



EARTH PEOPLE TECHNOLOGY, Inc

UnoMax CPLD DEVELOPMENT SYSTEM User Manual

The UnoMax CPLD development system provides an innovative method of developing and debugging programmable logic code. It also provides a high speed data transfer mechanism between user code and a host PC. The UnoMax CPLD development system provides a convenient, user-friendly work flow by connecting seamlessly with Intel/Altera's Quartus Prime Lite software. The user will develop the code in the Quartus environment on a Windows Personal Computer. The programmable logic code is then synthesized and loaded into the CPLD using only the Quartus Programmer tool and a standard USB cable. The Active Host SDK provides a highly configurable communications interface between the UnoMax and host PC. The board connects transparently with the Active Transfer Library in the CPLD code. This Active Host/Active Transfer combination eliminates the complexity of designing a USB communication system. The UnoMax CPLD development system is a unique combination of hardware and software.

Circuit designs, software and documentation are copyright © 2018, Earth People Technology, Inc

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

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

Table of Contents

1	Introduction and General Description	4
1.1	Test Driving the Active Host Test Application	5
1.2	EPT-5M57-AP-U3	8
1.2.1	High Speed USB Communications	10
1.2.2	Inputs and Outputs	11
1.2.3	JTAG	11
1.3	Active Host EndTerms	12
1.4	Active Transfer EndTerms	12
2	EPT Drivers.....	13
2.1	USB Driver	13
2.2	JTAG DLL Insert to Quartus Prime	19
2.2.1	Installing Quartus	19
2.2.2	Downloading Quartus	20
2.2.3	Quartus Installer	23
2.2.4	Adding the EPT_Blasters to Quartus Prime Lite.....	27
2.3	Active Host Application DLL	29
3	Active Transfer Library.....	32
3.1	EPT Active Transfer System Overview	33
3.2	Active Transfer Library	34
3.2.1	Active Trigger EndTerm	36
3.2.2	Active Transfer EndTerm	39
3.2.3	Active Block EndTerm	41
3.3	Timing Diagram for Active Transfer EndTerms	44
3.3.1	Active Trigger EndTerm Timing	44
3.3.2	Active Transfer EndTerm Timing.....	44
3.3.3	Active Block EndTerm Timing.....	45
4	Compiling, Synthesizing, and Programming CPLD	46
4.1	Setting up the Project and Compiling.....	46
4.1.1	Selecting Pins and Synthesizing.....	52
4.1.2	Programming the CPLD.....	59
5	Active Host Application.....	65
5.1	Trigger EndTerm	66
5.2	Transfer(Byte) EndTerm	66
5.3	Block EndTerm	66
5.4	Active Host DLL	67
5.4.1	Active Host Open Device.....	68
5.4.2	Active Host Read Callback Function	70
5.4.3	Active Host Triggers	71
5.4.4	Active Host Byte Transfers	73
5.4.5	Active Host Block Transfers.....	75



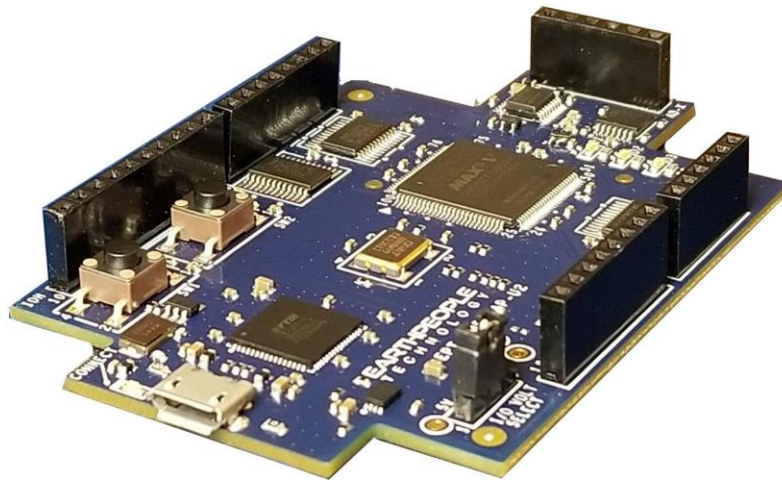
UnoMax CPLD Development System User

Manual

6	Assembling, Building, and Executing a .NET Project on the PC	79
6.1	Creating a Project	79
6.1.1	Setting up the C# Express Environment x64 bit	81
6.2	Assembling Files into the Project.....	87
6.2.1	Changing Project Name	88
6.2.2	Add Files to Project.....	90
6.2.3	Adding Controls to the Project.....	92
6.2.4	Adding the DLL's to the Project.....	95
6.2.5	Building the Project	96
6.2.6	Testing the Project.....	97
7	Using the Analog to Digital Converter	101
7.1	ELECTRICAL CHARACTERISTICS	103
7.2	3-Wire Serial Interface	103
7.3	Output Data Format	104
7.4	Register Descriptions	106
7.5	Conversion Register	107
7.6	Setup Register.....	108
7.7	Averaging Register.....	109
7.8	Reset Register.....	110
8	The UnoProLyzer Application.....	111
8.1	Accessing the UnoProLyzer Application	112
9	Developing Verilog Applications	123
10	APPENDIX I.....	123

1 Introduction and General Description

The Earth People Technology UnoMax CPLD development system hardware consists of a High Speed USB chip, a CPLD and a four channel ADC. The USB interface provides both JTAG programming for the CPLD and a High Speed communications path. The CPLD is an Intel/Altera MAXV chip providing 570 Logic Cells. The ADC is a four channel 300KSample/sec data converter. The software consists of the Active Host SDK for the PC. The firmware includes the Active Transfer Library which is used in the CPLD to provide advanced functions for control and data transfer to/from a PC connection.



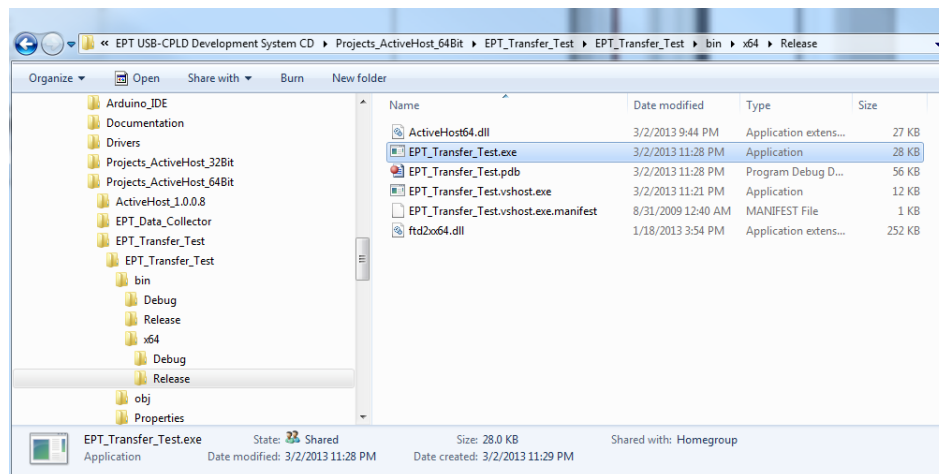
The UnoMax Development System allows users to write HDL code (either Verilog or VHDL) that will implement any digital logic circuit. The user's HDL code is compiled and synthesized and packaged into a programming file. The programming file is programmed into the CPLD using the JTAG channel of the USB to Serial chip, the

Manual

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 UnoMax 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 CPLD 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.

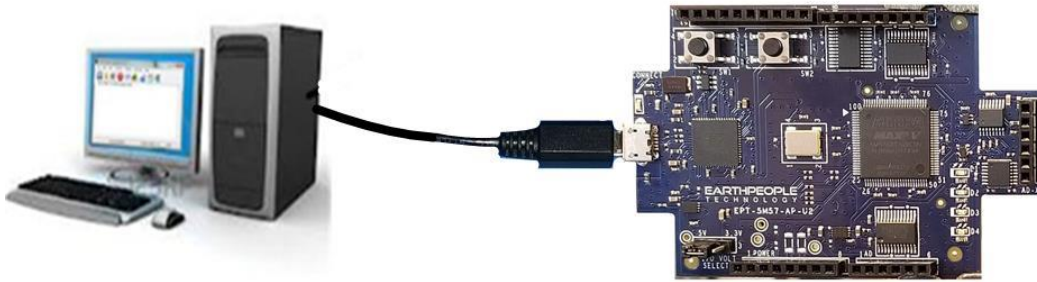
1.1 Test Driving the Active Host Test Application

The UnoMax board comes pre-loaded with the EPT_Transfer_Test HDL project in the CPLD. This project allows the user to test out the functions of the Active Host API and the board hardware.

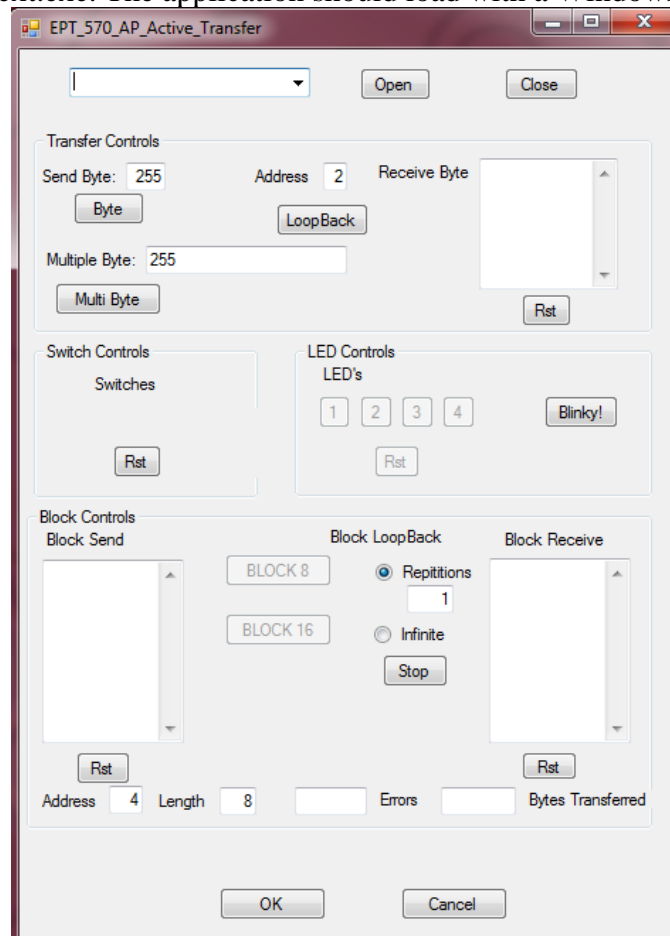


To test drive the application, connect the UnoMax to the Windows PC using Type A to Micro B USB cable. Load the driver for the board. See the section EPT Drivers for instructions on loading the EPT-5M57-AP-U3 driver.

Manual

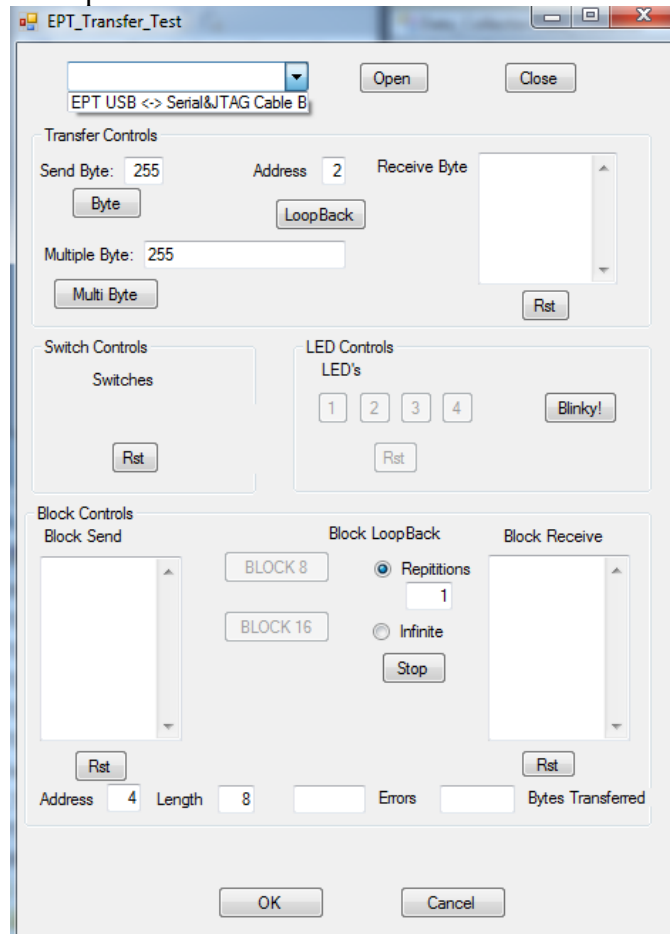


Next, open a Windows Explorer browser. Browse to the Projects_ActiveHost_xxBit\EPT_Transfer_Test\EPT_Transfer_Test\bin\X64\Release\ folder on the UNO_USB_CPLD_PROJECT_CD. Double click on the EPT_Transfer_Text.exe. The application should load with a Windows form.



Manual

With the application loaded, select the USB-CPLD 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 UnoMax 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.



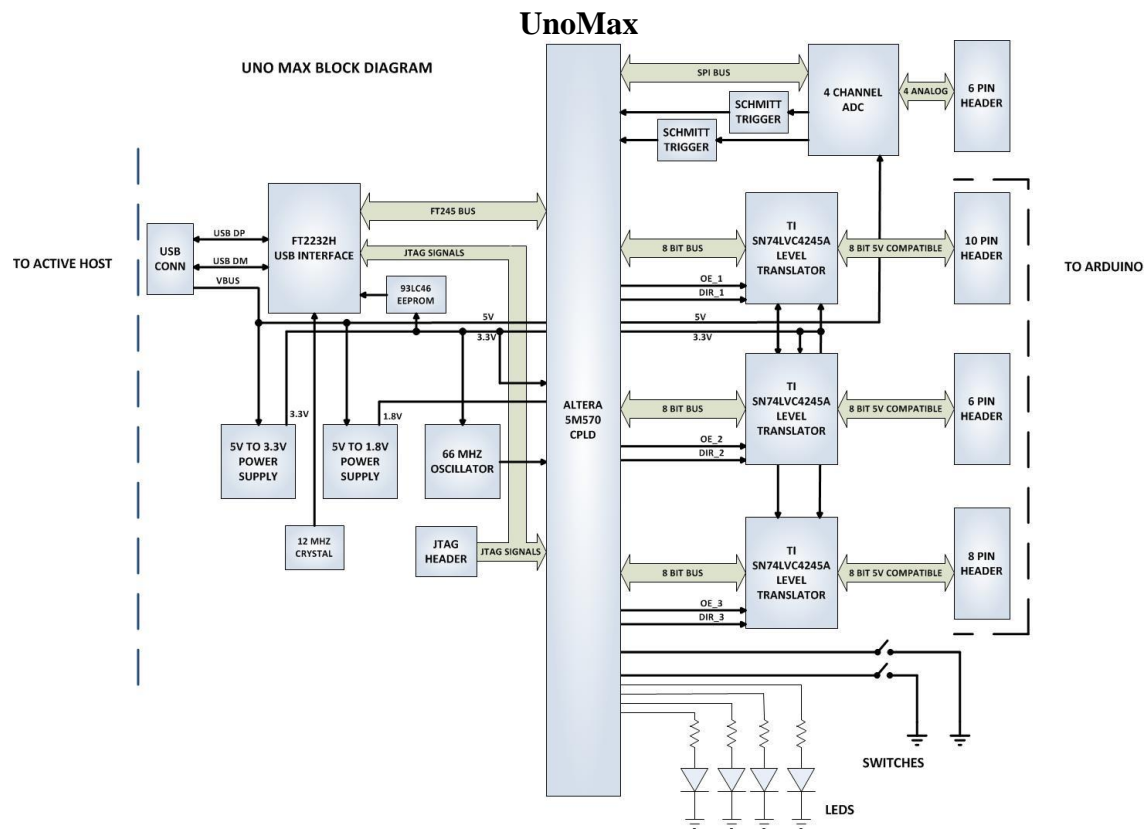
To exercise the Block Transfer EndTerm, click the “Block8” or “Block16” button in the Block Controls group. A pre-selected group of numbers appear in the Block Receive textbox.

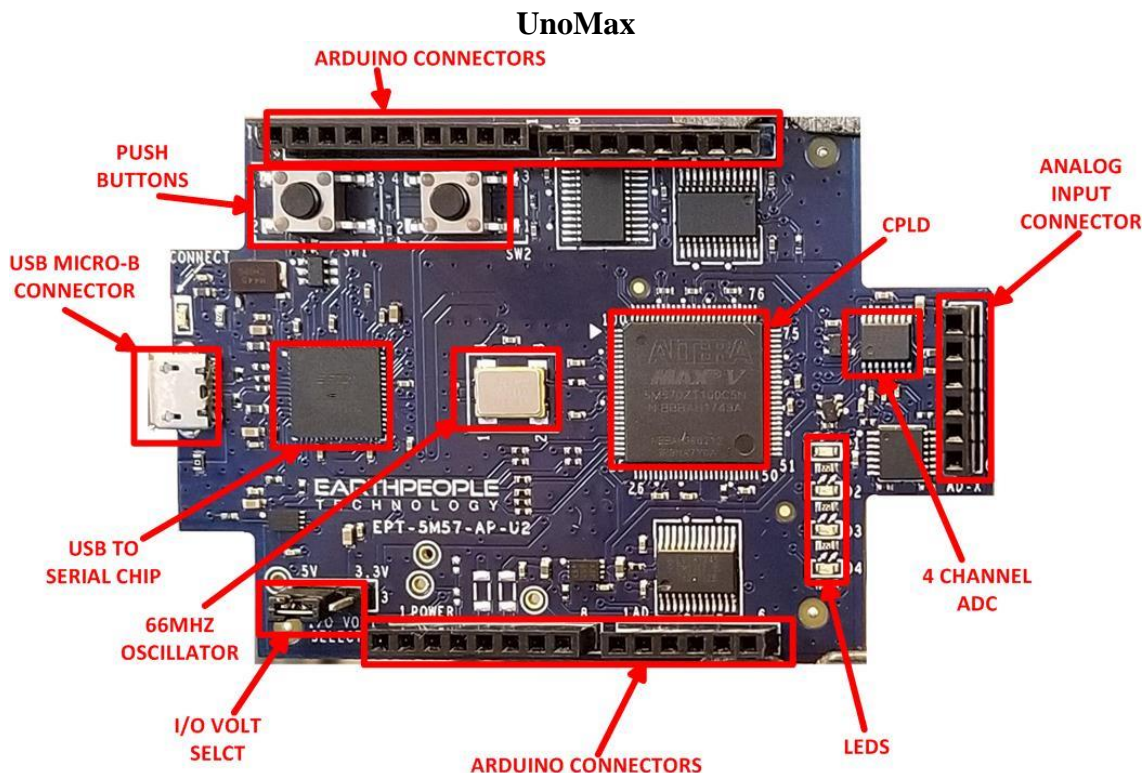
Press the PCB switches on the UnoMax to view the Switch Controls in action.

1.2 EPT-5M57-AP-U3

The UnoMax board (EPT-5M57-AP-U3) is equipped with an Intel/Altera 5M570 CPLD; which is programmed using the Quartus Prime software. The CPLD has 570 Logic Elements which is equivalent to 440 Macrocells. 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. A four channel ADC with 300KSamples/Second sample rate and dedicated 6 pin header. Twenty Four I/O's from the CPLD are attached to three 8 bit transceivers to provide 3.3V/5 Volt compatible I/O's. These 74LVC245 bidirectional voltage translator/bus transceivers are controlled by one enable and direction bit per transceiver. This means the direction of the individual bits of each transceiver cannot be selected; the direction is selected for all eight bits per transceiver. There are four green LED's and two Push Buttons that are controllable by the user code. The hardware features are as follows.

- Intel/Altera 5M570 CPLD with 440 Macrocells
- 4 Channel ADC 300KSamples/Second
- 66 MHz oscillator for driving USB data transfers and users code
- Three bidirectional voltage translator/bus transceivers
- 24 user Input/Outputs available as three 8 bit ports
- Ports have jumper selectable 3.3V/5 Volt Input/Output
- Four Green LED's accessible by the user
- Two PCB switches accessible by the user





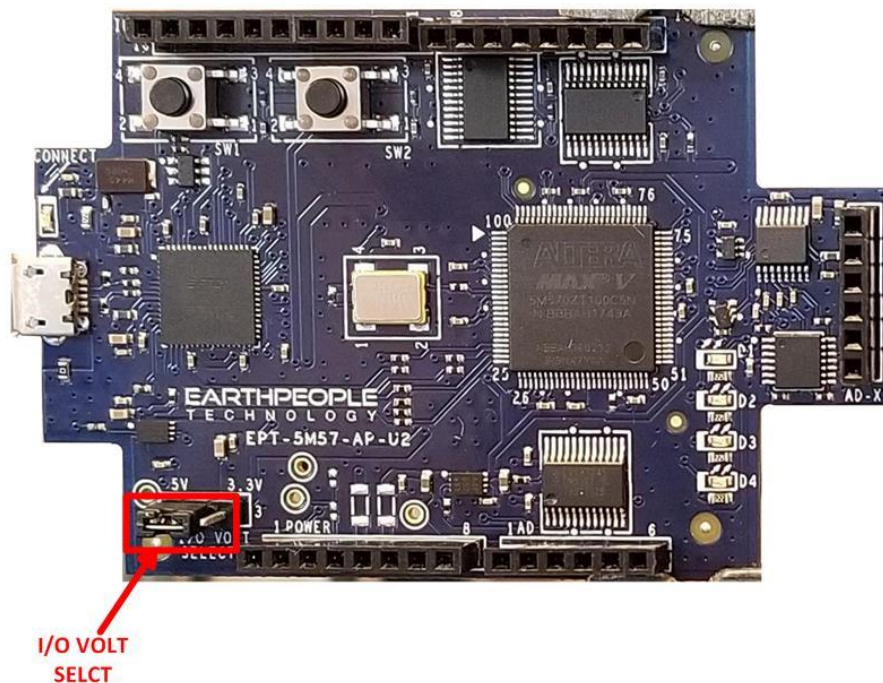
1.2.1 High Speed USB Communications

The UnoMax CPLD Development system connects an FT2232H Dual High Speed USB (480 Mbits/sec) chip to the CPLD. The CPLD uses a dedicated channel on the FT2232H for high speed transfers to the PC. Using the EPT Active Transfer Library, sustained speeds of 8 Mbytes/sec can be achieved. The transfers are bi-directional.

The FT2232H chip provides a means of data conversion from USB to serial/ parallel data and serial/parallel to USB for data being sent from the CPLD to the PC. Channel A is configured as a JTAG bus and Channel B is configured as an 8 bit parallel bus. CPLD Programming commands are transmitted via the JTAG bus (channel A). Channel B has one dual port 4Kbyte FIFO for transmission from Host PC to the CPLD, it also has one dual port 4Kbyte FIFO for receiving data from the CPLD to the Host PC. The FT2232H Mini Module provides its own 12 MHz clock and +3.3V and +1.8V power supplies. The +3.3V power supply output is used by the EPT-UnoMax for all of its +3.3V power budget.

1.2.2 Inputs and Outputs

There are 24 Inputs/Outputs which are selectable between +3.3V and +5 Volt. JMP1 is used to select which voltage the 24 Inputs/Outputs are set to.



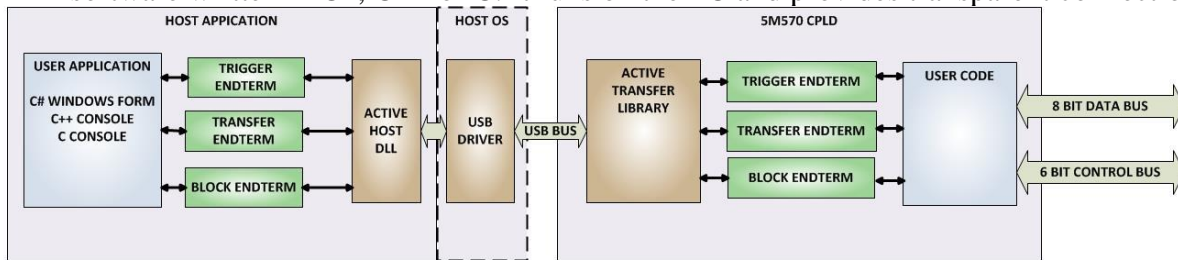
The I/O's are organized as three 8 bit directional ports. Each port must be defined as input or output. This means that all 8 bits of a port will point in the same direction, depending on the direction bit of the transceiver. The direction bit can be changed at any time, so that a port can change from input to output in minimum setup time of 6 nanoseconds. Each port also has an enable pin. This enable pin will enable or disable the bits of the port. If the port is disabled, the bits will "float".

1.2.3 JTAG

The UnoMax uses the second channel of the FT2232H chip as a dedicated CPLD programming port. The CPLD must be programmed via JTAG signals and the FT2232H has built in JTAG signals. The CPLD can be programmed directly from Quartus Prime Lite by using the "jtag_hw_mbftdi_blaster.dll". Just click on the Programmer button and select the EPT-Blaster.

1.3 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 CPLD 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 CPLD.

Receiving data from the CPLD is made simple by Active Host. Active Host transfers data from the CPLD 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 CPLD. The user application does not need to schedule a read from the USB or call any blocking threads.

1.4 Active Transfer EndTerms

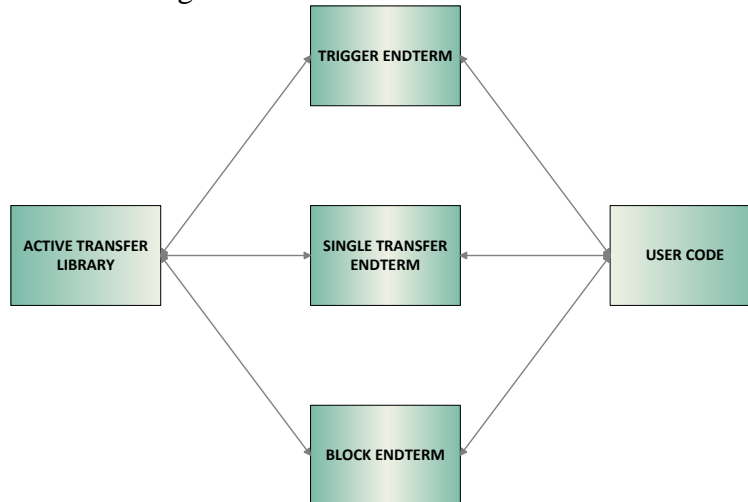
The Active Transfer Library is a portfolio of HDL modules that provides an easy to use yet powerful USB transfer mechanism. The user HDL code communicates with EndTerms in the form of modules. These EndTerm modules are commensurate with the Active Host EndTerms. There are three types of EndTerms in the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

- Trigger Endterm
- Transfer Endterm

Manual

- 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

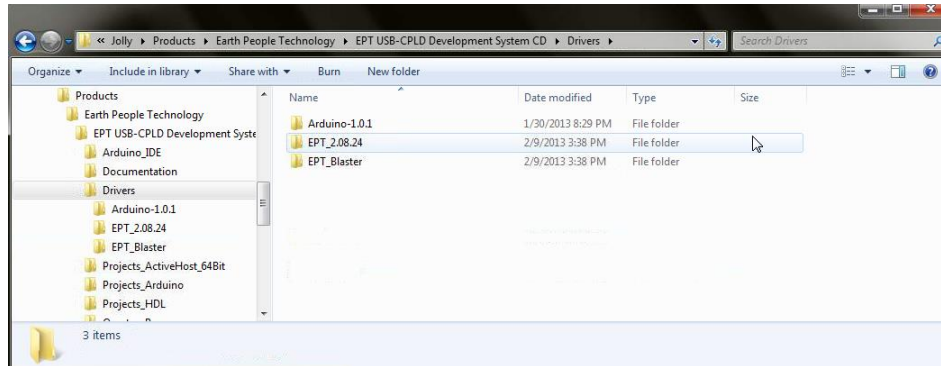
The EPT USB-CPLD Development system requires drivers for any interaction between PC and the UnoMax. The communication between the two consists of programming the CPLD and data transfer. In both cases, the USB Driver is required. This will allow Windows to recognize the USB Chip and setup a pathway for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

2.1 USB Driver

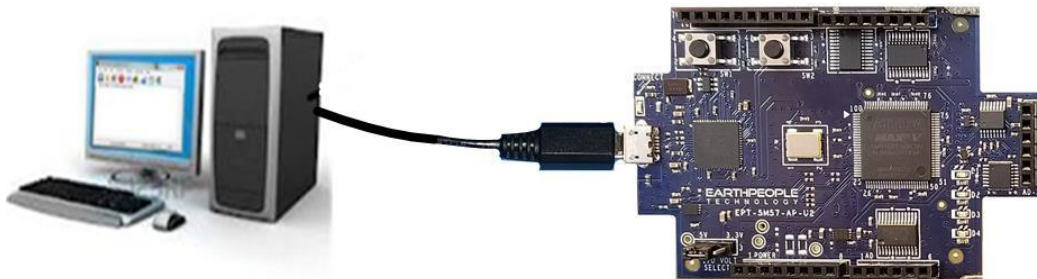
The UnoMax board uses an FTDI FT2232H USB to Serial chip. This chip provides the USB interface to the PC and the serial/FIFO interface to the CPLD. The FT2232H requires the use of the EPT USB driver. To install the driver onto your PC, use the EPT_2.08.24 Folder. The installation of the EPT_2.08.24 driver 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 USB-CPLD Development System DVD using Windows Explorer.

