DUEPROLOGIC

FPGA DEVELOPMENT SYSTEM
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 Prime 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.

http://www.earthpeopletechnology.com/

Table of Contents

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.

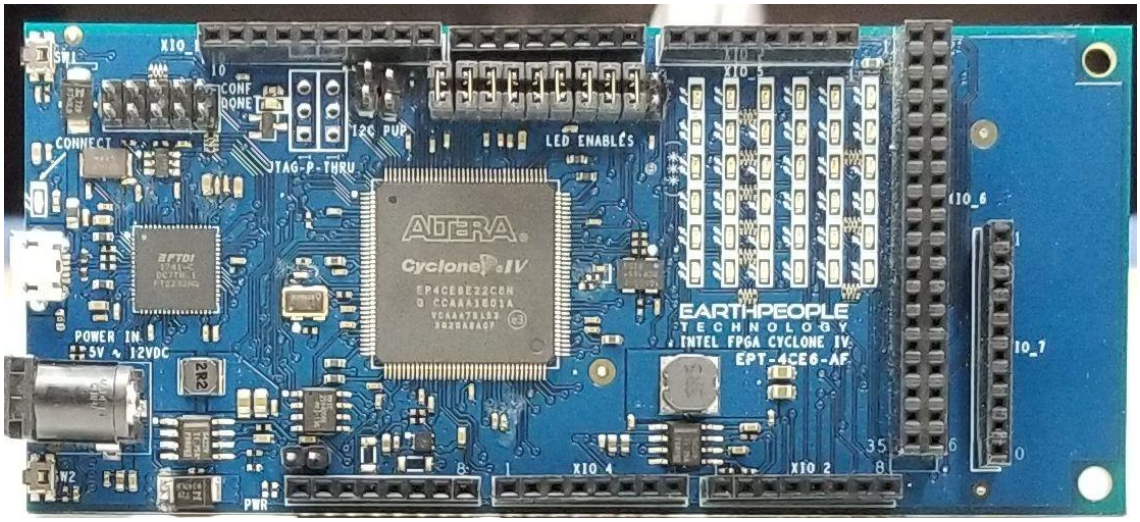
With the DPL's FPGA, projects can also more easily be attempted which rely on asynchronous, exceedingly fast, and even multiple separate concurrent logic structures operating in parallel which would have traditionally required a plethora of chip gates or multiple high speed microprocessors to implement 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.

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 .

1.1 Test Driving the Active Host Test Application

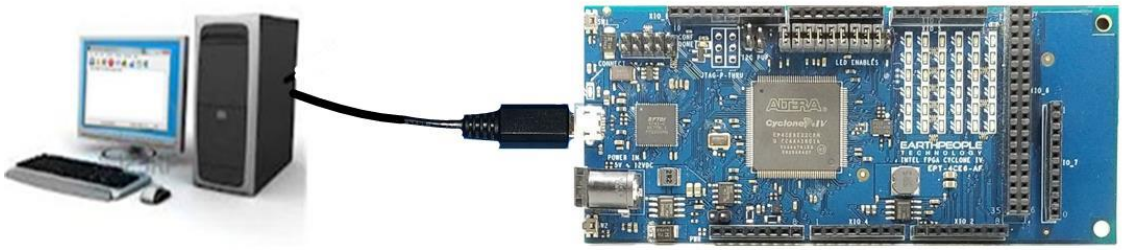
The



EPT-4CE6-AF-D2 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.

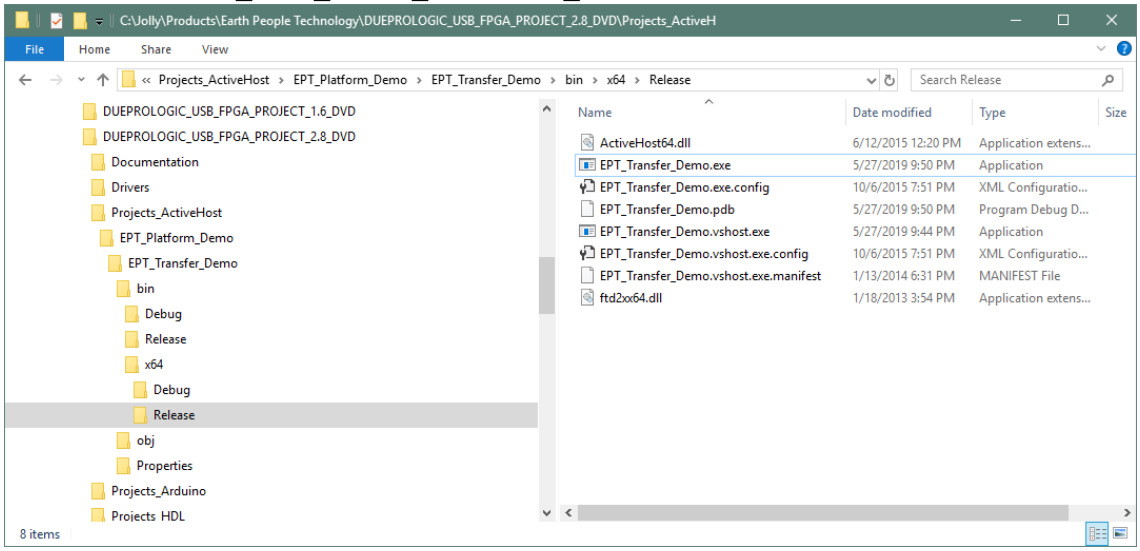
To test drive the application, connect the DPL to the Windows PC using a Micro B USB cable. Load the driver for the board. See the section "EPT Drivers" for instructions

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.

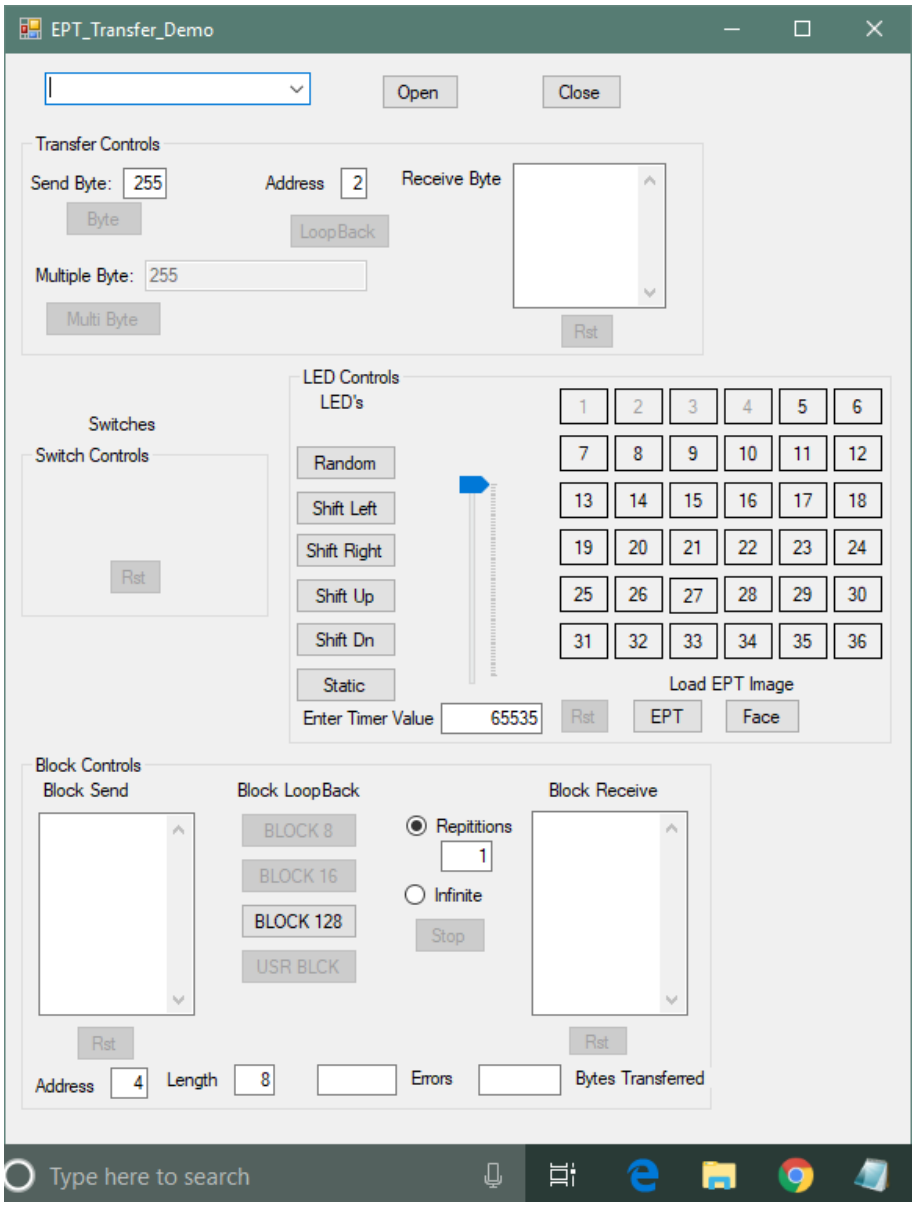


Next, open a Windows Explorer browser.

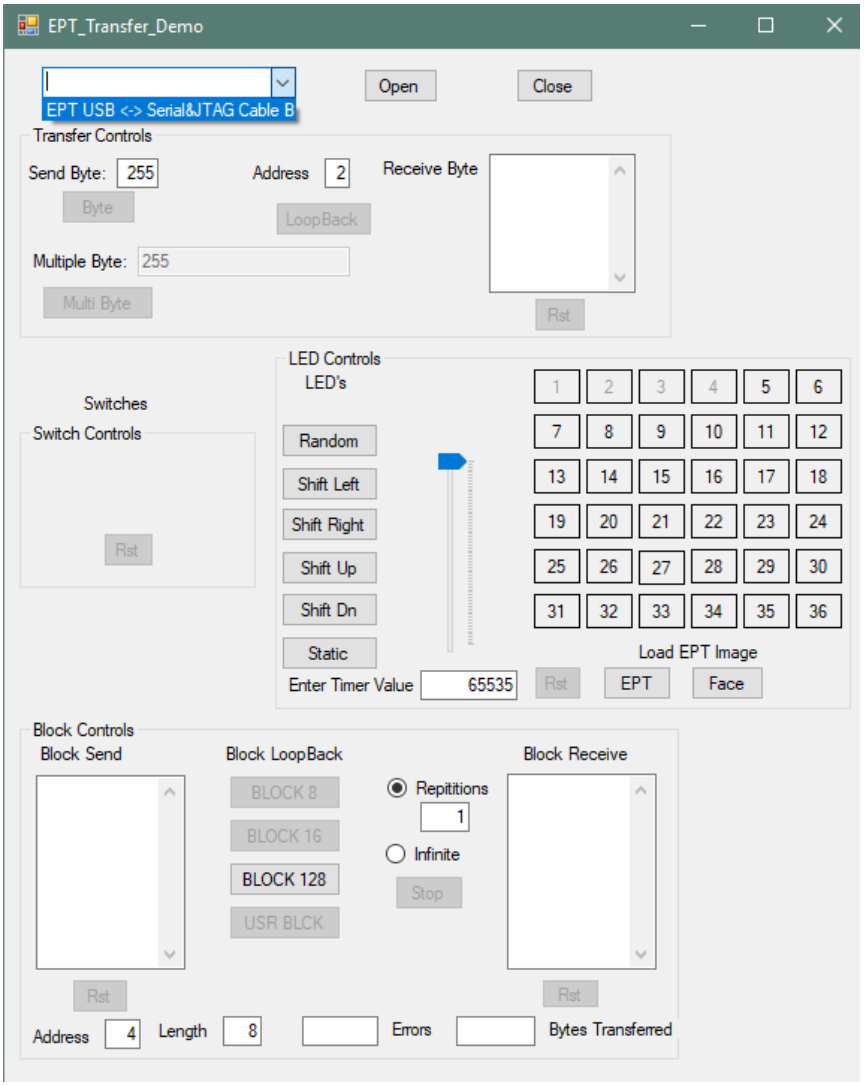
Browse to the Projects_ActiveHost\EPT_Platform_Demo\EPT_Transfer_Demo\bin\X64\Release\ folder on the DPL_USB_FPGA_PROJECT_DVD.



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.



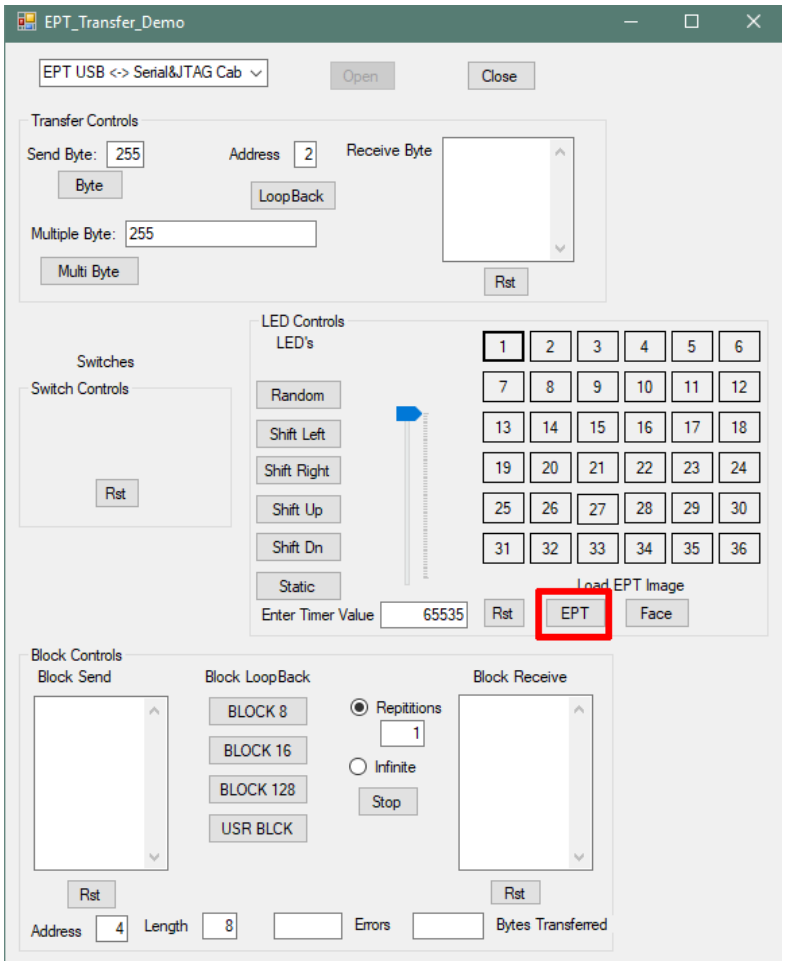
Leave the Address set at 2 for the Transfer Controls Group. And, leave the Address set at 4 for the Block Controls Group.

Click on one of the LED buttons in the middle of the window. The corresponding LED on the EPT-4CE6-AF-D2 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.

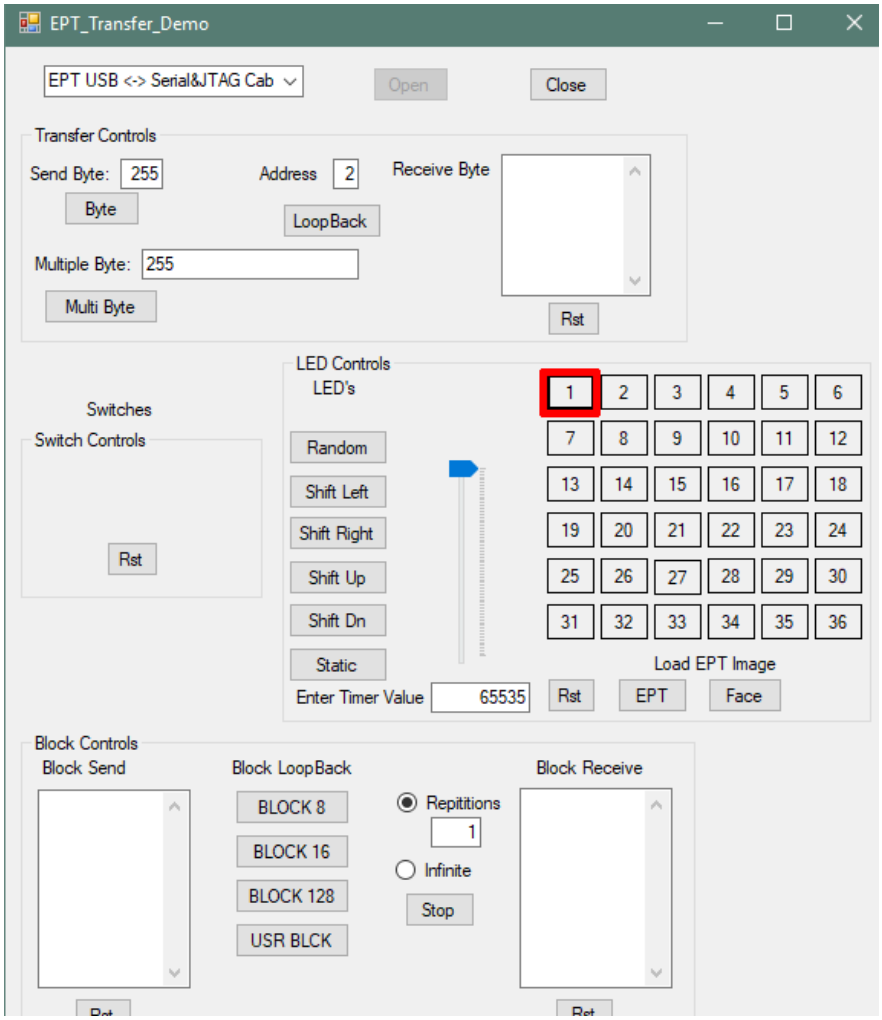
To exercise the Block Transfer EndTerm, click the “BLOCK 8”, “BLOCK 16” or “BLOCK 128” button in the Block Controls group. A pre-selected group of numbers appear in the Block Receive textbox.

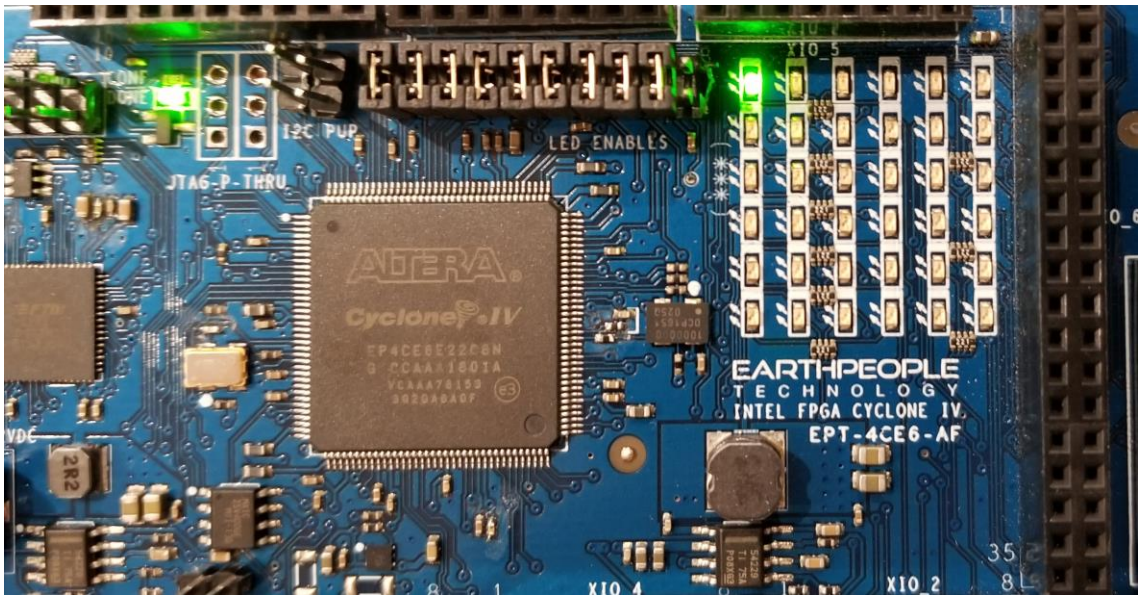
Under LED Controls, press the “EPT” button under the “Load EPT Image”.



The characters “EPT” will be displayed on the 6x6 LED array. All the characters cannot be displayed at once, so the “Shift Left” or “Shift Right” button must be pressed to see all characters.

Next press any of the numbered buttons in the “LED Controls” group. This will toggle the corresponding LED in the 6x6 LED Array. It performs this operation as a Block Write. So, an entire LED frame will be transferred to the DueProLogic each time a button is pressed.


EPT_Transfer_Demo
EPT USB <-> Serial&JTAG Cab
Open
Close
Transfer Controls
Send Byte: 255
Byte
Address 2
LoopBack
Receive Byte
Multiple Byte: 255
Multi Byte
Rst
LED Controls
LED's
Random
Shift Left
Shift Right
Shift Up
Shift Dn
Static
1 2 3 4 5 6
7 8 9 10 11 12
13 14 15 16 17 18
19 20 21 22 23 24
25 26 27 28 29 30
31 32 33 34 35 36
Load EPT Image
Enter Timer Value 65535
Rst
EPT
Face
Switches
Switch Controls
Rst
Block Controls
Block Send
Block LoopBack
BLOCK 8
BLOCK 16
BLOCK 128
USR BLCK
Repititions
1
Infinite
Stop
Block Receive

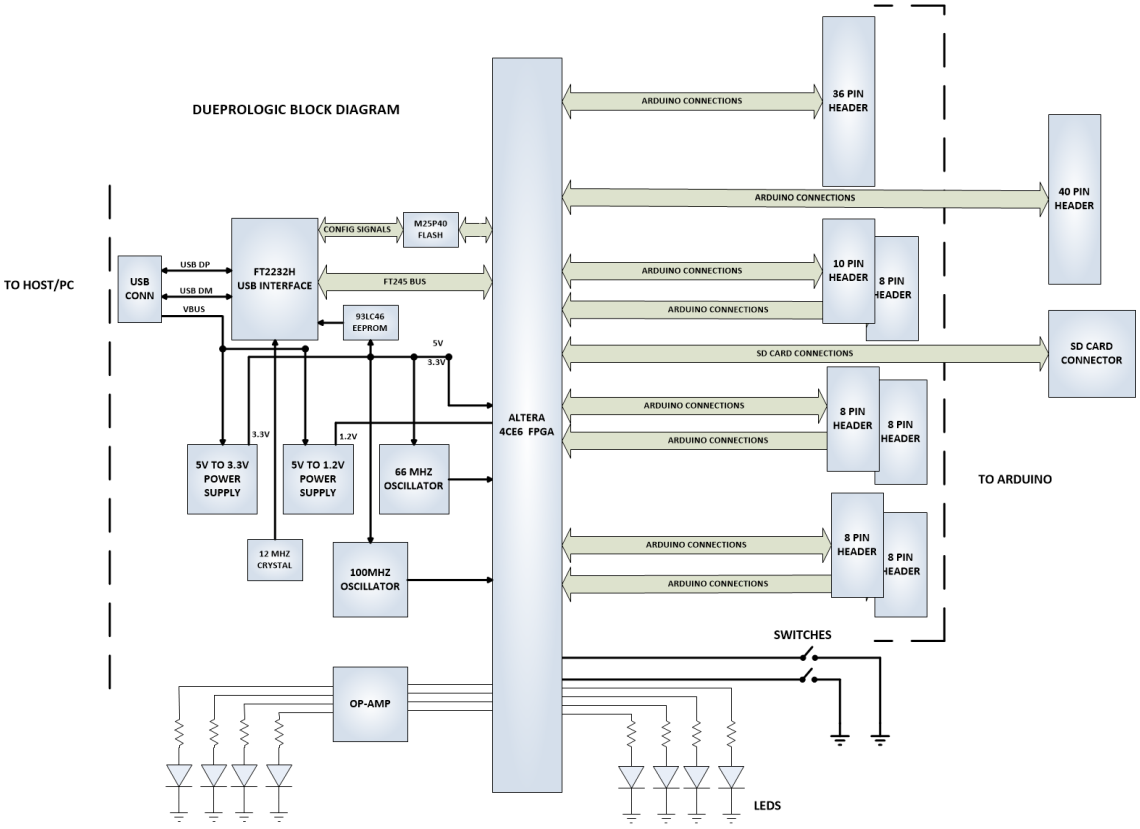


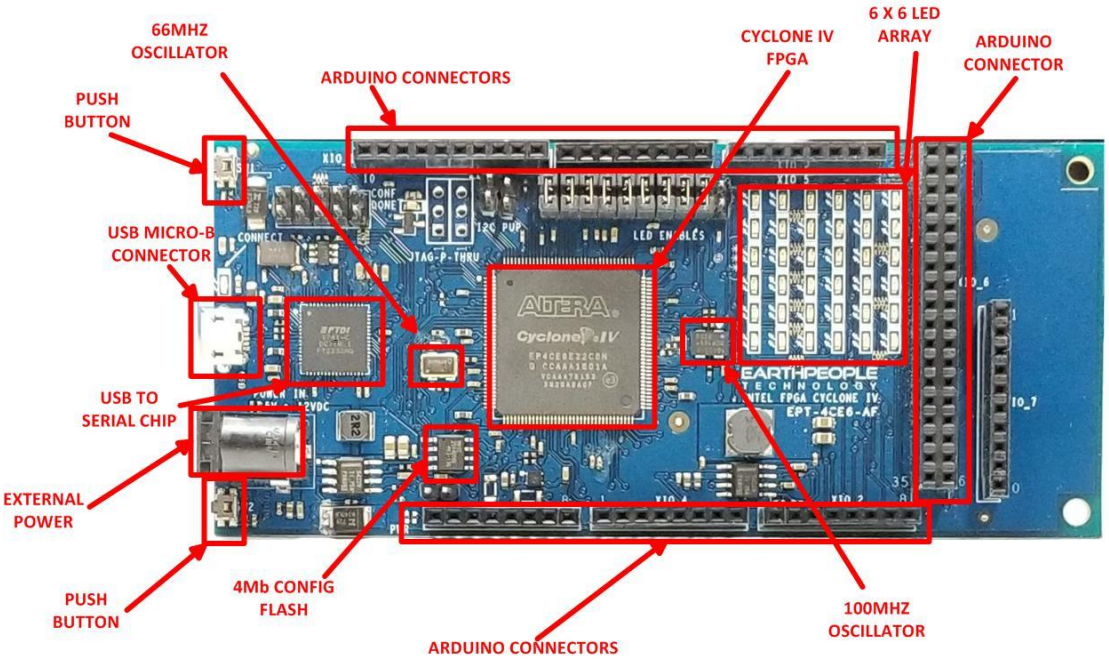
1.2 EPT-4CE6-AF-D2

The EPT-5M57-AP-U2 board is equipped with an Altera EP4CE6E22C8 FPGA; which is programmed using the Altera Quartus Prime 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)
- 36 Green LED Array accessible by the user
- Two PCB switches accessible by the user
- I/O connectors stack into the Arduino Due

EPT-4CE6-AF-D2


DUEPROLOGIC BLOCK DIAGRAM
TO HOST/PC
USB CONN
USB DP
USB DM
VBUS
FT2232H USB INTERFACE
CONFIG SIGNALS
M25P40 FLASH
FT245 BUS
93LC46 EEPROM
5V
3.3V
1.2V
5V TO 3.3V POWER SUPPLY
5V TO 1.2V POWER SUPPLY
66 MHZ OSCILLATOR
12 MHZ CRYSTAL
100MHZ OSCILLATOR
OP-AMP
ALTERA 4CE6 FPGA
ARDUINO CONNECTIONS
36 PIN HEADER
40 PIN HEADER
10 PIN HEADER
8 PIN HEADER
SD CARD CONNECTIONS
SD CARD CONNECTOR
TO ARDUINO
SWITCHES
LEDS



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

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-D2 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 Prime 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

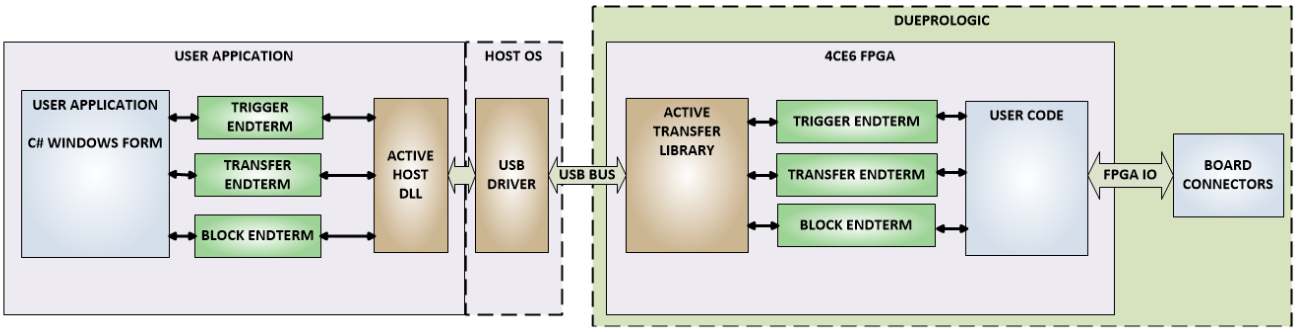
The DPL includes a standard SD Card Interface. The SD connector is on the bottom of the DPL. This interface allows the user to add expansion memory or the standard SD interface.

