(count % 2 == 0)
        digitalWrite(ledPin, HIGH);
    else
        digitalWrite(ledPin, LOW);
}
```

The Arduino IDE will compile the project, then transmit the machine level code into the ATmega328 SRAM to start the program. When this is complete, the Yellow L LED will blink about once per second.

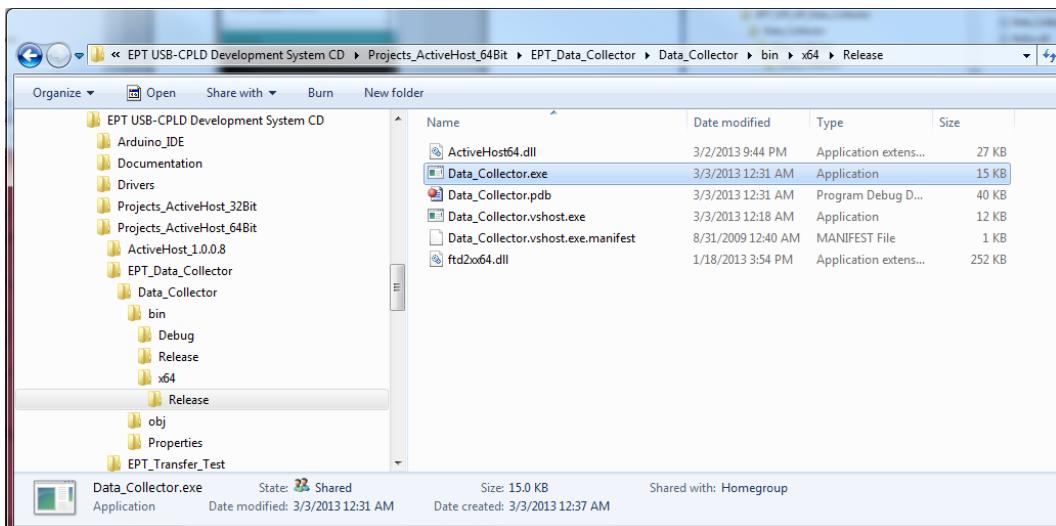
If this LED is blinking at the rate of once per second, the Arduino and the Data Collector project are ready for the EPT 570-AP-U2 code.

The FPGA should already be programmed with its Data Collector Project. If it isn't, follow the instructions in section 3.1.10.

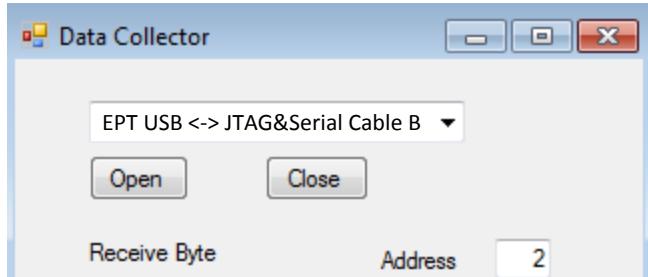
Open the EPT Data Collector on the PC by browsing to the Data Collector project folder. Locate the executable in the \bin\x64\Release folder.



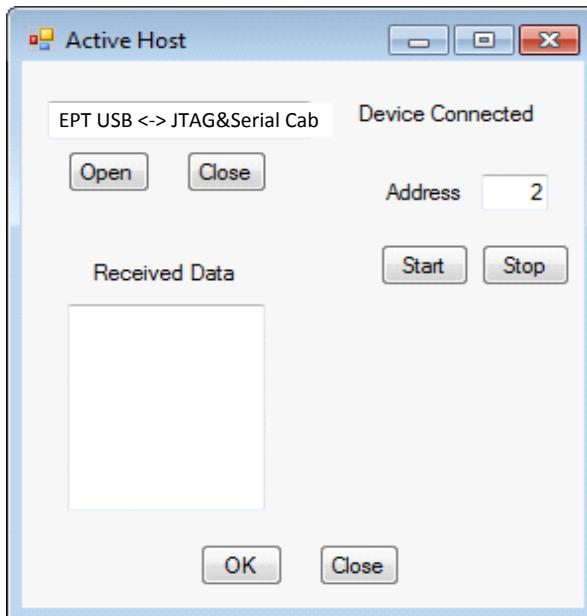
## FPGA Development System User Manual



Initiate the application by double clicking the application icon in the \Release folder of the project. The application will open and automatically load the Active Host dll. The application will locate the EPT 570-AP-U2 device. Next, the combo box at the top will be populated with the name of the device.

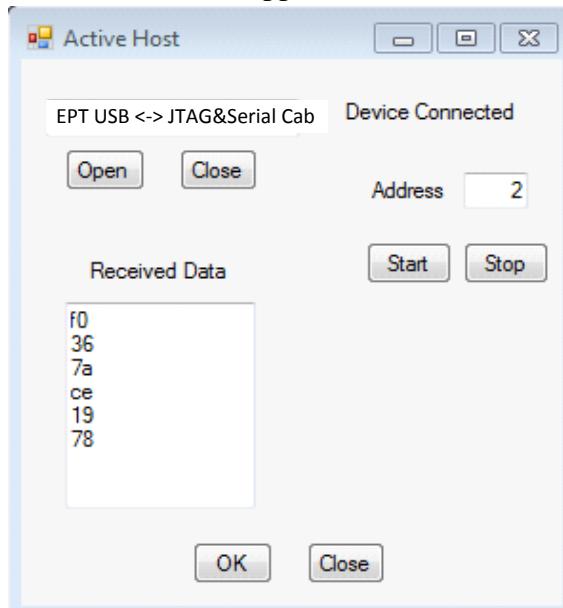


Select the EPT 570-AP device and click the Open button. If the Active Host application connects to the device, a label will indicate “Device Connected”. Next, select the address of the Active Transfer module in the FPGA. In our case it is “2”.



### 7.1.16 Testing the Project

To test our Data Collector project, just click on the Start button. As soon as the device connects, the data from the Arduino will appear in the received data textBox.



And that's all there is to the Data Collector Project. It's up to the user to use this project as a base to create much larger projects. You can easily make a volt meter using this project by turning off the Random number generator in the Arduino and reading the



## FPGA Development System User Manual

Analog Pins. Also, reformat the textBox display that it shows in decimal instead of the Hexadecimal display.



## FPGA Development System User Manual

### APPENDIX I

#### List of Abbreviations and Acronyms

EPT	Earth People Technology
FIFO	First In – First Out
FTDI	Future Technology Device International
HSP	Hyper Serial Port
I2C	Inter-Integrated Circuit
JTAG	Joint Test Action Group
PC	Personal Computer
FPGA	Complex Programmable Logic Device
USB	Universal Serial Bus

### APPENDIX II

#### Details of the Altera EPM570 FPGA