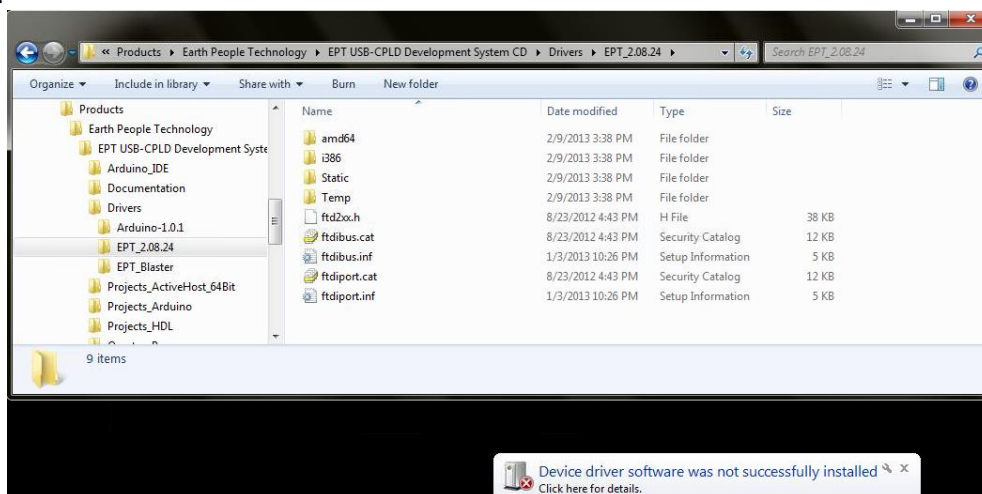
Manual



Plug in the UnoMax device into an available USB port.

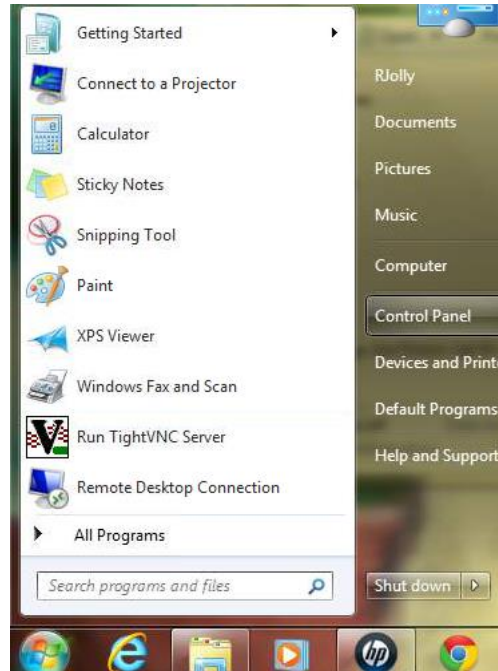


Windows will attempt to locate a driver for the USB device. When it does not find one, it will report a error, "Device driver software was not successfully installed". Ignore this error.

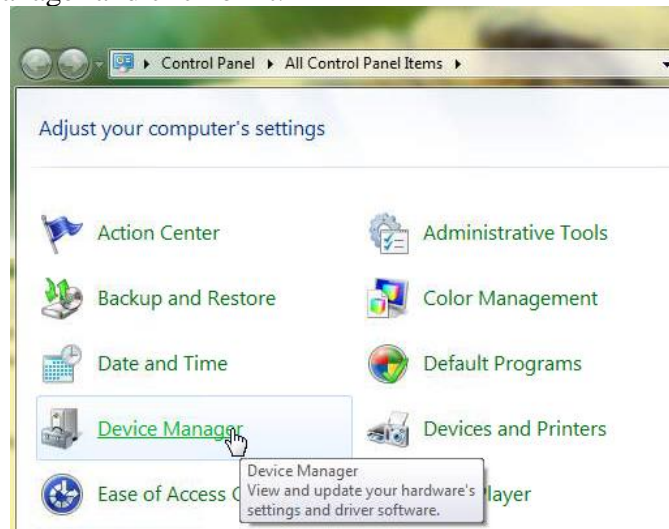


Go to Start->Control Panel

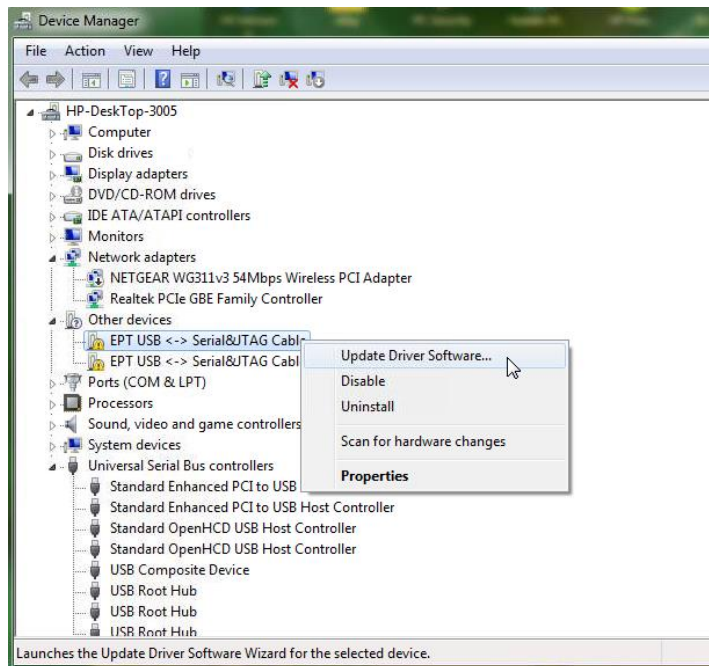
Manual



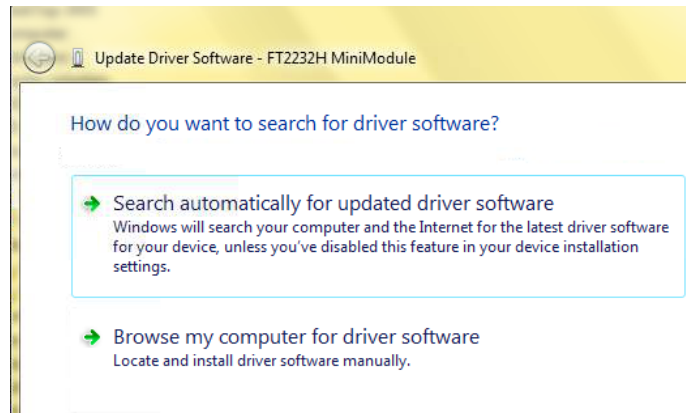
Locate Device Manager and click on it.



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

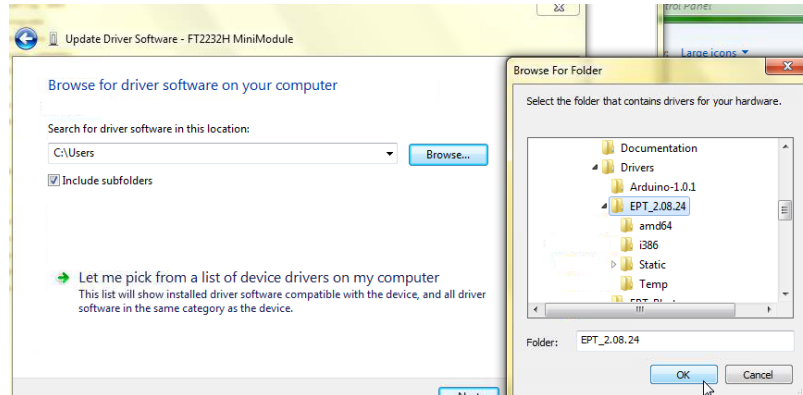


At the Update Driver Software Window, select “Browse my computer for driver software”.

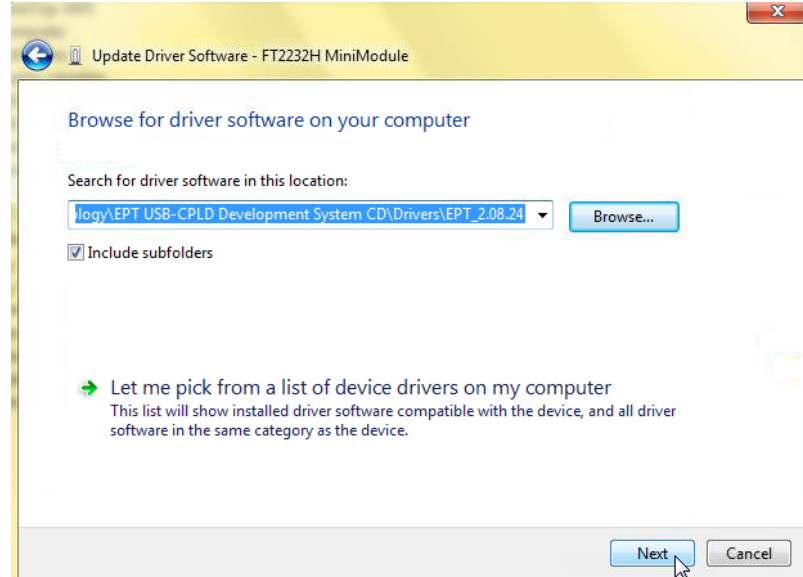


Click the Browse button and browse to the \Drivers\EPT_2.08.24 folder of the EPT USB-CPLD Development System CD. Click the Ok button.

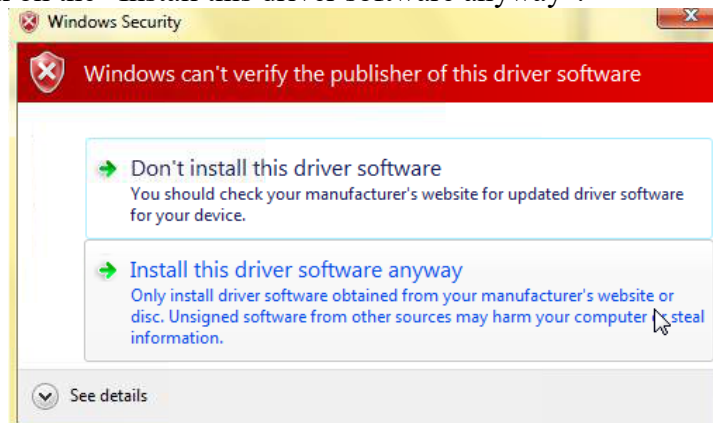
Manual



Click the Next button

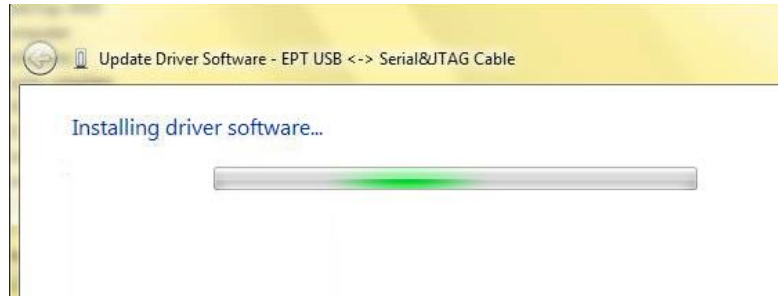


The next window is the Windows Security notice. The EPT driver is not signed by Windows. Click on the “Install this driver software anyway”.



Windows will add the EPT_2.08.24 driver to the System Registry.

Manual

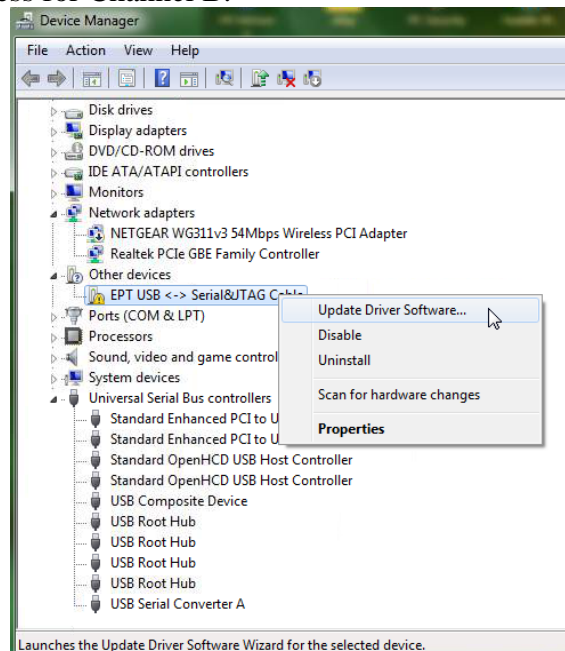


When Windows has completed the update driver the following screen will be displayed.



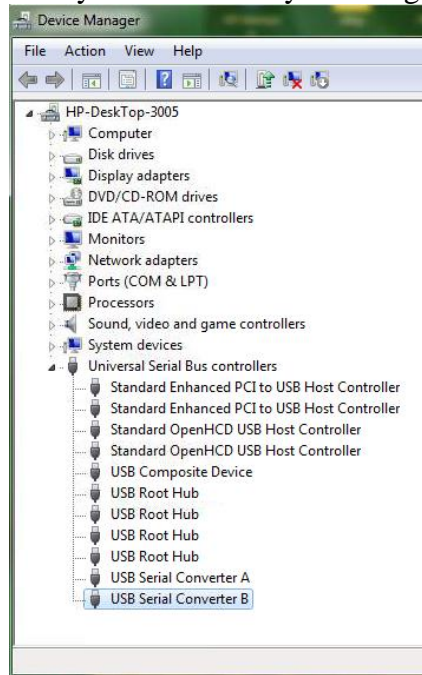
Channel A of the FT2232H chip is ready for use.

Next, repeat the process for Channel B.



Manual

The driver files will automatically install in the System Registry.



When this is complete, the drivers are installed and the UnoMax can be used with for programming and USB data transfers.

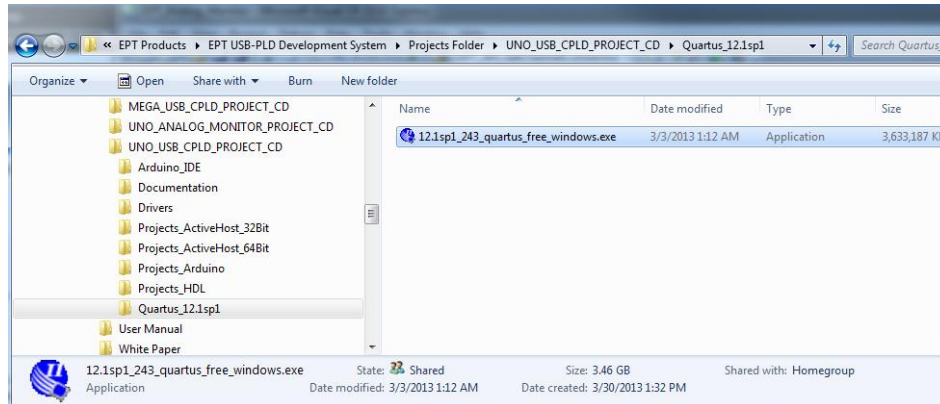
2.2 JTAG DLL Insert to Quartus Prime

The JTAG DLL Insert to Quartus Prime Lite allows the Programmer Tool under Quartus to recognize the UnoMax. The UnoMax can then be selected and perform programming of the CPLD. The file, `jtag_hw_mbftdi_blaster.dll` must be placed into the folder that hosts the `jtag_server` for Quartus.

2.2.1 Installing Quartus

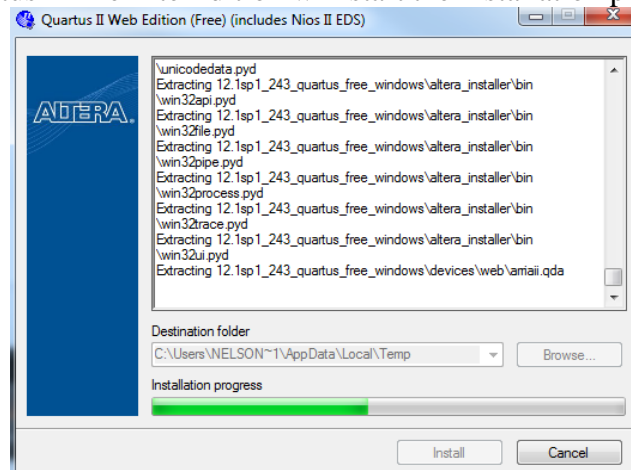
Locate the `Quartus_XXXX` folder (The XXXX signifies the version of Quartus Prime Lite on the DVD, the version of Prime is subject to change) on the EPT USB-CPLD Development System DVD.

Manual



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

If you don't need to download Quartus, double click on the xxx_quartus_free_widows.exe (the xxx is the build number of the file, it is subject to change). The Quartus Prime Lite Edition will start the installation process.



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

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

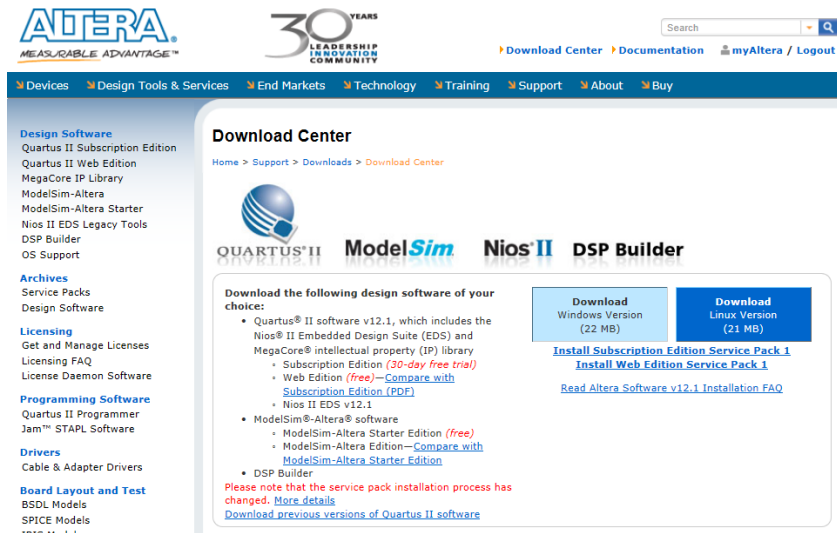
2.2.2 Downloading Quartus

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

<http://fpgasoftware.intel.com/?edition=lite>

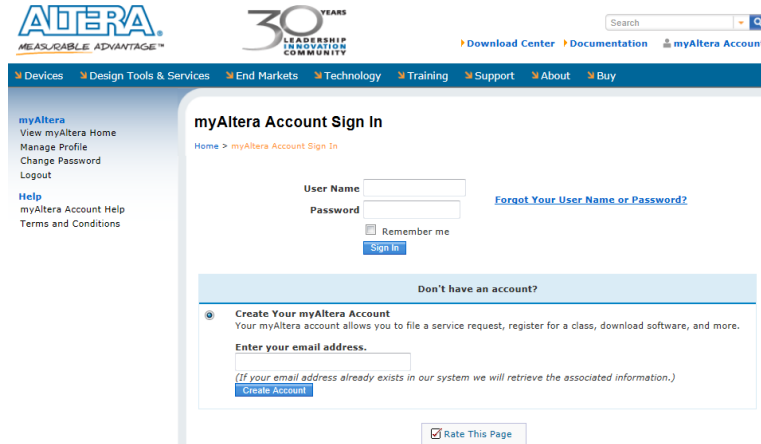
Click on the Download Windows Version.

Manual



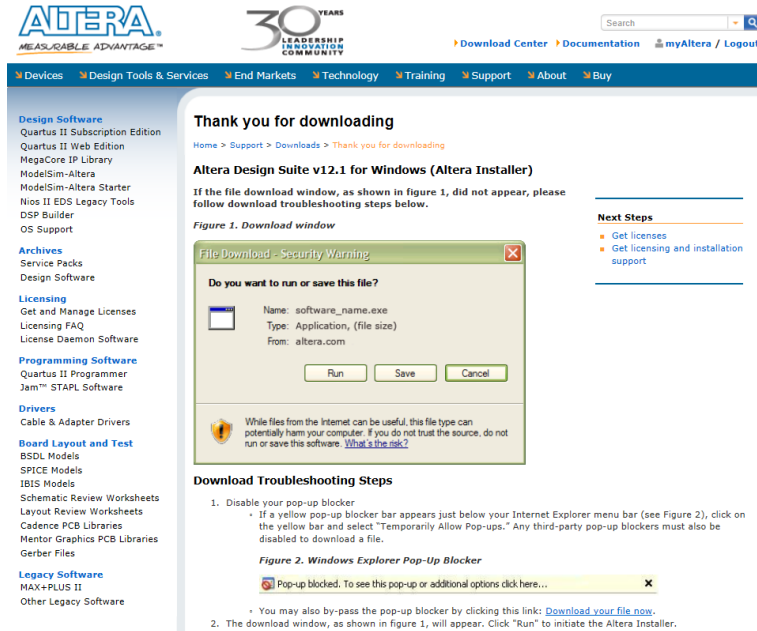
The screenshot shows the Altera Download Center page. The header includes the Altera logo, a 30th anniversary badge, and navigation links for Download Center, Documentation, myAltera, and Logout. A search bar is also present. The main content area is titled "Download Center" and lists various design software options: Quartus II, ModelSim, Nios II, and DSP Builder. It provides download links for Windows and Linux versions of the software, along with links to service packs and installation FAQs. A sidebar on the left contains links to Design Software, Archives, Licensing, Programming Software, Drivers, and Board Layout and Test.

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



The screenshot shows the myAltera Account Sign In page. It features a sign-in form with fields for User Name and Password, a "Remember me" checkbox, and a "Sign In" button. A link for "Forgot Your User Name or Password?" is also provided. Below the sign-in form, there is a section titled "Don't have an account?" with a link to "Create Your myAltera Account". This section explains that a myAltera account allows users to file service requests, register for classes, download software, and more. It includes a field for "Enter your email address" and a note that if the email already exists in the system, the associated information will be retrieved. A "Create Account" button is located at the bottom of this section. A "Rate This Page" button is also visible at the bottom of the page.

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.



Altera MEASURABLE ADVANTAGE™

30 YEARS LEADERSHIP IN INNOVATION COMMUNITY

Search []

[Download Center](#) [Documentation](#) [myAltera / Logout](#)

[Devices](#) [Design Tools & Services](#) [End Markets](#) [Technology](#) [Training](#) [Support](#) [About](#) [Buy](#)

Design Software
 Quartus II Subscription Edition
 Quartus II Web Edition
 MegaCore IP Library
 ModelSim-Altera
 ModelSim-Altera Starter
 Nios II EDS Legacy Tools
 DSP Builder
 OS Support

Archives
 Service Packs
 Design Software

Licensing
 Get and Manage Licenses
 Licensing FAQ
 License Daemon Software

Programming Software
 Quartus II Programmer
 Jam™ STAPL Software

Drivers
 Cable & Adapter Drivers

Board Layout and Test
 BSDL Models
 SPICE Models
 IBIS Models
 Schematic Review Worksheets
 Layout Review Worksheets
 Cadence PCB Libraries
 Mentor Graphics PCB Libraries
 Gerber Files

Legacy Software
 MAX+PLUS II
 Other Legacy Software

Thank you for downloading
[Home](#) > [Support](#) > [Downloads](#) > [Thank you for downloading](#)

Altera Design Suite v12.1 for Windows (Altera Installer)
 If the file download window, as shown in figure 1, did not appear, please follow download troubleshooting steps below.

Figure 1. Download window

File Download - Security Warning
 Do you want to run or save this file?
 Name: software_name.exe
 Type: Application, (file size)
 From: altera.com
 Run Save Cancel

While files from the Internet can be useful, this file type can potentially harm your computer. If you do not trust the source, do not run or save this software. [What's the risk?](#)

Download Troubleshooting Steps

1. Disable your pop-up blocker
 - If a yellow pop-up blocker bar appears just below your Internet Explorer menu bar (see Figure 2), click on the yellow bar and select "Temporarily Allow Pop-ups." Any third-party pop-up blockers must also be disabled to download a file.

Figure 2. Windows Explorer Pop-Up Blocker

Pop-up blocked. To see this pop-up or additional options click here...

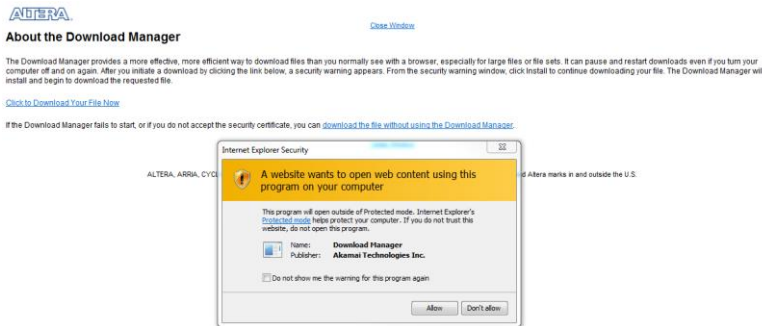
- You may also by-pass the pop-up blocker by clicking this link: [Download your file now.](#)

2. The download window, as shown in figure 1, will appear. Click "Run" to initiate the Altera Installer.

Next Steps

- [Get licenses](#)
- [Get licensing and installation support](#)

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



Altera MEASURABLE ADVANTAGE™

About the Download Manager
[Close Window](#)

The Download Manager provides a more effective, more efficient way to download files than you normally see with a browser, especially for large files or file sets. It can pause and restart downloads even if you turn your computer off and on again. After you initiate a download by clicking the link below, a security warning appears. From the security warning window, click Install to continue downloading your file. The Download Manager will install and begin to download the requested file.

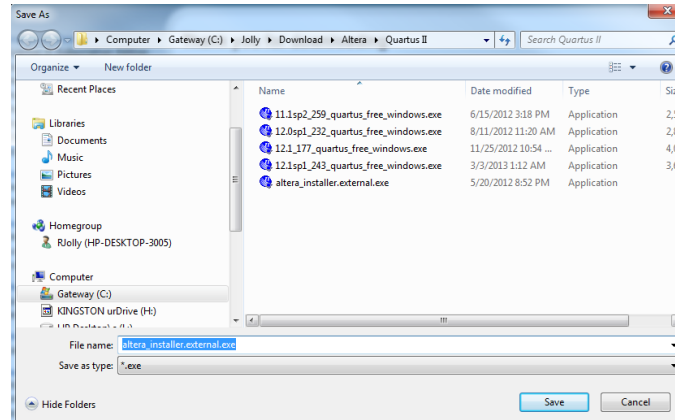
[Click to Download Your File Now](#)

If the Download Manager fails to start, or if you do not accept the security certificate, you can [download the file without using the Download Manager](#).

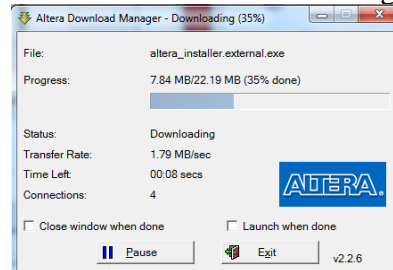
Internet Explorer Security
 A website wants to open web content using this program on your computer.
 This program will open outside of Protected mode. Internet Explorer's Protected mode helps protect your computer. If you do not trust this website, do not open this program.
 Name: Download Manager
 Publisher: Alkamel Technologies Inc.
 Do not show me the warning for this program again
 Allow Don't allow

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

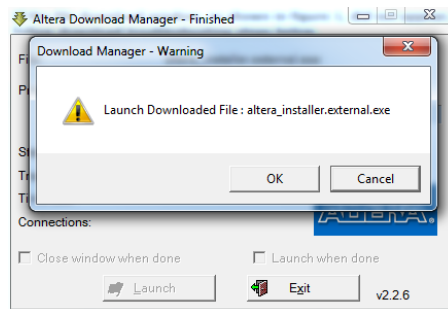
Manual



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



When it finishes, click the “Launch” button.



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

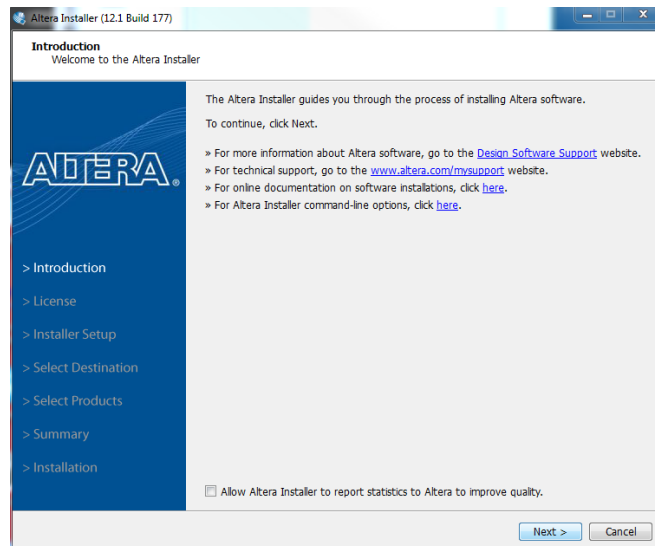
2.2.3 Quartus Installer

Click “Next” on the Introduction Window.