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 Prime.

1.4 Active Host EndTerms

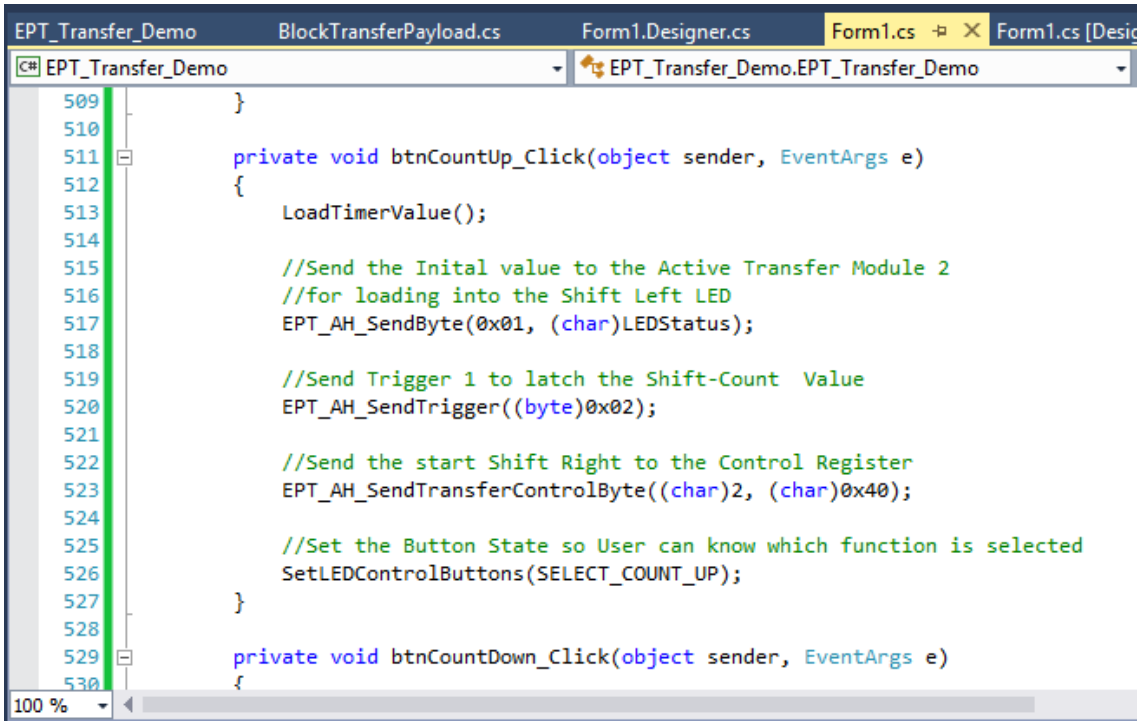
The Active Host SDK is provided as a dll which easily interfaces to application software written in C#, C++ or C. It runs on the PC and provides transparent connection