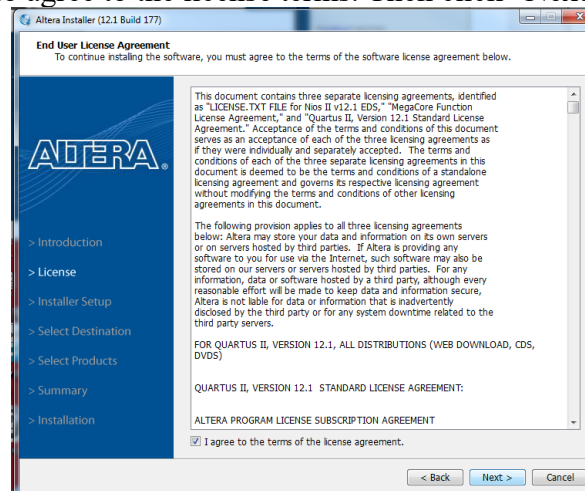


Manual

UnoMax CPLD Development System User

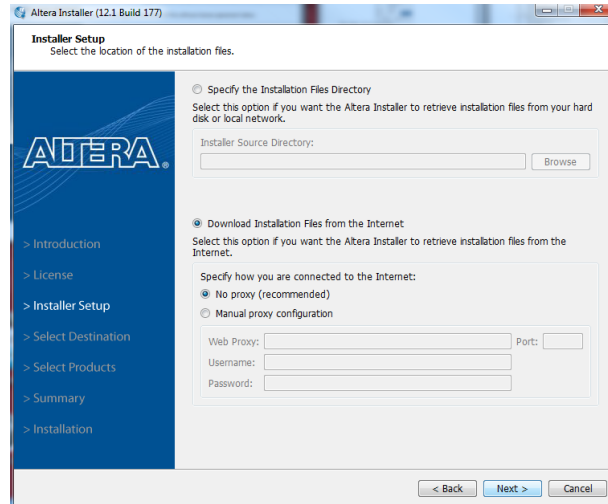


Click the checkbox to agree to the license terms. Then click “Next”.

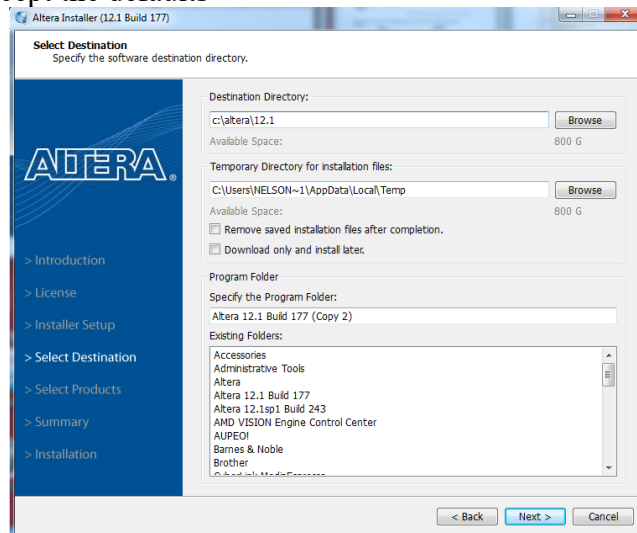


Click “Next” and accept the defaults.

Manual

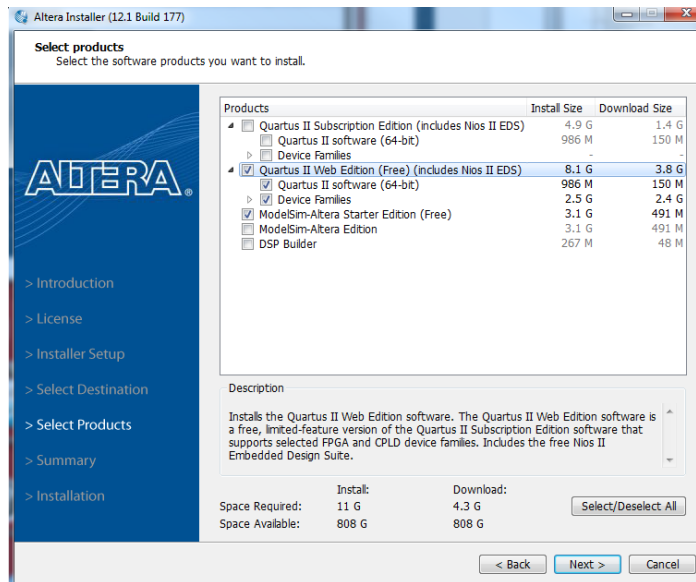


Click “Next” to accept the defaults

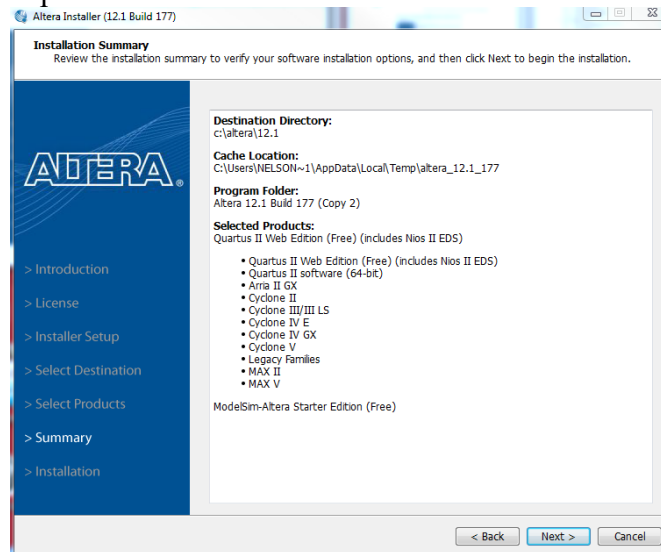


At the Select Products Window, de-select the Quartus Prime Lite Subscription Edition by clicking on its check box so that the box is not checked. Then click on the check box by the Quartus Prime Lite Edition (Free).

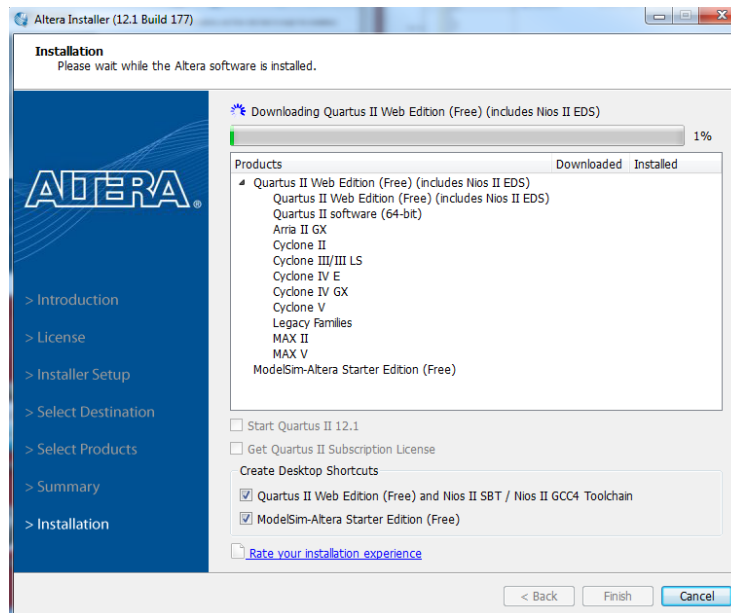
Manual



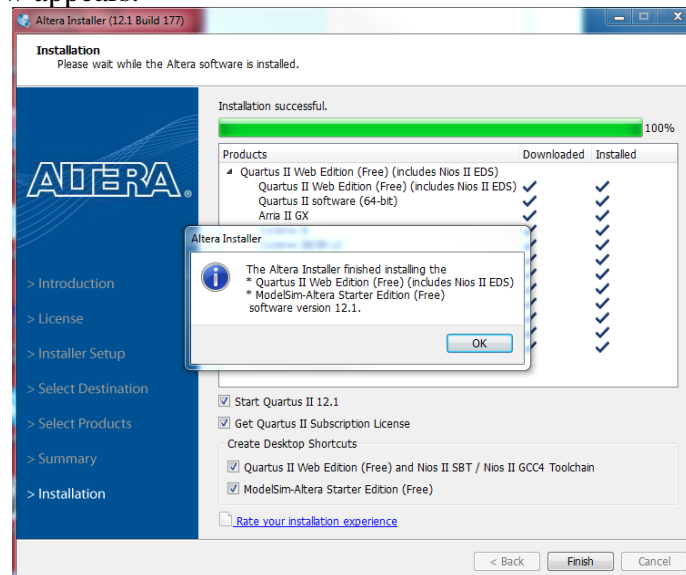
Click “Next” to accept the defaults



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.

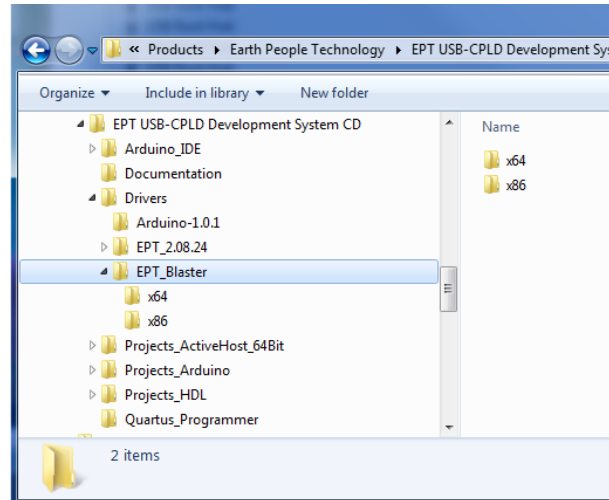


Click “Ok”, then click “Finish”. The Quartus Prime Lite is now installed and ready to be used.

2.2.4 Adding the EPT_Blaster to Quartus Prime Lite

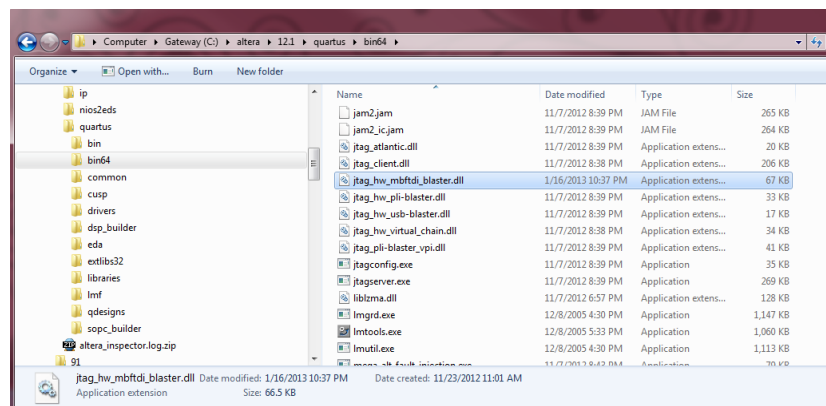
Close out the Quartus Prime Lite application. Locate the \Drivers\EPT_Blaster folder on the EPT USB-CPLD Development System DVD.

Manual



Windows 64 bit

1. Open the C:\EPT USB-CPLD Development System CD\Drivers\EPT_Blasters\x64 folder.
2. Select the file “jtag_hw_mbftdi_blasters.dll” and copy it.
3. Browse over to C:/intelFPGA_lite\xxxx\quartus\bin64.
4. Right click in the folder and select Paste
5. Click Ok.
6. Open the Quartus Prime Lite application.



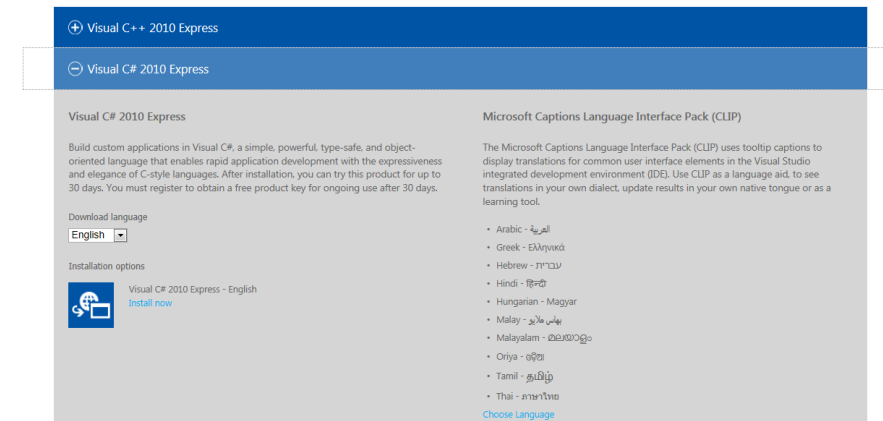
The DLL is installed and the JTAG server should recognize it. Go to the section “Programming the CPLD” of this manual for testing of the programming. If the driver is not found in the Programmer Tool->Hardware Setup box, see the JTAG DLL Insert to Quartus Prime Lite Troubleshooting Guide.

2.3 Active Host Application DLL

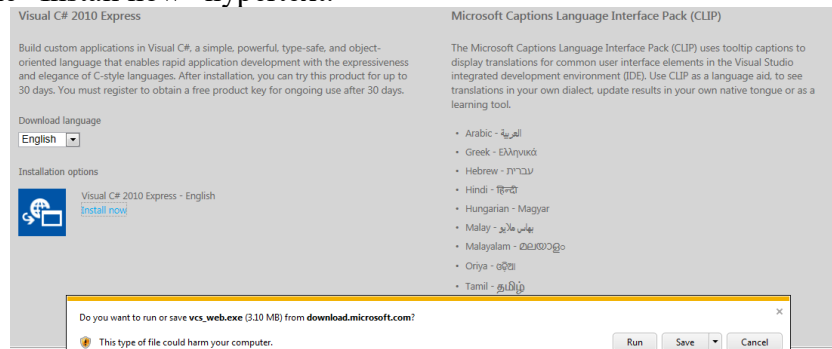
Download the latest version of Microsoft Visual C# xxxx Express (Where xxxx signifies the latest version of Visual C# Express, the following guide explains the install of the Visual C# 2010 Express) environment from Microsoft. It's a free download.

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

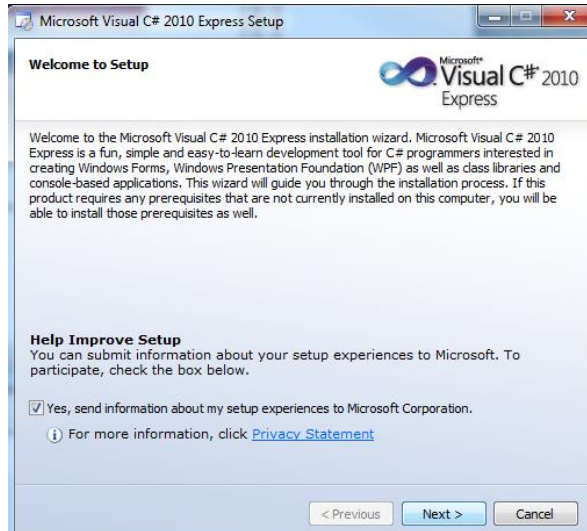
Go to the website and click on the “+” icon next to the Visual C# 2010 Express.



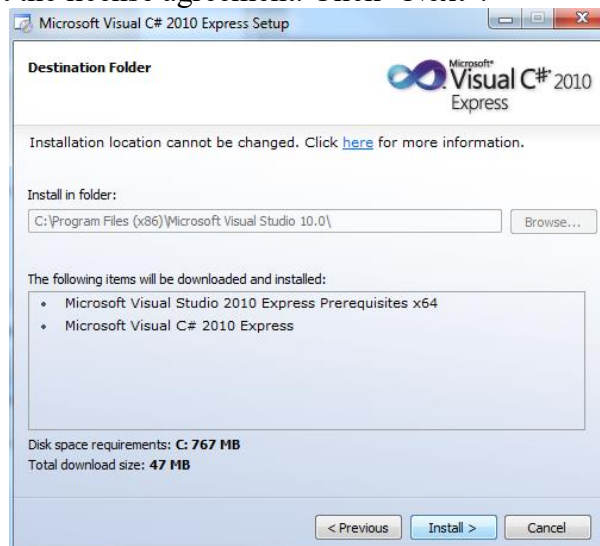
Click on the “Install now” hypertext.



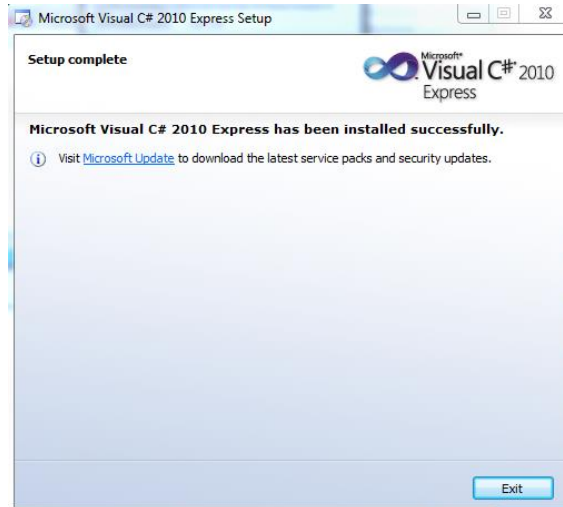
Click the “Run” button.



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

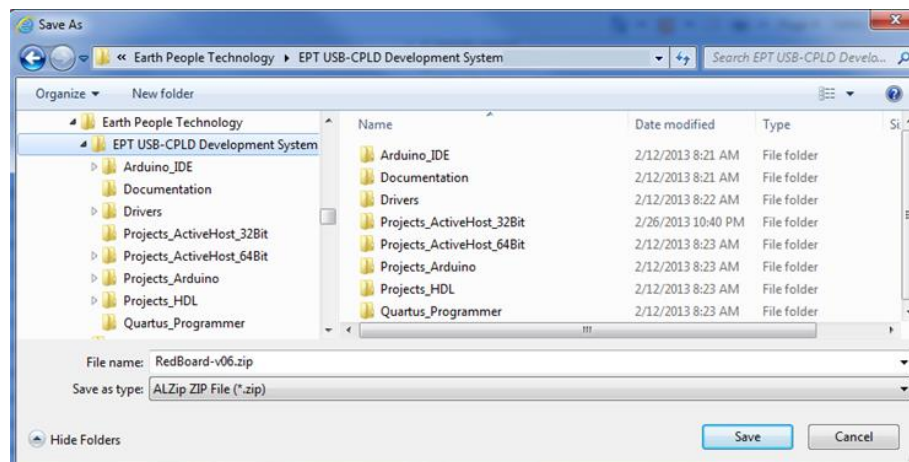


Visual C# 2010 Express will install. This may take up to twenty minutes depending on your internet connection.



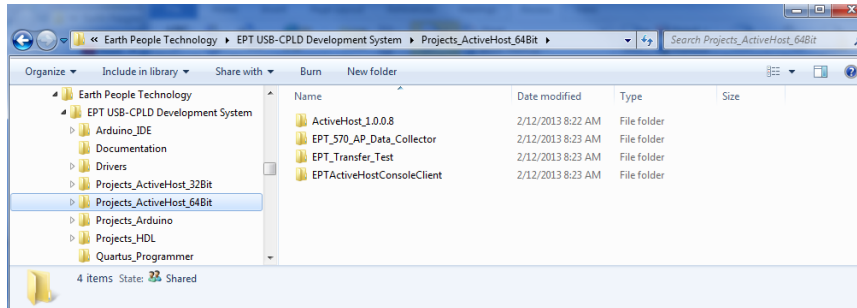
The installed successfully window will be displayed when Visual C# Express is ready to use.

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 UnoMax. 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 USB-CPLD Development System DVD.

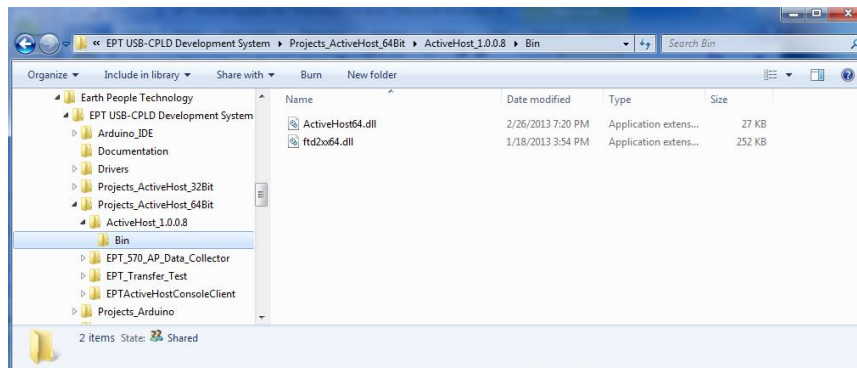


Locate the Projects_ActiveHost_64Bit folders in the EPT USB-CPLD Development System using Windows Explorer.

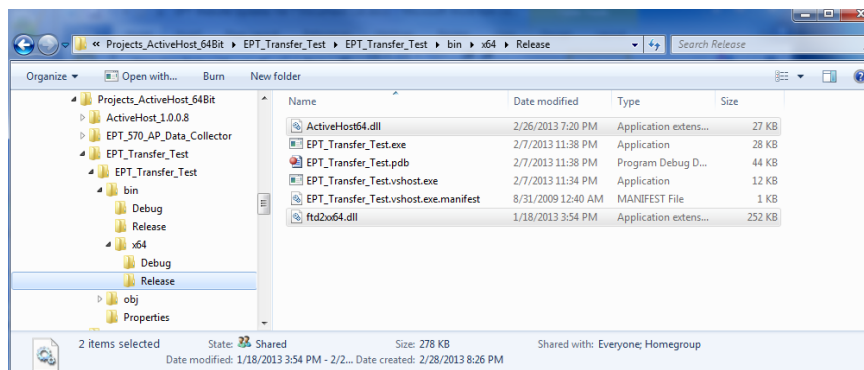
Manual



Locate the Projects_ActiveHost_64Bit \ActiveHost_1.0.0.8\Bin folder and copy the ActiveHost64.dll and the ftd2xx64.dll.



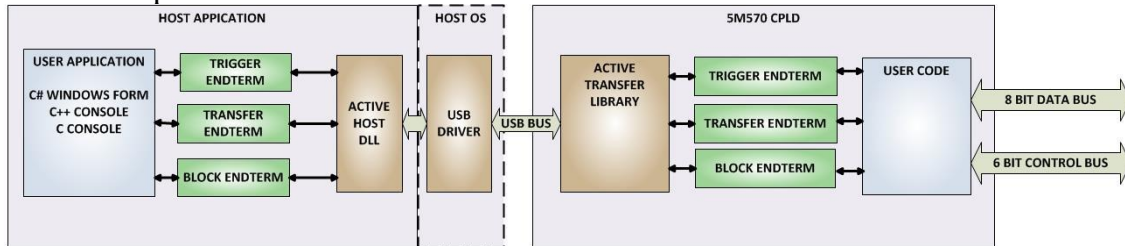
Save the DLL's in the bin\x64\Release folder of the user project under the Microsoft C# Express project. See the Active Host Application section of the EPT USB-CPLD Development System User Manuals for instructions on how to add the dll to the Microsoft C# Express project.



3 Active Transfer Library

The Active Transfer Library is an HDL library designed to transfer data to and from the UnoMax via High Speed USB. It is a set of pre-compiled HDL files that the user will

add to their project before building it. The description of what the library does and how to use its components are described in this manual.



3.1 EPT Active Transfer System Overview

The Active Transfer System components consist of the following:

- active_transfer_library.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

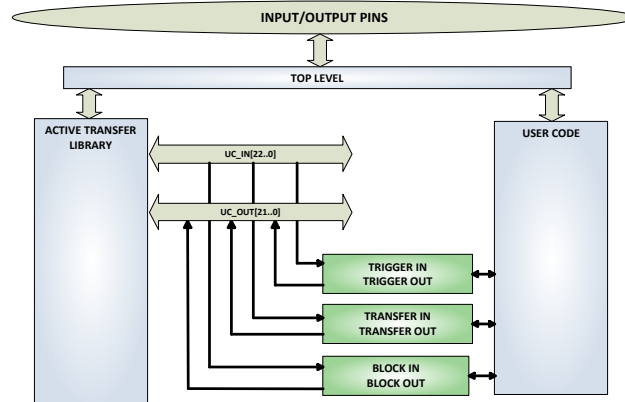
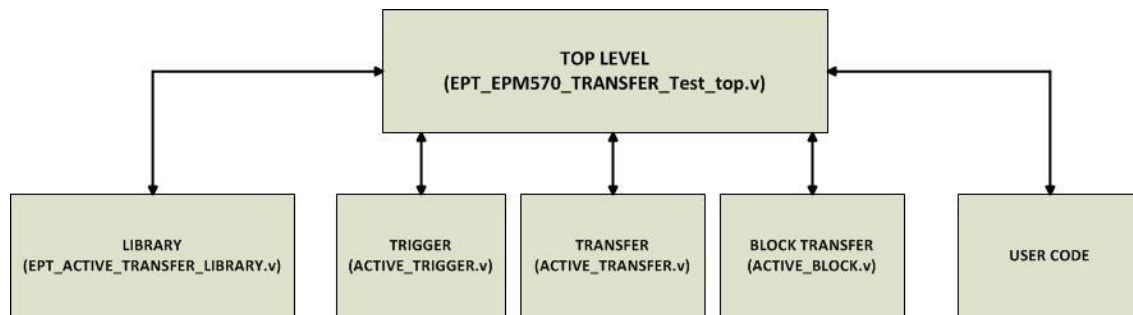


Figure 6 shows how the modules of the EPT Active Transfer Library attach to the overall user project. The EPT Active_Transfer_Library.vqm, Active_Trigger.vqm, Active_Transfer.vqm and Active_Block.vqm modules are instantiated in the top level of the user project. The User_Code.v module is also instantiated 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 CPLD. The user code can send a transfer to the Host, and the Host can send a transfer to the user code. This provides significant control for both data transfers and

signaling from the user code to PC. The Triggers are used to send momentary signals that can turn on (or off) functions in user code or PC. The Active Transfer is used to send a single byte. And the Active Block is used to send a block of data. The Active_Transfer and Active_Block modules have addressing built into them. This means the user can declare up to 8 individual instantiations of Active_Transfer or Active_Block, and send/receive data to each module separately.

3.2 Active Transfer Library

The Active Transfer Library contains the command, control, and data transfer mechanism that allows users to quickly build powerful communication schemes in the CPLD. Coupled with the Active Host application on the PC, this tool 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.





```
1  //#####
2  //#
3  //# Copyright   Earth People Technology Inc. 2012
4  //#
5  //#
6  //# File Name:   EPT_FT232_Transfer_Test_top.v
7  //#
8  //# Revision History:
9  //#      DATE      VERSION    DETAILS
10 //#      07/5/12    A          Created    RJJ
11 //#
12 //#
13 //#
14 //#####
15 `ifndef SIM
16     `include "../src/define.v"
17     `include "../Testbench/tb_define.v"
18 `endif
19
20 `timescale 1ns/1ps
21
22
23
24 //*****
25 // * Module Declaration
26 //*****
27
28 module ept_EPM570_Transfer_Test_top (
29
30
31     input wire [1:0]    aa,
32     input wire [1:0]    bc_in,
33
34     •
35     •
36     •
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88 //-----
89 // Instantiate the EPT Library
90 //-----
91
92     active_transfer_library    EPT_LIBRARY_TOP_INST
93
94     (
95         .aa                (aa),
96         .bc_in              (bc_in),
97         .bc_out              (bc_out),
98         .bd_inout            (bd_inout),
99
100        .UC_IN                (UC_IN),
101        .UC_OUT                (UC_OUT),
102
103        .TEST_SIGNAL_1        (data_byte_ready),
104        .STATE_OUT              (ft_245_state_machine),
105        .TEST_BUS                (register_decode),
106        .ENDPOINT_STATE_OUT      (endpoint_registers_state),
107        .ENDPOINT_TEST_BUS        (endpoint_write_to_host)
108    );
109
110 //-----
111 // Instantiate the EPT Modules
112 //-----
113
114     wire [22*3-1:0]    uc_out_m;
115     eptWireOR # (.N(3)) wireOR (UC_OUT, uc_out_m);
116     active_trigger      ACTIVE_TRIGGER_INST
117     (
118         .uc_clk            (CLK_66),
```

Manual

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 CPLD.

3.2.1 Active Trigger EndTerm

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

```

743 wire [22*3-1:0] uc_out_m;
744 eptWireOR # (.N(3)) wireOR (UC_OUT, uc_out_m);
745 active_trigger ACTIVE_TRIGGER_INST
746 (
747     .uc_clk          (CLK_66),
748     .uc_reset        (RST),
749     .uc_in           (UC_IN),
750     .uc_out          (uc_out_m[ 0*22 +: 22 ]),
751
752     .trigger_to_host (trigger_to_host),
753     .trigger_to_device (trigger_in_byte)
754
755 );
756
  
```

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.