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 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         );
```

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.

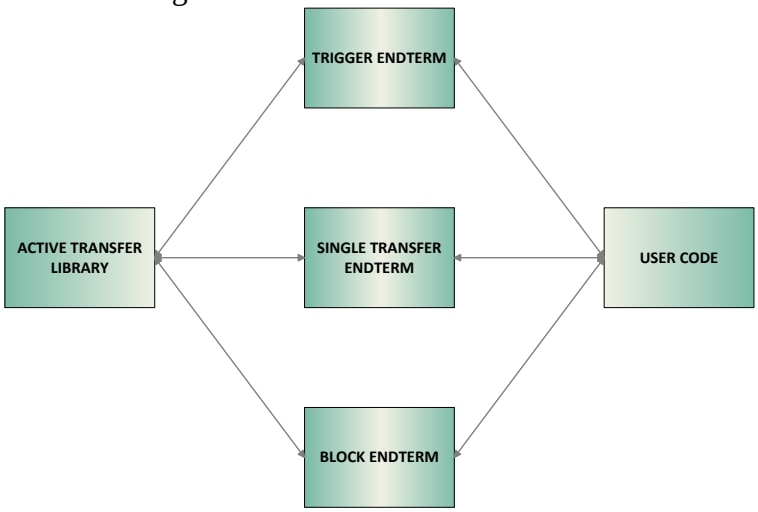
1.5 Active Transfer EndTerms

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.

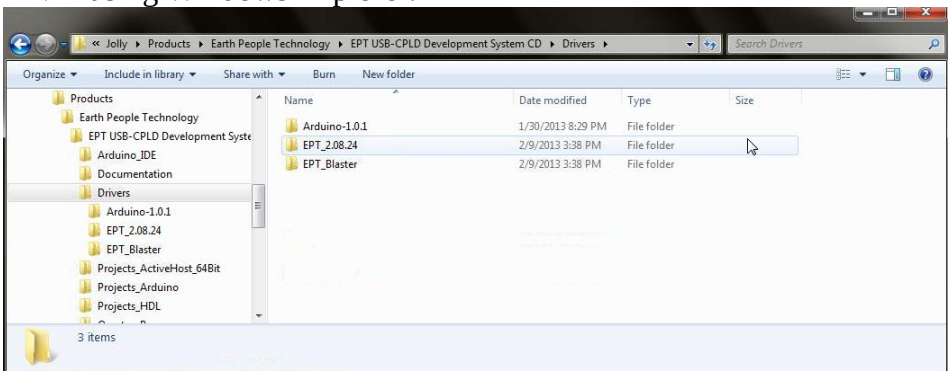
2 EPT Drivers

The EPT FPGA Development system requires drivers for any interaction between PC and the EPT-4CE6-AF-D2. 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.

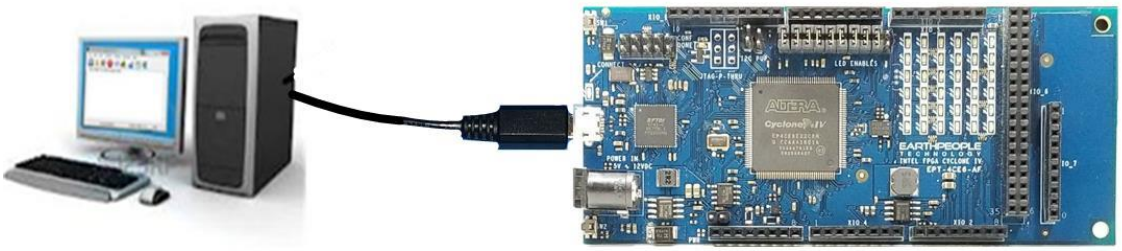
2.1 USB Driver

The EPT-4CE6-AF-D2 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.

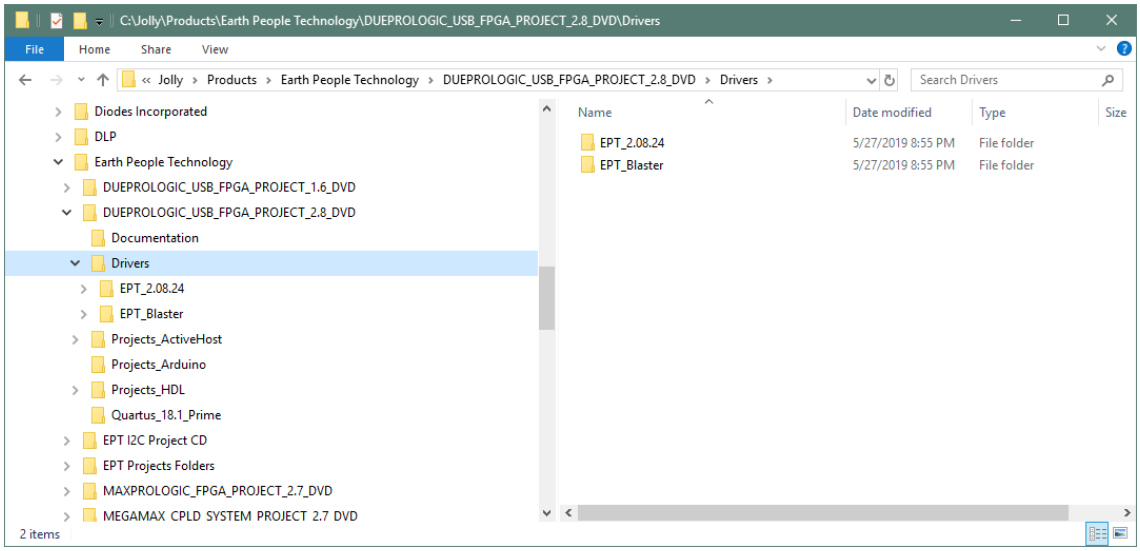
Locate the EPT_2.08.24 folder in the Drivers folder of the EPT FPGA Development System DVD using Windows Explorer.



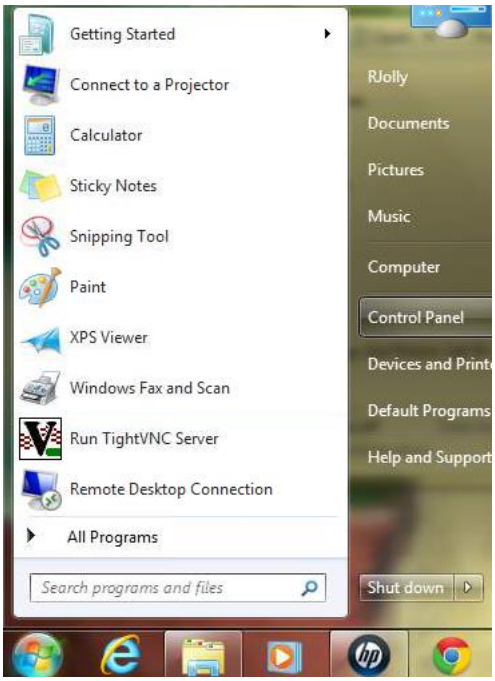
Plug in the EPT-4CE6-AF-D2 device into an available USB port.



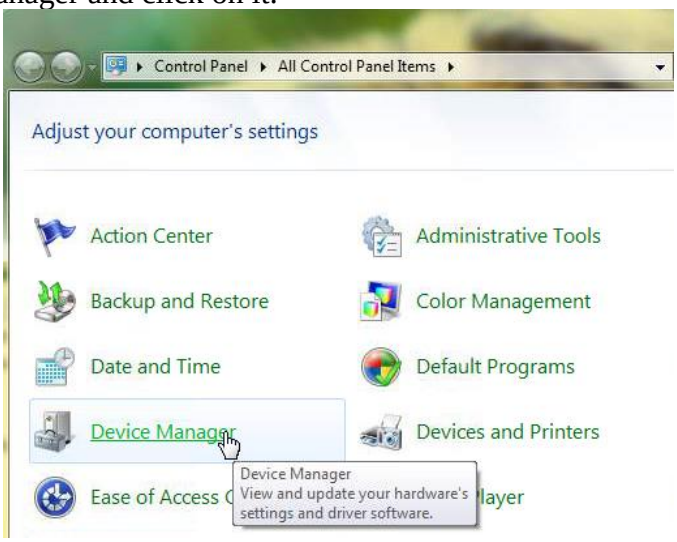
Windows will attempt to locate a driver for the USB device. When it does not find one, it will report a error, “Device driver software was not successfully installed”. Ignore this error.



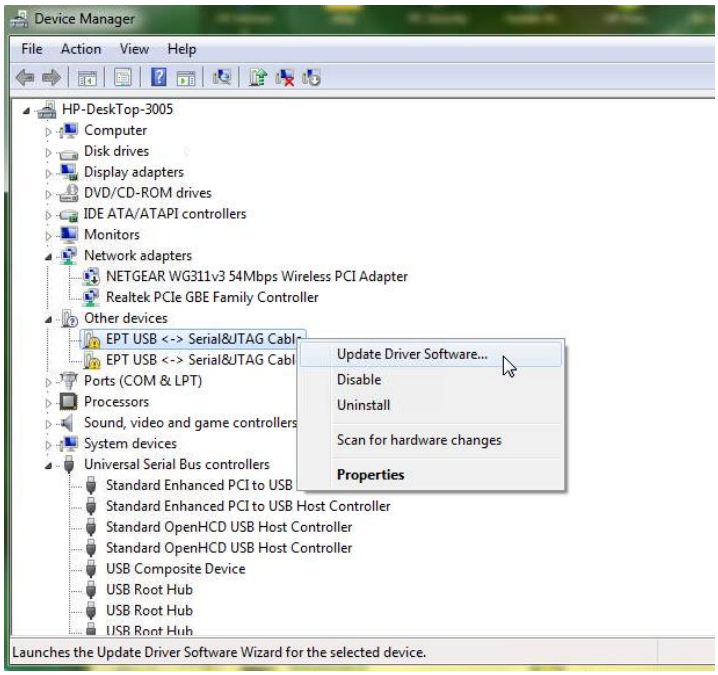
Go to Start->Control Panel



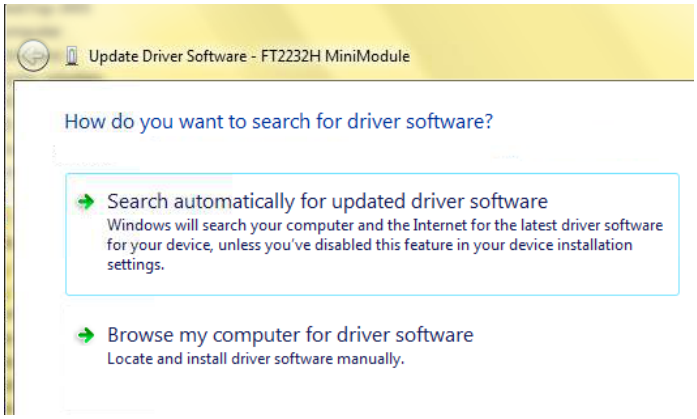
Locate Device Manager and click on it.



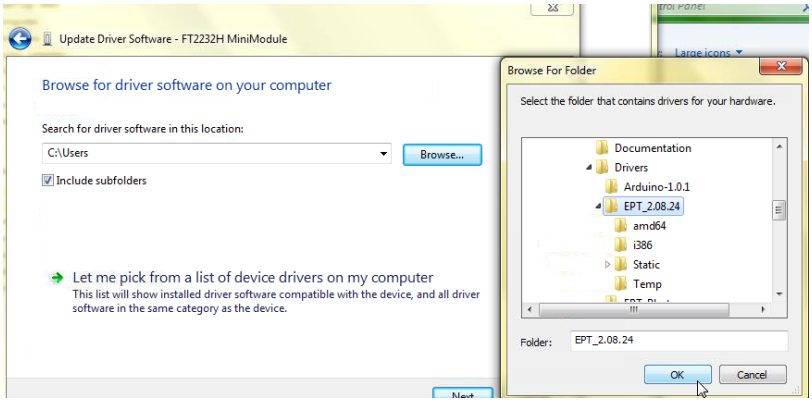
Locate the entry under “Other devices”. Right click “EPT USB <->Serial&JTAG Cable” and select “Update Driver Software…”.



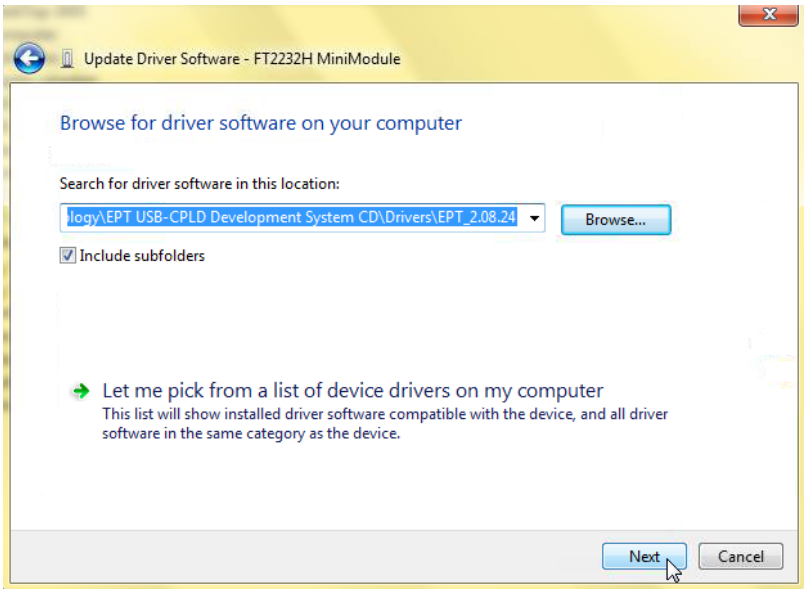
At the Update Driver Software Window, select “Browse my computer for driver software”.



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.



Click the Next button



3 The next window is the Windows Security notice. The EPT driver is not signed by Windows. How to Disable Driver Signature Verification on 64-Bit Windows 10

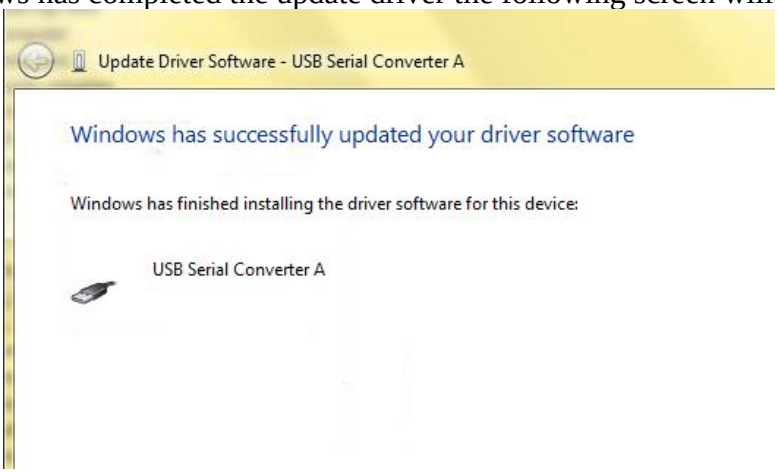
Windows 10 implements extra protection against malicious driver files that intend to do harm to the users PC. This implementation unfortunately locks out any third party driver file if it is unsigned. Currently all EPT boards have unsigned drivers. Follow the instructions below to allow Windows 10 to allow third party unsigned drivers to be installed on your PC.

- Under Windows 10, You must restart the computer in the “options menu” mode. This will allow you to go through the process of allowing all unsigned drivers to self install on your Windows 10.

- First, the computer must be restarted into the so-called “options menu”. The easiest way to get there is via the “Run”-dialog, which is opened by means of the key combination Win+R. The command you have to enter, to boot into the Options menu is as follows:
- shutdown.exe /r /o /f /t 00 Caution: This command starts the reboot process immediately!
- You’ll find the explanation of the individual parameters below:
    - shutdown.exe – It’s a Command-Line application which is inclusive with Windows. It does various kinds of restarts and shutdowns.
    - /r – means “restart”
    - /o – means “the PC should start in the Option menu”
    - /f – means “restart directly and close all opened programs immediately”
    - /t 00 – shows the time until the restart happens (in seconds). In this case 0 seconds, which equals an immediate restart Then click on the Recovery option on the left hand side.
- After you have successfully rebooted into the “Options menu”, click “Troubleshoot” and then “Advanced options”.
- Now click on the “startup settings”-button and the press “reboot”.
- After another reboot, you’re at the startup settings page. Here you can choose between various options, which can be entered by pressing the respective number key. For our needs, you have to click option 7 – “Disable driver signature enforcement”. This deactivates the driver check and enables you to install unsigned drivers in Windows 10.
- In the last step and after another restart, you are able to install unsigned drivers by using Windows “Device Manager”. Don’t get confused. There still will be a question in the beginning, as you can see in the screenshot above, but nevertheless the installation of the driver will be possible without problems.

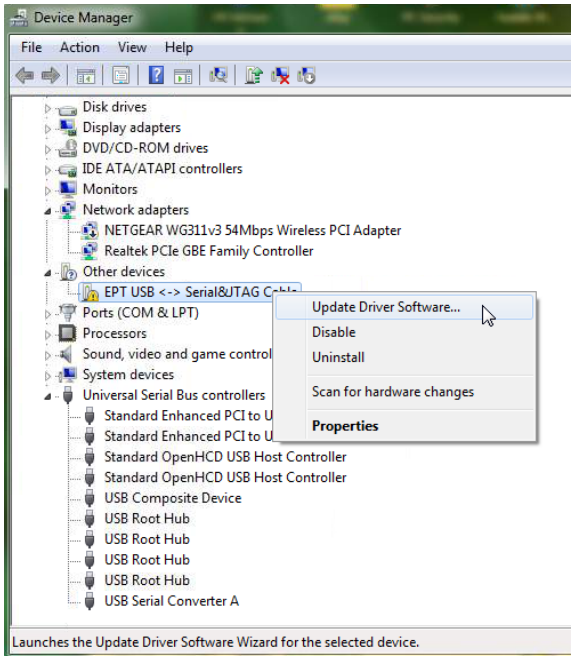
After these tasks are successfully completed, you can proceed with the following sections.
Windows will add the EPT_2.08.24 driver to the System Registry.

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

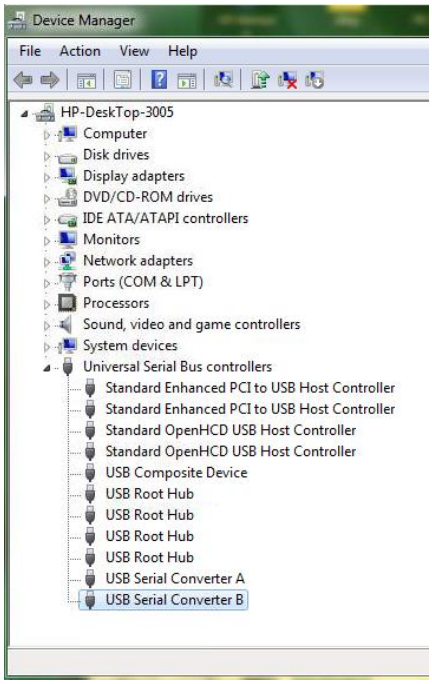


Channel A of the EPT-4CE6-AF-D2 is ready for use.

Next, repeat the process for Channel B.



The driver files will automatically install in the System Registry.



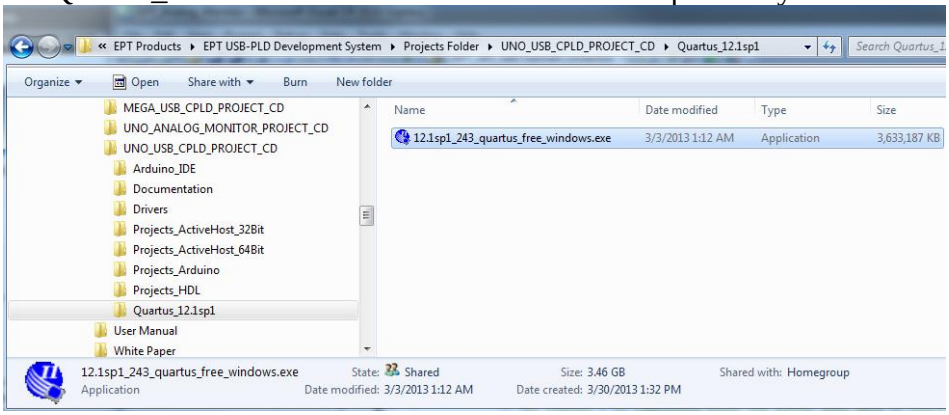
When this is complete, the drivers are installed and the EPT-4CE6-AF-D2 can be used with for programming and USB data transfers.

3.1 JTAG DLL Insert to Quartus Prime

The JTAG DLL Insert to Quartus Prime allows the Programmer Tool under Quartus to recognize the EPT-4CE6-AF-D2. The EPT-4CE6-AF-D2 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 7, 8, and 10 64-bit.

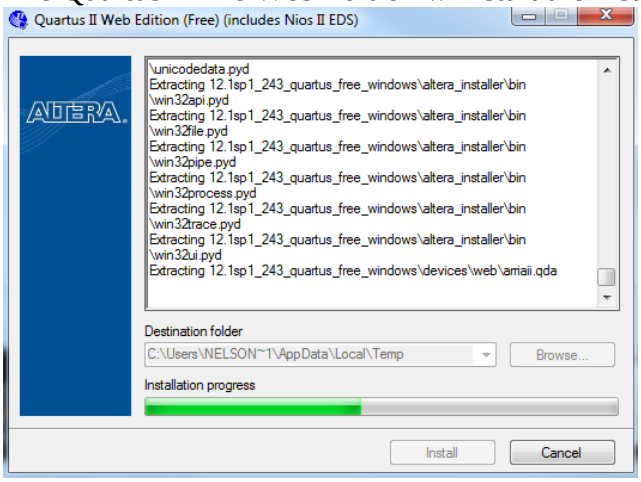
3.1.1 Installing Quartus

Locate the Quartus_Prime folder on the EPT FPGA Development System DVD.



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

If you don't need to download Quartus, double click on the Prime_xxx_quartus_free_widows.exe (the xxx is the build number of the file, it is subject to change). The Quartus Prime 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.

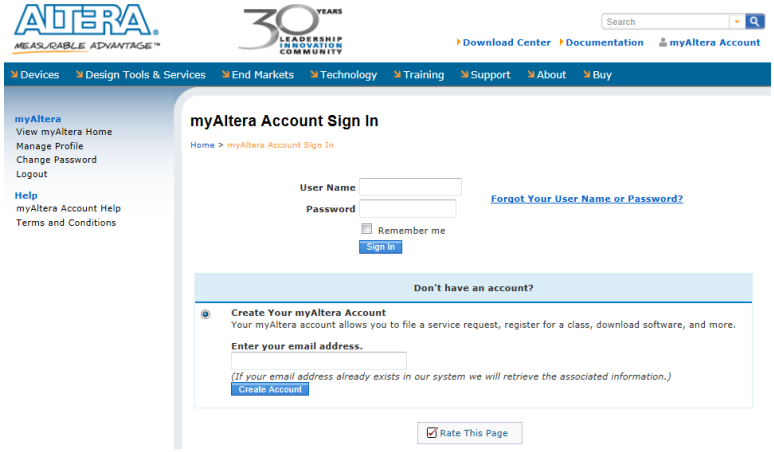
3.1.2 Downloading Quartus

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:

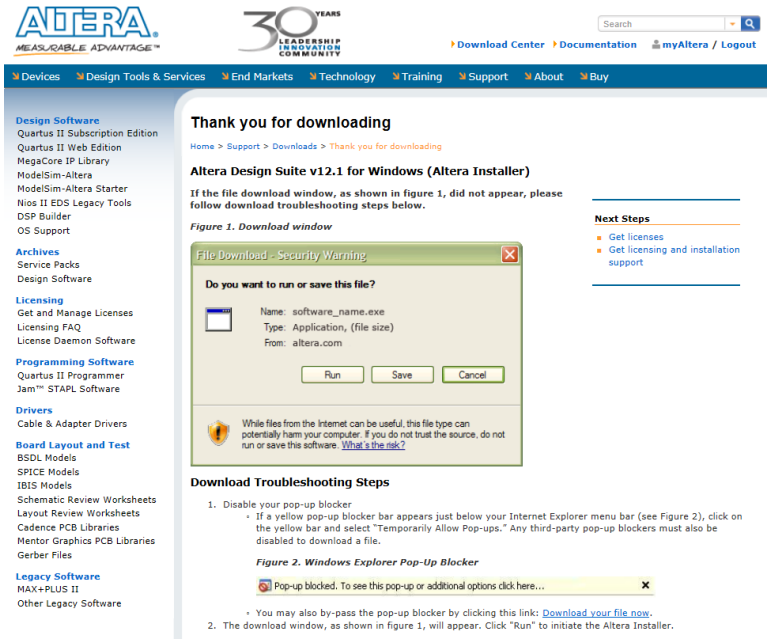
https://www.altera.com/download/dnl-index.jsp