Manual

```

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.



Manual

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

3.2.2 Active Transfer EndTerm

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

```

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

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

Manual

```

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

Manual

transfer_received port is asserted. Upon assertion, the user code should read the byte from the transfer_to_device port into a local register.

```

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

3.2.3 Active Block EndTerm

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

Manual

```

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

```

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

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



Manual

```
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
```


Manual

To receive a buffer from the Host, the user code should monitor the `transfer_received` port for assertion. When the bit is asserted, the next rising edge of `transfer_ready` will indicate that the byte at `transfer_to_device` is ready for the user code to read.

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

3.3 Timing Diagram for Active Transfer EndTerms

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

3.3.1 Active Trigger EndTerm Timing

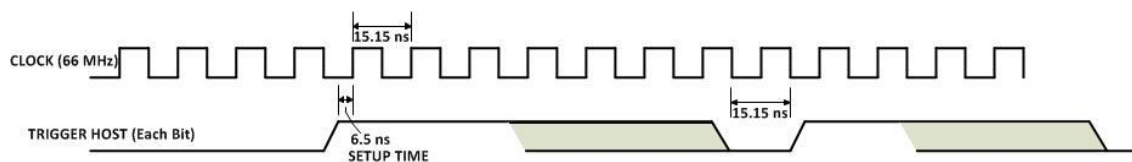


Figure xx Active Trigger to Host Timing

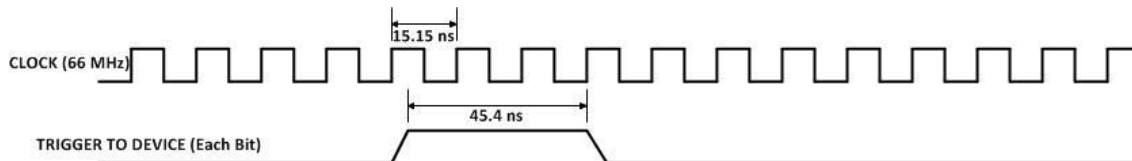


Figure xx Active Trigger to Device Timing

3.3.2 Active Transfer EndTerm Timing

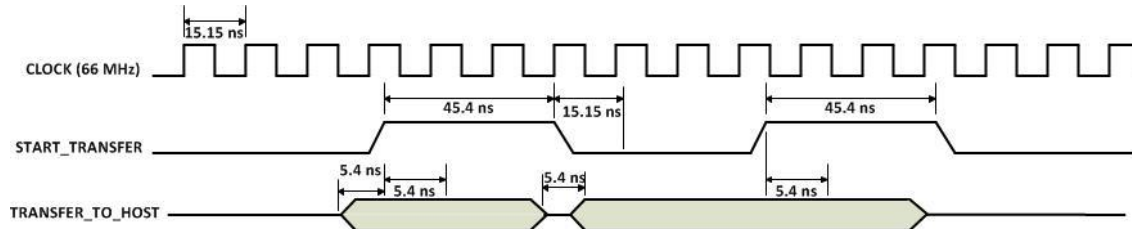


Figure xx Active Transfer To Host Timing

Manual

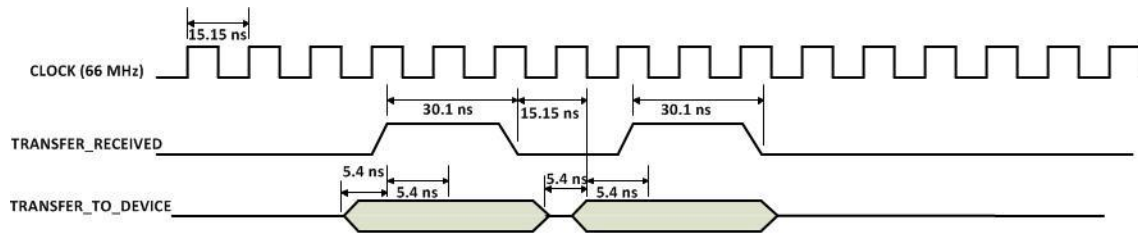


Figure xx Active Transfer To Device Timing

3.3.3 Active Block EndTerm Timing

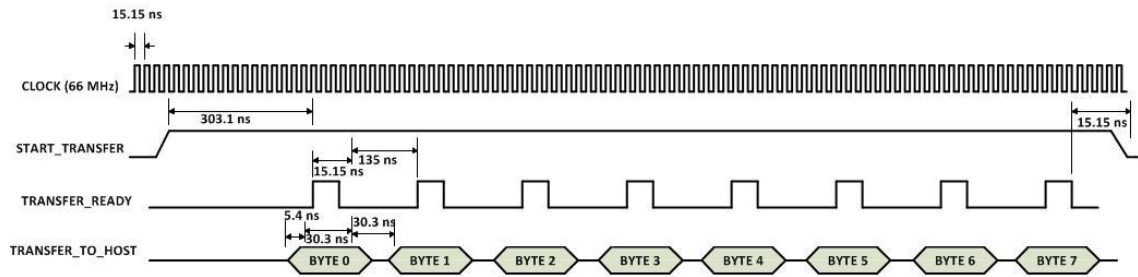


Figure xx Active Block To Host Timing

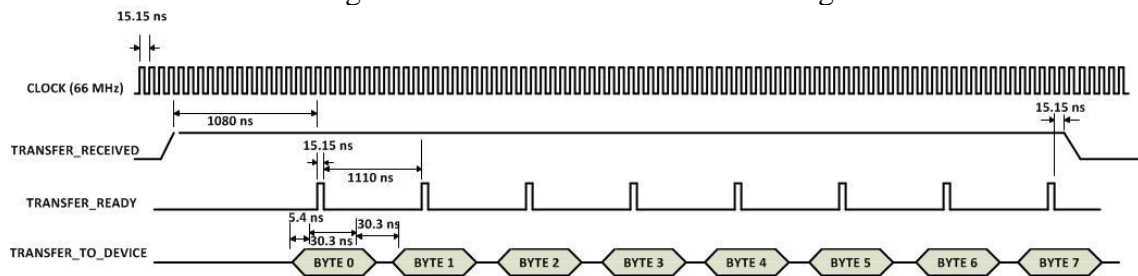
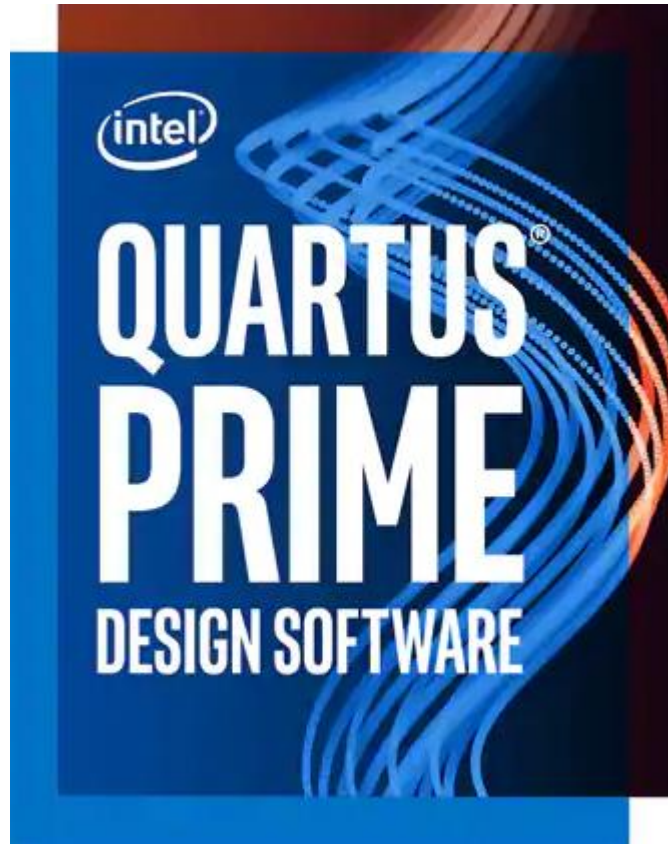


Figure xx Active Block To Device Timing

4 Compiling, Synthesizing, and Programming CPLD

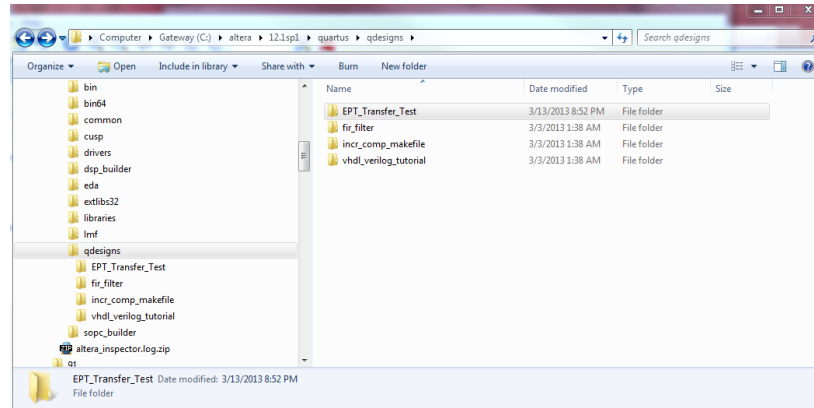


The CPLD on the UnoMax can be programmed with the Active Transfer Library and custom HDL code created by the user. Programming the CPLD requires the use of the Quartus Prime Lite software and a standard USB cable. There are no extra parts to buy, just plug in the USB cable. Once the user HDL code is written according to the syntax rules of the language (Verilog and VHDL) it can be compiled and synthesized using the Quartus Prime Lite 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 CPLD.

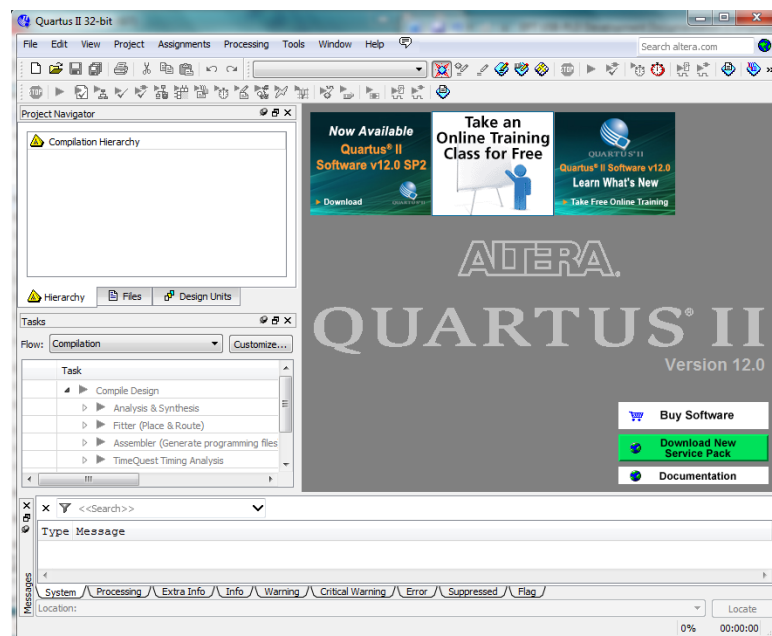
4.1 Setting up the Project and Compiling

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

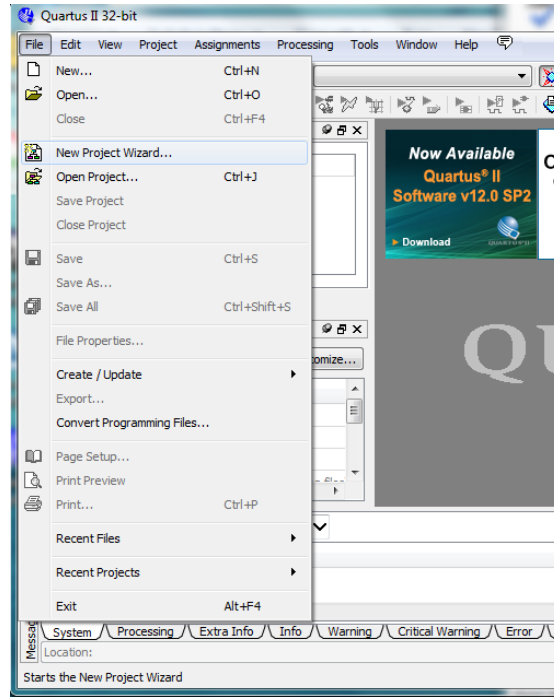
Manual



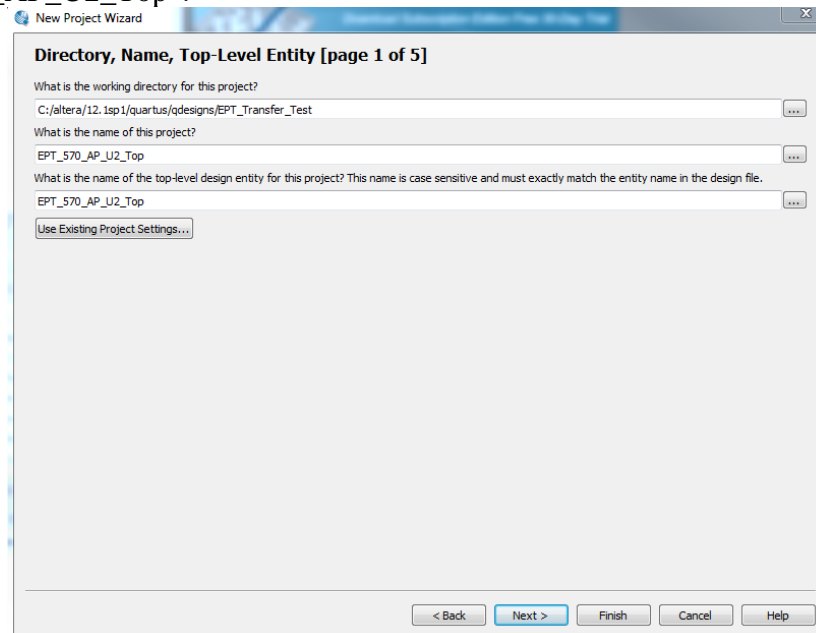
Open Quartus Prime Lite by clicking on the icon .



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



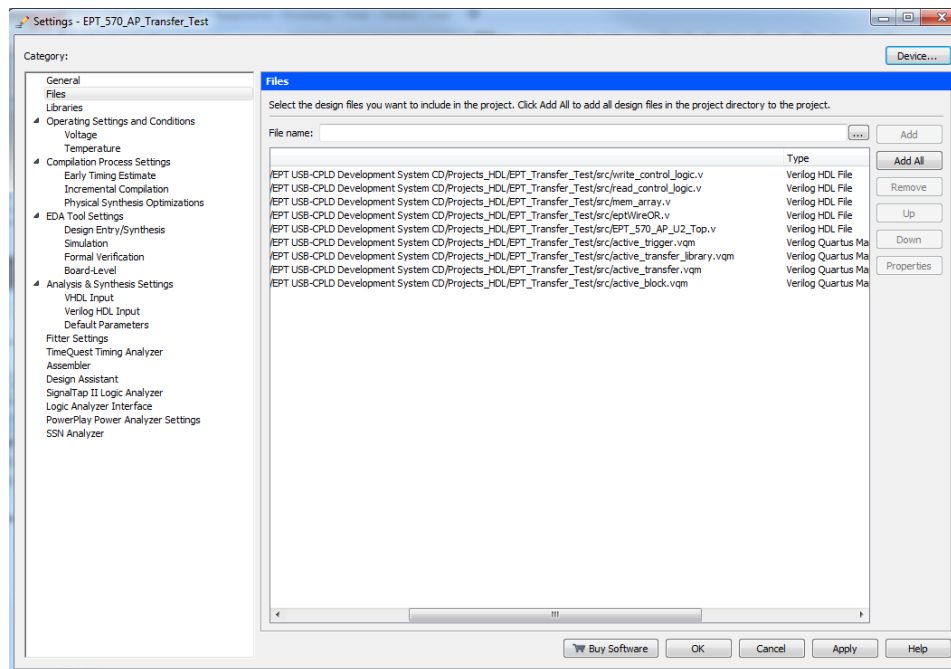
At the Top-Level Entity page, browse to the C:/intelFPGA_lite /xxx/quartus/qdesigns directory to store your project. Type in a name for your project “EPT_570_AP_U2_Top”.



Manual

Select Next. At the Add Files window: Browse to the \Projects_HDL\EPT_Transfer_Test \src folder of the EPT USB-CPLD Development System CD. Copy the files from the \src directory.

- Active_block.vqm
- Active_transfer.vqm
- Active_trigger.vqm
- Active_transfer_library.vqm
- eptWireOr.v
- mem_array.v
- read_control_logic.v
- write_control_logic.v
- EPT_570_AP_U2_Top.v



Select Next, at the Device Family group, select MAX V for Family. In the Available Devices group, browse down to 5M570ZT100C5 for Name.

New Project Wizard

Family & Device Settings [page 3 of 5]

Select the family and device you want to target for compilation.

Device family

Family: **MAX II**

Devices: **All**

Target device

☐ Auto device selected by the Fitter

☒ Specific device selected in 'Available devices' list

☐ Other: n/a


Show in 'Available devices' list

Package: **Any**

Pin count: **Any**


Speed grade: **Any**

Name filter:

☒ Show advanced devices ☐ HardCopy compatible only 

Available devices:

Name	Core Voltage	LEs	UFM blocks
EPMS570T100C3	3.3V	570	1
EPMS570T100C4	3.3V	570	1
EPMS570T100C5	3.3V	570	1
EPMS570T100I5	3.3V	570	1
EPMS570T144A5	3.3V	570	1
EPMS570T144C3	3.3V	570	1
EPMS570T144C4	3.3V	570	1
EPMS570T144C5	3.3V	570	1

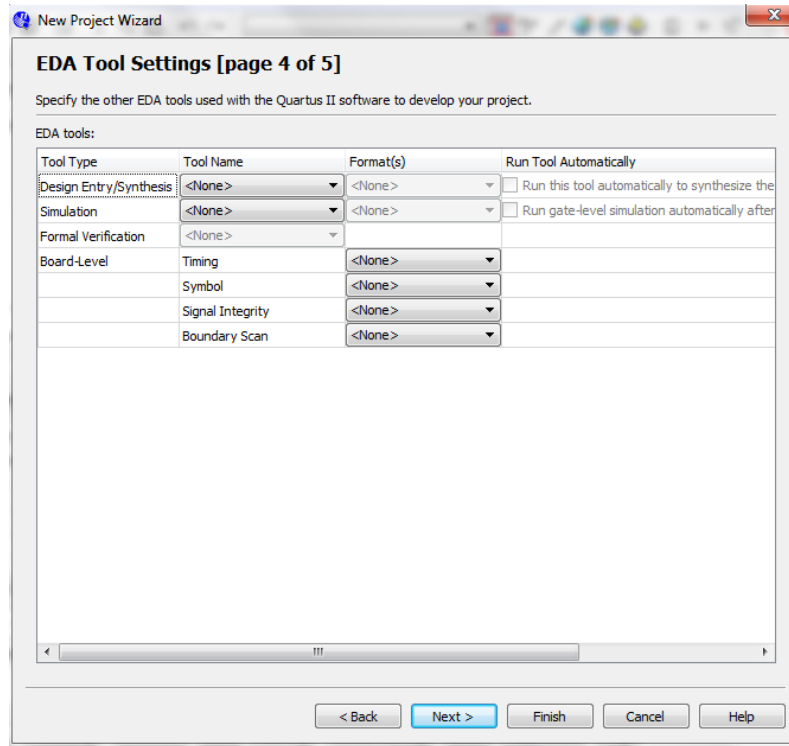
Companion device 

HardCopy:

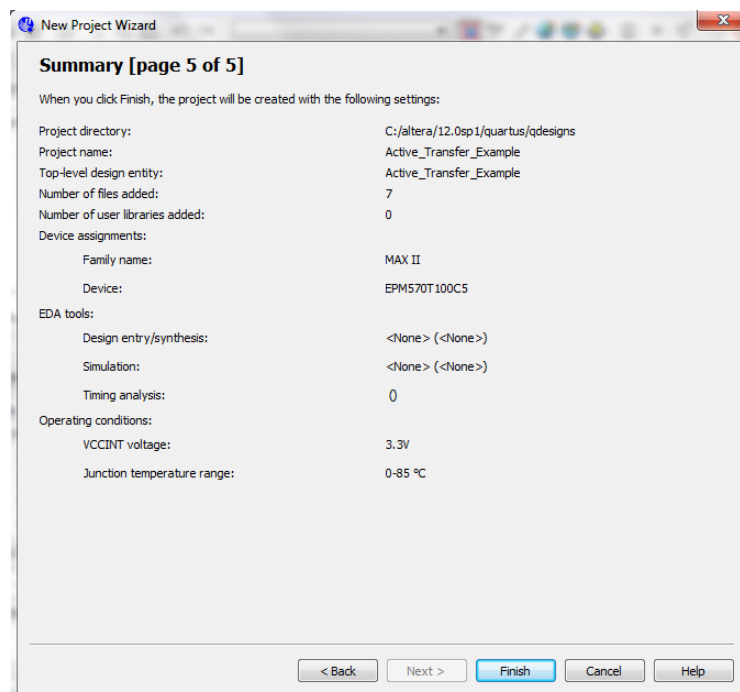
☐ Limit DSP & RAM to HardCopy device resources

< Back **Next >** Finish Cancel Help

Select Next, leave defaults for the EDA Tool Settings.



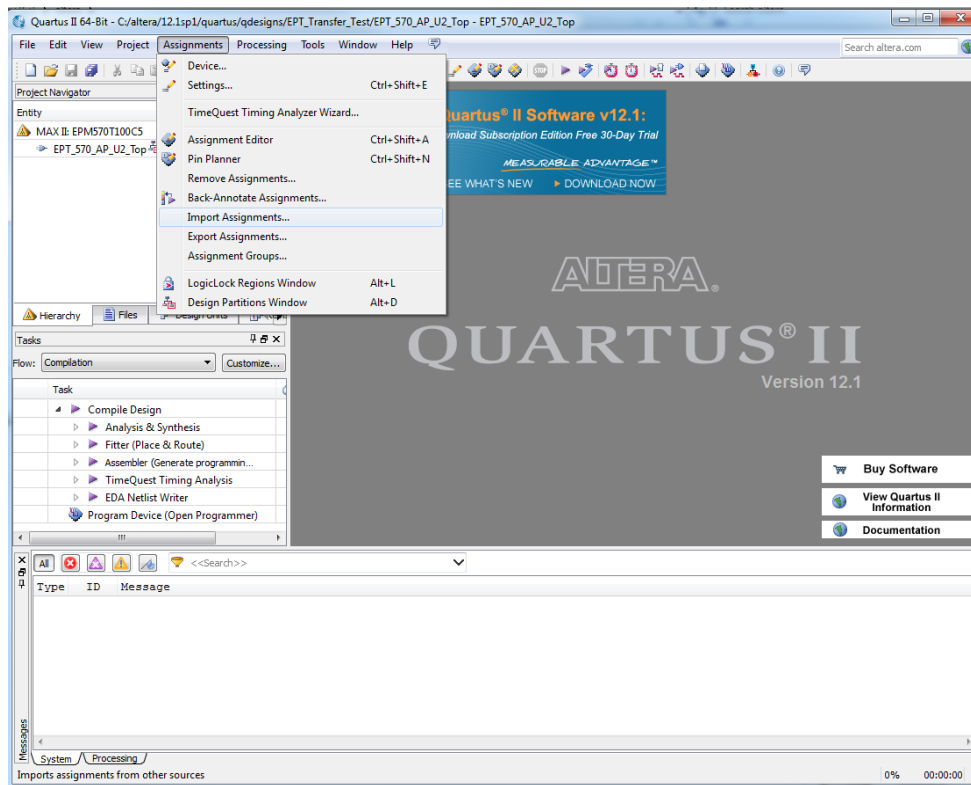
Select Next, then select Finish. You are done with the project level selections.



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

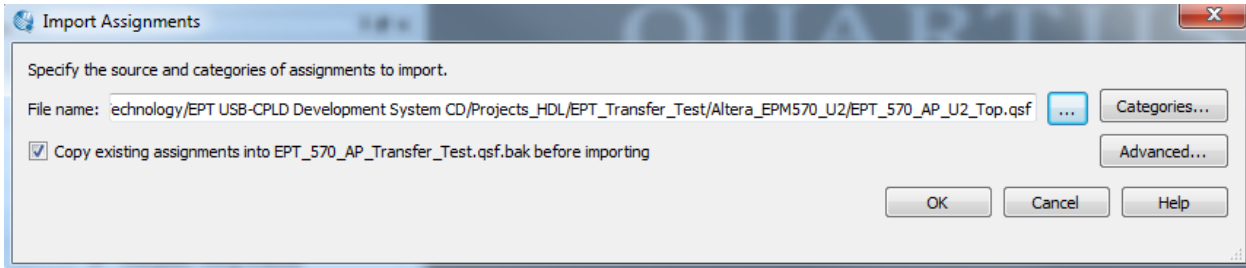
4.1.1 Selecting Pins and Synthesizing

With the project created, we need to assign pins to the project. The signals defined in the top level file (in this case: EPT_570_AP_U2_Top.v) will connect directly to pins on the CPLD. The Pin Planner Tool from Quartus Prime Lite 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.

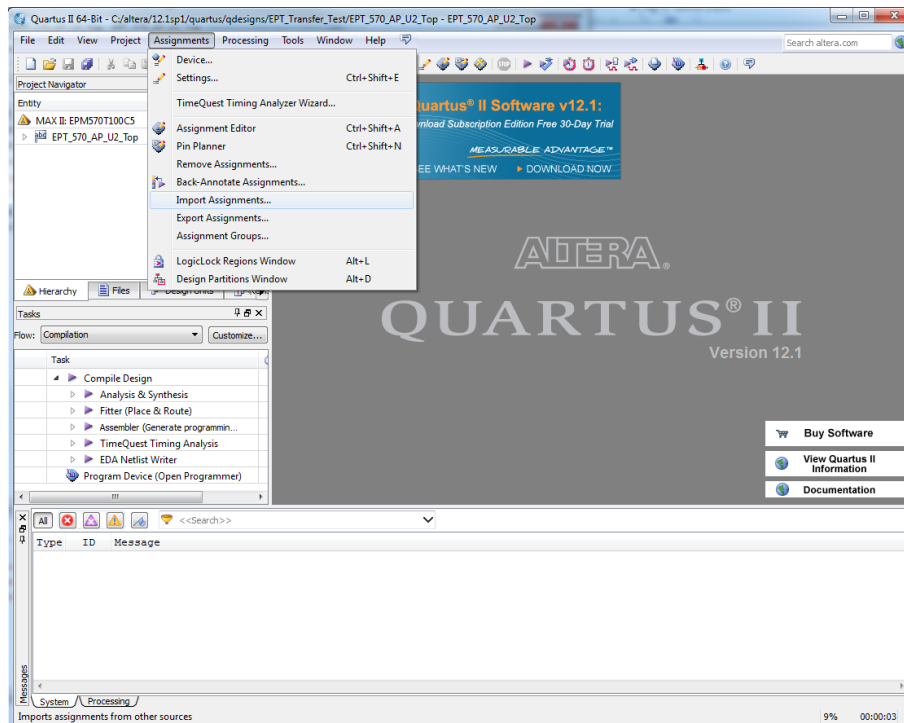


At the Import Assignment dialog box, Browse to the
 \Projects_HDL\EPT_Transfer_Test \ EPT -5M57-AP-U2_TOP folder of the EPT USB-CPLD Development System CD. Select the “EPT_570_AP_U2_Top.qsf” file.

Manual

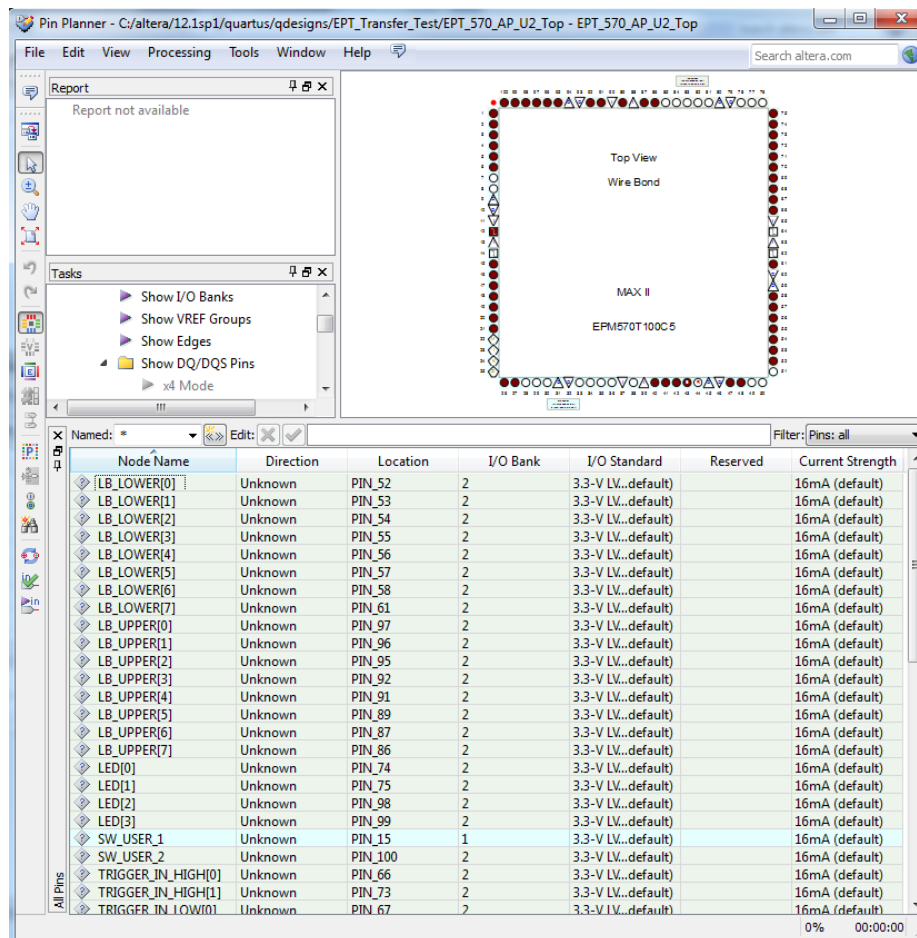


Click Ok. Under Assignments, Select Pin Planner. Verify the pins have been imported correctly.