Click on the Download Windows Version.

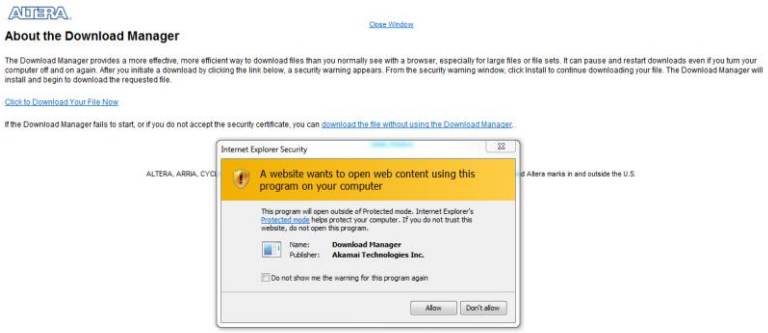
The next page will require you to sign into your "myAltera" account. If you do not have one, follow the directions under the box, "Don't have an account?"



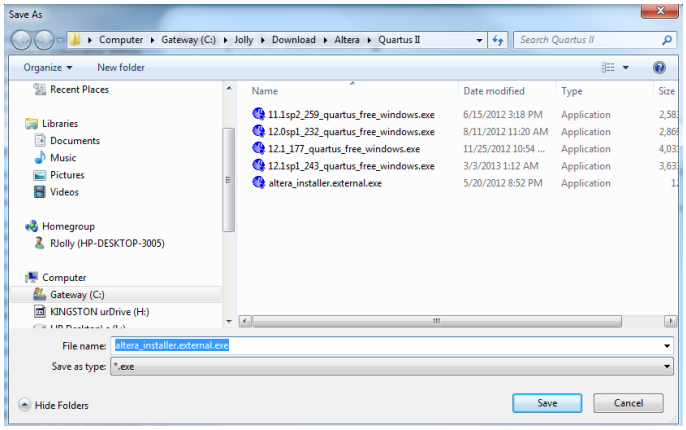
Once you have created your myAltera account, enter the User Name and Password. The next window will ask you to allow pop ups so that the file download can proceed.



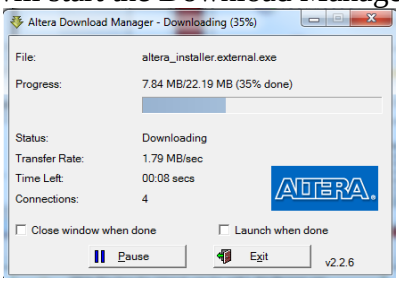
Click on the “Allow Once” button. The next window will appear. It is the Download Manager.



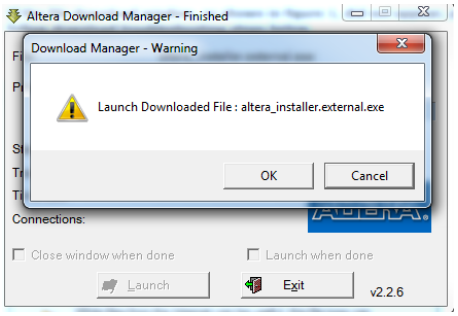
Click on the ”Allow” button. This will bring up the Save As dialog box. Save the altera_installer.external.exe to a download file.



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



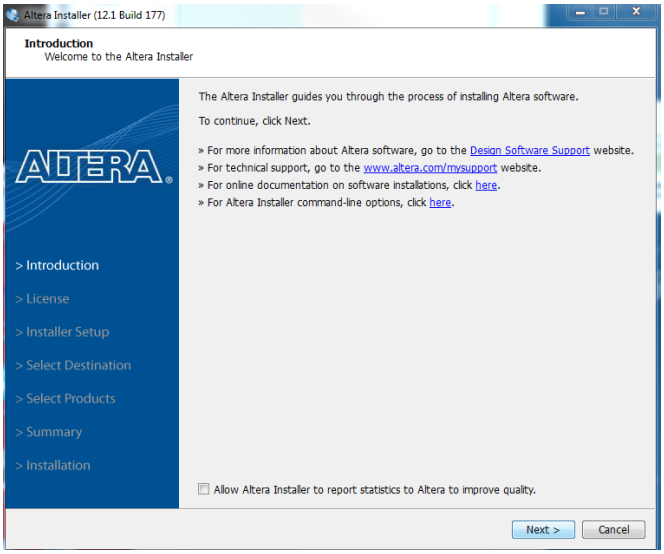
When it finishes, click the "Launch" button.



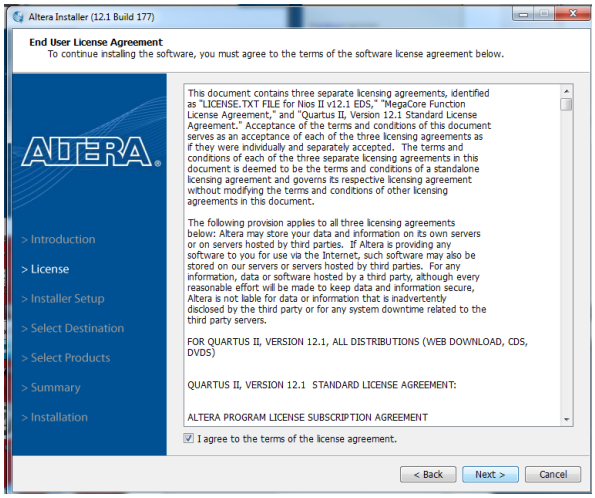
Click "Ok" and "Yes" to the following screen.

3.1.3 Quartus Installer

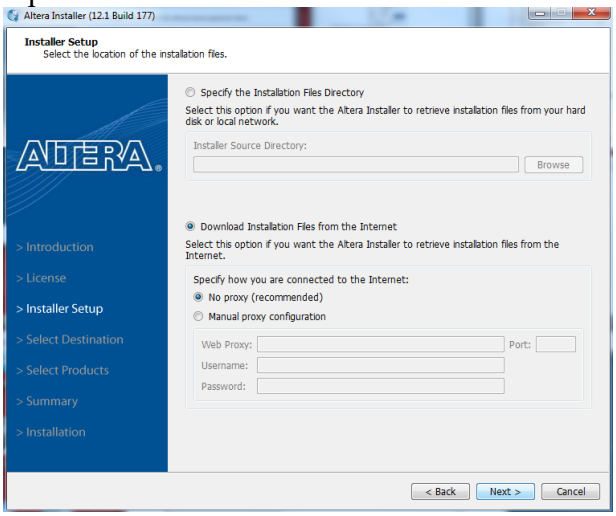
Click "Next" on the Introduction Window.



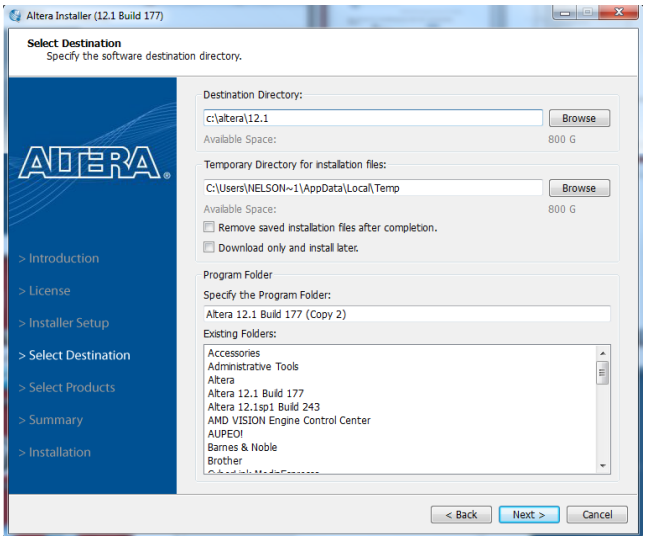
Click the checkbox to agree to the license terms. Then click "Next".



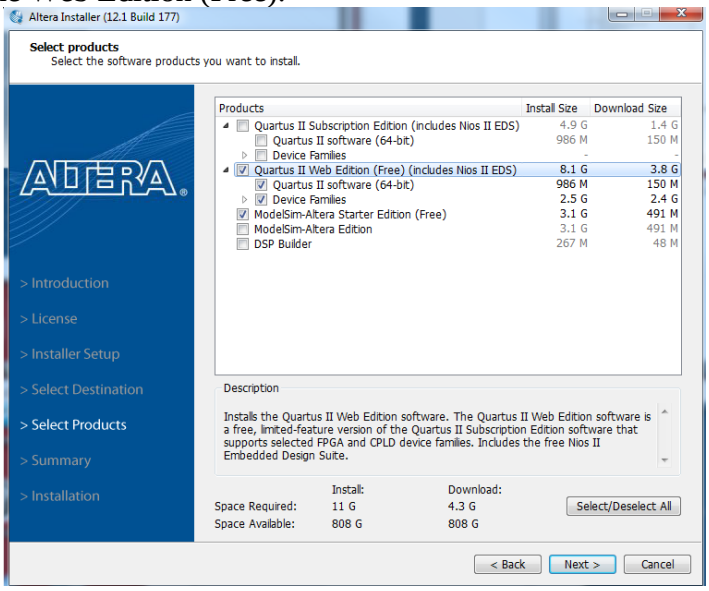
Click “Next” and accept the defaults.



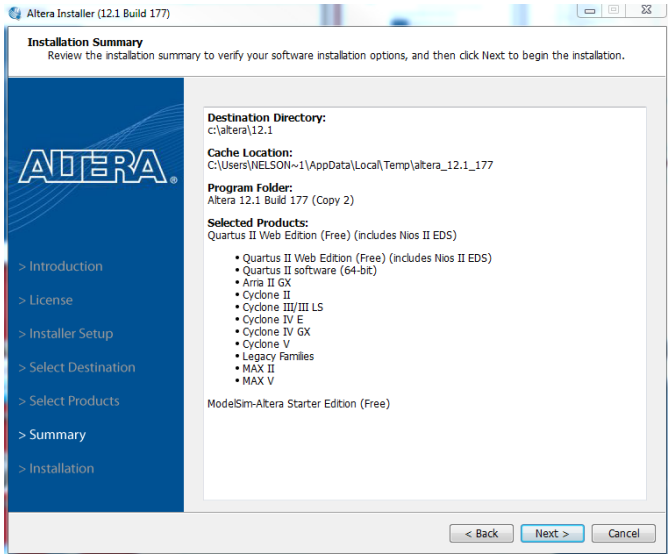
Click “Next” to accept the defaults



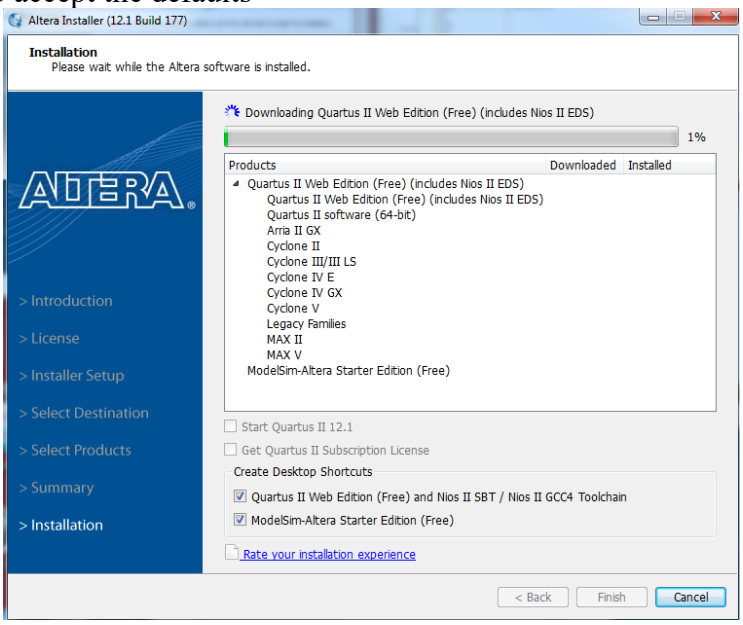
At the Select Products Window, de-select the Quartus Prime Supbscription Edition by clicking on its check box so that the box is not checked. Then click on the check box by the Quartus Prime Web Edition (Free).



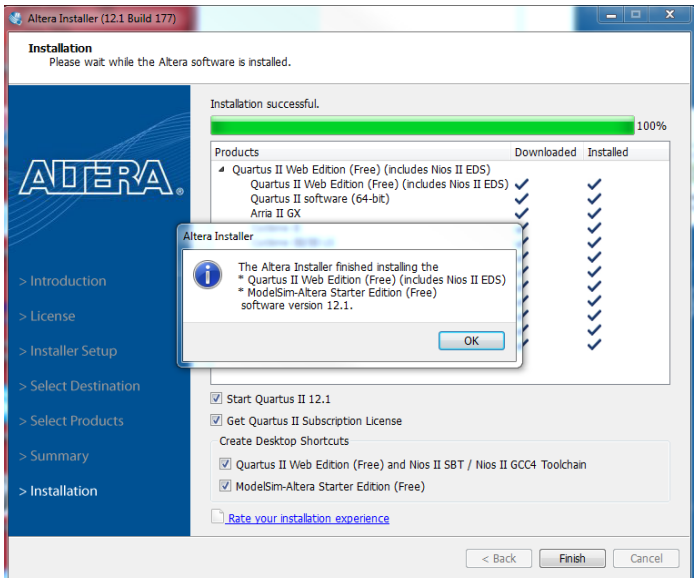
Click “Next” to accept the defaults



Click “Next” to accept the defaults



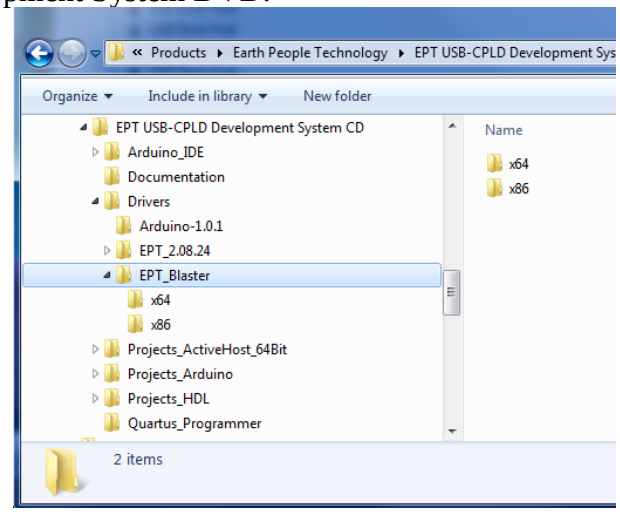
Wait for the download to complete. The file is 3.5 GB, so this could take a couple of hours depending on your internet connection. When installation is complete, the following window appears.



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

3.1.4 Adding the EPT_Blaster to Quartus Prime

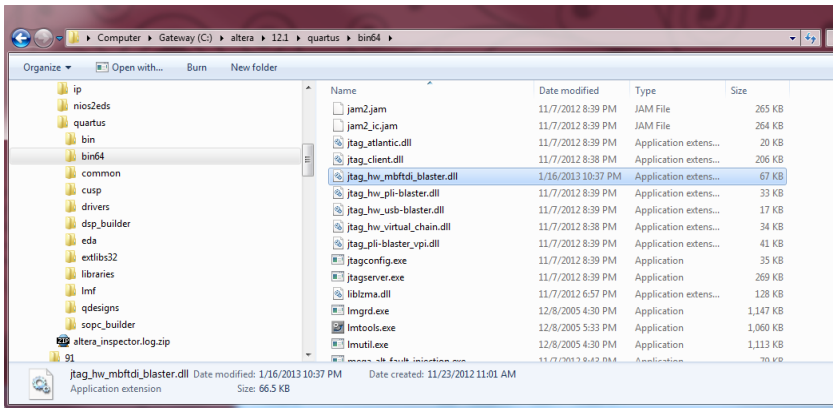
Close out the Quartus Prime application. Locate the \Drivers\EPT_Blaster folder on the EPT FPGA Development System DVD.



Follow these directions:

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 Prime application.



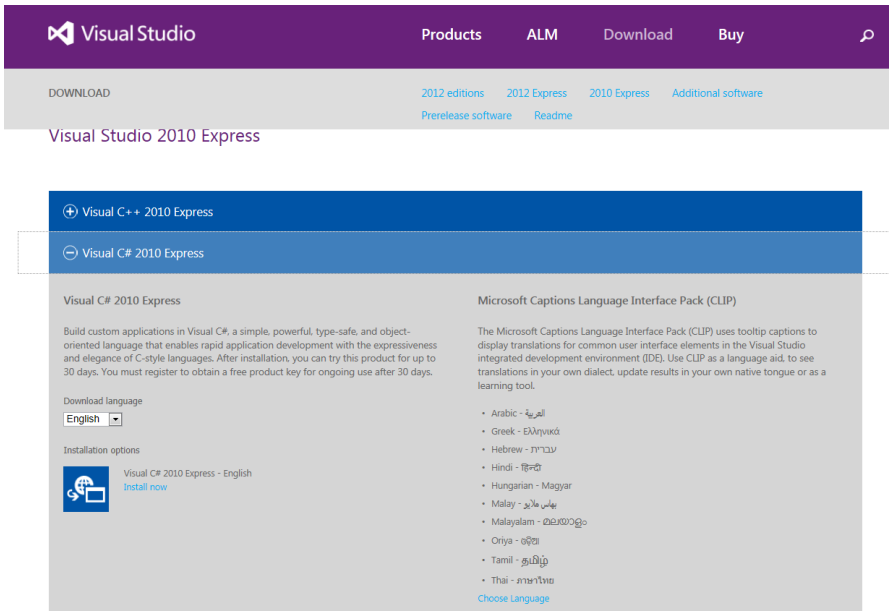
The DLL is installed and the JTAG server should recognize it. Go to the section "Programming the FPGA" of this manual for testing of the programming. If the driver is not found in the Programmer Tool->Hardware Setup box, see the JTAG DLL Insert to Quartus Prime Troubleshooting Guide.

3.2 Active Host Application DLL

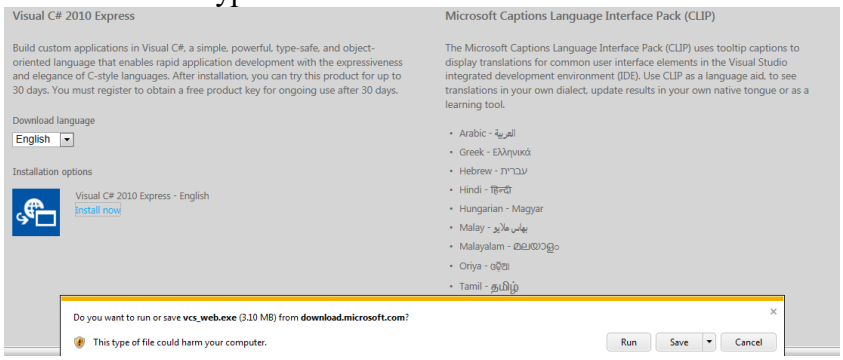
Download the latest version of Microsoft Visual C# Express environment from Microsoft. It's a free download.

https://visualstudio.microsoft.com/vs/express/

Go to the website and click on the "+" icon next to the Visual C# Express.

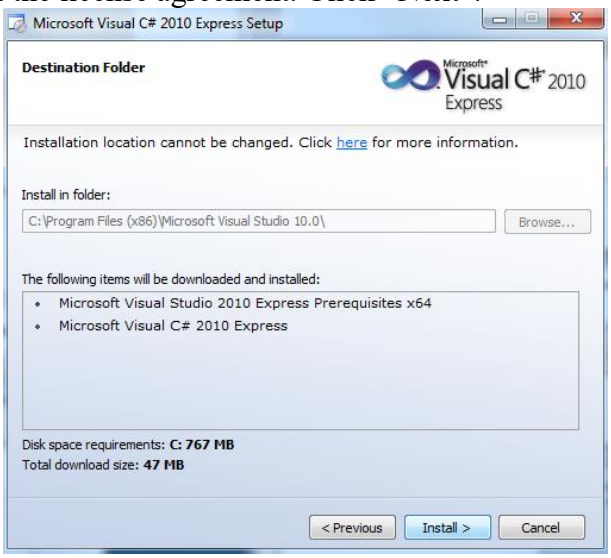


Click on the “Install now” hypertext.

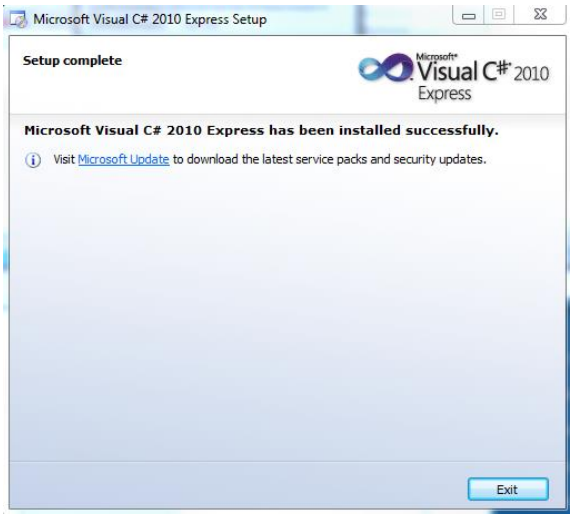


Click the “Run” button.

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

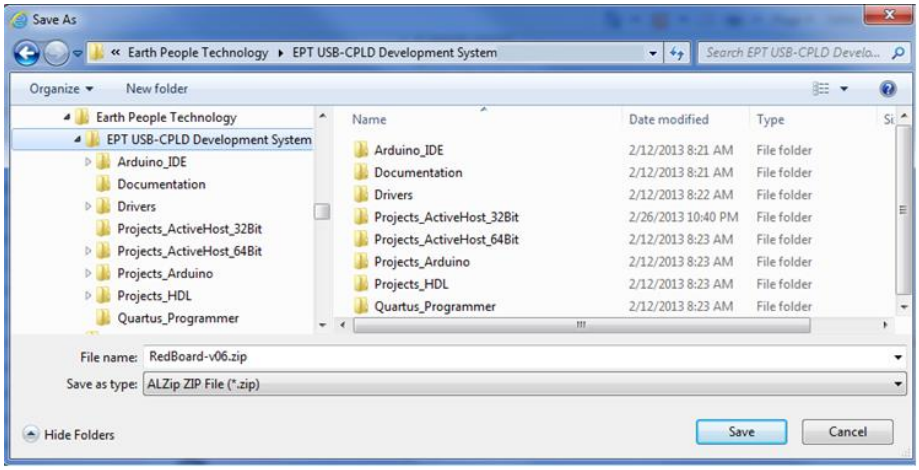


Visual C# 2010 Express will install. This may take up to twenty minutes depending on your internet connection.

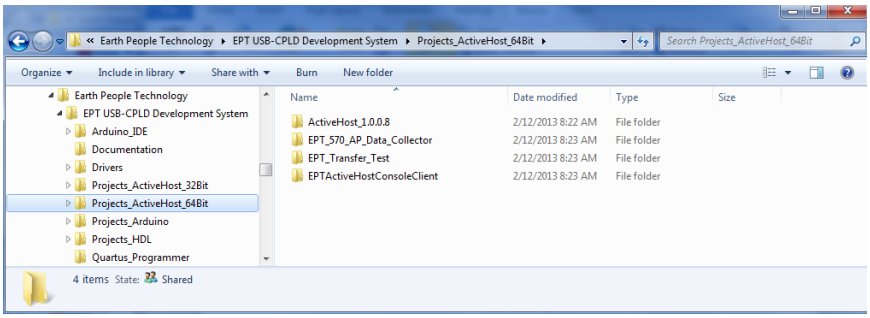


The installed successfully window will be displayed when Visual C# Express is ready to use.

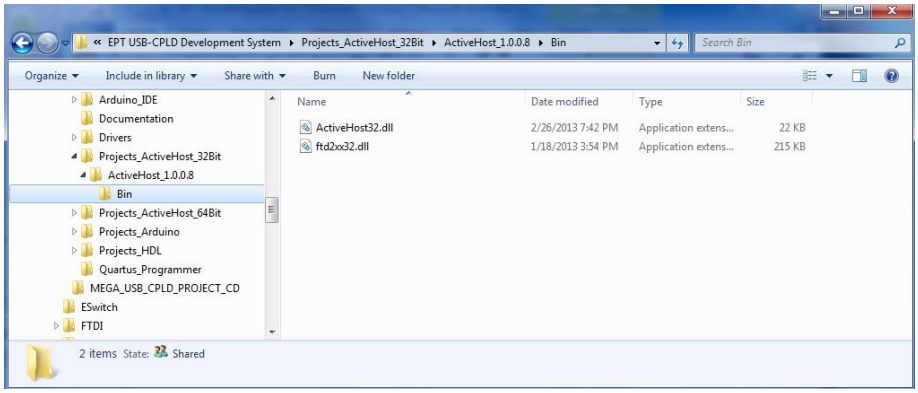
To use the Active Host Application Software, the Active Host DLL and the ftd2xx DLL must be included in the Microsoft Visual project. The Active Host Application Software will allow the user to create a custom applications on the PC using the EndTerms to perform Triggers and Data Transfer to/from the EPT-4CE6-AF-D2. 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.



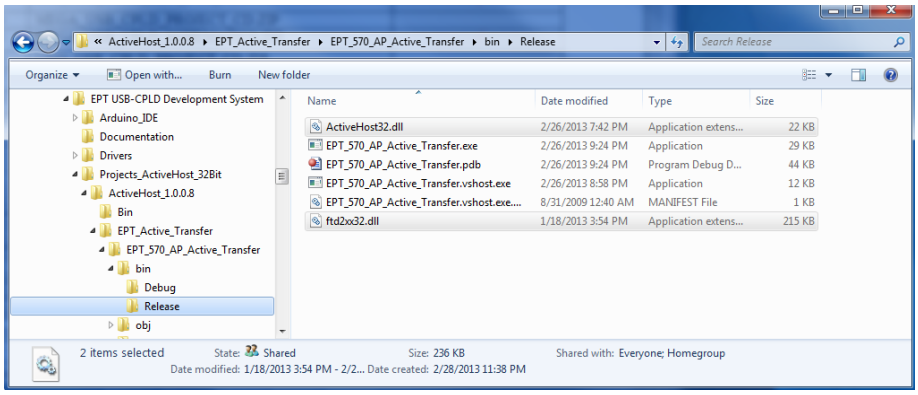
Locate the Projects_ActiveHost_64Bit and \Projects_ActiveHost_32Bit folders in the EPT FPGA Development System using Windows Explorer.



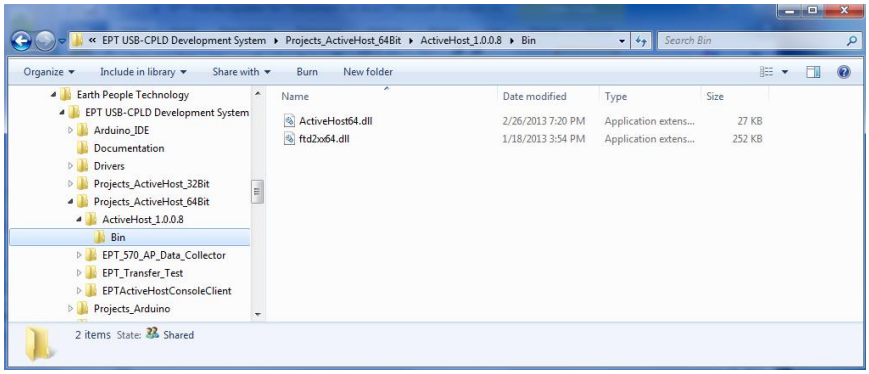
Locate the Projects_ActiveHost_32Bit \ActiveHost_1.0.0.11\Bin folder and copy the ActiveHost32.dll and the ftd2xx32.dll.



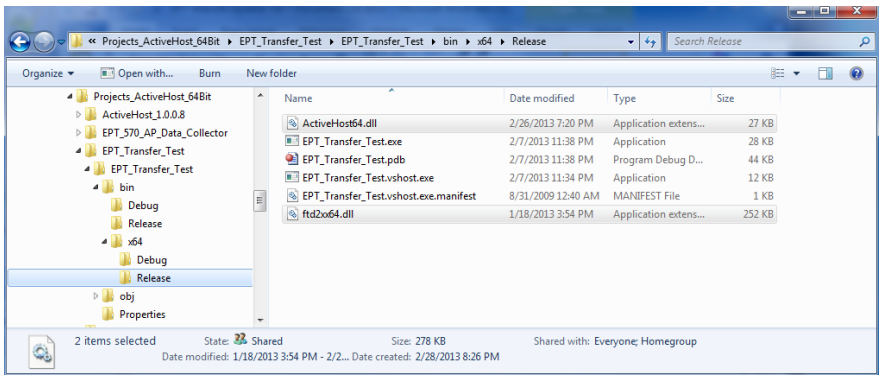
Save the DLL's in the \bin\Release folder for the 32 bit Windows 7 OS of the user project under the Microsoft C# Express project. See the Active Host Application for instructions on how to add the dll to the Microsoft C# Express project.



Locate the Projects_ActiveHost_64Bit \ActiveHost_1.0.0.11\Bin folder and copy the ActiveHost64.dll and the ftd2xx64.dll.

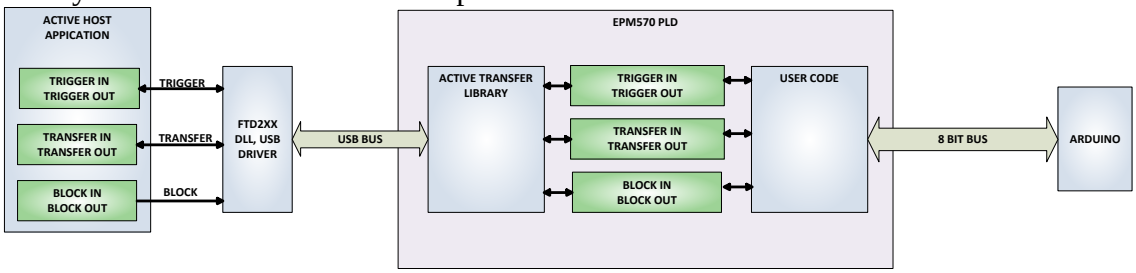


Save the DLL's in the bin\x64\Release folder of the user project under the Microsoft C# Express project. See the Active Host Application section of the EPT FPGA Development System User Manuals for instructions on how to add the dll to the Microsoft C# Express project.



4 FPGA Active Transfer Library

The Active Transfer Library is an HDL library designed to transfer data to and from the EPT-4CE6-AF-D2 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.



4.1 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

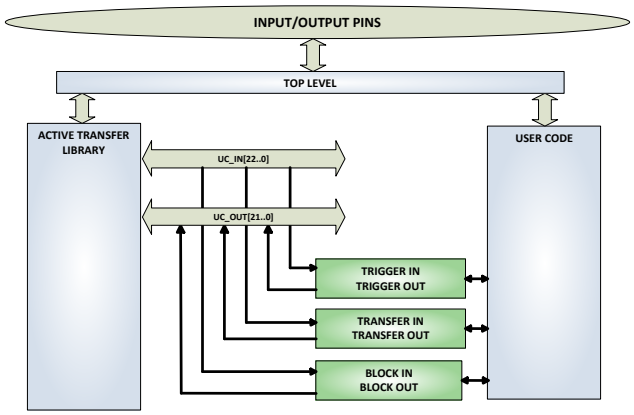
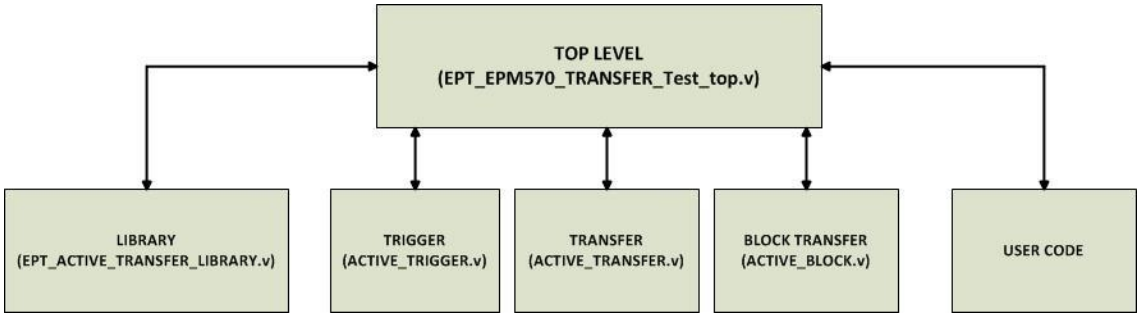


Figure 6 shows how the modules of the EPT Active Transfer Library attach to the overall user project. The EPT Active_Transfer_Library.vqm, Active_Trigger.vqm, Active_Transfer.vqm and Active_Block.vqm modules are instantiated in the top level

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.

4.2 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.

4.2.1 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
```

4.2.2 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(TRANSFER_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
```

4.2.3 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 theUSB chip. This process continues until the number of bytes desicribed 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]

4.3 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.

4.3.1 Active Trigger EndTerm Timing

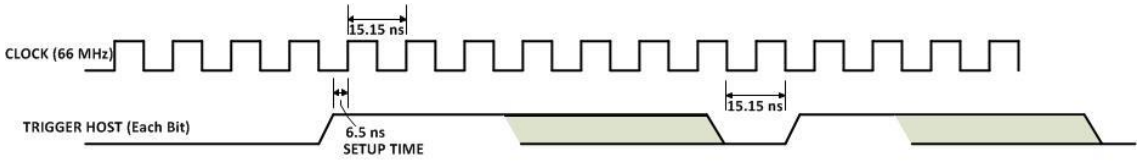


Figure xx Active Trigger to Host Timing

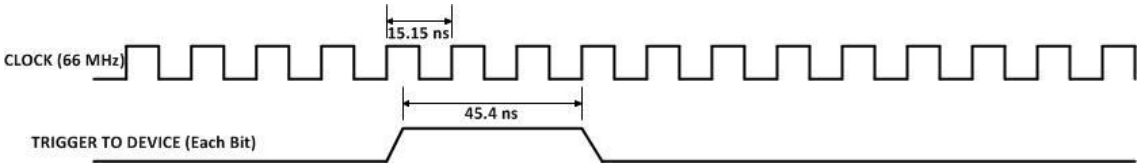


Figure xx Active Trigger to Device Timing

4.3.2 Active Transfer EndTerm Timing

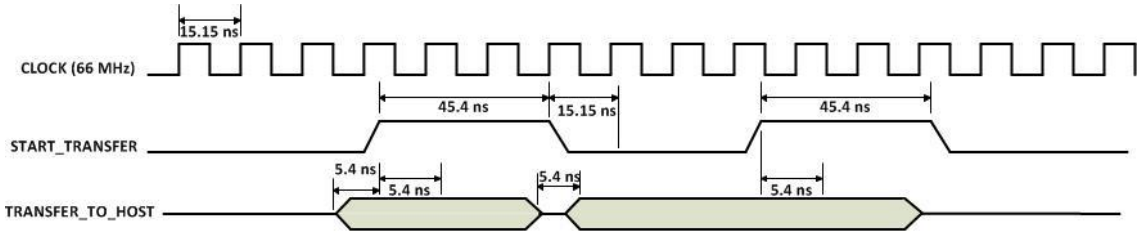


Figure xx Active Transfer To Host Timing

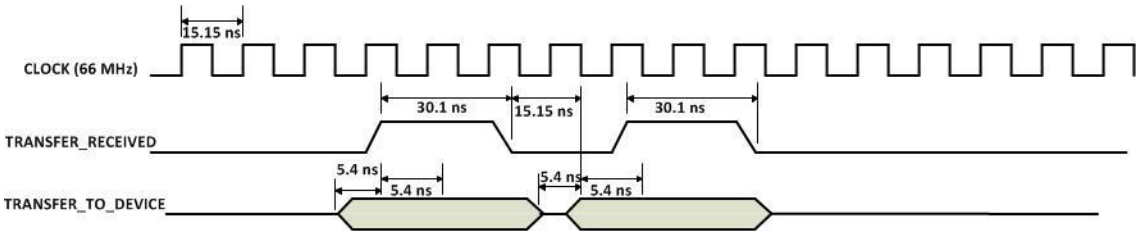


Figure xx Active Transfer To Device Timing

4.3.3 Active Block EndTerm Timing

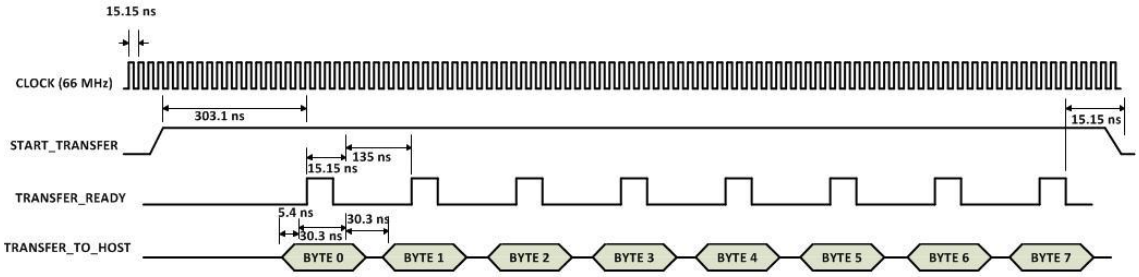


Figure xx Active Block To Host Timing

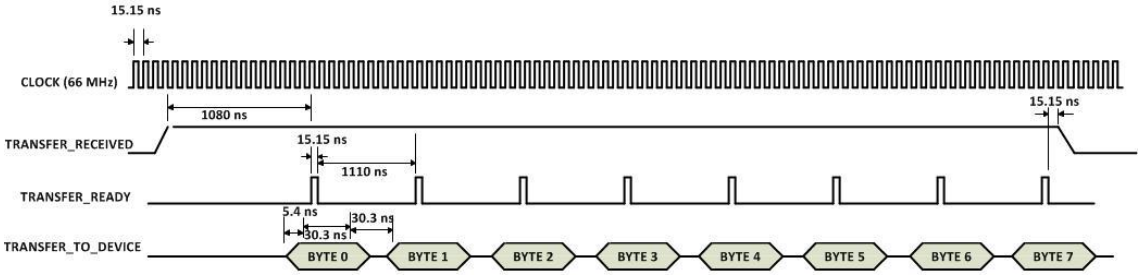


Figure xx Active Block To Device Timing

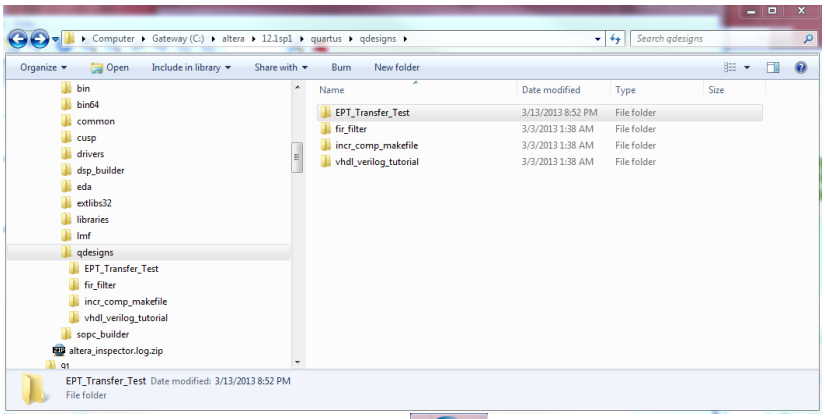
5 Compiling, Synthesizing, and Programming FPGA

The FPGA on the EPT-4CE6-AF-D2 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 Prime 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 Prime 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.

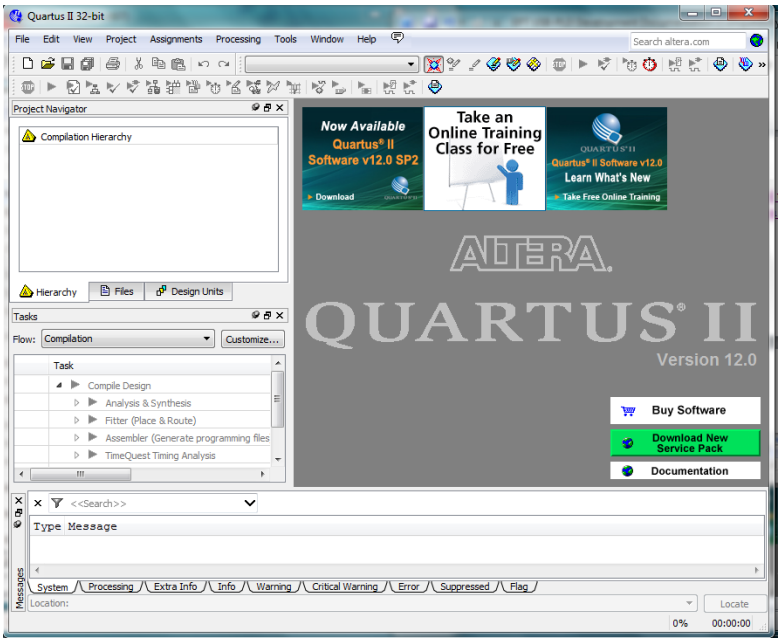
5.1 Setting up the Project and Compiling

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

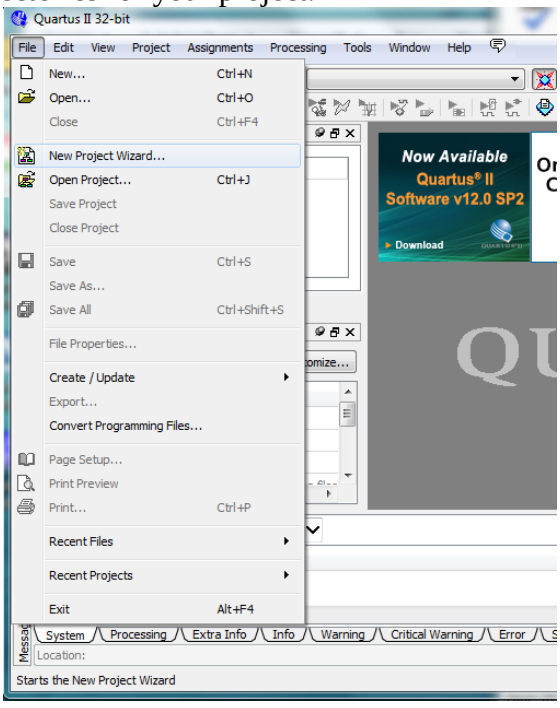


Open Quartus Prime by clicking on the icon 

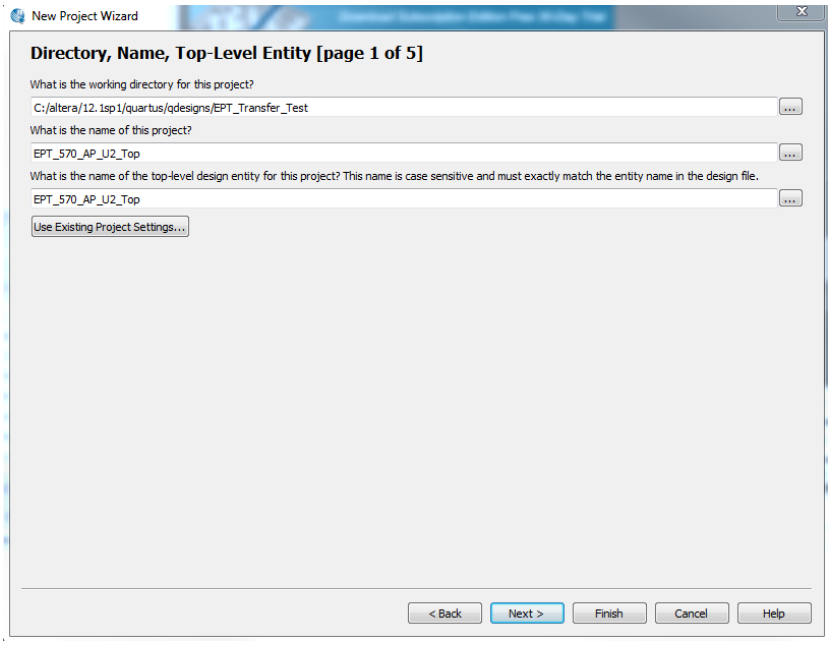
.



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

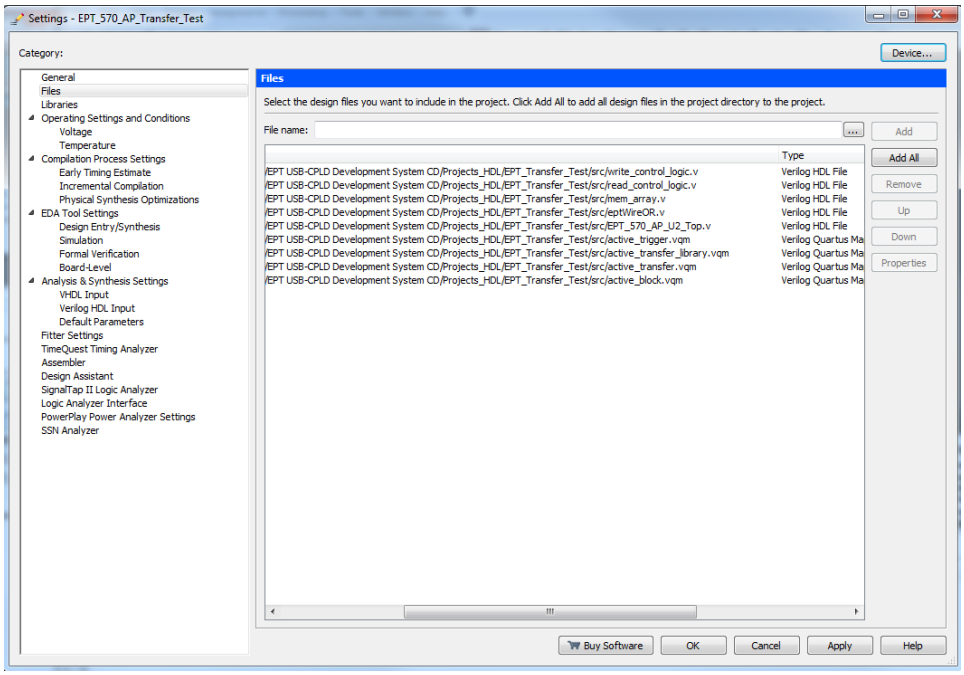


At the Top-Level Entity page, browse to the c:/altera/xxx/quartus/qdesigns directory to store your project. Type in a name for your project “EPT_4CE6_AF_D1_Top”.

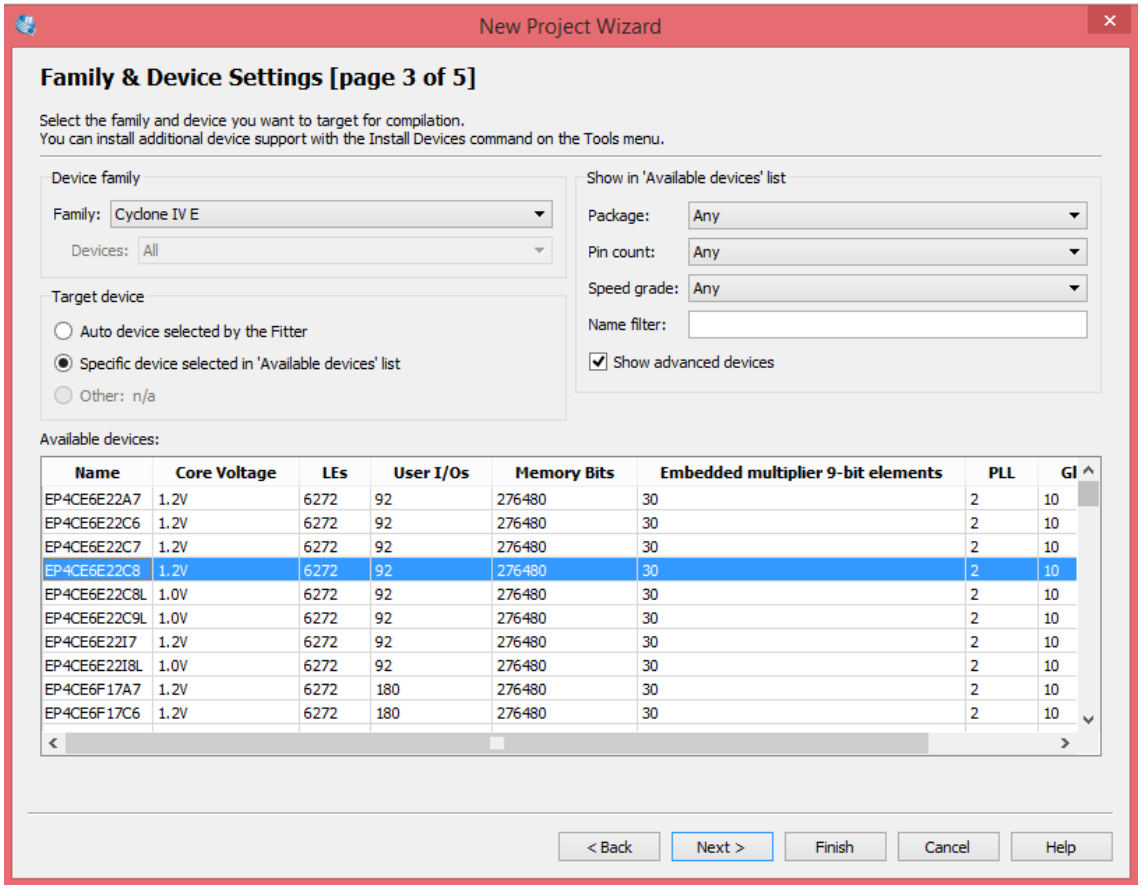


Select Next. At the Add Files window: Browse to the \Projects_HDL\EPT_Transfer_Demo \src folder of the EPT FPGA Development System DVD. Copy the files from the \src directory.

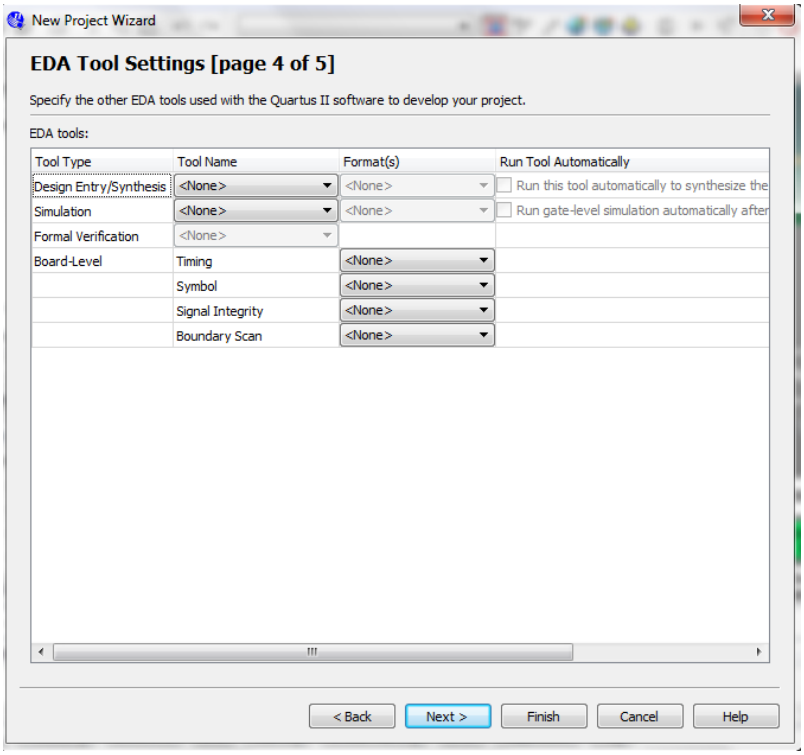
- Active_block.vqm
- Active_transfer.vqm
- Active_trigger.vqm
- Active_transfer_library.v
- ft_245_state_machine.v
- endpoint_registers.vqm
- eptWireOr.v
- mem_array.v
- read_control_logic.v
- write_control_logic.v
- EPT_4CE6_AF_D1_Top.v



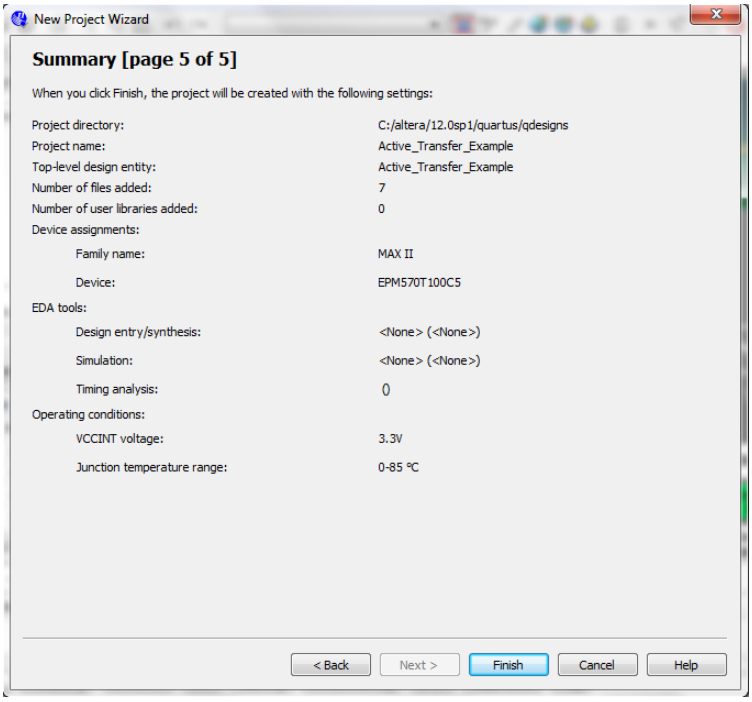
Select Next, at the Device Family group, select Cyclone IV for Family. In the Available Devices group, browse down to EP4CE6E22C8 for Name.



Select Next, leave defaults for the EDA Tool Settings.



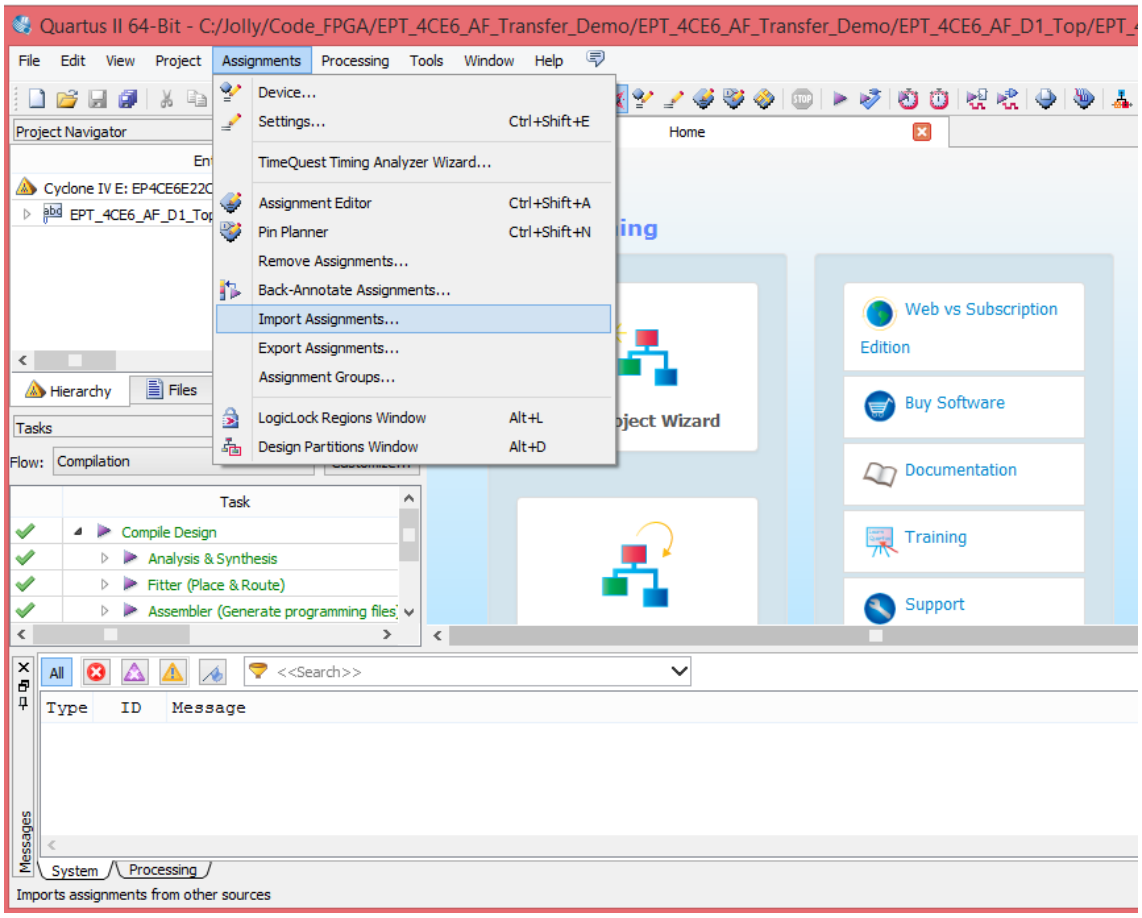
Select Next, then select Finish. You are done with the project level selections.



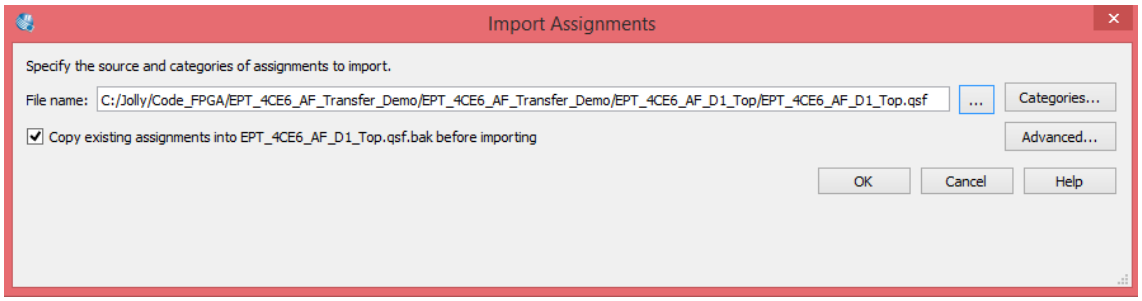
Next, we will select the pins and synthesize the project.

5.1.1 Selecting Pins and Synthesizing

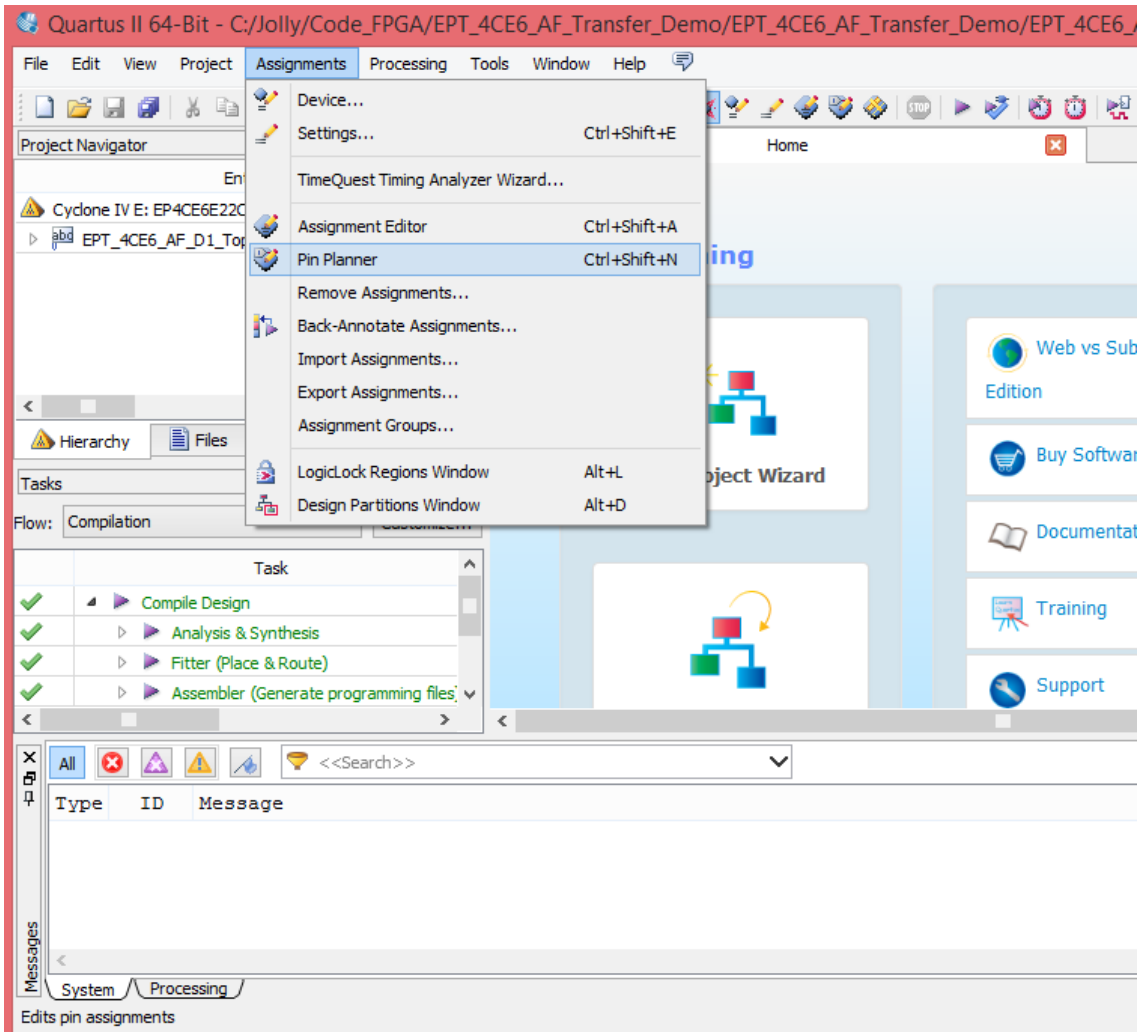
With the project created, we need to assign pins to the project. The signals defined in the top level file (in this case: EPT_4CE6_AF_D1_Top.v) will connect directly to pins on the FPGA. The Pin Planner Tool from Quartus Prime will add the pins and check to verify that our pin selections do not violate 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.



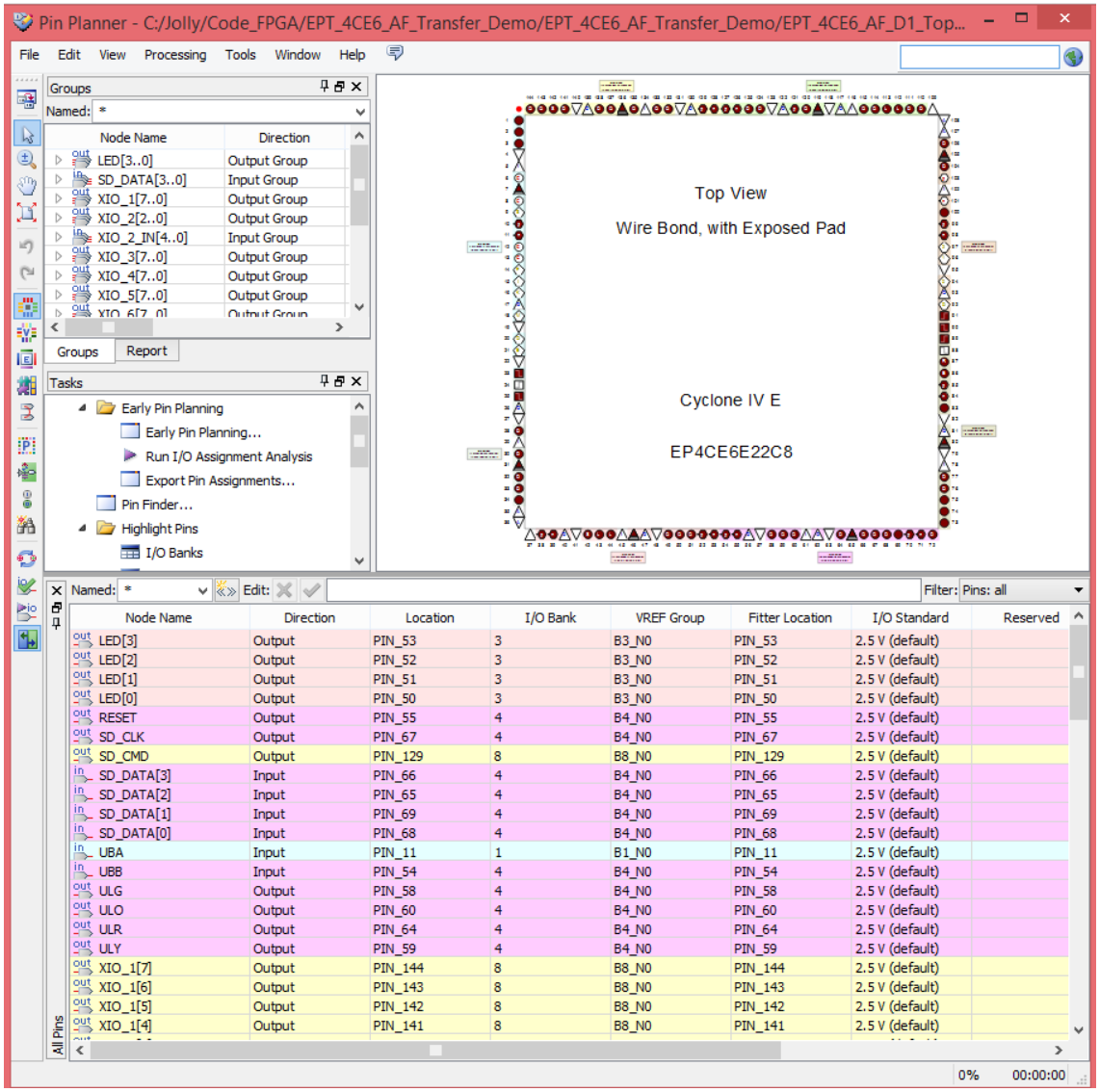
At the Import Assignment dialog box, Browse to the \Projects_HDL\EPT_Transfer_Demo \ EPT-4CE6-AF-D2_TOP folder of the EPT FPGA Development System DVD. Select the “EPT-4CE6-AF-D2_Top.qsf” file.



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



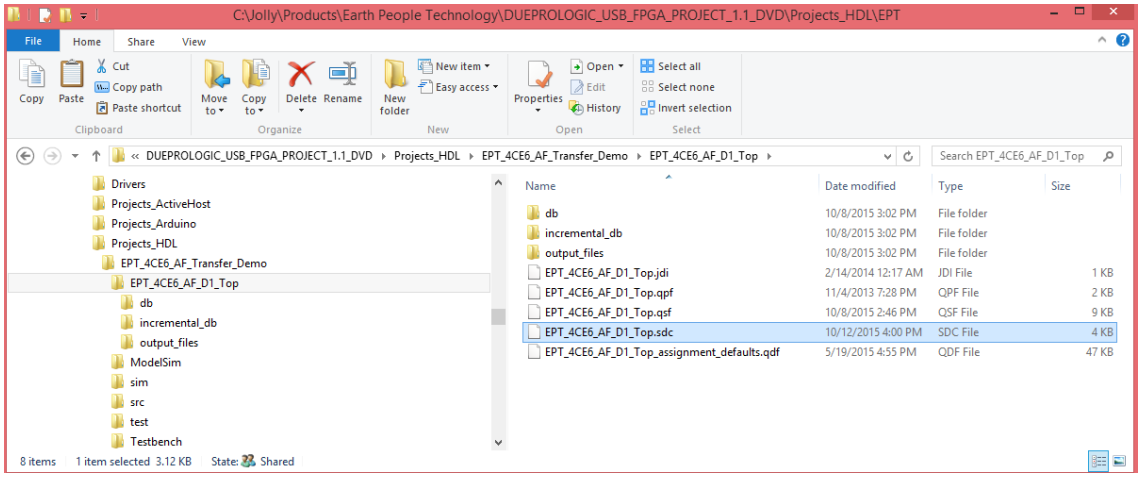
The pin locations should not need to be changed for EPT USB FPGA Development System. However, if you need to change any pin location, just click on the “location” column for the particular node you wish to change. Then, select the new pin location from the drop down box.



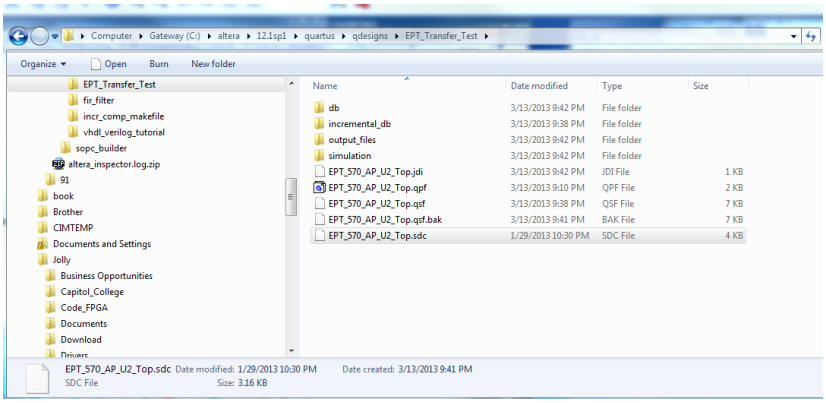
Exit the Pin Planner. Next, we need to add the Synopsys Design Constraint file. This file contains timing constraints which forces the built in tool called TimeQuest Timing Analyzer to analyze the path of the synthesized HDL code with setup and hold times of the internal registers. It takes note of any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

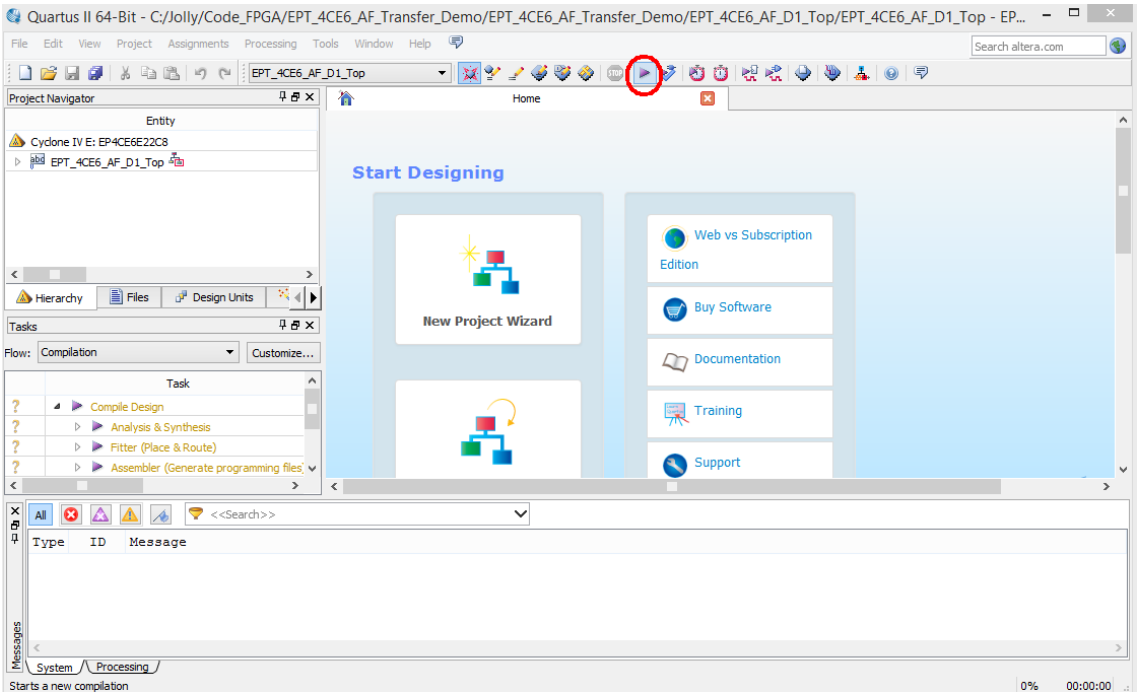
Browse to the \Projects_HDL\EPT_Transfer_Demo \ EPT-4CE6-AF-D2_TOP folder of the EPT FPGA Development System DVD. Select the "EPT-4CE6-AF-D2_Top.sdc" file.