Manual

The pin locations should not need to be changed for EPT USB CPLD 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

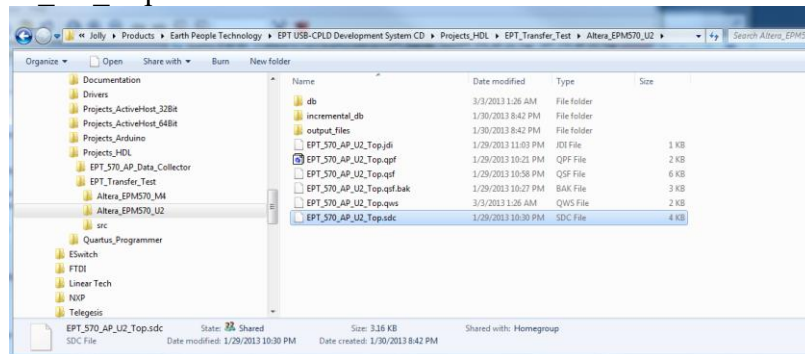


UnoMax CPLD Development System User

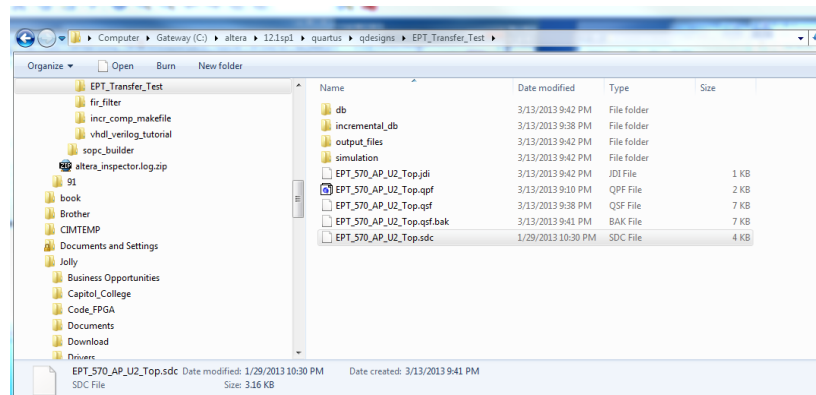
Manual

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_Test \ EPT_5M57_AP_U2_TOP folder of the EPT USB-CPLD Development System DVD. Select the “EPT_570_AP_U2_Top.sdc” file.

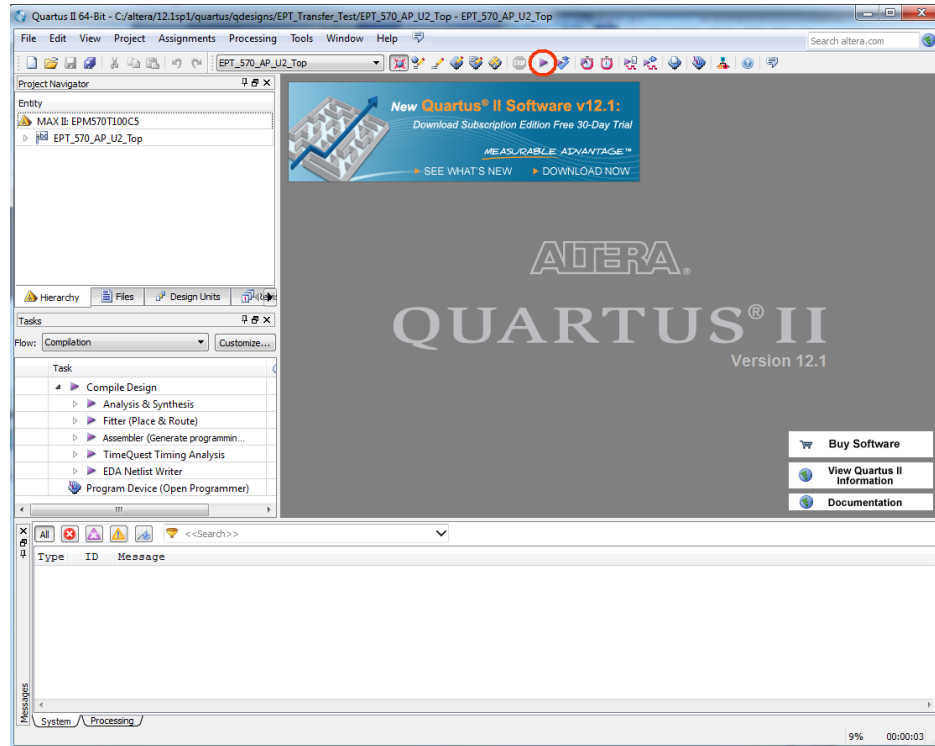


Copy the file and browse to C:\intelFPGA_lite
\xxx\quartus\qdesigns\EPT_Transfer_Test directory. Paste the file.



Select the Start Compilation button.

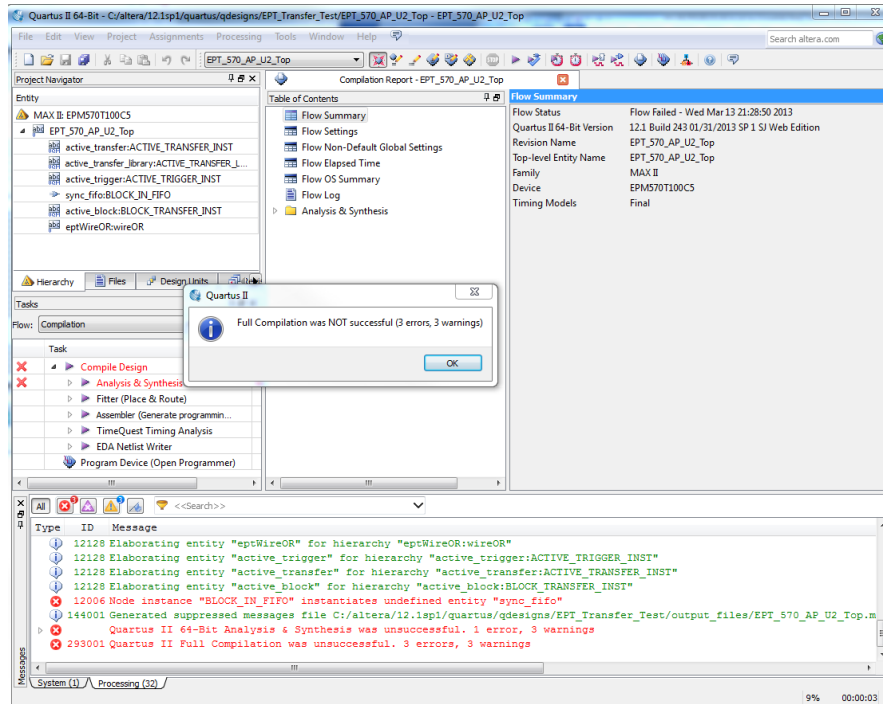
Manual



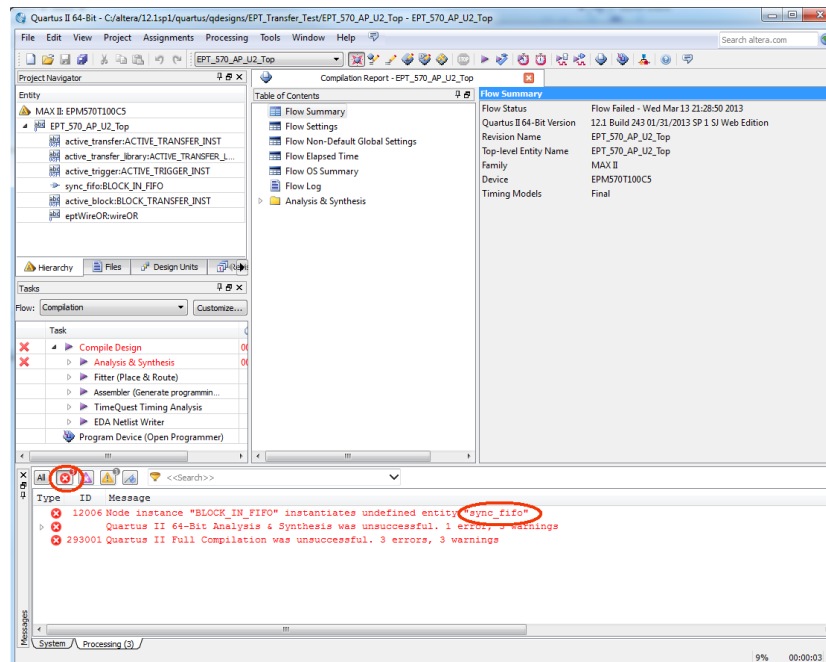
If you forget to include a file or some other error you should expect to see a screen similar to this:



Manual

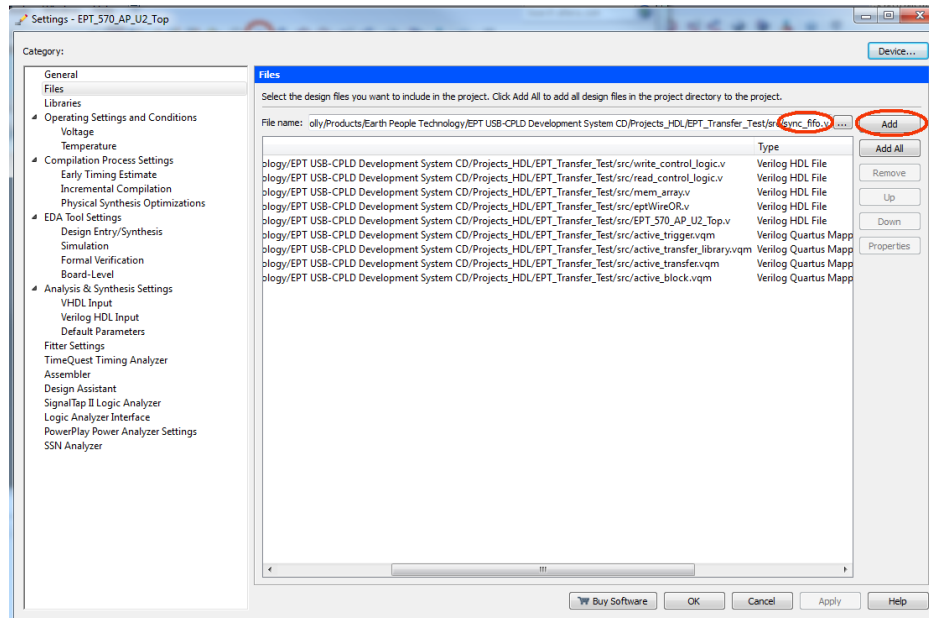


Click Ok, then select the “Error” tab to see the error.

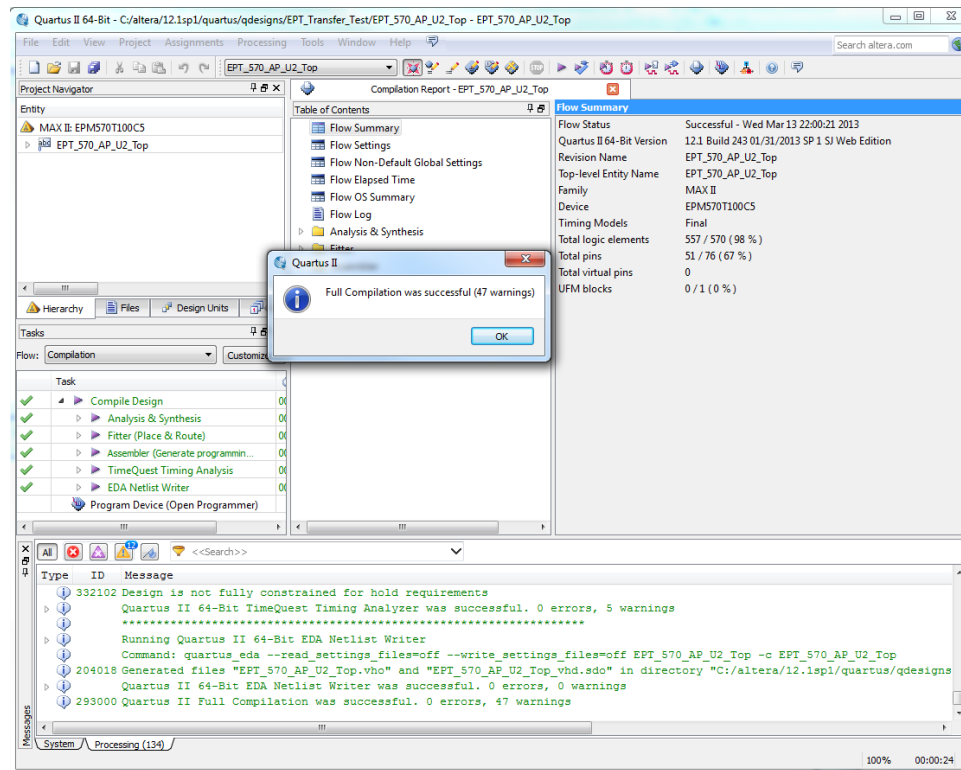


Manual

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



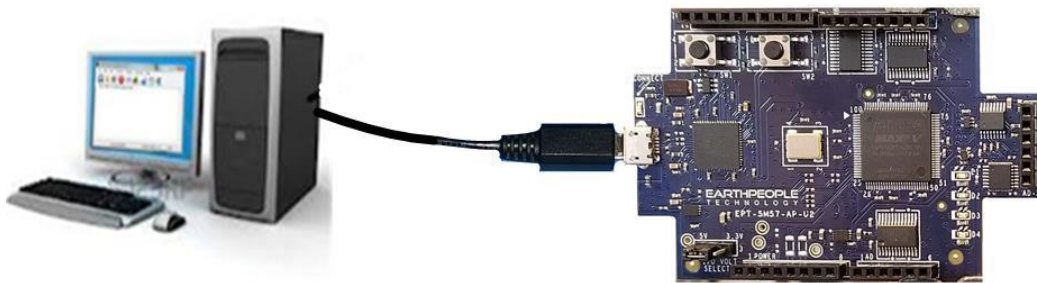
Click Ok then re-run the Compile process. After successful completion, the screen should look like the following:



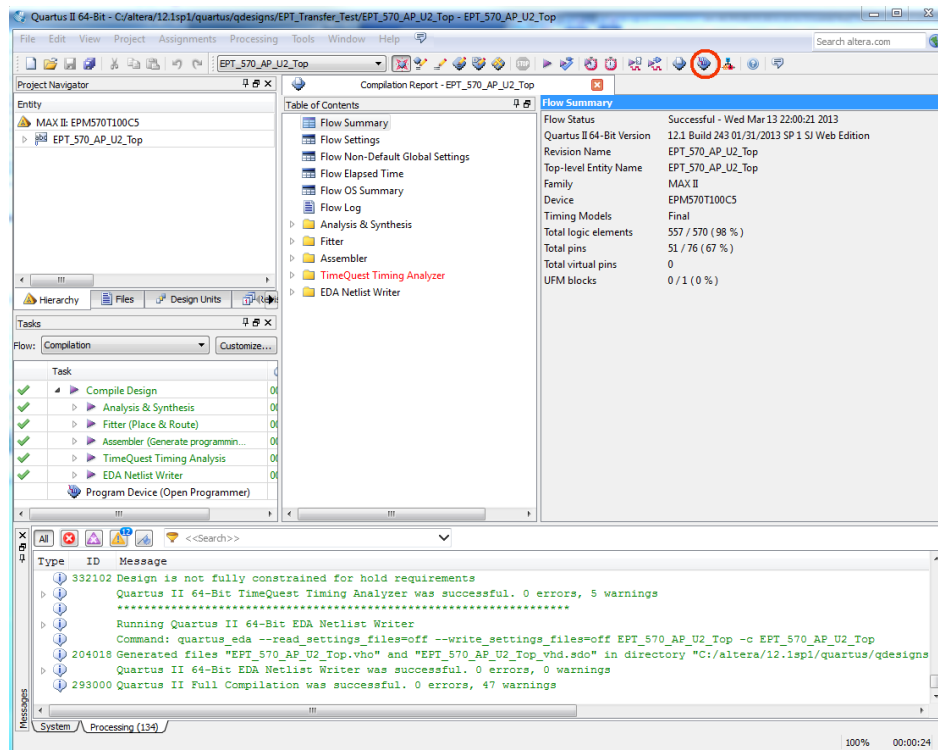
At this point the project has been successfully compiled, synthesized and a programming file has been produced. See the next section on how to program the CPLD.

4.1.2 Programming the CPLD

Programming the CPLD is quick and easy. All that is required is a standard USB cable with a Micro Type B connector on one end and the EPT_Blaster Driver DLL. Connect the UnoMax to the PC, open up Quartus Prime Lite, open the programmer tool, and click the Start button. To program the CPLD, follow the steps to install the USB Driver and the JTAG Driver Insert for Quartus Prime Lite.

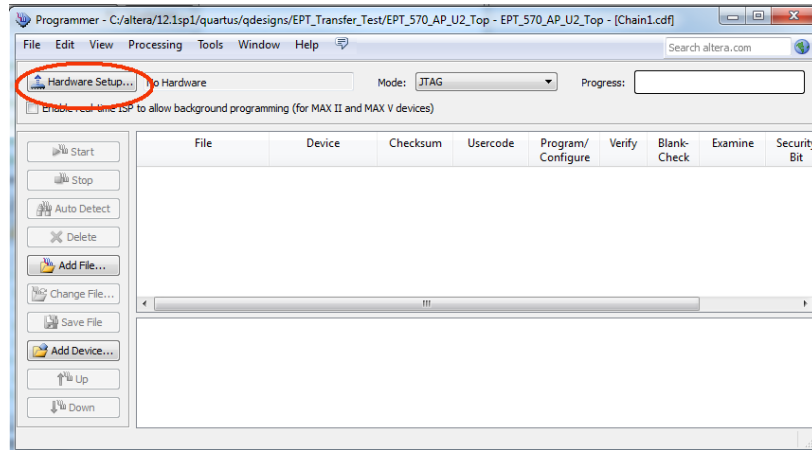


If the project created in the previous sections is not open, open it. Click on the Programmer button.

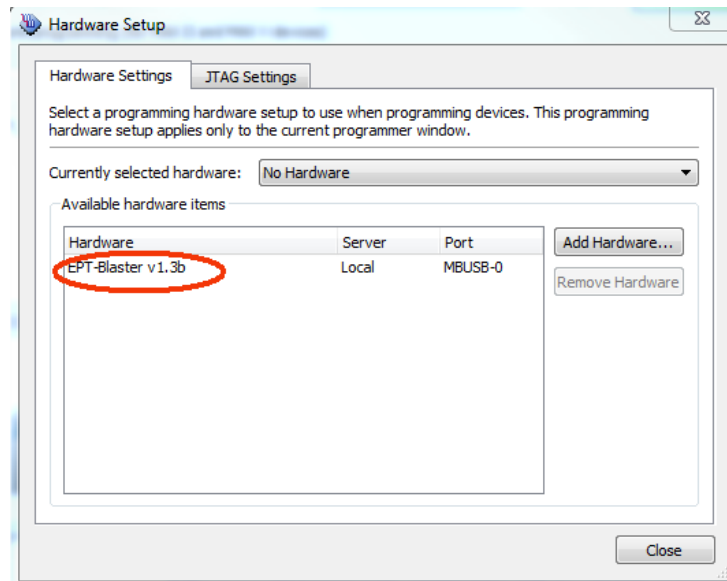


The Programmer Window will open up with the programming file selected. Click on the Hardware Setup button in the upper left corner.

Manual

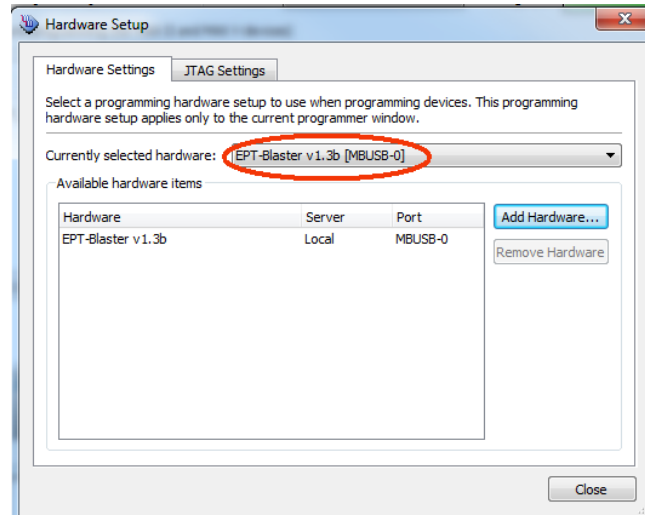


The Hardware Setup Window will open. In the “Available hardware items”, double click on “EPT-Blaster v1.6b”.

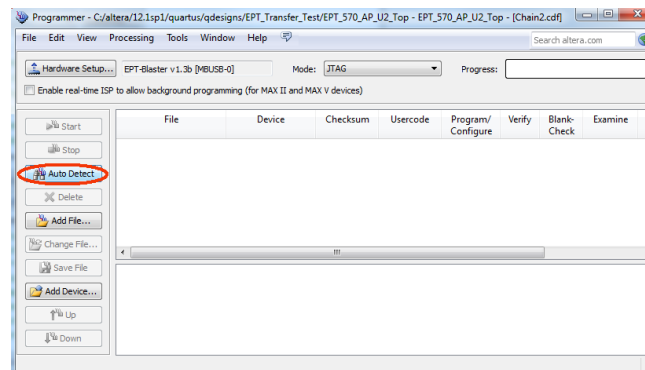


If you successfully double clicked, the “Currently selected hardware:” dropdown box will show the “EPT-Blaster v1.6b”.

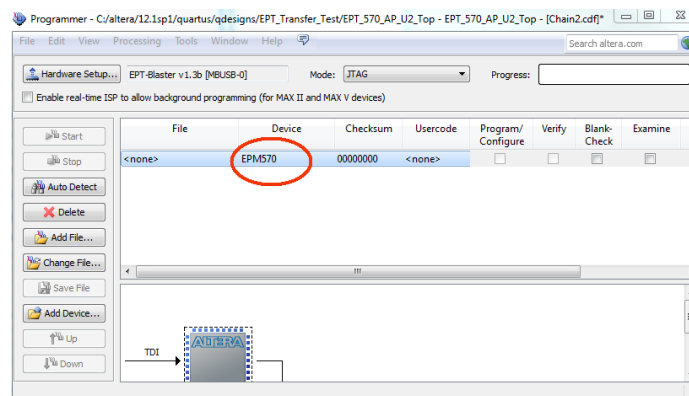
Manual



Click on the Auto-Detect button. This will verify that the EPT-Blaster driver can connect with the 5M570 device.

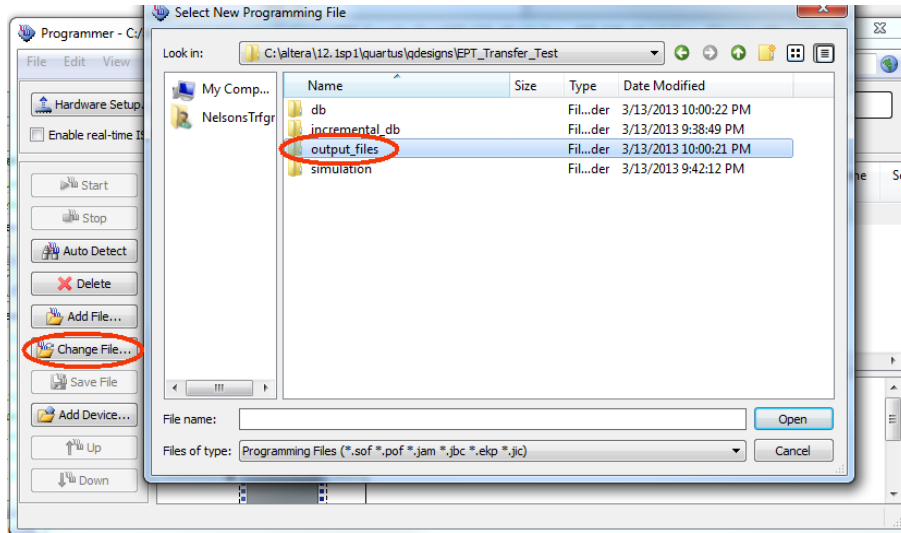


Select the 5M570 under “Device”.

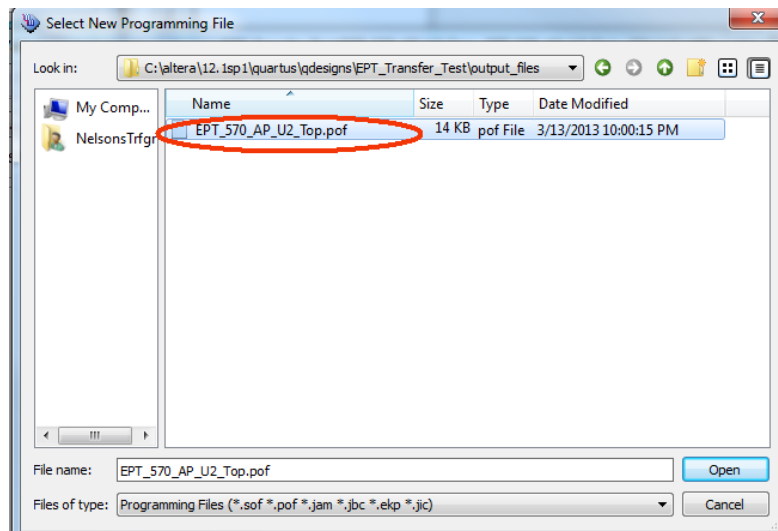


Manual

Click on the “Change File” button and browse to the output_files folder.



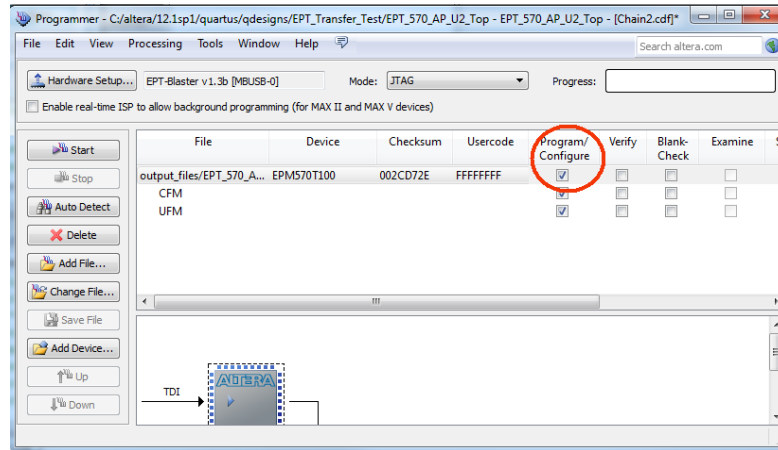
Click on the EPT_5M57_AP_U2_Top.pof file to select it.



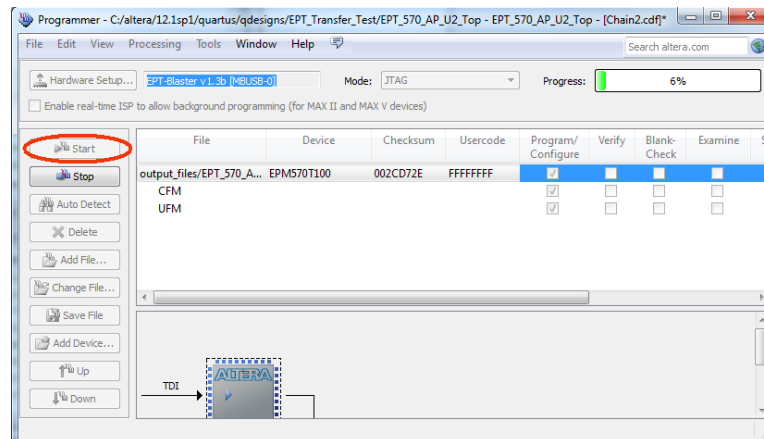
Click the Open button in the lower right corner.

Next, select the checkbox under the “Program/Configure” of the Programmer Tool. The checkboxes for the CFM and UFM will be selected automatically.