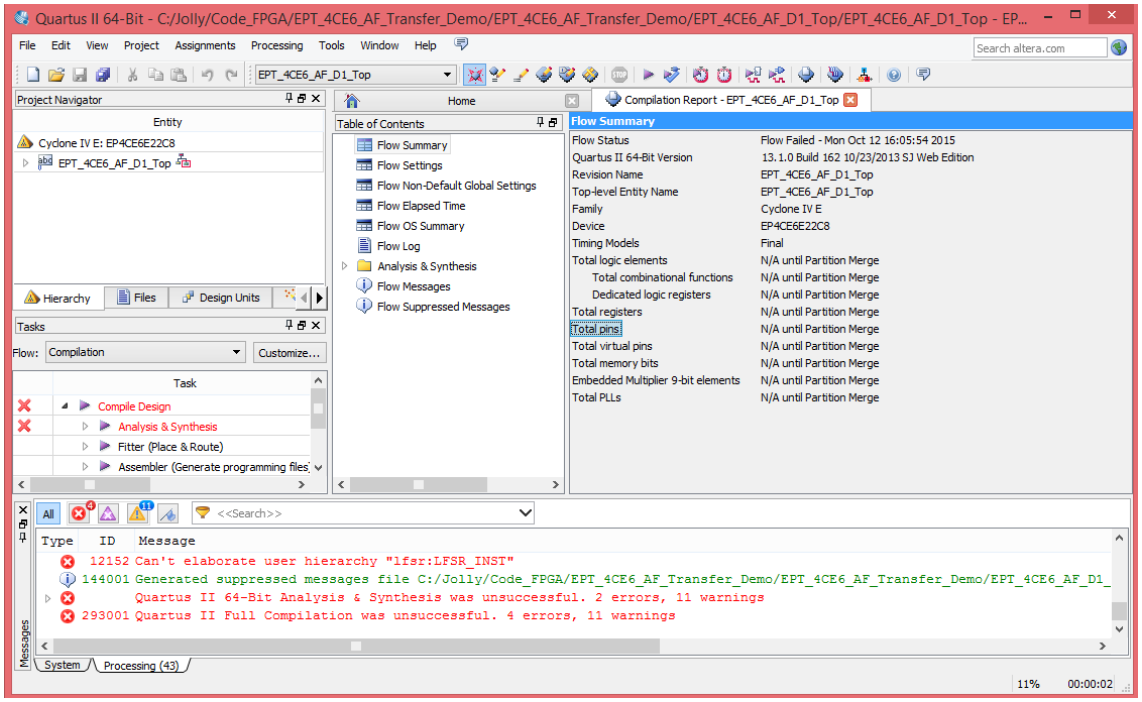
Copy the file and browse to c:\altera\xxx\quartus\qdesigns\EPT_Transfer_Demo directory. Paste the file.



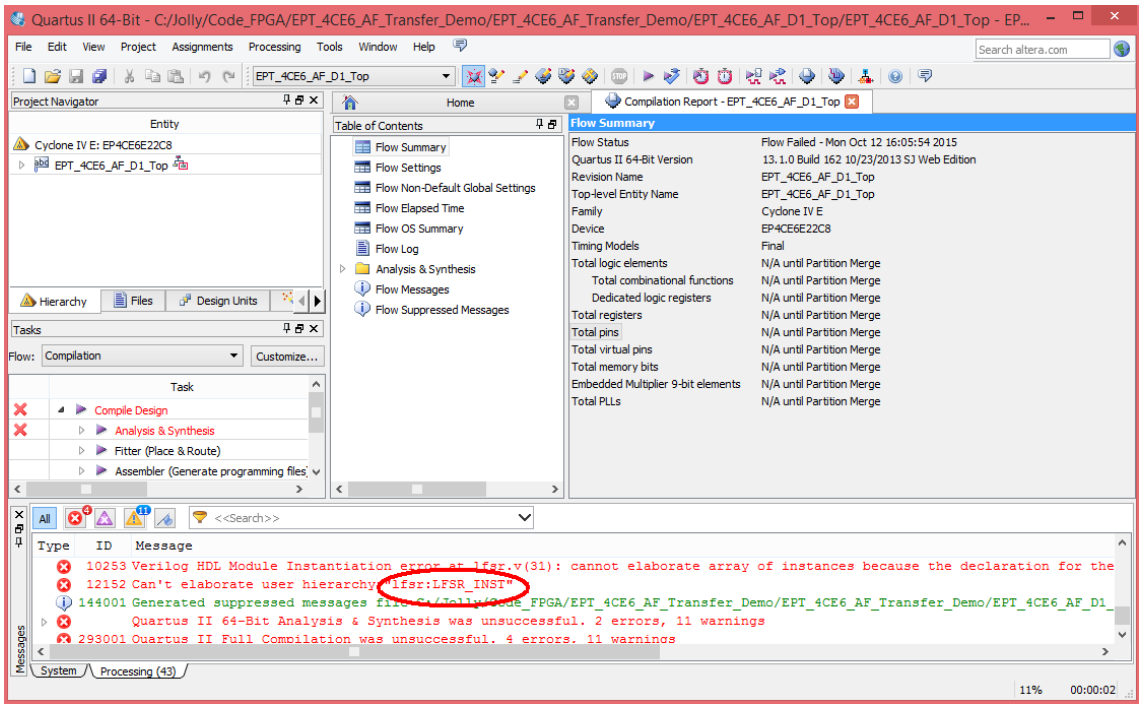
Select the Start Compilation button.



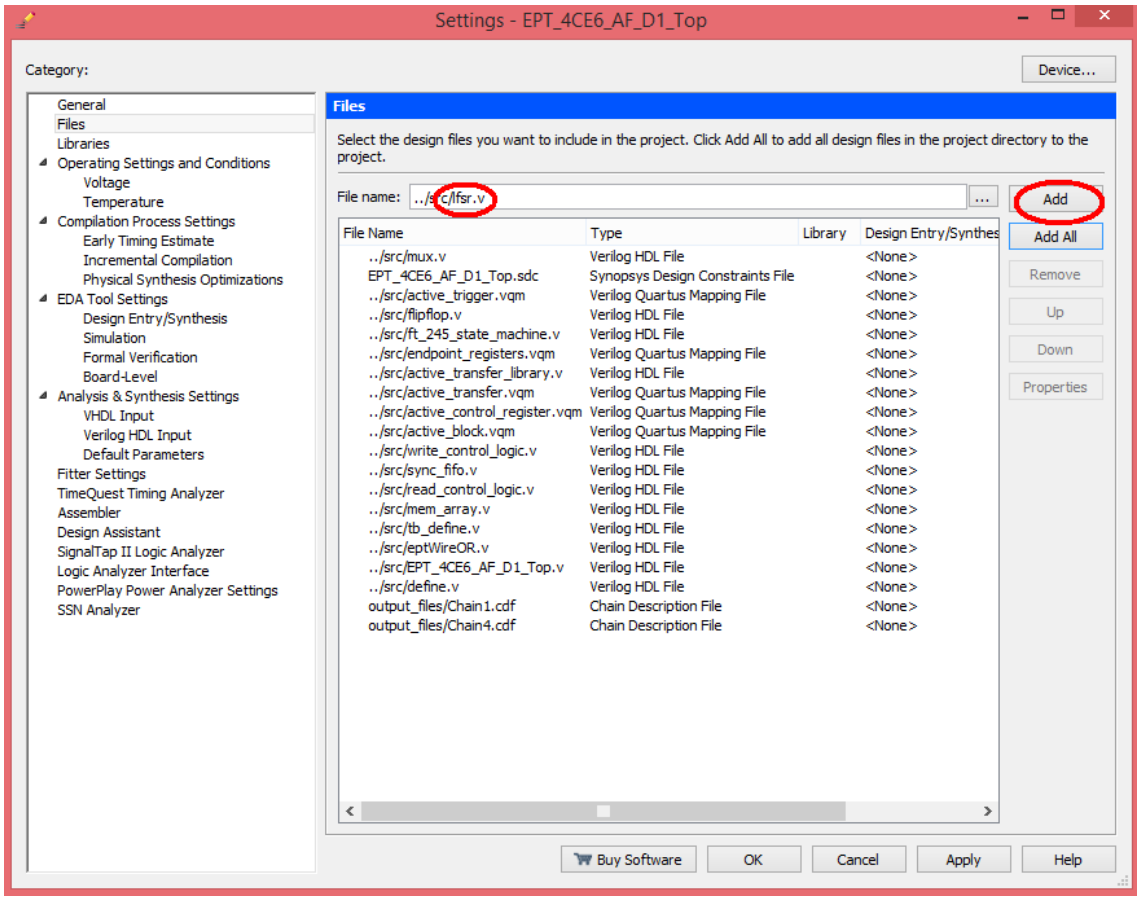
If you forget to include a file or some other error you should expect to see a screen similar to this:



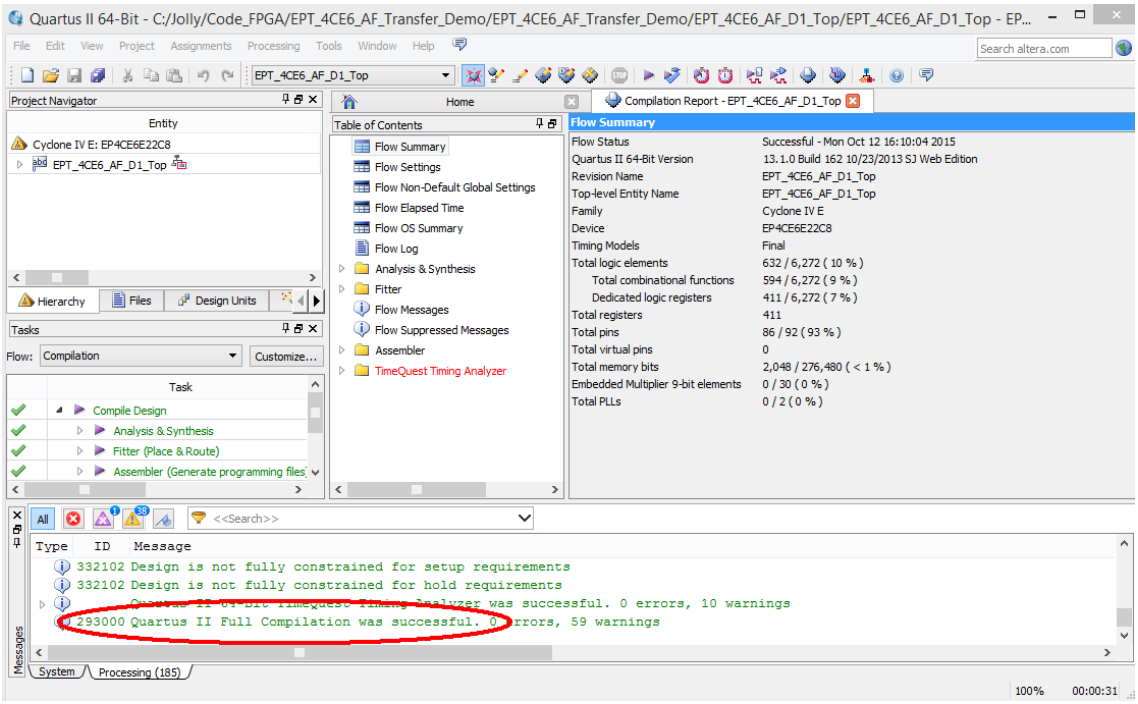
Click Ok, the select the “Error” tab to see the error.



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.



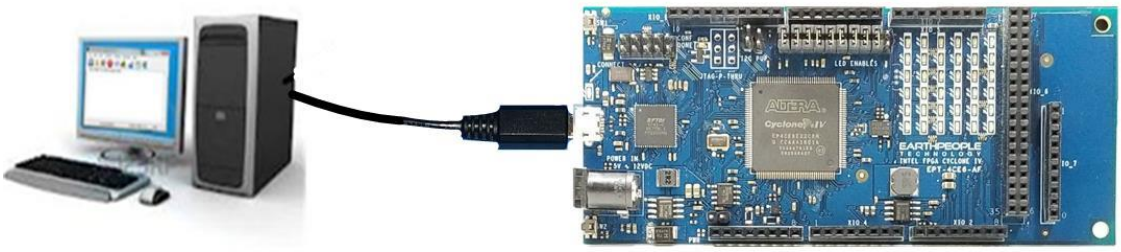
Click Ok then re-run the Compile process. After successful completion, the screen should look like the following:



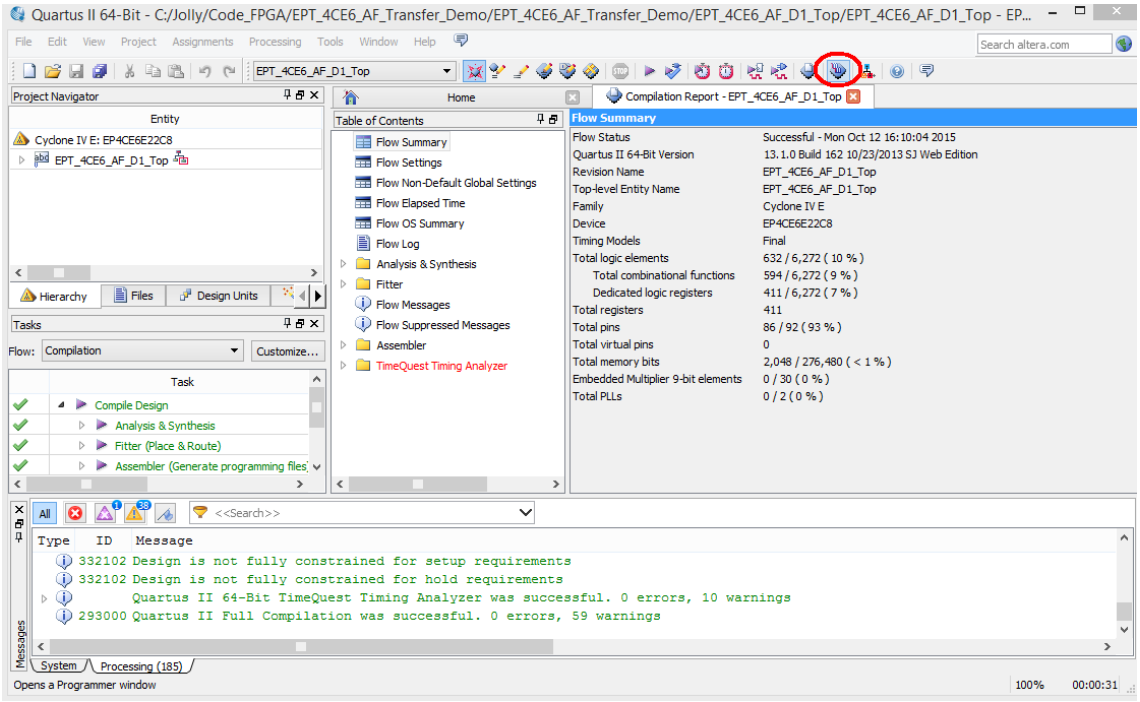
At this point the project has been successfully compiled, synthesized and a programming file has been produce. See the next section on how to program the FPGA.

5.1.2 Configuring the FPGA

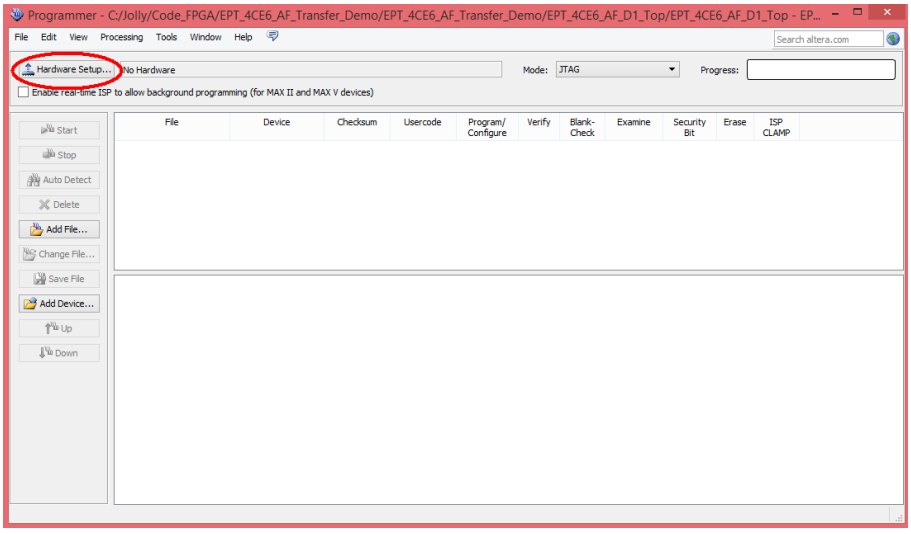
Configuring the FPGA is quick and easy. All that is required is a standard USB Micro B cable and the EPT_Blaster Driver DLL. Connect the EPT-4CE6-AF-D2 to the PC, open up Quartus Prime, open the programmer tool, and click the Start button. To program the DPL Configuration Flash, follow the steps to install the USB Driver and the JTAG Driver Insert for Quartus Prime.



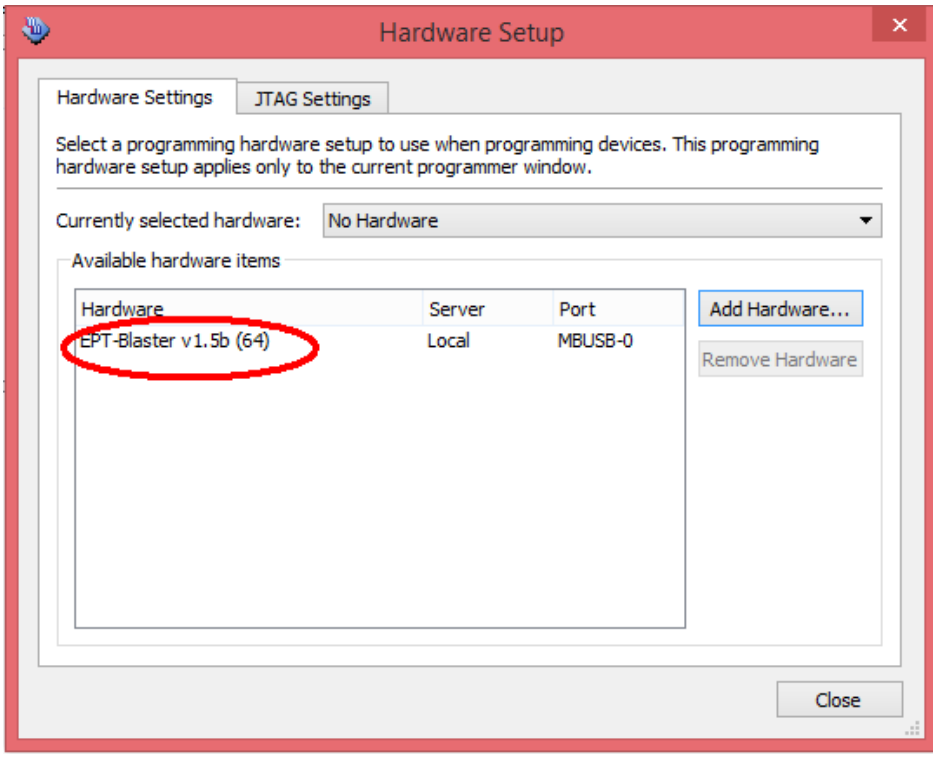
If the project created in the previous sections is not open, open it. Click on the Programmer button.



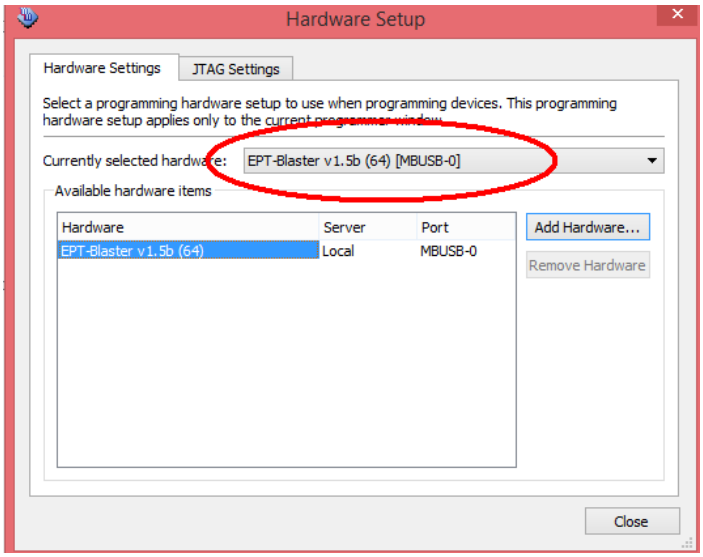
The Programmer Window will open up with the programming file selected. Click on the Hardware Setup button in the upper left corner.



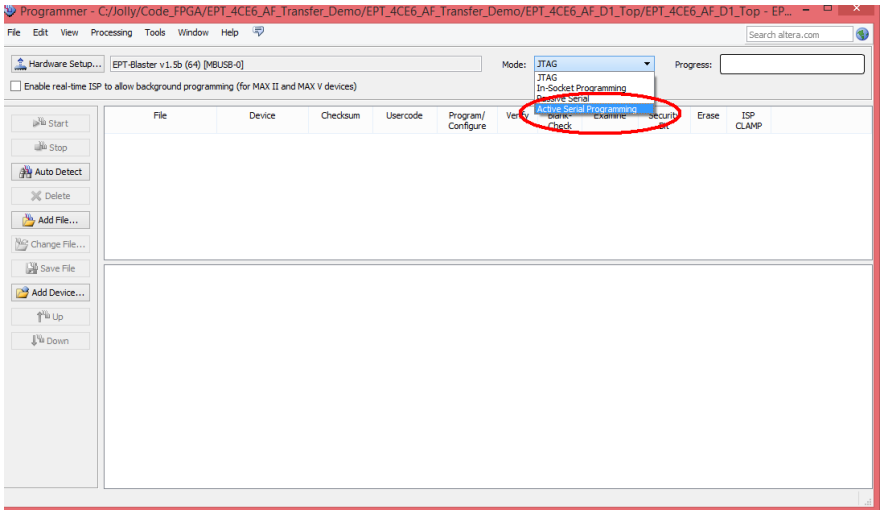
The Hardware Setup Window will open. In the “Available hardware items”, double click on “EPT-Blaster v1.5b”.



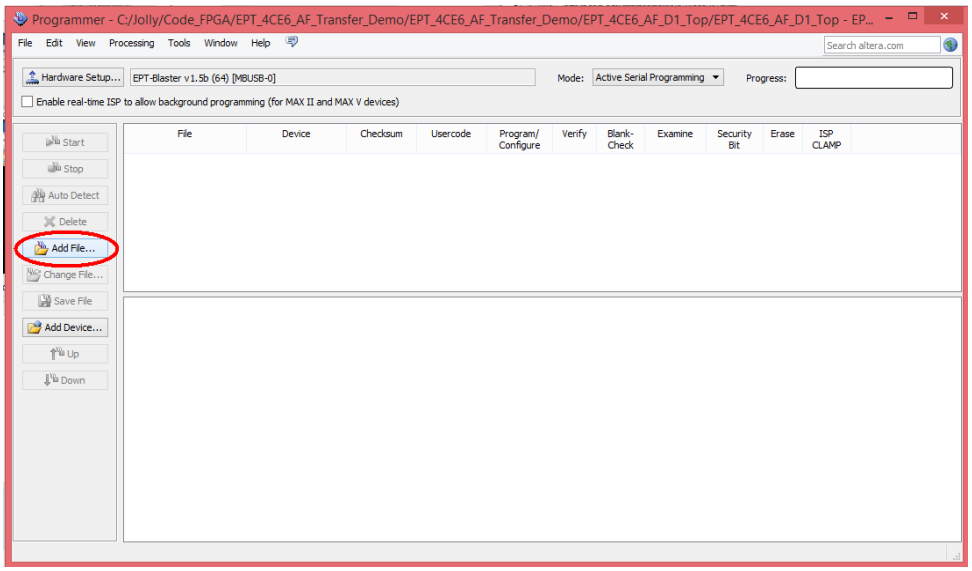
If you successfully double clicked, the “Currently selected hardware:” dropdown box will show the “EPT-Blaster v1.5b”.

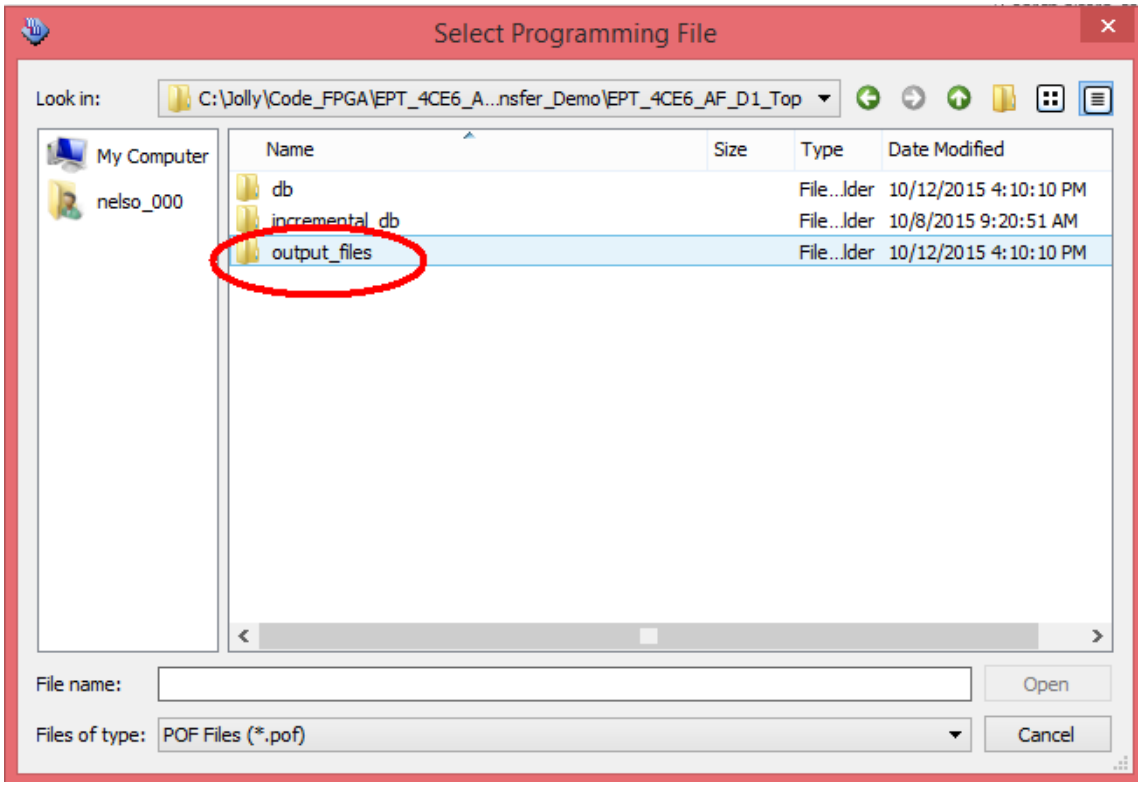


Click on the “Mode:” drop down box. Select the “Active Serial Programming” option.

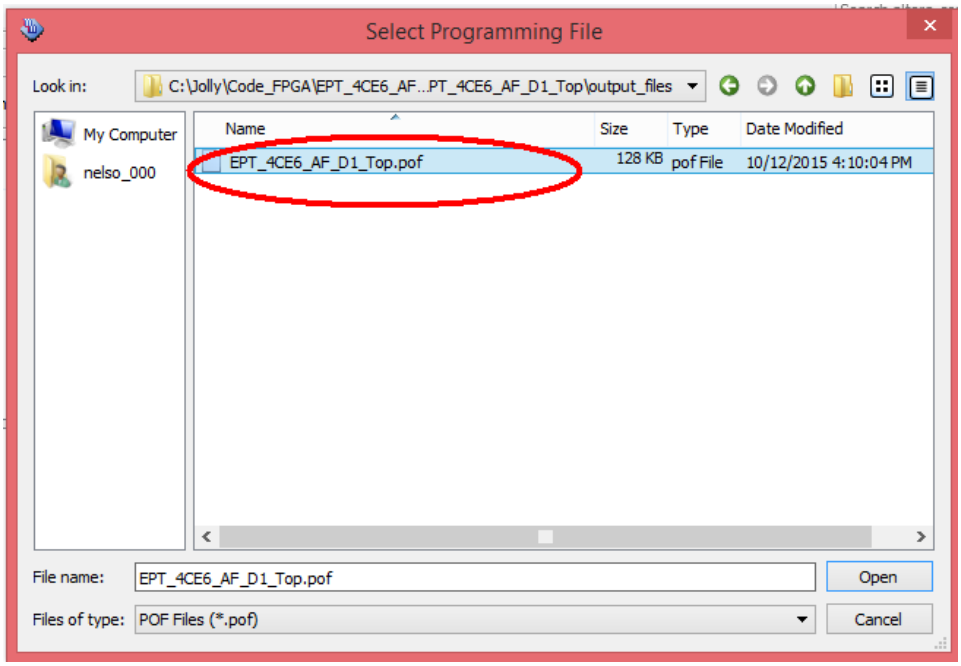


Click on the “Add File” button



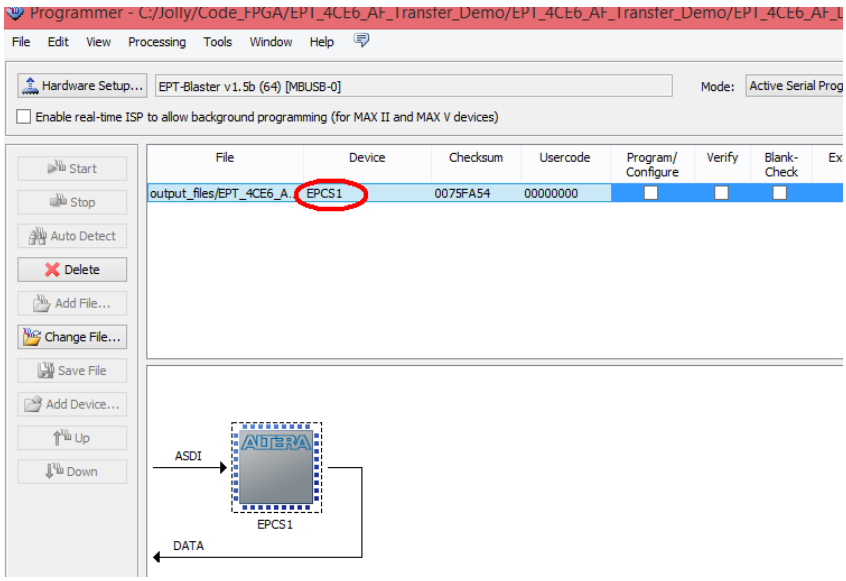


At the Browse window, double click on the output files folder.

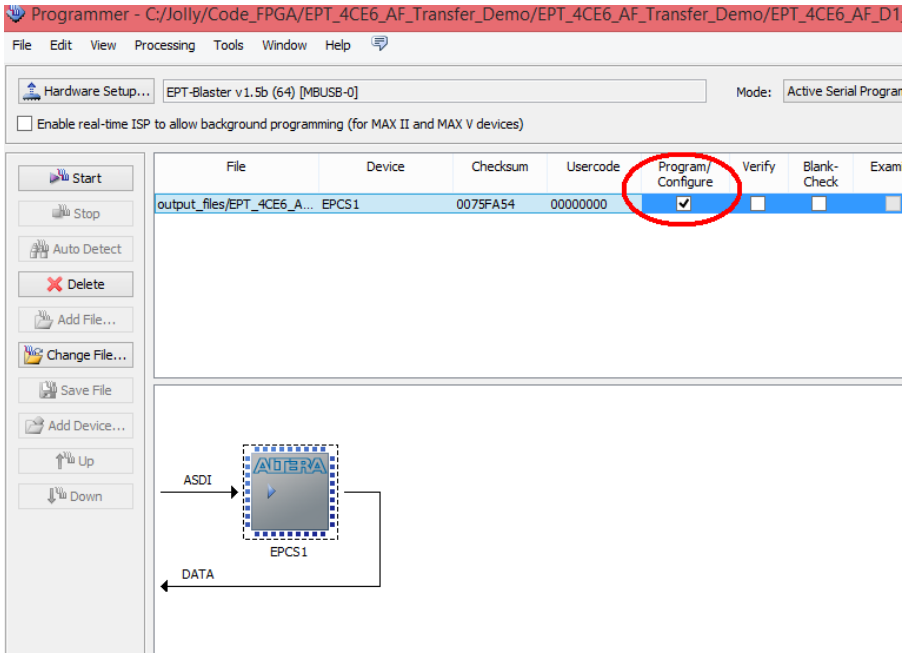


Double click on the “EPT_4CE6_D1_Top.pof” file. Click the Open button in the lower right corner.

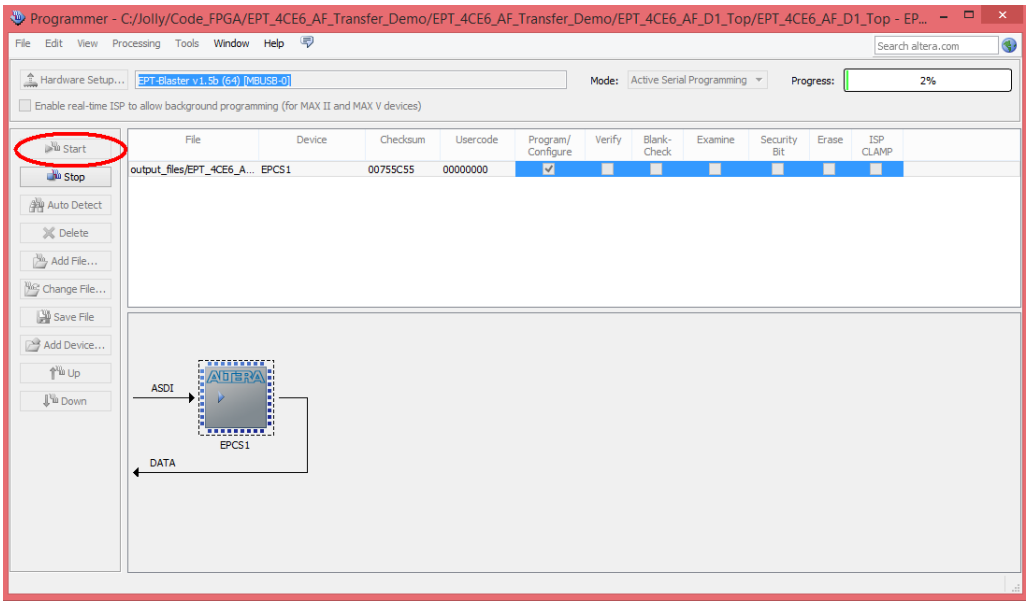
Select the EPCS1 under “Device”.



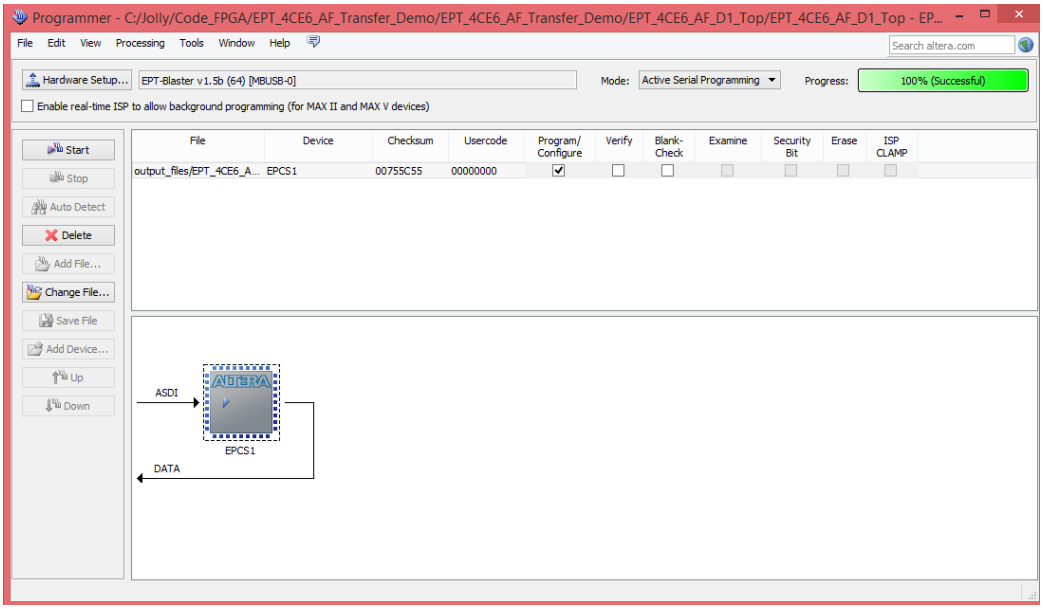
Next, selet the checkbox under the “Program/Configure” of the Programmer Tool.



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-D2 is programmed and ready for use. To test that the FPGA is properly programmed, bring up the Active Transfer Demo Tool. Click on one

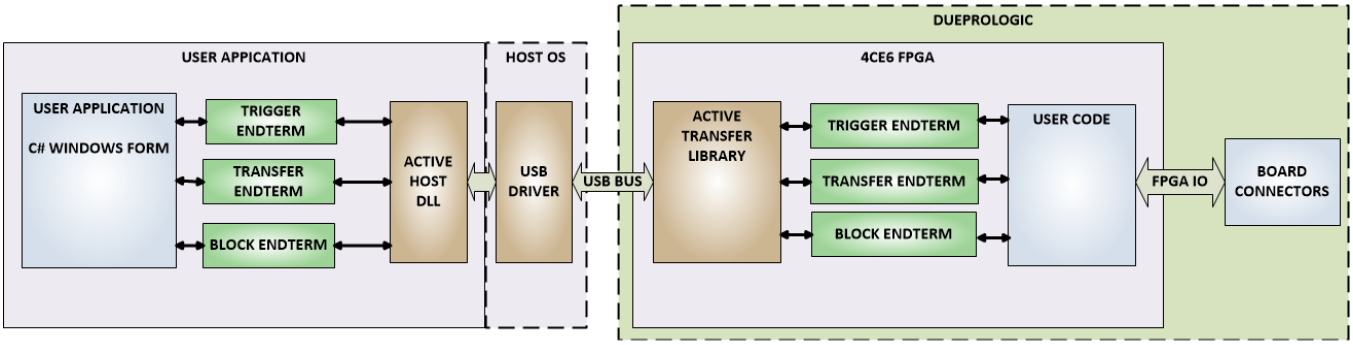
of the LED's and verify that the LED selected lights up. Press one of the switches on the board and ensure that the switch is captured on the Active Host Test Tool. Now you are ready to connect to the Arduino Due and write some code to transfer data between microcontroller and PC.

6 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.



6.1 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.

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

6.2 Transfer(Byte) EndTerm

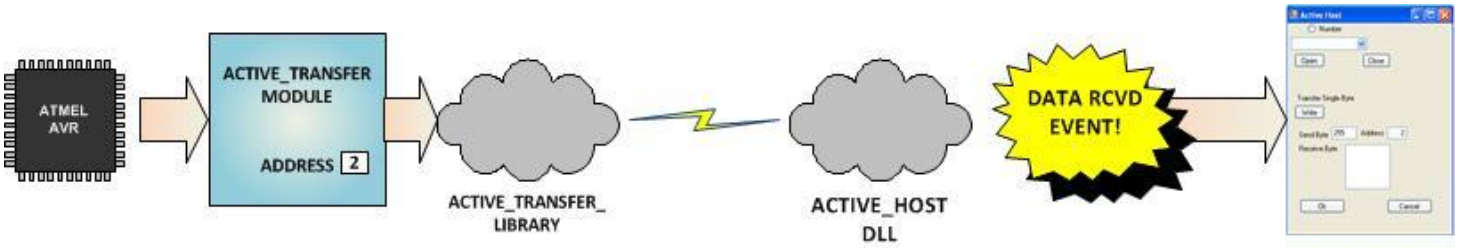
The Transfer EndTerm is a software component that provides a direct path from the users application to the commensurate Transfer EndTerm in the FPGA. It is used to transfer a byte to and from the FPGA. Eight separate Transfer EndTerm modules can be instantiated in the FPGA. Each module is addressed by 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.

6.3 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.

6.4 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()

6.4.1 Active Host Open Device

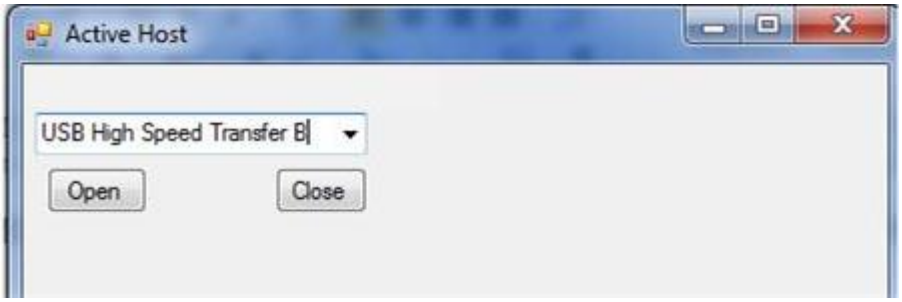
To use the library functions for data transfer and triggering, an Earth People Technology device must be opened. The first function called when the Windows Form loads up is the <project_name>_Load(). This function is called automatically upon the completion of the Windows Form, so there is no need to 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

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;
}
```