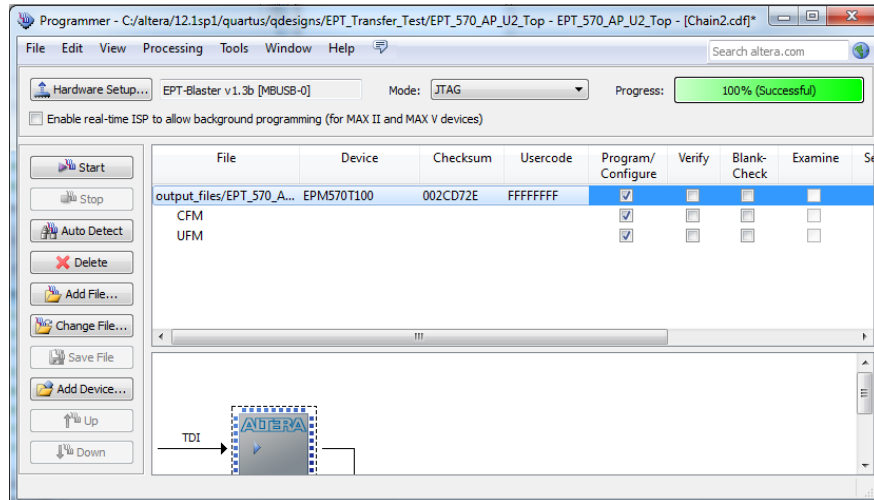
Manual



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



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



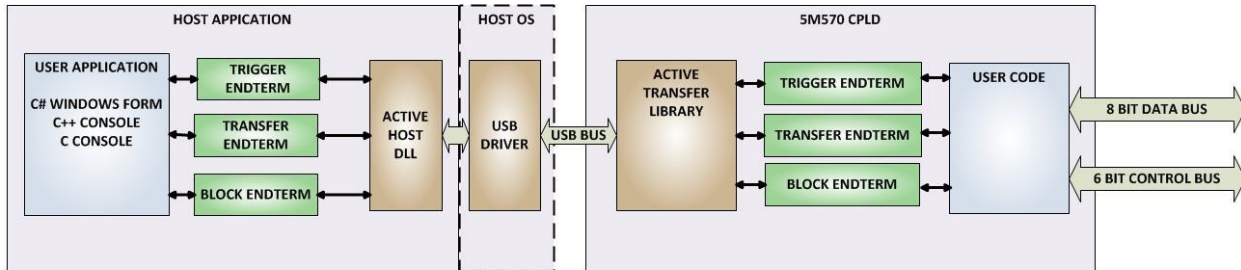
At this point, the UnoMax is programmed and ready for use. To test that the CPLD is properly programmed, bring up the Active Host Test Tool. Click on one of the LED's and verify that the LED selected lights up. Press one of the switches on the board and ensure that the switch is captured on the Active Host Test Tool. Now you are ready to write some code to transfer data between CPLD and PC.

5 Active Host Application

The Active Host SDK is provided as a dll which easily interfaces to application software written in C#, C++ or C. It runs on the PC and provides transparent connection from PC application code through the USB driver to the user CPLD 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 CPLD. The Trigger Endterms are used as "switches". The user code can set a Trigger bit in the CPLD and cause an event to occur. The Transfer Endterm sends one byte to the CPLD. The Block Endterm sends a block of bytes. By using one of the Active Host Endterms, the user can create a dynamic, bi-directional, and configurable data transfer design.



5.1 Trigger EndTerm

The Trigger EndTerm is a software component that provides a direct path from the users application to the commensurate Trigger EndTerm in the CPLD. 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.

5.2 Transfer(Byte) EndTerm

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

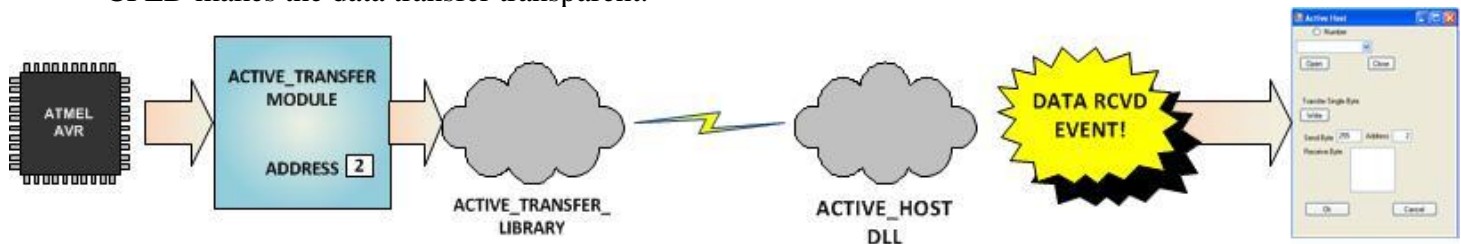
5.3 Block EndTerm

The Block EndTerm is a software component that provides a direct path from the users application to the commensurate Block EndTerm in the CPLD. The Block EndTerm is used to transfer a complete block to the CPLD. Block size is limited to 1 to 256 bytes. Eight separate Block EndTerm modules can be instantiated in the CPLD. 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 CPLD. Receiving a block is just as easy, a callback function is registered at initialization. When the CPLD transmits a block using its EndTerm, the callback function is called in the

user application. The incoming Transfers are stored in a circular buffer in memory. This allows the user code to fetch the transfers with out losing bytes.

5.4 Active Host DLL

The Active_Host DLL is designed to transfer data from the CPLD 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 CPLD. This method of automatically moving data from the user code Endterm in the CPLD makes the data transfer transparent.



The data seamlessly appears in Host PC memory from the UnoMax. 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 CPLD by use of functions contained in the Active Host dll. The functions to access the CPLD 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()

Manual

- EPT_AH_CloseDeviceByIndex()
- EPT_AH_CloseDeviceByName()
- EPT_AH_SendTrigger ()
- EPT_AH_SendByte ()
- EPT_AH_SendBlock ()
- EPT_AH_SendTransferControlByte()
- EPT_AH_RegisterReadCallback ()
- EPT_AH_GetLastError()
- EPT_AH_PerformSelfTest()
- EPT_AH_LEDBlinky()
- EPT_AH_SetDebugMode()
- EPT_AH_RegisterReadCallbackForChannel()
- EPT_AH_FlushDeviceChannelBuffer()
- EPT_AH_GetDeviceChannelFreeBufferBytes()
- EPT_AH_GetDeviceChannelPendingBufferBytes()
- EPT_AH_SetChannelConnectionFlag()
- EPT_AH_GetChannelConnectionFlag()

5.4.1 Active Host Open Device

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

```
// Main object loader
private void EPT_570_AP_Active_Transfer_Load (object sender, System.EventArgs e)
{
    // Call the List Devices function
    ListDevices();
}
```

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

Manual

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

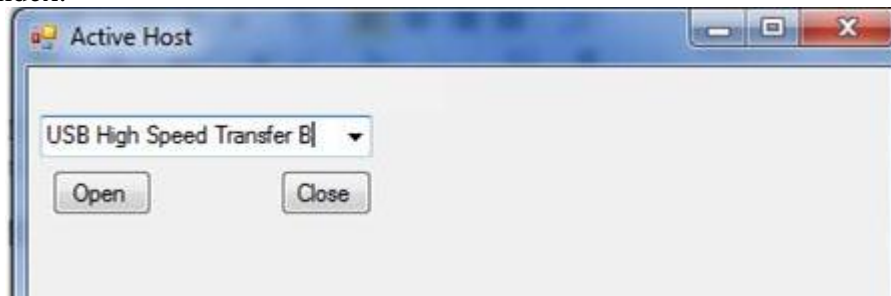
    // Open the DLL
    result = EPT_AH_Open(null, null, null);
    if (result != 0)
    {
        MessageBox.Show("Could not attach to the ActiveHost library");
        return 0;
    }

    // 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

Manual

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

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

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

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

5.4.2 Active Host Read Callback Function

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

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

    // Select current device
    EPT_AH_SelectActiveDeviceByIndex(device_id);

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

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

5.4.3 Active Host Triggers

The user application can send a trigger to the CPLD 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 CPLD.

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

To detect a trigger from the CPLD, 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

Manual

```

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

    // Select current device
    EPT_AH_SelectActiveDeviceByIndex(device_id);

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

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

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

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

```

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

```

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

```

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

5.4.4 Active Host Byte Transfers

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

```

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

```

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

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

Manual

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));
}
```

Manual

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

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

```
public void TransferOutReceive()
{
    string WriteRcvChar = "";
    WriteRcvChar = String.Format("{0}", (int)EPTReceiveData.Payload);
    tbDataBytes.AppendText(WriteRcvChar + ' ');
    tbAddress.Text = String.Format("{0:x2}", (uint)System.Convert.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 be copied and pasted into a user's project.

5.4.5 Active Host Block Transfers

The Active Host Block Transfer is designed to transfer blocks of data between Host and CPLD 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

Manual

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 CPLD.

```
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 CPLD 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.

Manual

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));
}
```

Manual

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

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

```
public void 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 be copied and pasted into a user's project.

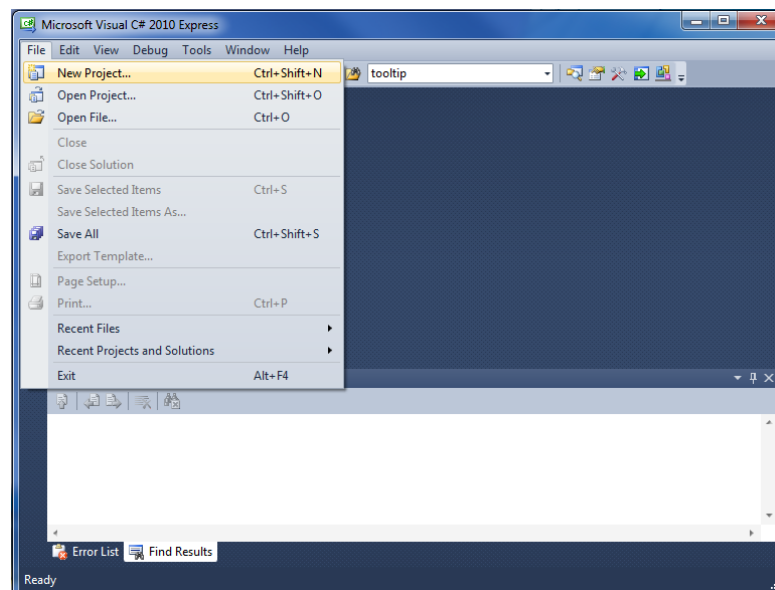
6 Assembling, Building, and Executing a .NET Project on the PC

The Active Host Application DLL is used to build a custom standalone executable on the PC that can perform Triggers and Transfer data to/from the UnoMax. A standalone project can be range from a simple program to display and send data from the user to/from the CPLD. Or it can more complex to include receiving data, processing it, and start or end a process on the CPLD. 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, such as events and controls, will be explained as the explanation progresses. Any language can be used with the Active Host Application DLL.

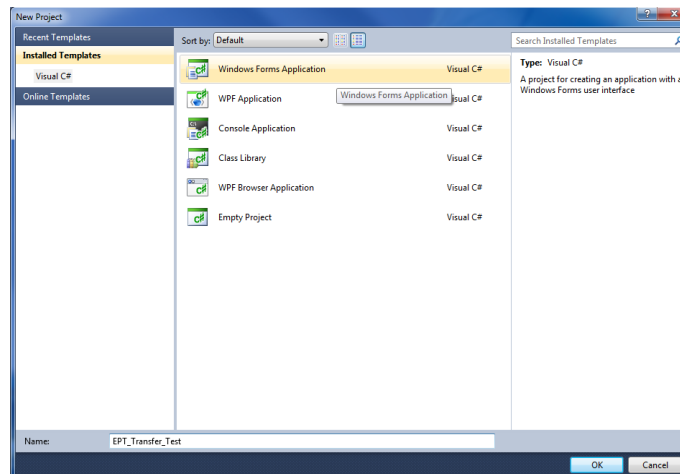
6.1 Creating a Project

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

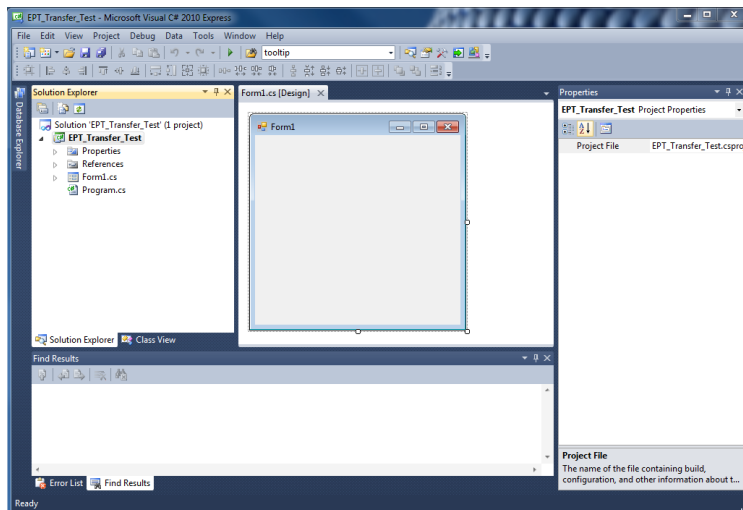


At the New Project window, select the Windows Forms Application. Then, at the Name: box, type in EPT_Transfer_Test

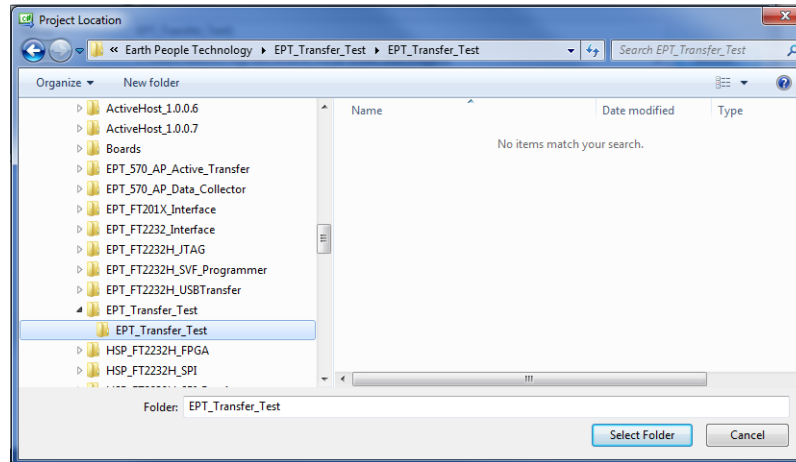
Manual



The project creation is complete.

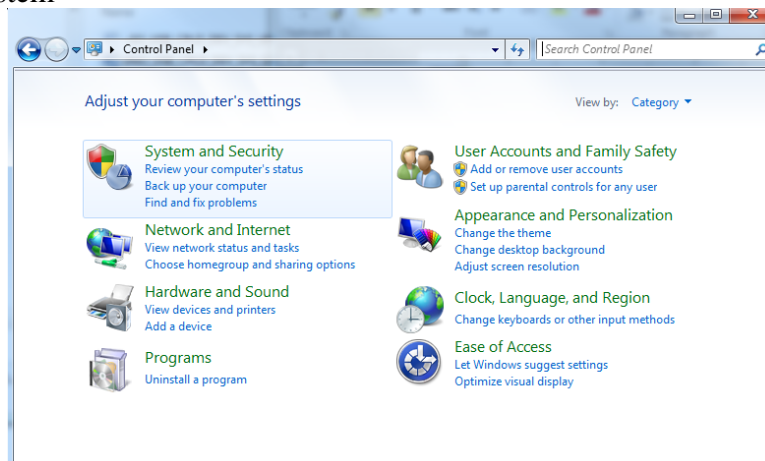


Save the project, go to File->Save as, browse to a folder to create EPT_Transfer_Test folder. The default location is c:\Users\<Users Name>\documents\visual studio 2010\Projects.



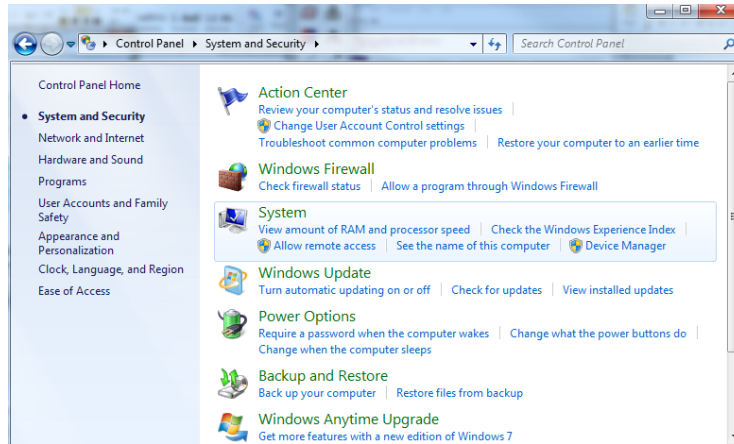
6.1.1 Setting up the C# Express Environment x64 bit

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

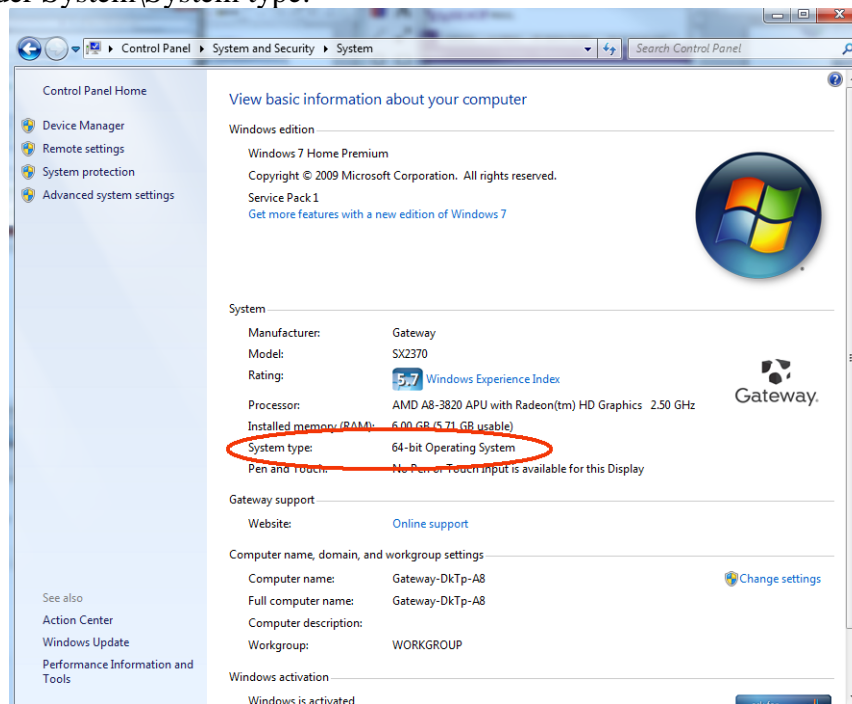


Click on System.

Manual

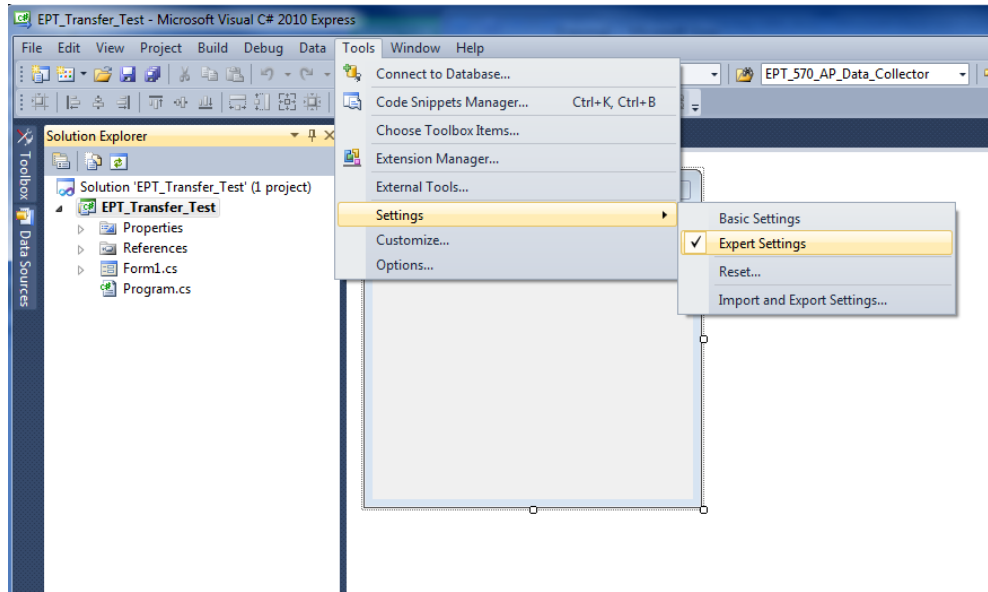


Check under System\System type:

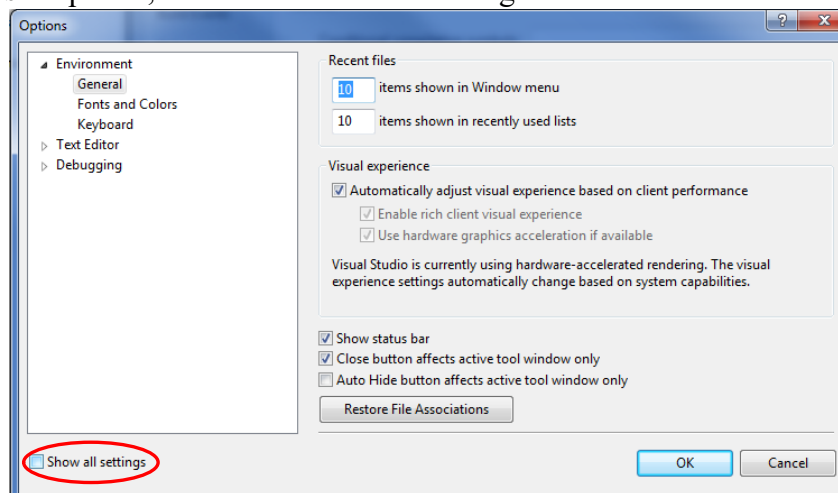


First, we need tell C# Express to produce 64 bit code if we are running on a x64 platform. Go to Tools->Settings and select Expert Settings

Manual

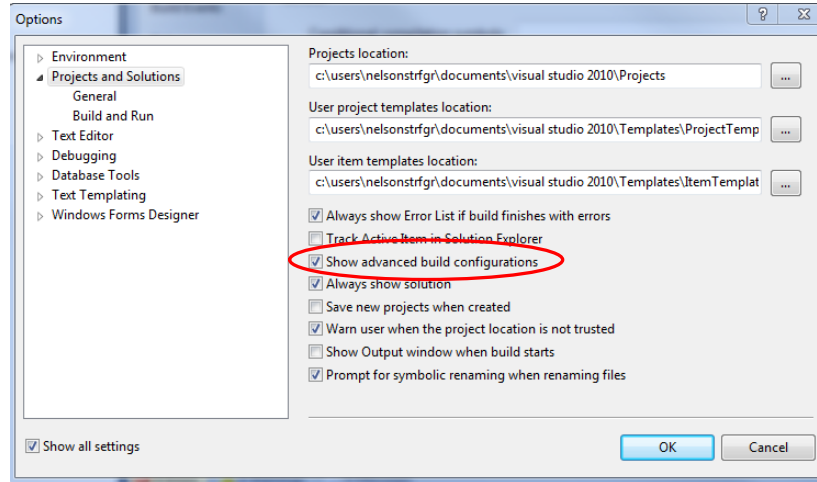


Go to Tools->Options, locate the “Show all settings” check box. Check the box.

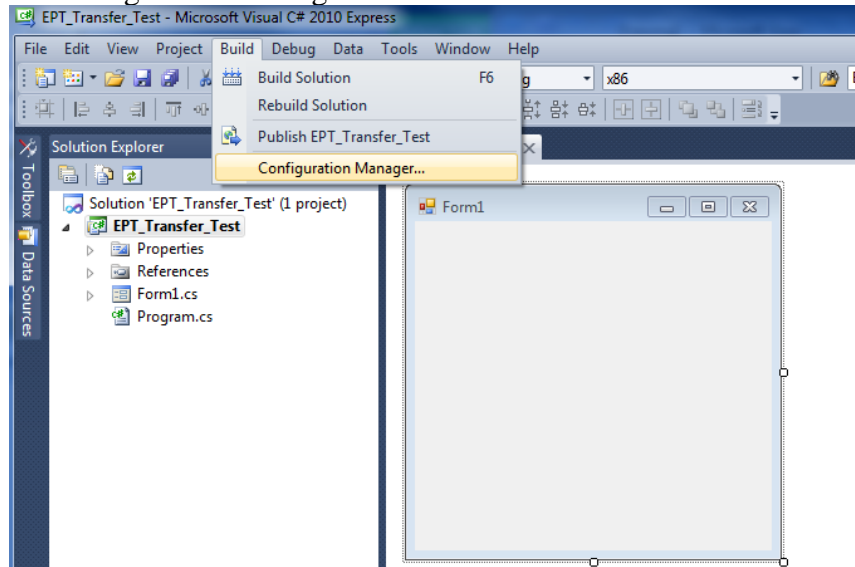


In the window on the left, go to “Projects and Solutions”. Locate the “Show advanced build configurations” check box. Check the box.

Manual

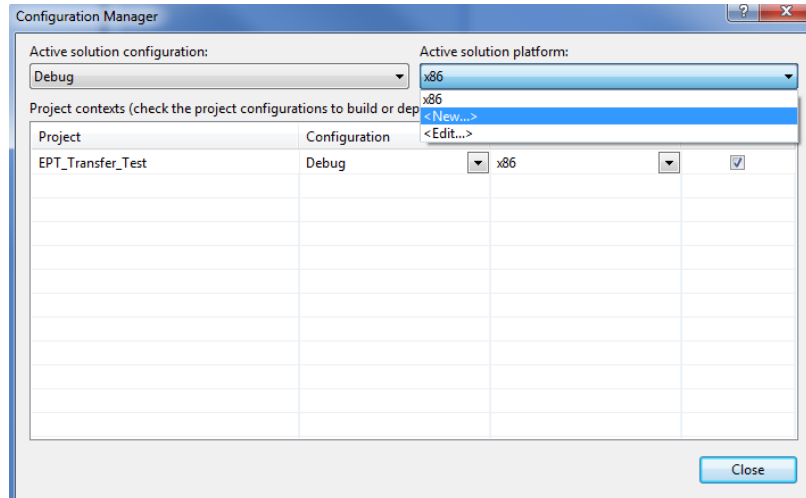


Go to Build->Configuration Manager.

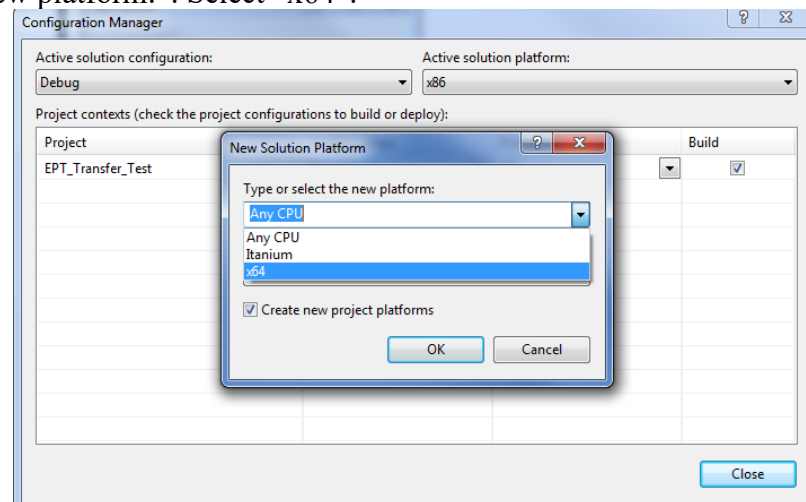


In the Configuration Manager window, locate the “Active solution platform:” label, select “New” from the drop down box.

Manual

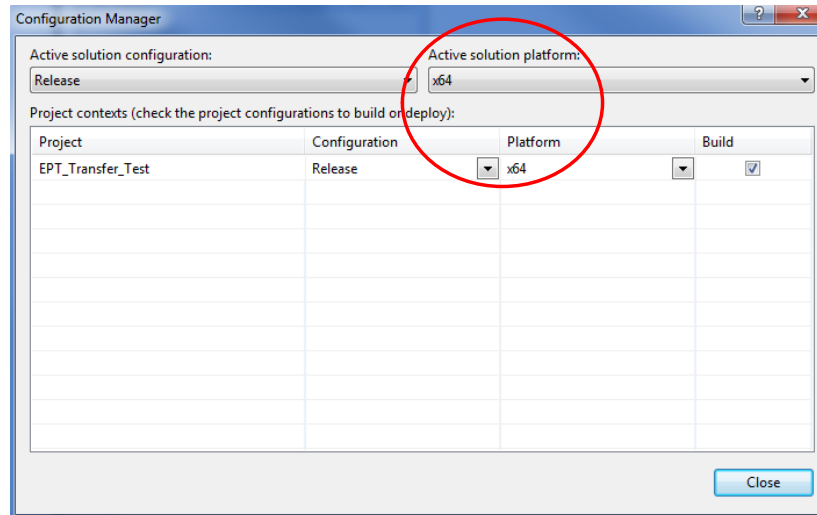


In the New Solution Platform window, click on the drop down box under “Type or select the new platform:”. Select “x64”.

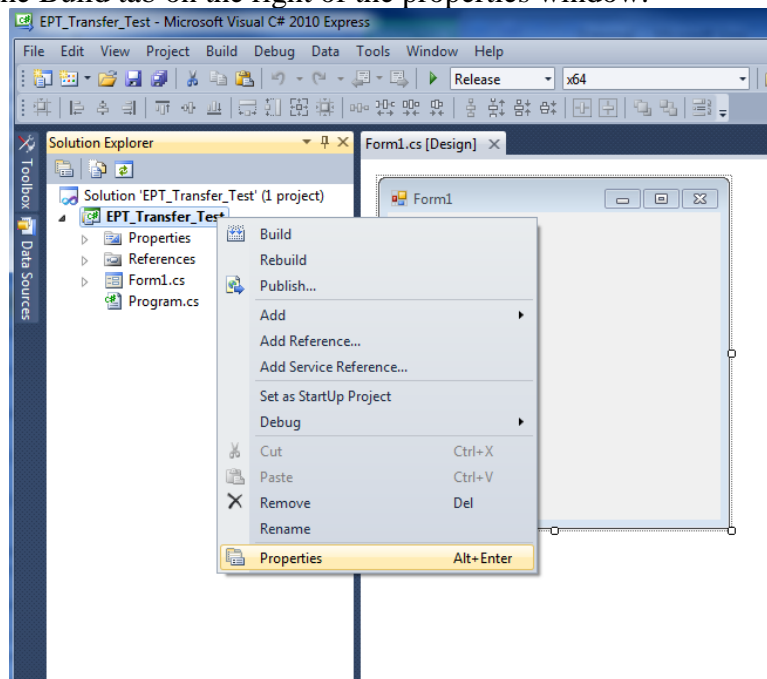


Click the Ok button. Verify that the “Active Solution Platform” and the “Platform” tab are both showing “x64”.

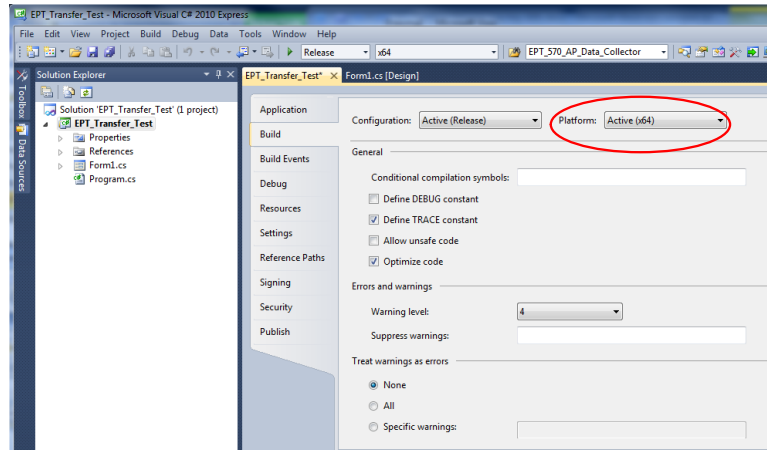
Manual



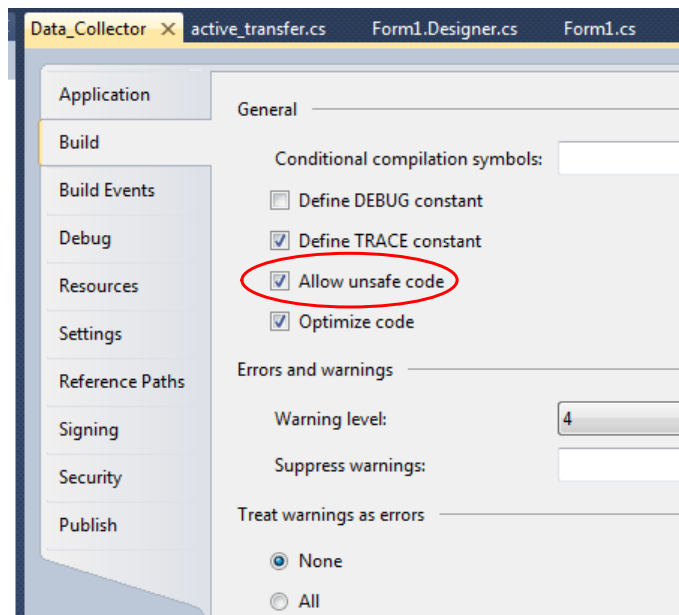
Also, select “Release” under “Active solution configuration”. Click Close. Then, using the Solution Explorer, you can right click on the project, select Properties and click on the Build tab on the right of the properties window.



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



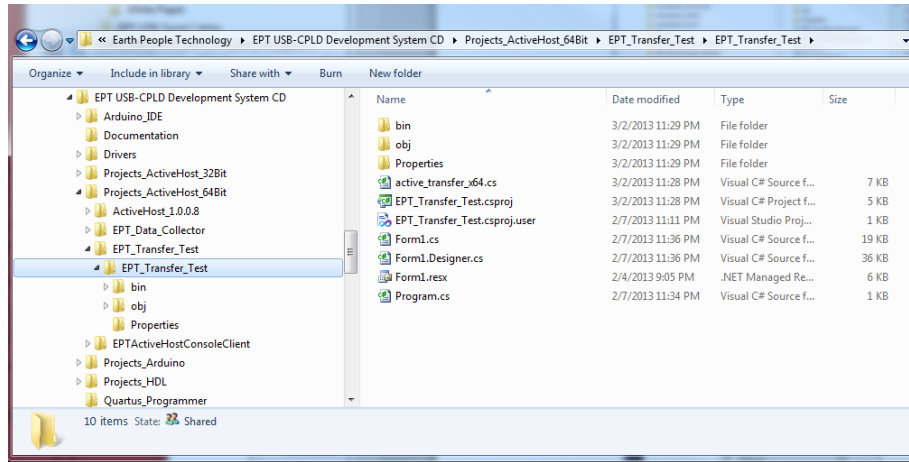
Next, unsafe code needs to be allowed so that C# can be passed pointer values from the Active Host. Click on the Build tab and locate the “Allow unsafe code” check box. Check the box



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

6.2 Assembling Files into the Project

Locate the EPT USB-CPLD Development System CD installed on your PC. Browse to the EPT_Transfer_Test 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_Test project.

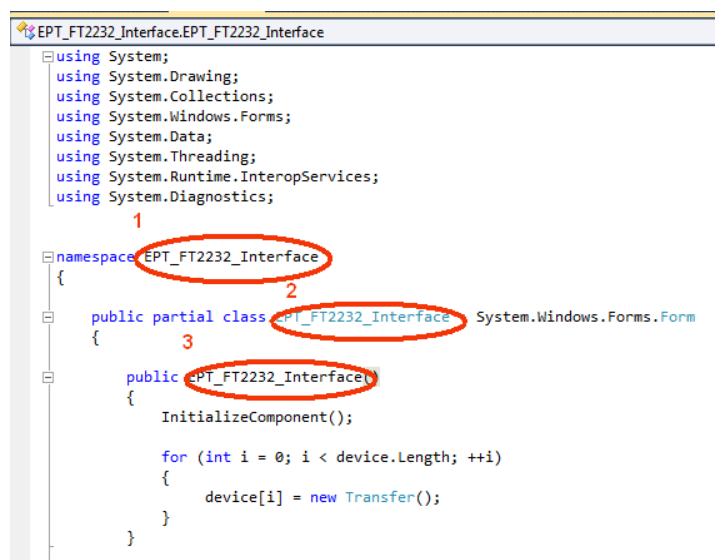


6.2.1 Changing Project Name

NOTE

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

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

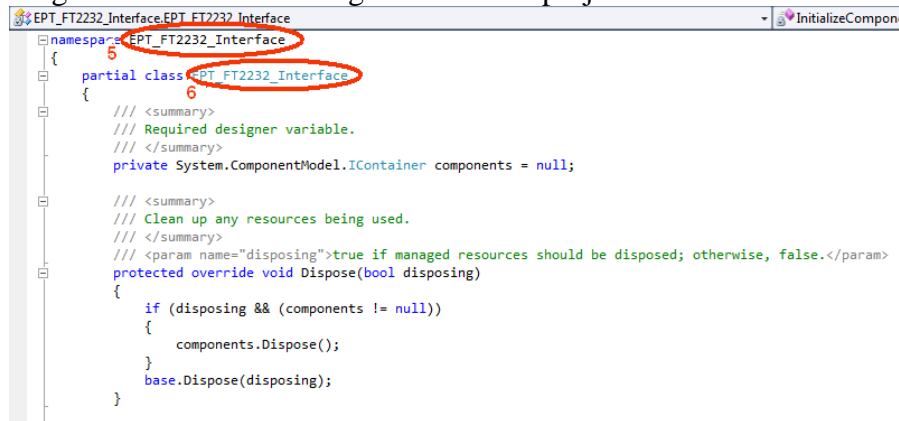


4. Change EPT_Transfer_Test_Load of Form1.cs to new <project name>_Load

Manual

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

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



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

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

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