6.4.2 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));
      }
   }
```

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.

6.4.3 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:
            lLableSwitch1.Text = "Switch 1\n Pressed";
            break;
        case 0x02:
            lLableSwitch2.Text = "Switch 2\n Pressed";
            break;
        case 0x04:
            lLableSwitch1.Text = "";
            lLableSwitch2.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 copied and pasted into a user's project.

6.4.4 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 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 it's transmit FIFO. If data exists, the Host will command the Device to send it's 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.

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));
}
```

```
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.ToUInt32(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.

6.4.5 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.

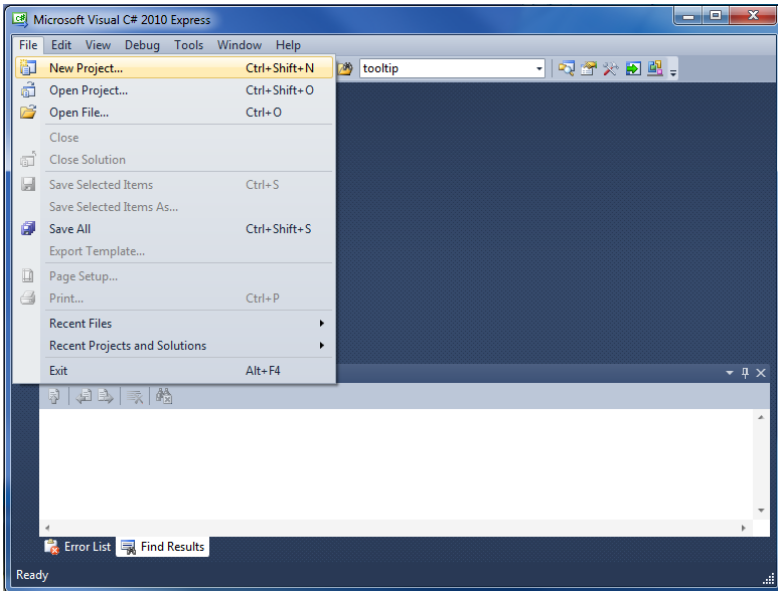
7 Assembling, Building, and Executing a .NET Project on the PC

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-D2. 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

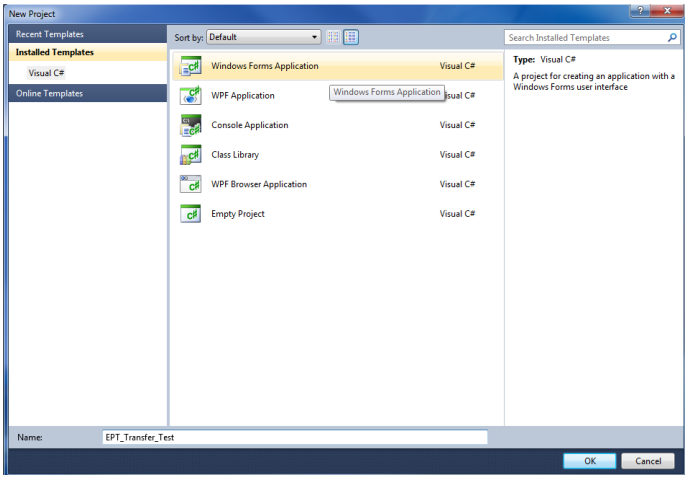
implication (such as events and controls), will be explained as the explanation progresses. Any language can be used with the Active Host Application DLL.

7.1 Creating a Project

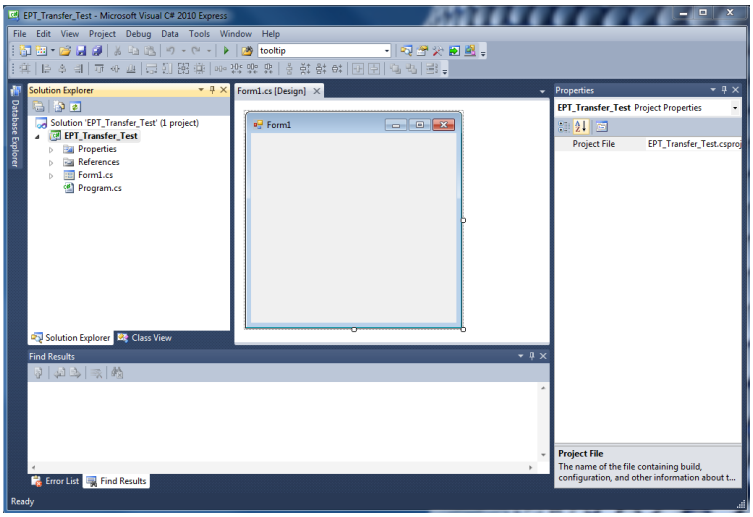
Once the application is installed, open it up. Click on File->New Project.



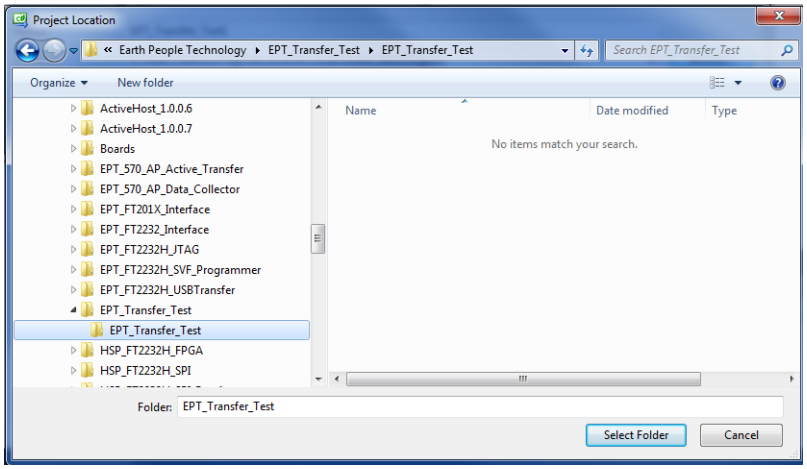
At the New Project window, select the Windows Forms Application. Then, at the Name: box, type in EPT_Transfer_Demo



The project creation is complete.

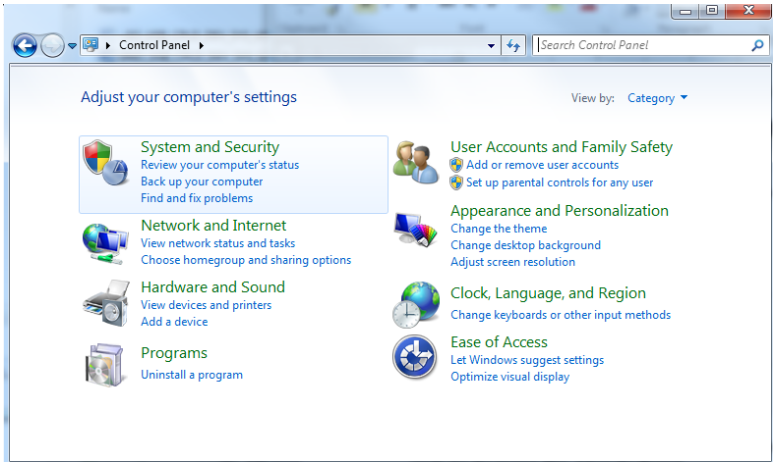


Save the project, go to File->Save as, browse to a folder to create EPT_Transfer_Demo folder. The default location is c:\Users\<Users Name>\documents\visual studio 2010\Projects.

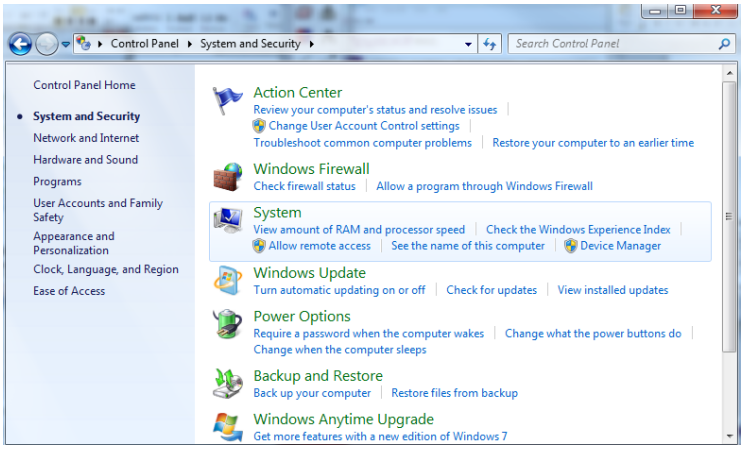


7.1.1 Setting up the C# Express Environment x64 bit

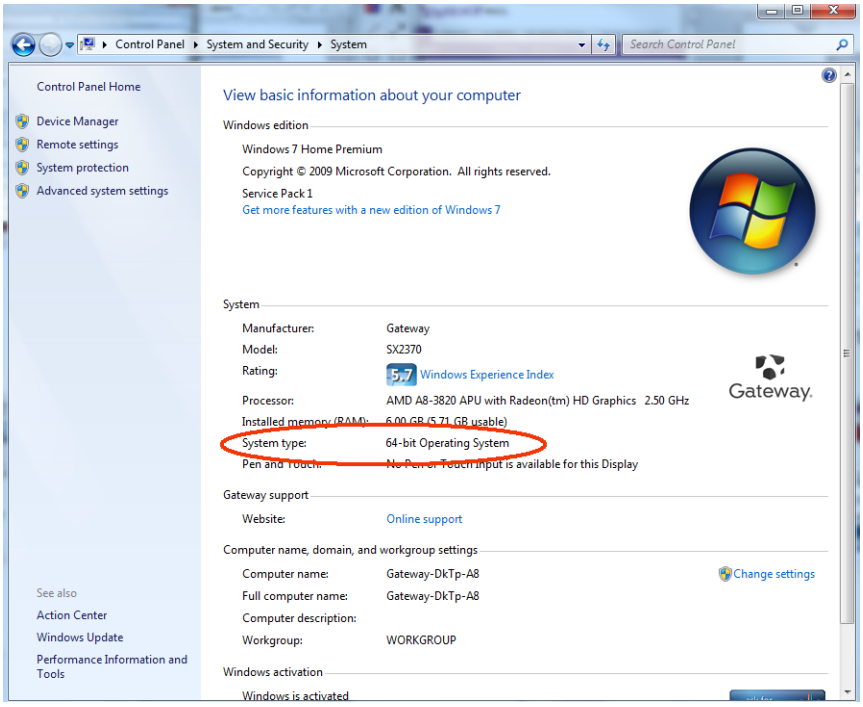
The project environment must be set up correctly in order to produce an application that runs correctly on the target platform. If your system supports 64 bit operation, perform the following steps. Otherwise if your system is 32 bit skip to the Section, Assembling Files into the Project. Visual C# Express 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



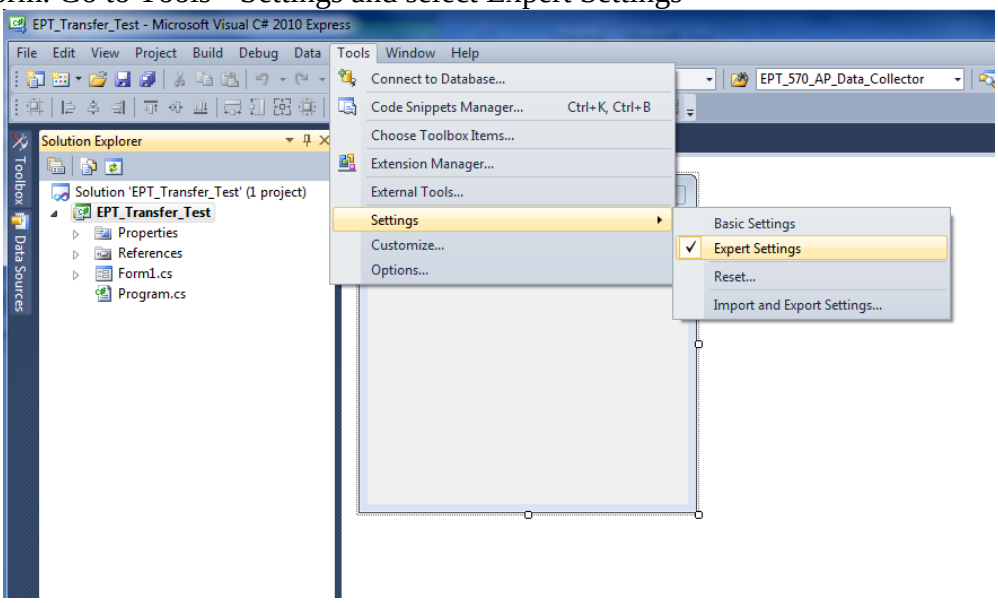
Click on System.



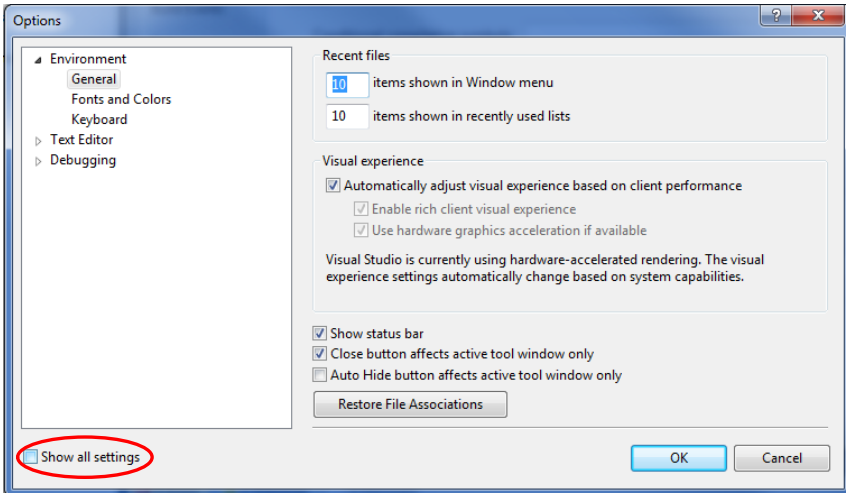
Check under System\System type:



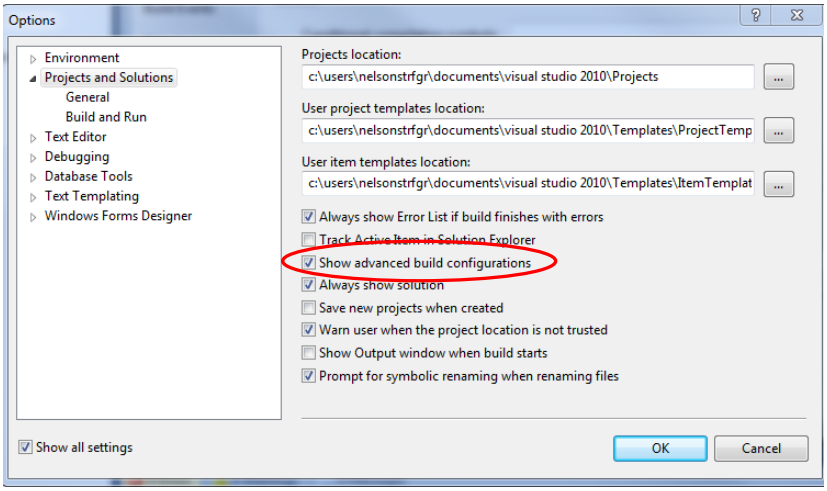
First, we need tell C# Express to produce 64 bit code if we are running on a x64 platform. Go to Tools->Settings and select Expert Settings



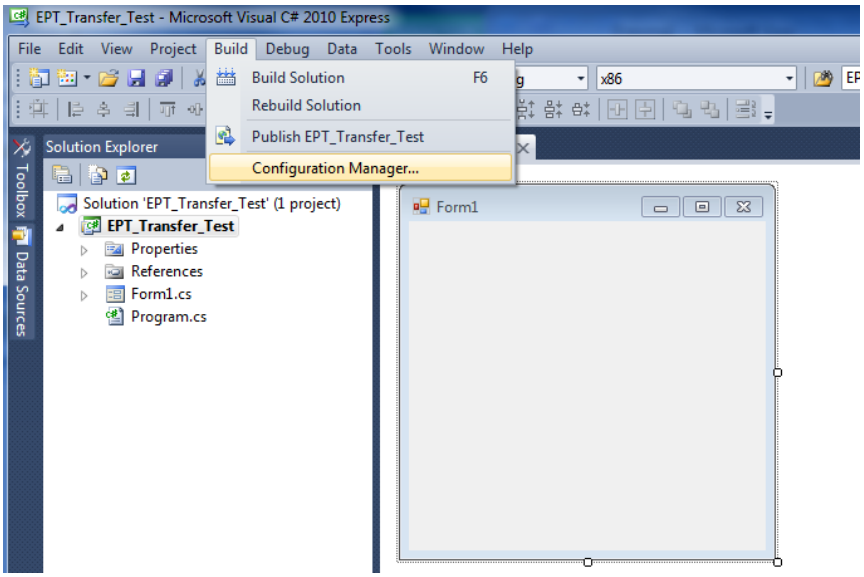
Go to Tools->Options, locate the “Show all settings” check box. Check the box.



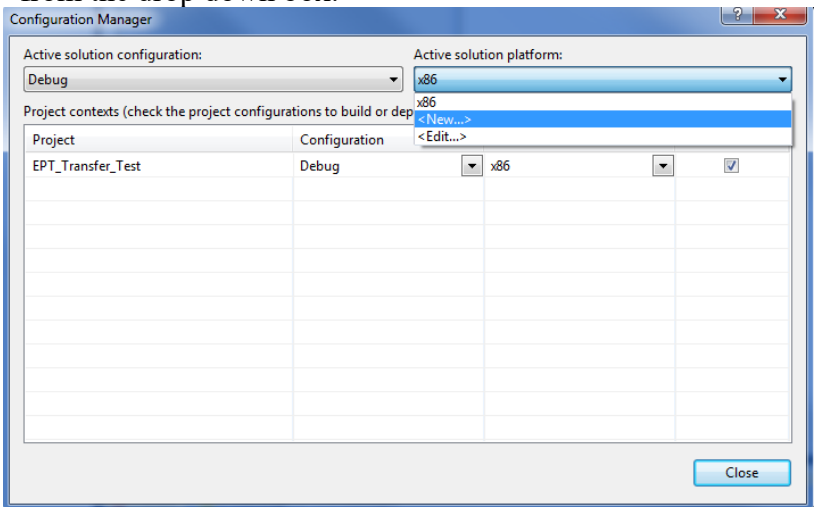
In the window on the left, go to “Projects and Solutions”. Locate the “Show advanced build configurations” check box. Check the box.



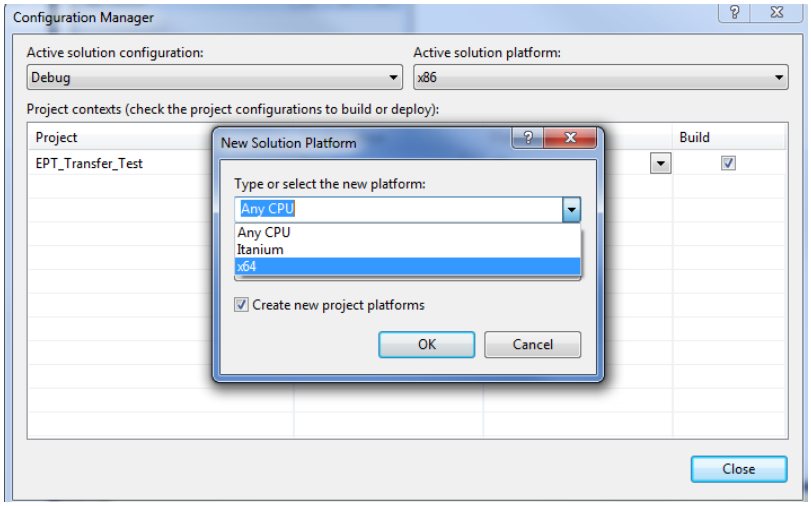
Go to Build->Configuration Manager.



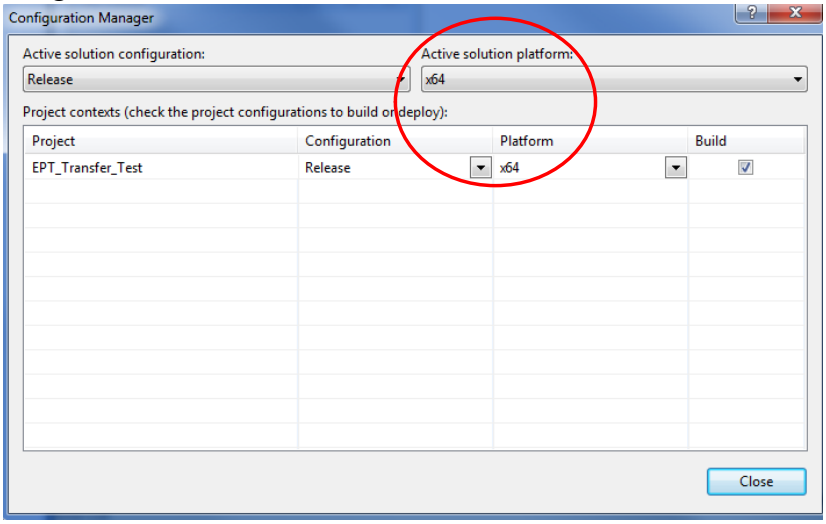
In the Configuration Manager window, locate the "Active solution platform:" label, select "New" from the drop down box.



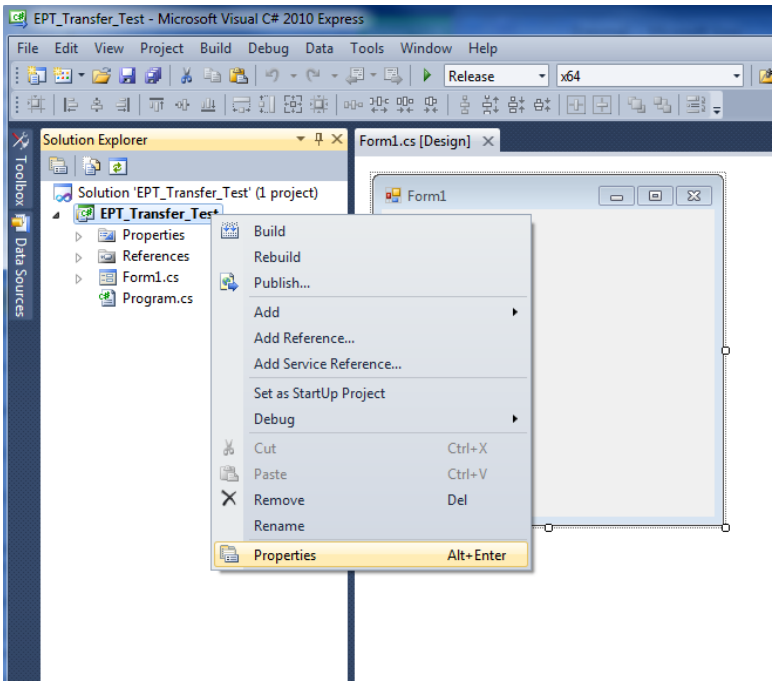
In the New Solution Platform window, click on the drop down box under "Type or select the new platform:". Select "x64".



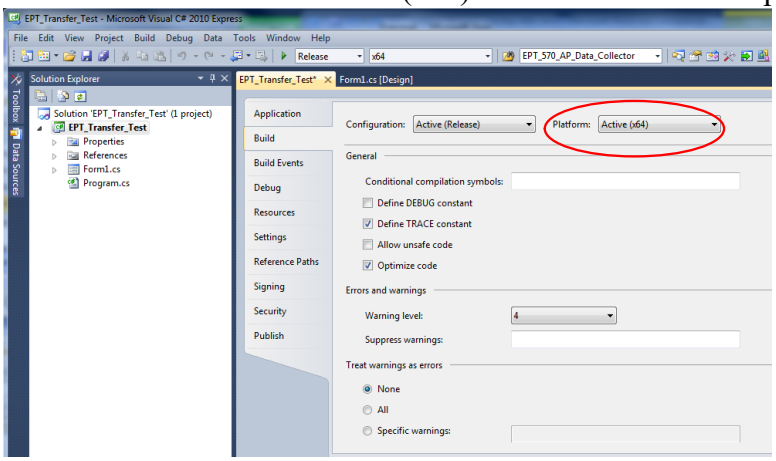
Click the Ok button. Verify that the “Active Solution Platform” and the “Platform” tab are both showing “x64”.



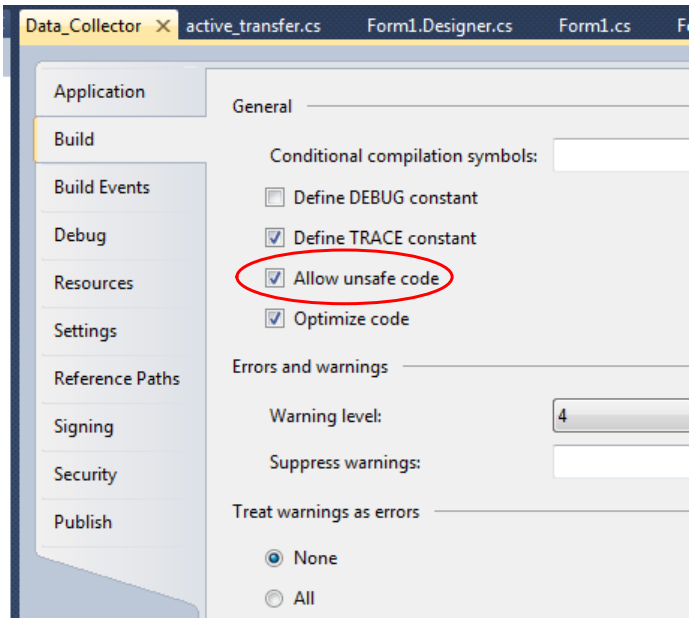
Also, select “Release” under “Active solution configuration”. Click Close.
Then, using the Solution Explorer, you can right click on the project, select Properties and click on the Build tab on the right of the properties window.



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



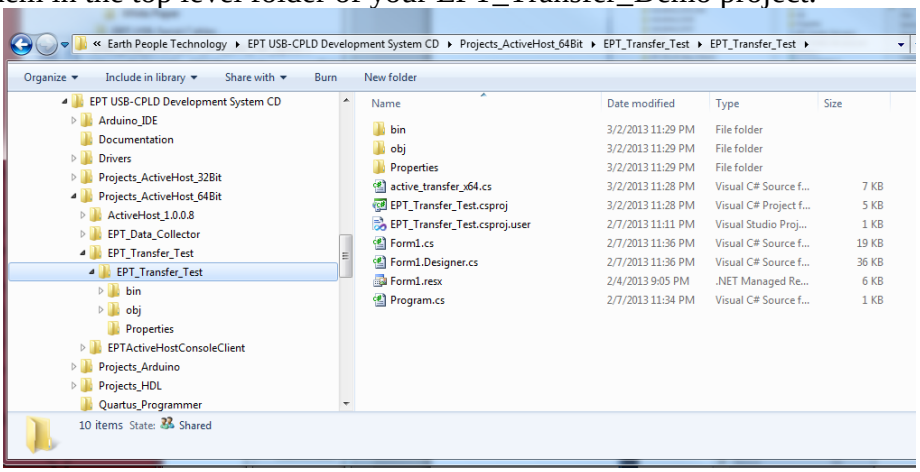
Next, unsafe code needs to be allowed so that C# can be passed pointer values from the Active Host. Click on the Build tab and locate the "Allow unsafe code" check box. Check the box



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.

7.2 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.

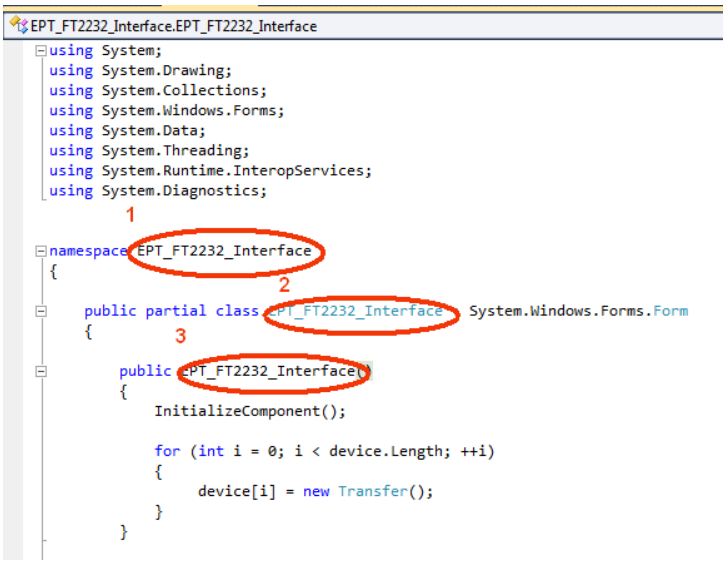


7.2.1 Changing Project Name

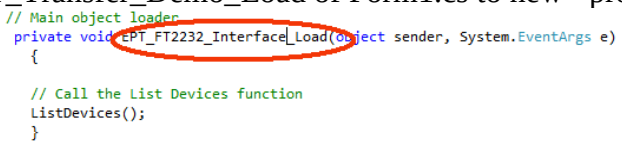
NOTE

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 chages can be made by modifying the following:

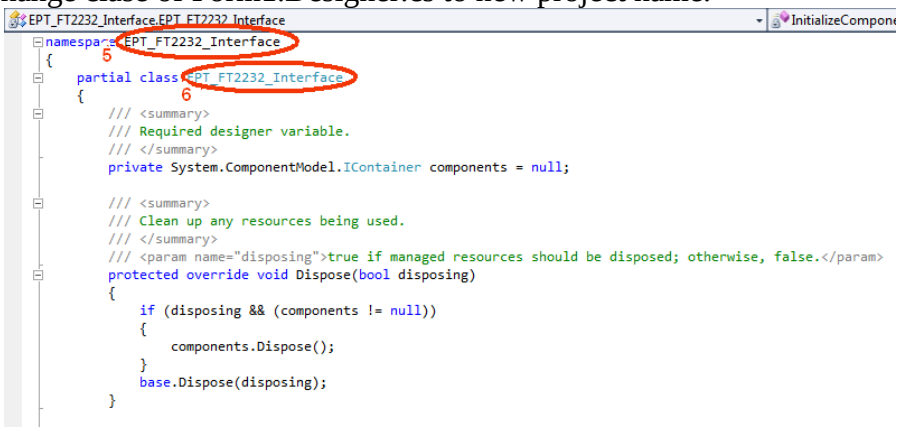
1. Change namespace of Form1.cs to new project name.
2. Change class of Form1.cs to new project name.
3. Change constructor of Form1.cs to new project name.



4. Change EPT_Transfer_Demo_Load of Form1.cs to new <project name>_Load



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



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.

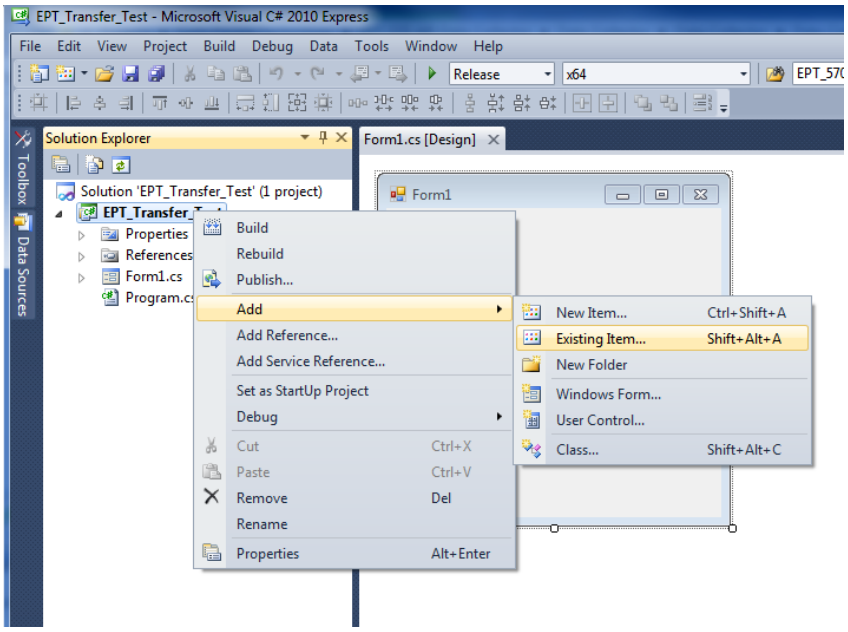
```
this.Controls.Add(this.btnTrigger3);
this.Controls.Add(this.btnTrigger2);
this.Controls.Add(this.btnTrigger1);
this.Controls.Add(this.btnCloseDevice);
this.Controls.Add(this.btnOpenDevice);
this.Controls.Add(this.cmbDevList);
this.Controls.Add(this.LEDBox);
this.Controls.Add(this.gbTriggerOut);
this.Controls.Add(this.gbTransferControl);
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.gbTransferControl.PerformLayout();
this.groupBox1.ResumeLayout(false);
this.groupBox1.PerformLayout();
this.ResumeLayout(false);
this.PerformLayout();
```

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

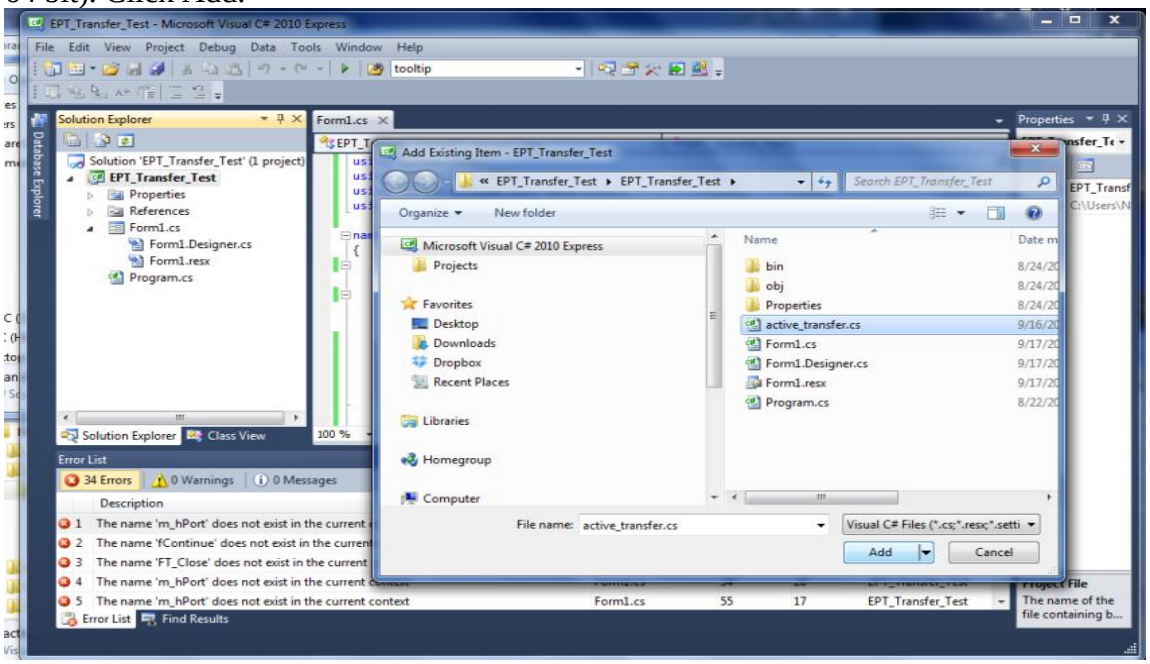
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;

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());
        }
    }
}
```

7.2.2 Add Files to Project

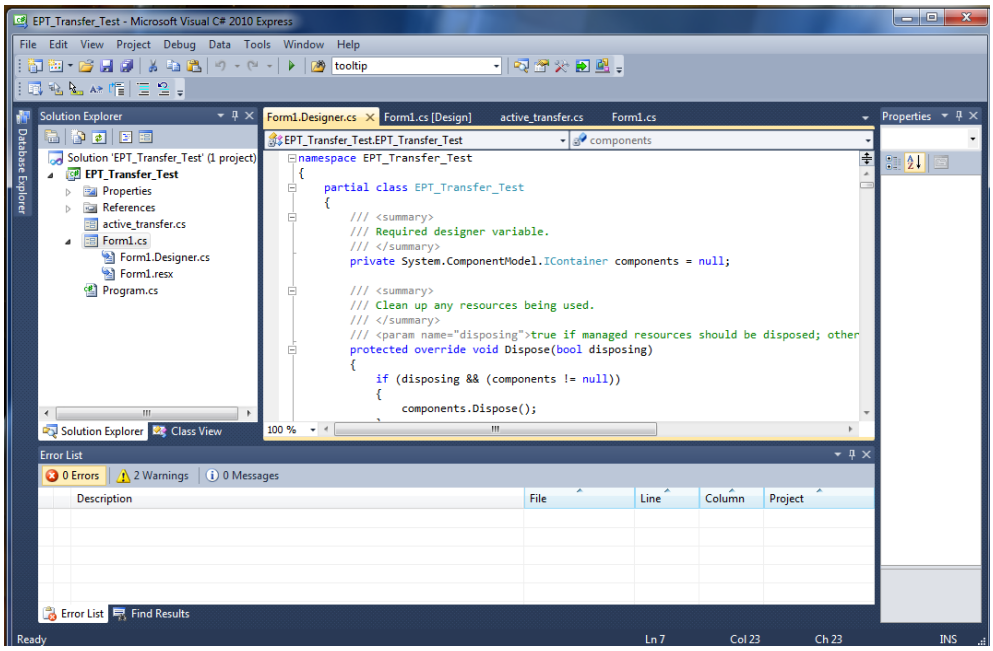
Open the EPT_Transfer_Demo project. Right click on the project in the Solutions Explorer. Select Add->Existing Item.



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.

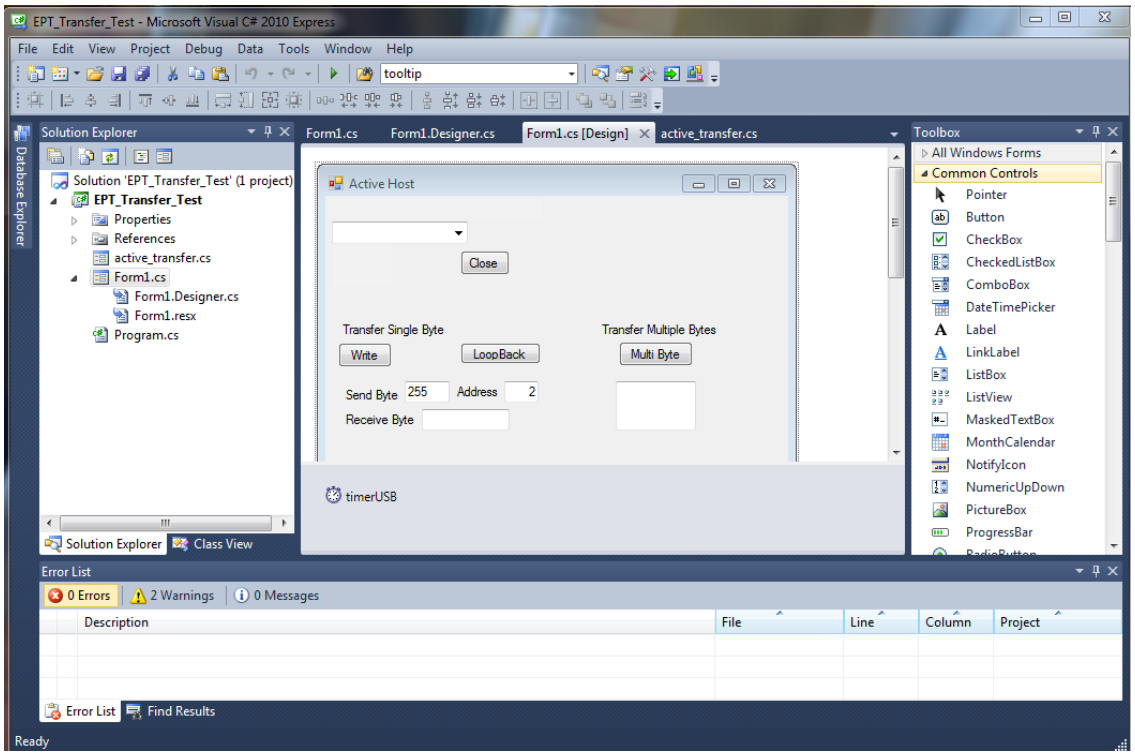


7.2.3 Adding Controls to the Project

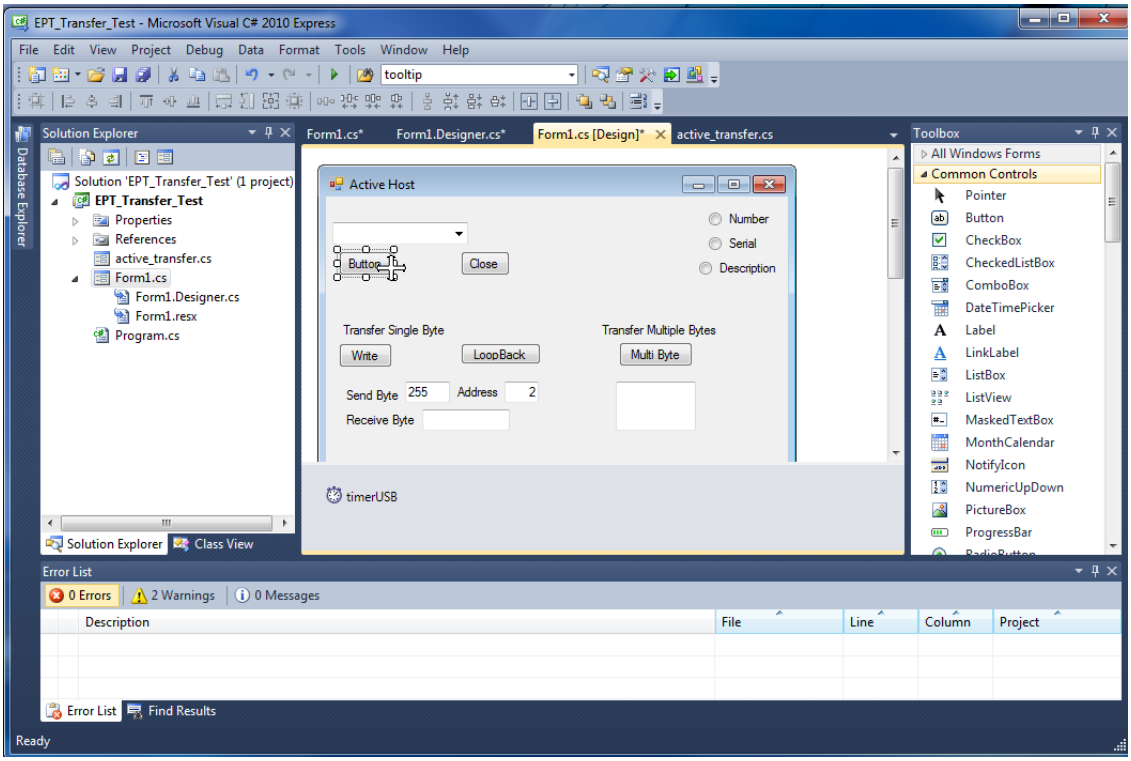
Although, the C# language is very similar to C Code, there are a few major differences. The first is C# .NET environment is event based. A second is C# utilizes classes. This guide will keep the details of these items hidden to keep things simple. However, a brief introduction to events and classes will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

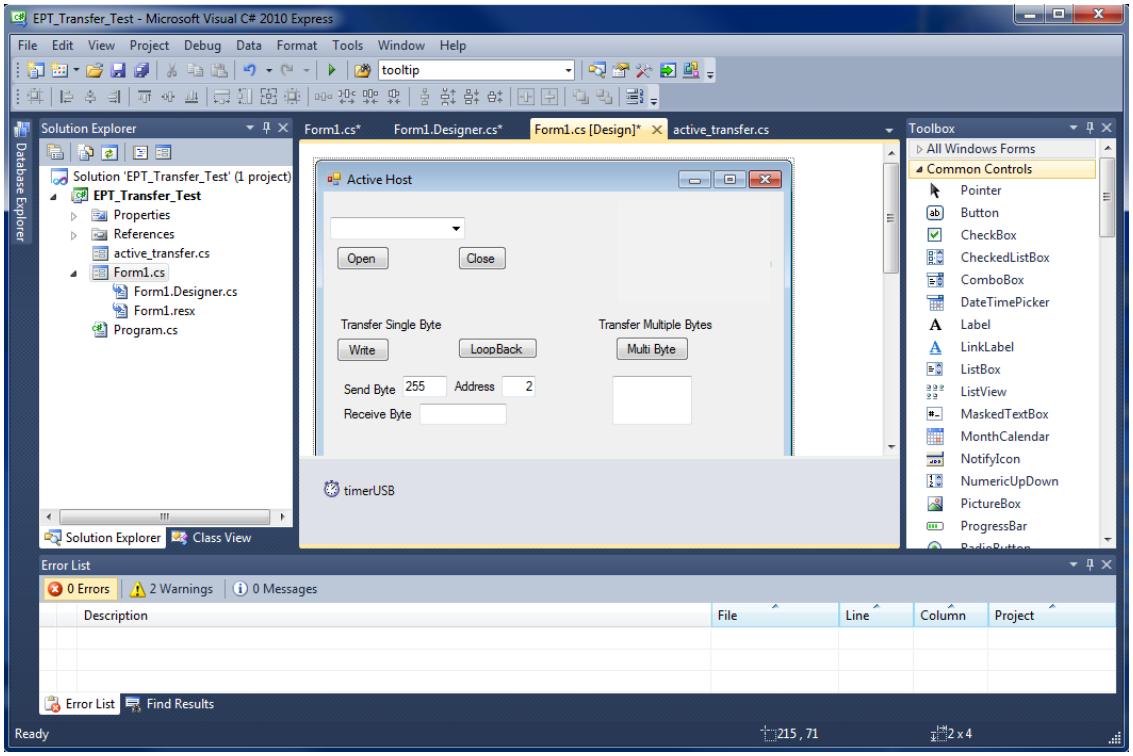
The easiest way to add a button to a form is to double click the Form1.cs in the Solution Explorer. Click on the  button to launch the Toolbox.



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

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;
}
```

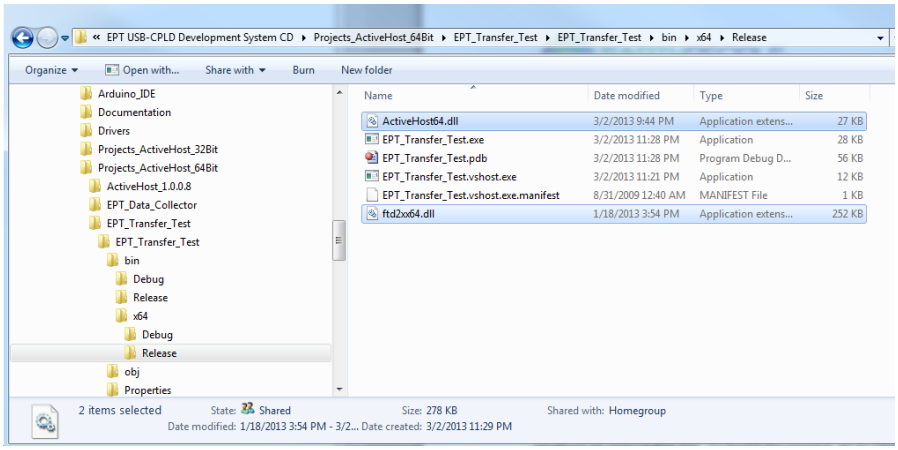
When btnOpenDevice_Click is called, it calls the function "OpenDevice()". This function is defined in the dll and will connect to the device selected in the combo box. This is a quick view of how to create, add files, and add controls to a C# project. The user is encouraged to spend some time reviewing 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.

7.2.4 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

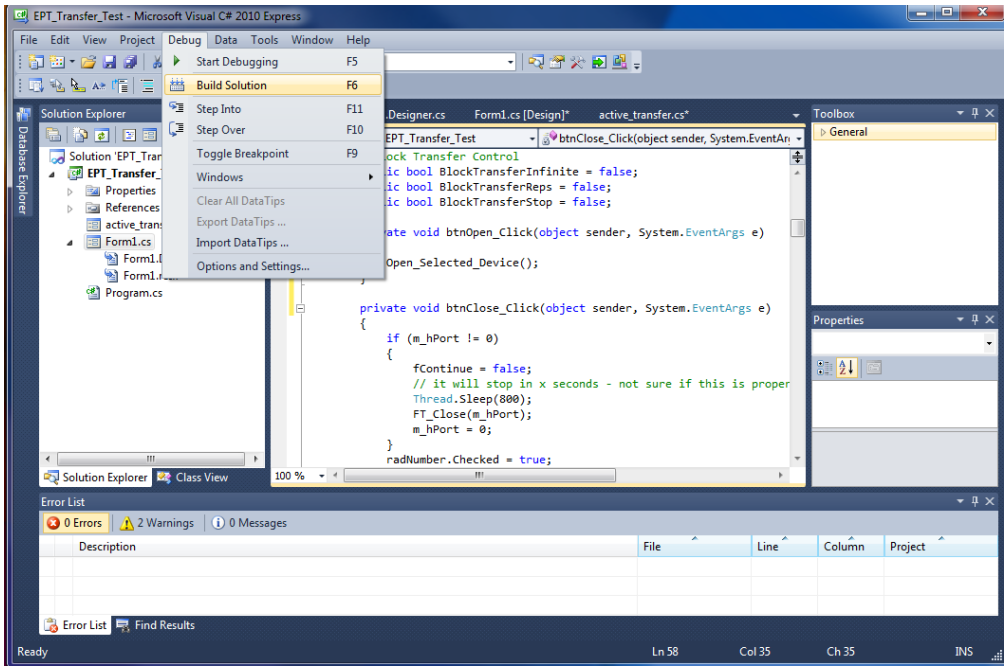
and install them in the bin\x64\x64 folder of your EPT_Transfer_Demo project.



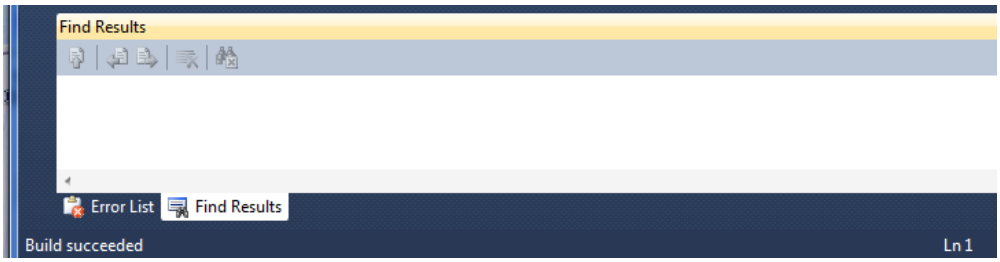
Save the project.

7.2.5 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.

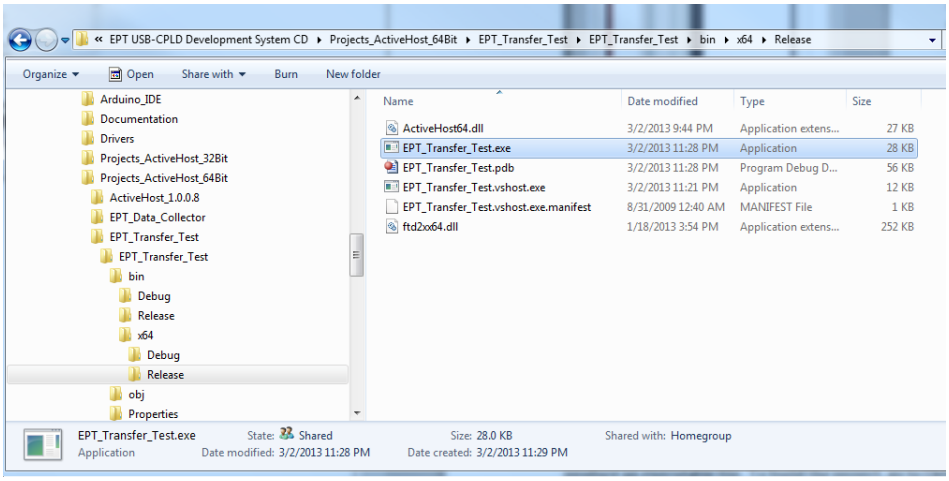


The C# Express compiler will start the building process. If there are no errors with code syntax, function usage, or linking, then the environment responds with “Build Succeeded”.

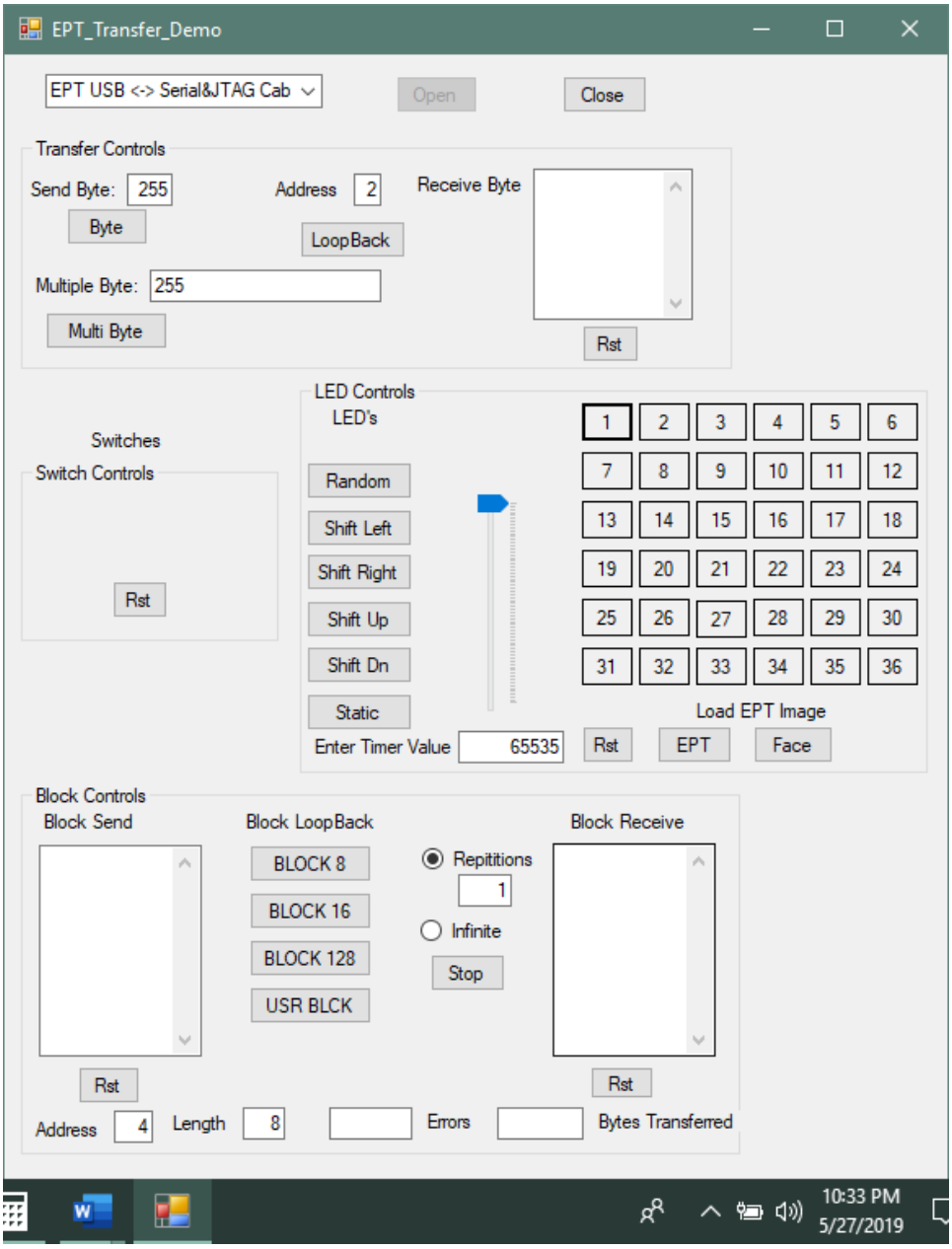


7.2.6 Testing the Project

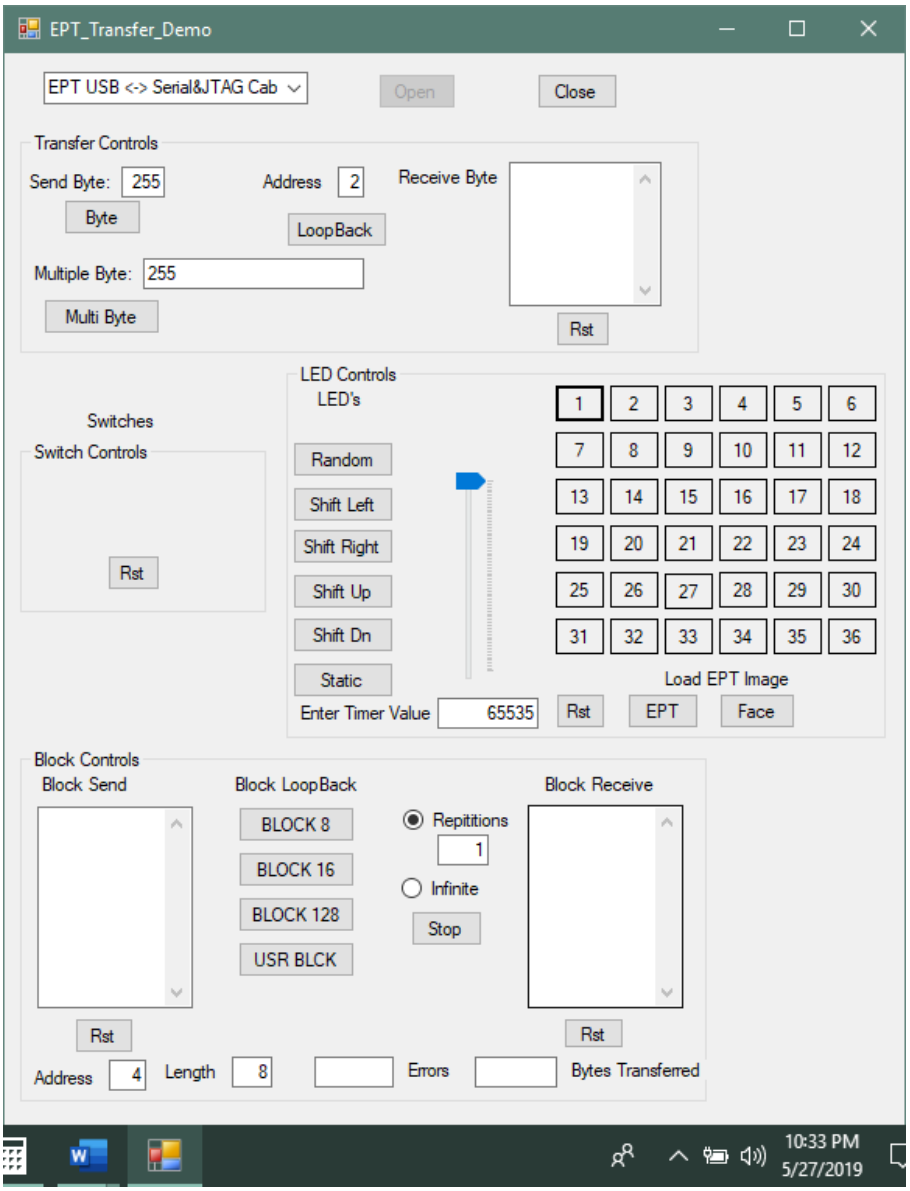
Once the project has been successfully built, it produces an *.exe file. The file will be saved in the Release or Debug folders.



The EPT_Transfer_Text.exe file can now be tested using the EPT-4CE6-AF-D2 board. To test the file, connect the EPT-4CE6-AF-D2 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.



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-D2 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.