```
this.Controls.Add(this.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.

```

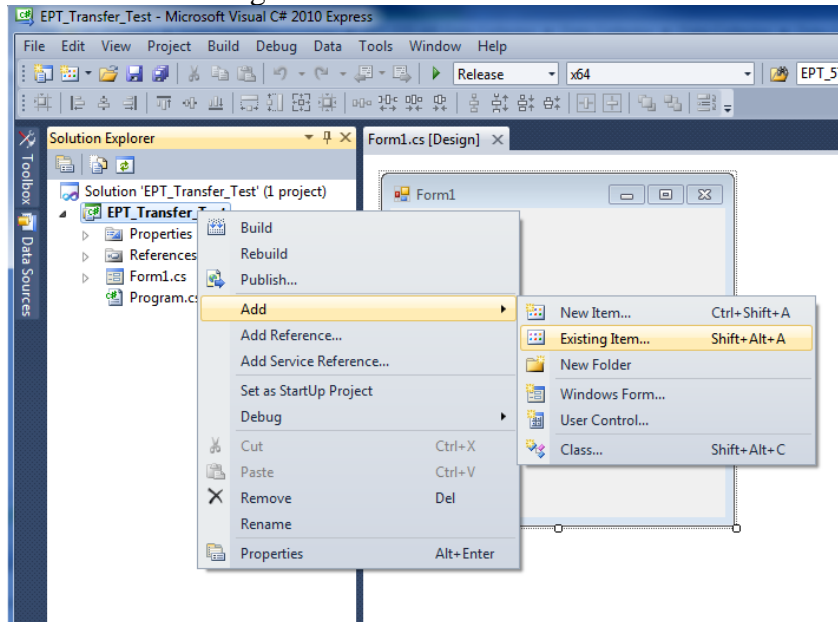
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;

namespace EPT_FT232_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_FT232_Interface());
        }
    }
}

```

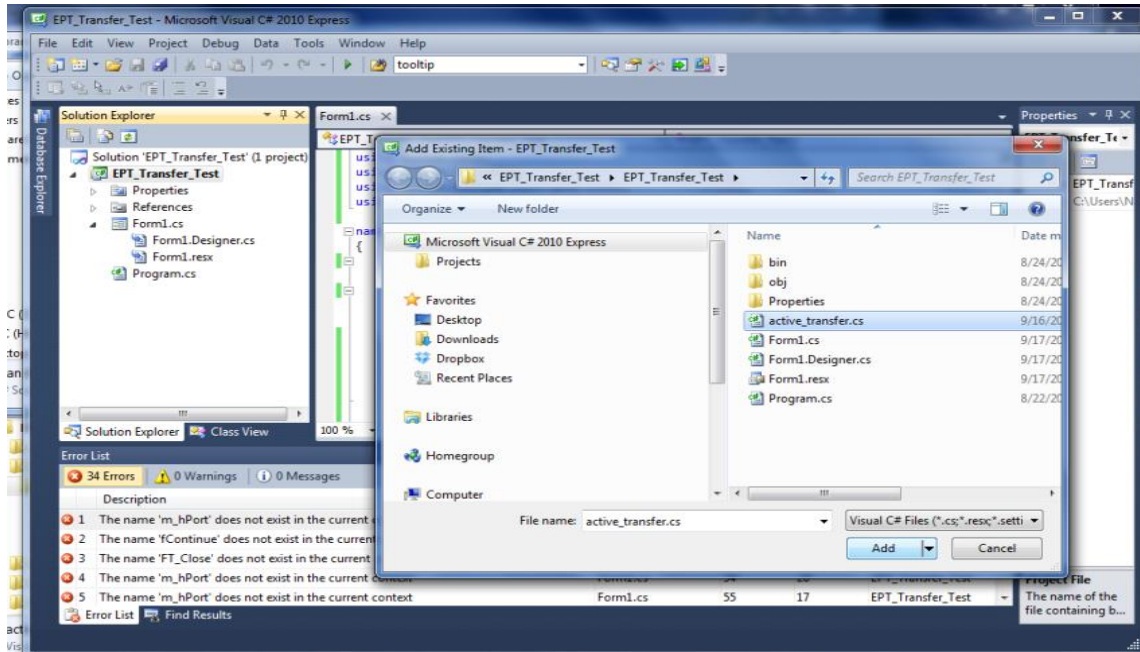
6.2.2 Add Files to Project

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

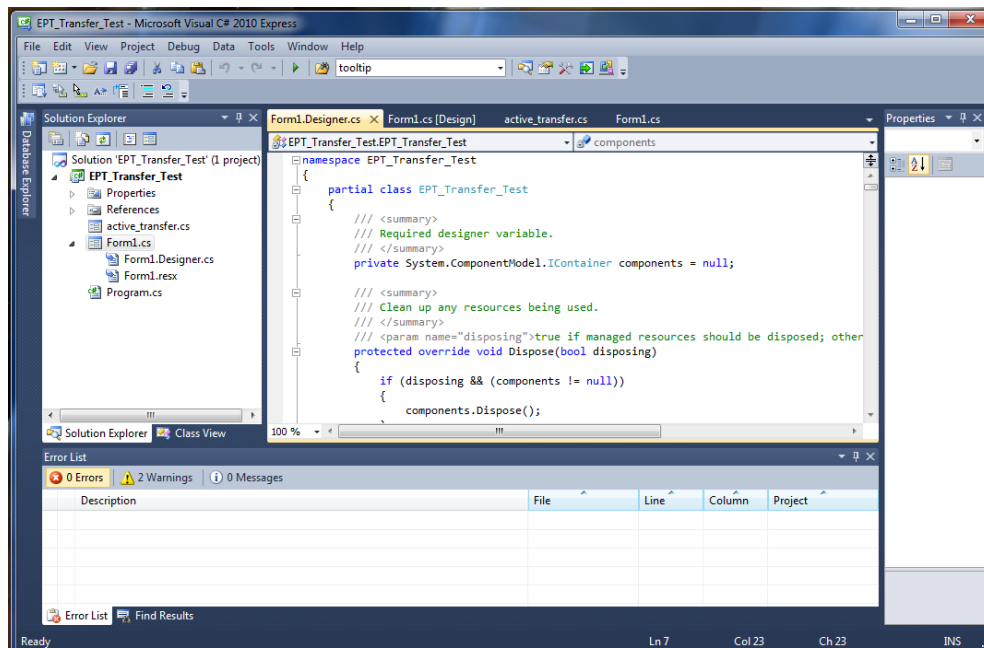


Browse to the EPT_Transfer_Test project folder and select the active_transfer_xx.cs file (choose either the 32 bit or 64 bit version, depending on whether your OS is 32 or 64 bit). Click Add.

Manual




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

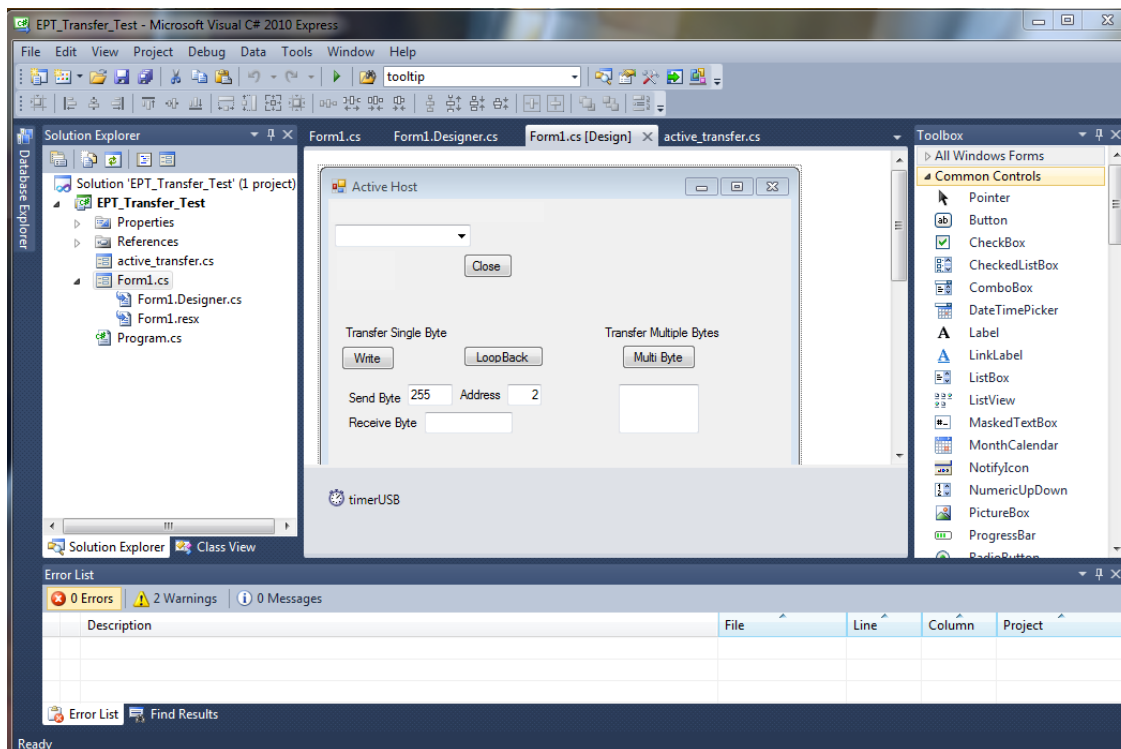


6.2.3 Adding Controls to the Project

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

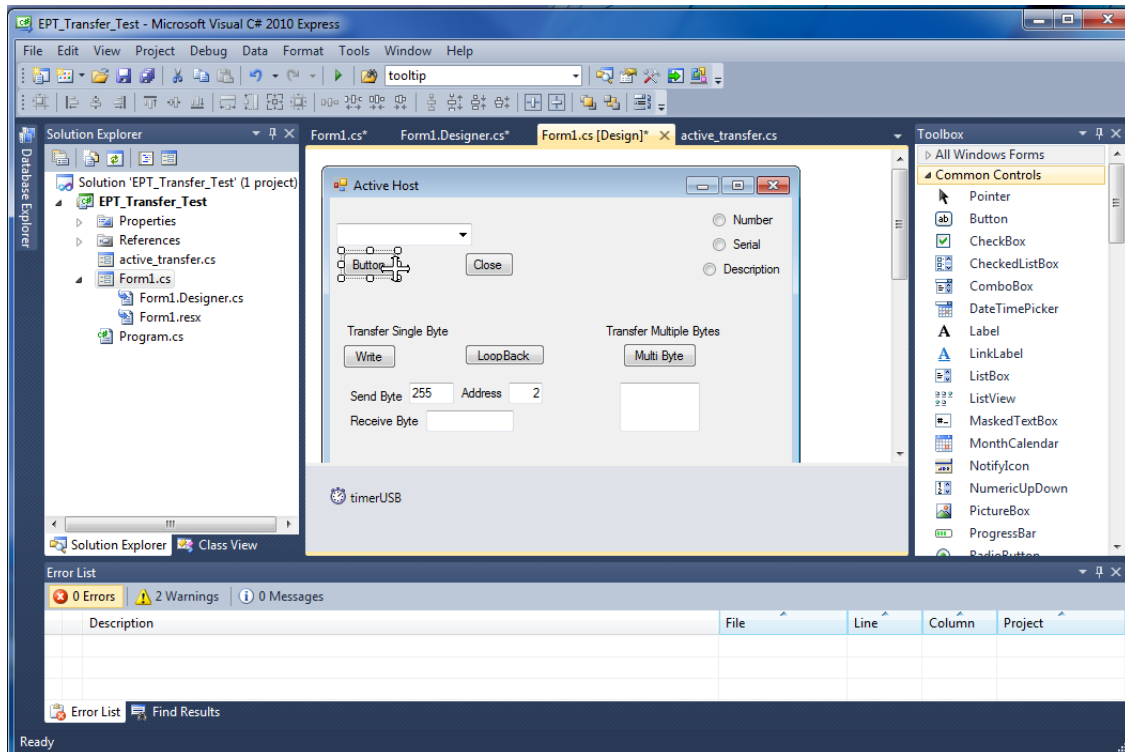
Event based programming means the software responds to events created by the user, a timer event, external events such as serial communication into PC, internal events such as the OS, or other events. The events we are concerned with for our example program are user events and the timer event. The user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

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



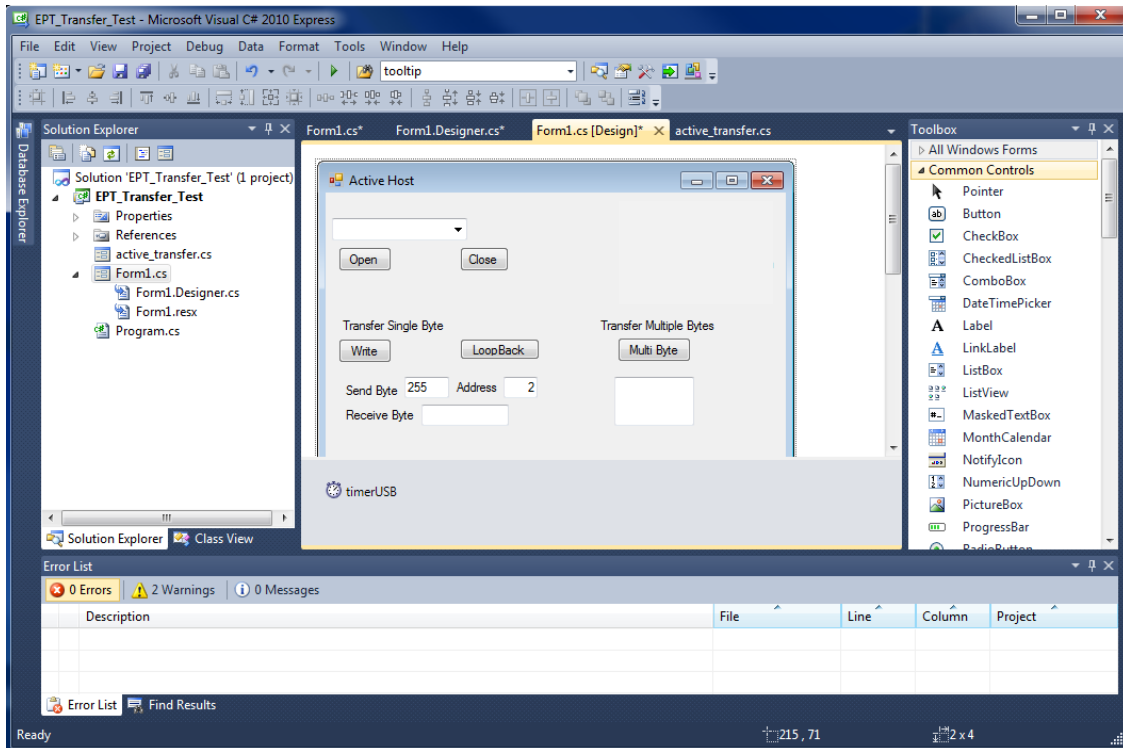
Locate the button on the Toolbox, grab and drag the button onto the Form1.cs [Design] and drop it near the top.

Manual



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.

Manual



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-CPLD Development System.

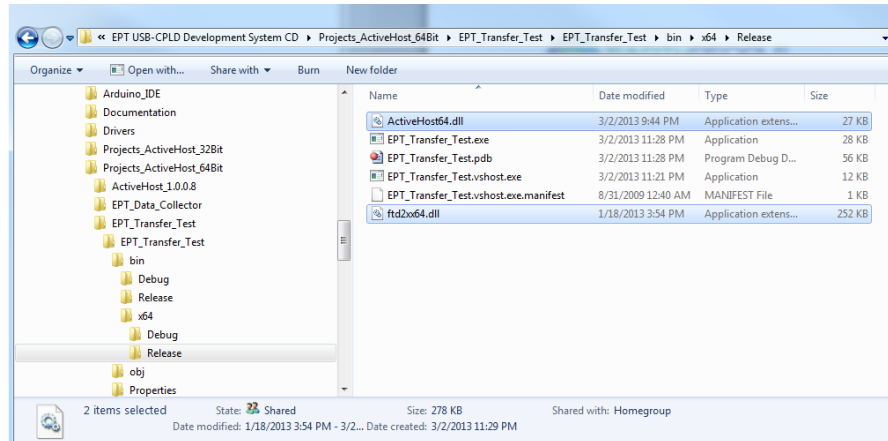
6.2.4 Adding the DLL's to the Project

Locate the EPT USB-CPLD Development System CD 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_Test project.

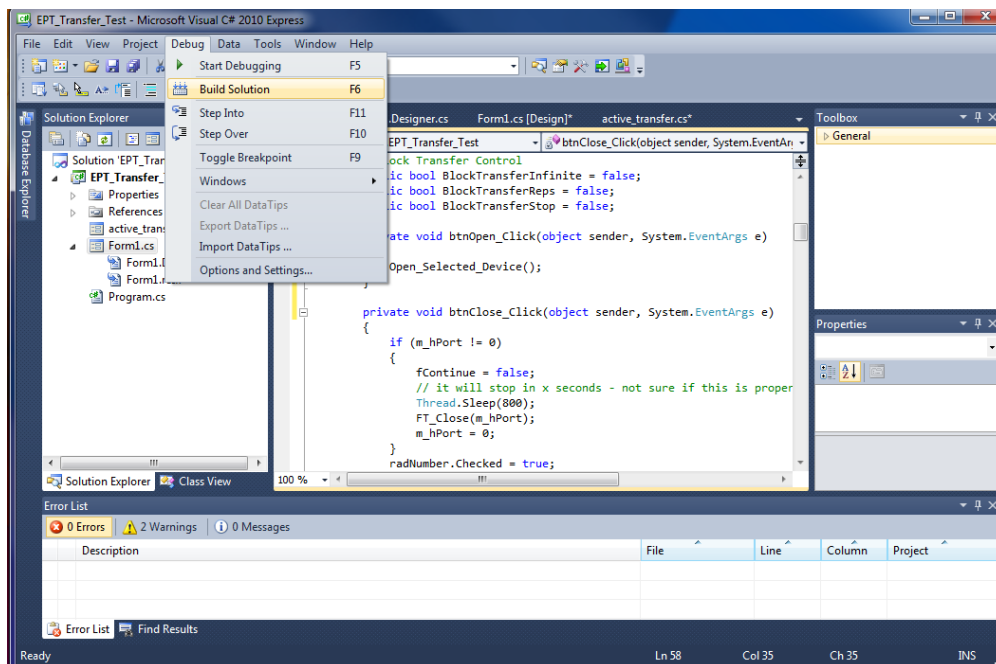
Manual



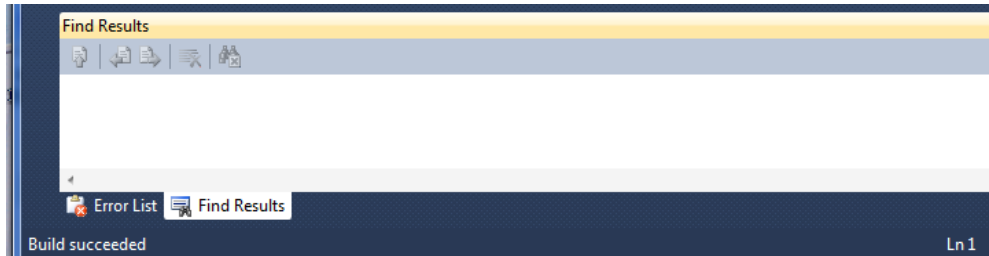
Save the project.

6.2.5 Building the Project

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

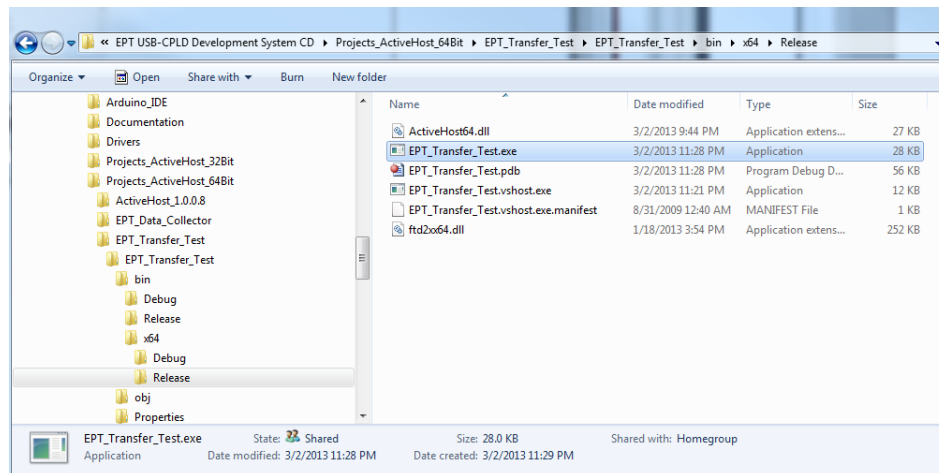


The C# Express compiler will start the building process. If there are no errors with code syntax, function usage, or linking, then the environment responds with “Build Succeeded”.



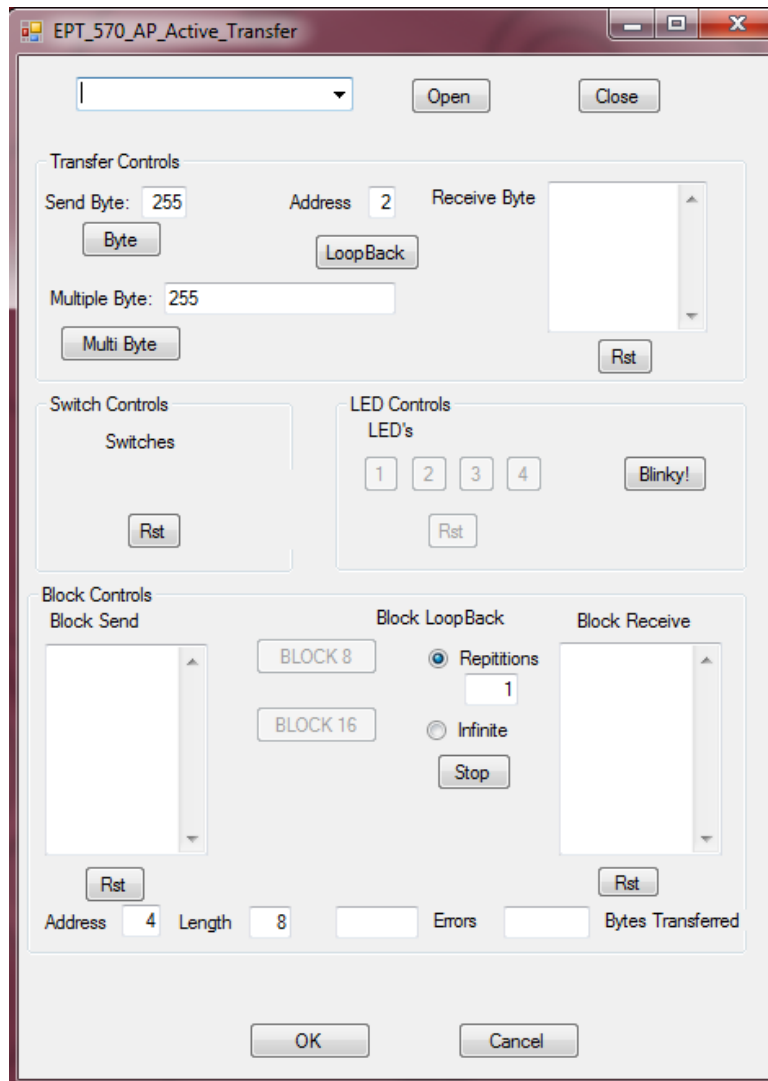
6.2.6 Testing the Project

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

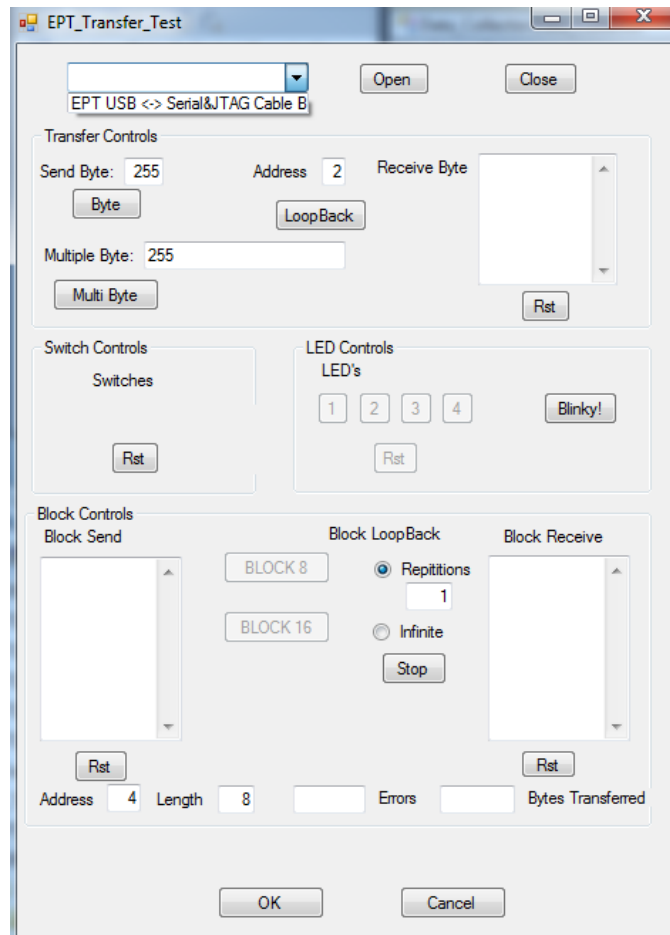


The EPT_Transfer_Text.exe file can now be tested using the UnoMax board. To test the file, connect the UnoMax to the Windows PC using Type A to Type Micro B USB cable. Make sure the driver for the board loads. If the USB driver fails to load, the Windows OS will indicate that no driver was loaded for 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.

Manual



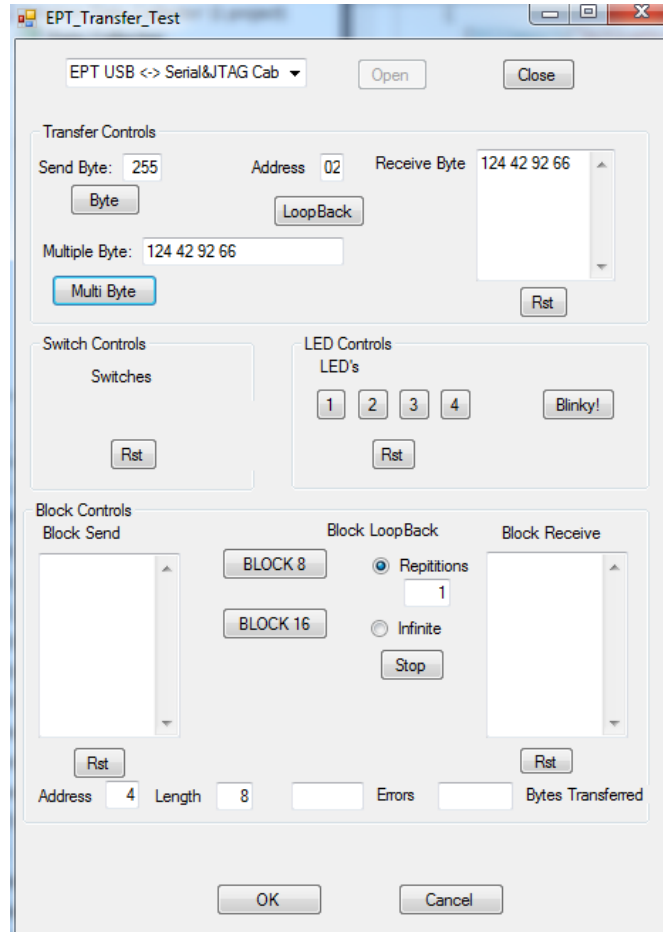
With the application loaded, select the USB-CPLD board from the dropdown combo box and click on the “Open” button.



Click on one of the LED buttons in the middle of the window. The corresponding LED on the UnoMax 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.

Manual



EPT USB <-> Serial&JTAG Cab [Open] [Close]

Transfer Controls

Send Byte: 255 Address: 02 Receive Byte: 124 42 92 66

[Byte] [LoopBack]

Multiple Byte: 124 42 92 66

[Multi Byte] [Rst]

Switch Controls

Switches

[Rst]

LED Controls

LED's

1 2 3 4 [Blinky!]

[Rst]

Block Controls

Block Send

[BLOCK 8] [BLOCK 16]

Block LoopBack

☒ Repititions 1

☐ Infinite [Stop]

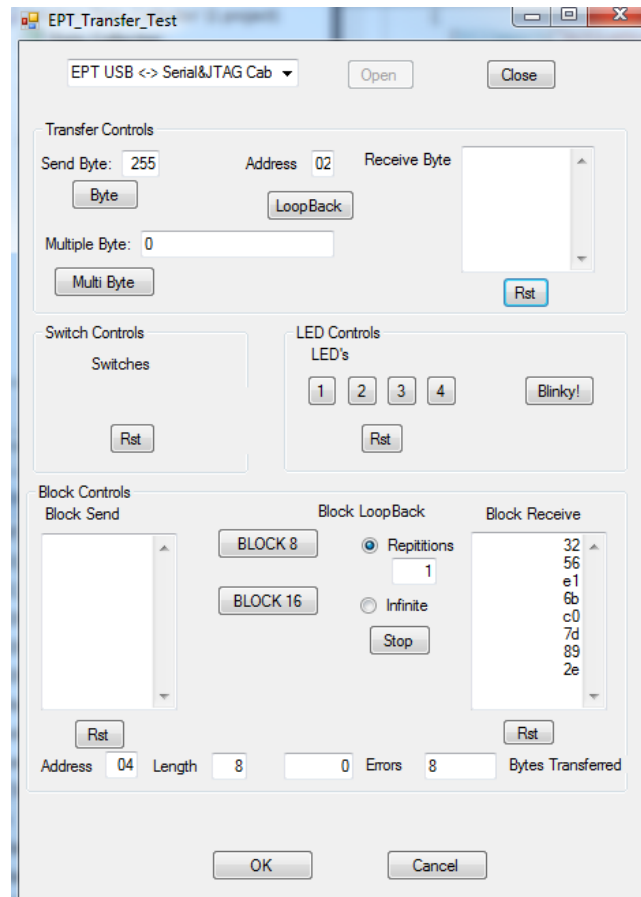
Block Receive

[Rst] [Rst]

Address: 4 Length: 8 Errors: Bytes Transferred:

[OK] [Cancel]

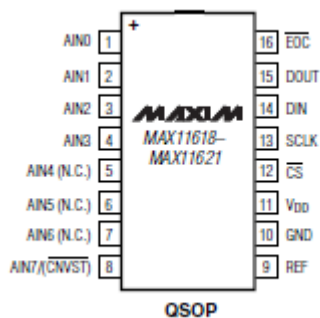
To exercise the Block Transfer EndTerm, click the “Block8” or “Block16” button in the Block Controls group. A pre-selected group of numbers appear in the Block Receive textbox.



Press the PCB switches on the UnoMax to view the Switch Controls in action.

7 Using the Analog to Digital Converter

The EPT 5M57-AP-U2 has an onboard Four Channel, 10 Bit, 300 KSamples/second Analog to Digital Converter. It has a serial SPI communications that allow the host to send setup commands and retrieve the sampled data.



Manual

PIN	NAME	FUNCTION
1–4	AIN0–AIN3	Analog Inputs
5, 6, 7	N.C.	No Connection
8	CONVST	Active – low Conversion Start Input
9	REF	Reference Input
10	GND	Ground
11	VDD	Power Input
12	CS	Active Low Chip Select Input. When CS is Low the interface is enabled. When CS is high MOSI is high impedance
13	SCLK	Serial Clock input. Clocks data in and out of the serial interface.
14	MISO	Serial Data input. MISO data is latched into the interface on the rising edge of SCLK
15	MOSI	Serial Data Output. Data is clocked out on the falling edge of SCLK. High impedance when CS is connected to VDD.
16	EOC	End of Conversion Output. Data is valide after EOC pulls low.

7.1 ELECTRICAL CHARACTERISTICS

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
CONVERSION RATE						
Power-Up Time	t _{PU}	External reference		0.8		μs
		Internal reference (Note 5)		65		
Acquisition Time	t _{ACQ}		0.6			μs
Conversion Time	t _{CONV}	Internally clocked		3.5		μs
		Externally clocked (Note 6)	2.7			
External Clock Frequency	f _{SCLK}	Externally clocked conversion	0.1		4.8	MHz
		Data I/O			10	
Aperture Delay				30		ns
Aperture Jitter				< 50		ps
ANALOG INPUT						
Input Voltage Range		Unipolar	0		V _{REF}	V
Input Leakage Current		V _{IN} = V _{DD}		±0.01	±1	μA
Input Capacitance		During acquisition time (Note 7)		24		pF
INTERNAL REFERENCE						
REF Output Voltage		MAX11618/MAX11620/MAX11624	4.024	4.096	4.168	V
		MAX11619/MAX11621/MAX11625	2.48	2.50	2.52	
REF Temperature Coefficient	T _{CREF}	MAX11618/MAX11620/MAX11624		±20		ppm/°C
		MAX11619/MAX11621/MAX11625		±30		
Output Resistance				6.5		kΩ
REF Output Noise				200		μVRMS
REF Power-Supply Rejection	PSRR			-70		dB
EXTERNAL REFERENCE						
REF Input Voltage Range	V _{REF}		1.0		V _{DD} + 50mV	V
REF Input Current	I _{REF}	V _{REF} = 2.5V (MAX11619/MAX11621/MAX11625); V _{REF} = 4.096V (MAX11618/MAX11620/MAX11624), f _{SAMPLE} = 300ksps		40	100	μA
		V _{REF} = 2.5V (MAX11619/MAX11621/MAX11625); V _{REF} = 4.096V (MAX11618/MAX11620/MAX11624), f _{SAMPLE} = 0		±0.1	±5	

7.2 3-Wire Serial Interface

The MAX11618–MAX11621/MAX11624/MAX11625 feature a serial interface compatible with SPI/QSPI and MICROWIRE devices. For SPI/QSPI, ensure the CPU serial interface runs in master mode so it generates the serial clock signal. Select the SCLK frequency of 10MHz or less, and set clock polarity (CPOL) and phase (CPHA) in the μP control registers to the same value. The MAX11618 operate with SCLK idling high or low, and thus operate with CPOL = CPHA = 0 or CPOL = CPHA = 1. Set CS low to latch input data at DIN on the rising edge of SCLK. Output data at DOUT is updated on the falling edge of SCLK. Results are output in binary format.

Serial communication always begins with an 8-bit input data byte (MSB first) loaded from DIN. A high-to-low transition on CS initiates the data input operation. The input data byte and the subsequent data bytes are clocked from DIN into the serial interface on the rising edge of SCLK. Tables 1–5 detail the register descriptions. Bits 5 and 4, CKSEL1 and CKSEL0, respectively, control the clock modes in the setup register. Choose between four different clock modes for various ways to start a conversion and determine whether the acquisitions are internally or externally timed. Select clock mode 00 to configure CNVST/AIN_ to act as a conversion start and use it to request the programmed, internally timed conversions without tying up the serial bus. In clock mode 01, use CNVST to request conversions one channel at a time, controlling the sampling speed without tying up the serial bus. Request and start internally timed conversions through the serial interface by writing to the conversion register in the default clock mode 10. Use clock mode 11 with SCLK up to 4.8MHz for externally timed acquisitions to achieve sampling rates up to 300ksps. Clock mode 11 disables scanning and averaging.

The device feature an active-low, end-of-conversion output. EOC goes low when the ADC completes the last requested operation and is waiting for the next input data byte (for clock modes 00 and 10). In clock mode 01, EOC goes low after the ADC completes each requested operation. EOC goes high when CS or CNVST goes low. EOC is always high in clock mode 11.

7.3 Output Data Format

Figures 4–7 illustrate the conversion timing for the MAX11618–MAX11621/MAX11624/MAX11625. The 10-bit conversion result is output in MSB-first format with four leading zeros followed by 10-bit data and two trailing zeros. DIN data is latched into the serial interface on the rising edge of SCLK. Data on DOUT transitions on the falling edge of SCLK. Conversions in clock modes 00 and 01 are initiated by CNVST. Conversions in clock modes 10 and 11 are initiated by writing an input data byte to the conversion register. Data output is binary.

Manual

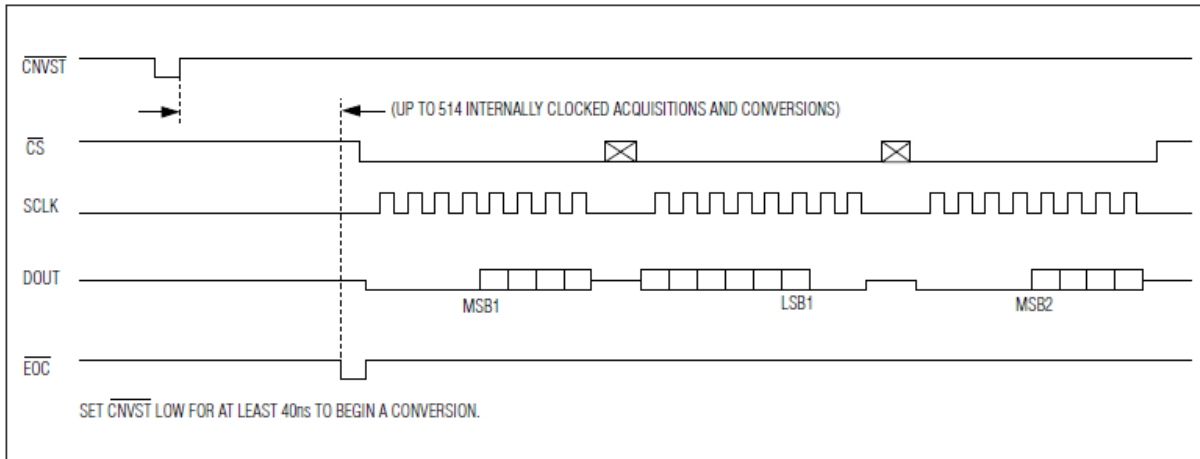


Figure 4. Clock Mode 00

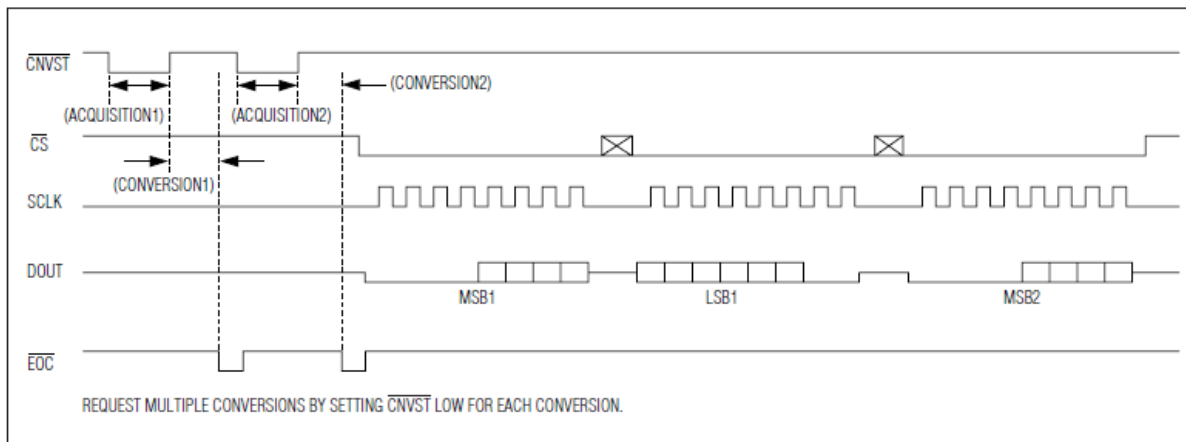


Figure 5. Clock Mode 01

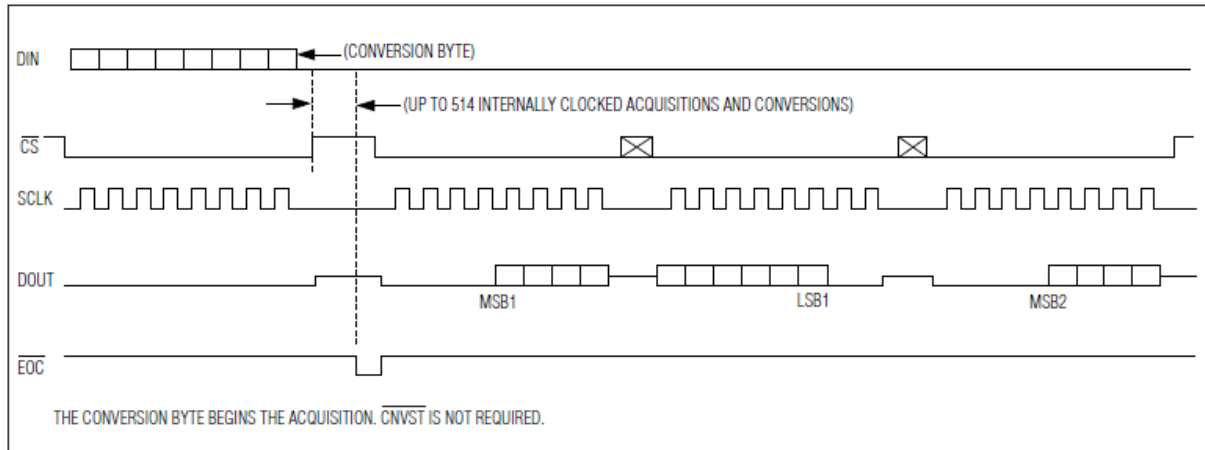


Figure 6. Clock Mode 10

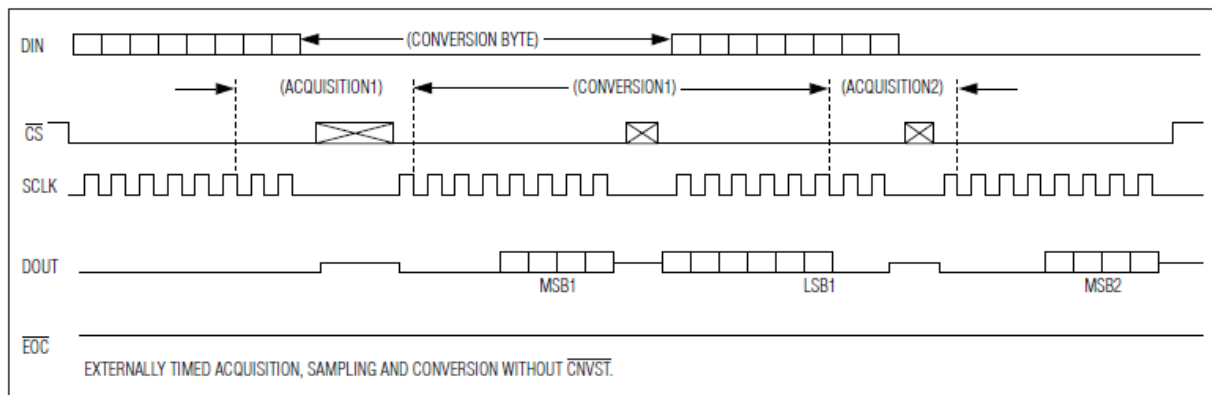


Figure 7. Clock Mode 11

7.4 Register Descriptions

The MAX11618 communicate between the internal registers and the external circuitry through the SPI-/QSPI-compatible serial interface. Table 1 details the registers and the bit names. Tables 2–5 show the various functions within the conversion register, setup register, averaging register, and reset register.

Table 1. Input Data Byte (MSB First)

REGISTER NAME	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
Conversion	1	CHSEL3	CHSEL2	CHSEL1	CHSEL0	SCAN1	SCAN0	X
Setup	0	1	CKSEL1	CKSEL0	REFSEL1	REFSEL0	X	X
Averaging	0	0	1	AVGON	NAV1	NAV0	NSCAN1	NSCAN0
Reset	0	0	0	1	RESET	X	X	X

X = Don't care.

7.5 Conversion Register

Select active analog input channels per scan and scan modes by writing to the conversion register. Table 2 details channel selection and the four scan modes.

Request a scan by writing to the conversion register when in clock mode 10 or 11, or by applying a low pulse to the CNVST pin when in clock mode 00 or 01.

A conversion is not performed if it is requested on a channel that has been configured as CNVST. Select scan mode 00 or 01 to return one result per single-ended channel within the requested range. Select scan mode 10 to scan a single input channel numerous times, depending on NSCAN1 and NSCAN0 in the averaging register (Table 4). Select scan mode 11 to return only one result from a single channel.

Table 2. Conversion Register*

BIT NAME	BIT	FUNCTION
—	7 (MSB)	Set to 1 to select conversion register.
CHSEL3	6	Analog input channel select.
CHSEL2	5	Analog input channel select.
CHSEL1	4	Analog input channel select.
CHSEL0	3	Analog input channel select.
SCAN1	2	Scan mode select.
SCAN0	1	Scan mode select.
—	0 (LSB)	Don't care.

CHSEL1	CHSEL0	SELECTED CHANNEL (N)
0	0	AIN0
0	1	AIN1
1	0	AIN2
1	1	AIN3

SCAN1	SCAN0	SCAN MODE (CHANNEL N IS SELECTED BY BITS CHSEL3–CHSEL0)
0	0	Scans channels 0 through N.
0	1	Scans channels N through the highest numbered channel.
1	0	Scans channel N repeatedly. The averaging register sets the number of results.
1	1	No scan. Converts channel N once only.

7.6 Setup Register

Write a byte to the setup register to configure the clock, reference, and power-down modes. Table 3 details the bits in the setup register. Bits 5 and 4 (CKSEL1 and CKSEL0) control the clock mode, acquisition and sampling, and the conversion start. Bits 3 and 2 (REFSEL1 and REFSEL0) control internal or external reference use.

Table 3. Setup Register*

BIT NAME	BIT	FUNCTION
—	7 (MSB)	Set to zero to select setup register.
—	6	Set to 1 to select setup register.
CKSEL1	5	Clock mode and $\overline{\text{CNVST}}$ configuration. Resets to 1 at power-up.
CKSEL0	4	Clock mode and $\overline{\text{CNVST}}$ configuration.
REFSEL1	3	Reference mode configuration.
REFSEL0	2	Reference mode configuration.
—	1	Don't care.
—	0 (LSB)	Don't care.

*See below for bit details.

CKSEL1	CKSEL0	CONVERSION CLOCK	ACQUISITION/SAMPLING	$\overline{\text{CNVST}}$ CONFIGURATION
0	0	Internal	Internally timed	$\overline{\text{CNVST}}$
0	1	Internal	Externally timed through $\overline{\text{CNVST}}$	$\overline{\text{CNVST}}$
1	0	Internal	Internally timed	AIN15/AIN11/AIN7**
1	1	External (4.8MHz max)	Externally timed through SCLK	AIN15/AIN11/AIN7**

**For the MAX11618/MAX11619, $\overline{\text{CNVST}}$ has its own dedicated pin.

REFSEL1	REFSEL0	VOLTAGE REFERENCE	AutoShutdown
0	0	Internal	Reference off after scan; need wake-up delay.
0	1	External single ended	Reference off; no wake-up delay.
1	0	Internal	Reference always on; no wake-up delay.
1	1	Reserved	Reserved. Do not use.

7.7 Averaging Register

Write to the averaging register to configure the ADC to average up to 32 samples for each requested result, and to independently control the number of results requested for single-channel scans. Table 2 details the four scan modes available in the conversion register. All four scan modes allow averaging as long as the AVGON bit, bit 4 in the averaging register, is set to 1. Select scan mode 10 to scan the same channel multiple times. Clock mode 11 disables averaging.

Table 4. Averaging Register*

BIT NAME	BIT	FUNCTION
—	7 (MSB)	Set to 0 to select averaging register.
—	6	Set to 0 to select averaging register.
—	5	Set to 1 to select averaging register.
AVGON	4	Set to 1 to turn averaging on. Set to zero to turn averaging off.
NAVGI	3	Configures the number of conversions for single-channel scans.
NAVGO	2	Configures the number of conversions for single-channel scans.
NSCAN1	1	Single-channel scan count. (Scan mode 10 only.)
NSCAN0	0 (LSB)	Single-channel scan count. (Scan mode 10 only.)

*See below for bit details.

AVGON	NAVGI	NAVGO	FUNCTION
0	X	X	Performs 1 conversion for each requested result.
1	0	0	Performs 4 conversions and returns the average for each requested result.
1	0	1	Performs 8 conversions and returns the average for each requested result.
1	1	0	Performs 16 conversions and returns the average for each requested result.
1	1	1	Performs 32 conversions and returns the average for each requested result.

X = Don't care.

NSCAN1	NSCAN0	FUNCTION (APPLIES ONLY IF SCAN MODE 10 IS SELECTED)
0	0	Scans channel N and returns 4 results.
0	1	Scans channel N and returns 8 results.
1	0	Scans channel N and returns 12 results.
1	1	Scans channel N and returns 16 results.

7.8 Reset Register

Write to the reset register (as shown in Table 5) to clear the FIFO or to reset all registers to their default states. Set the RESET bit to 1 to reset the FIFO. Set the reset bit to zero to return the MAX11618 to the default power-up state.

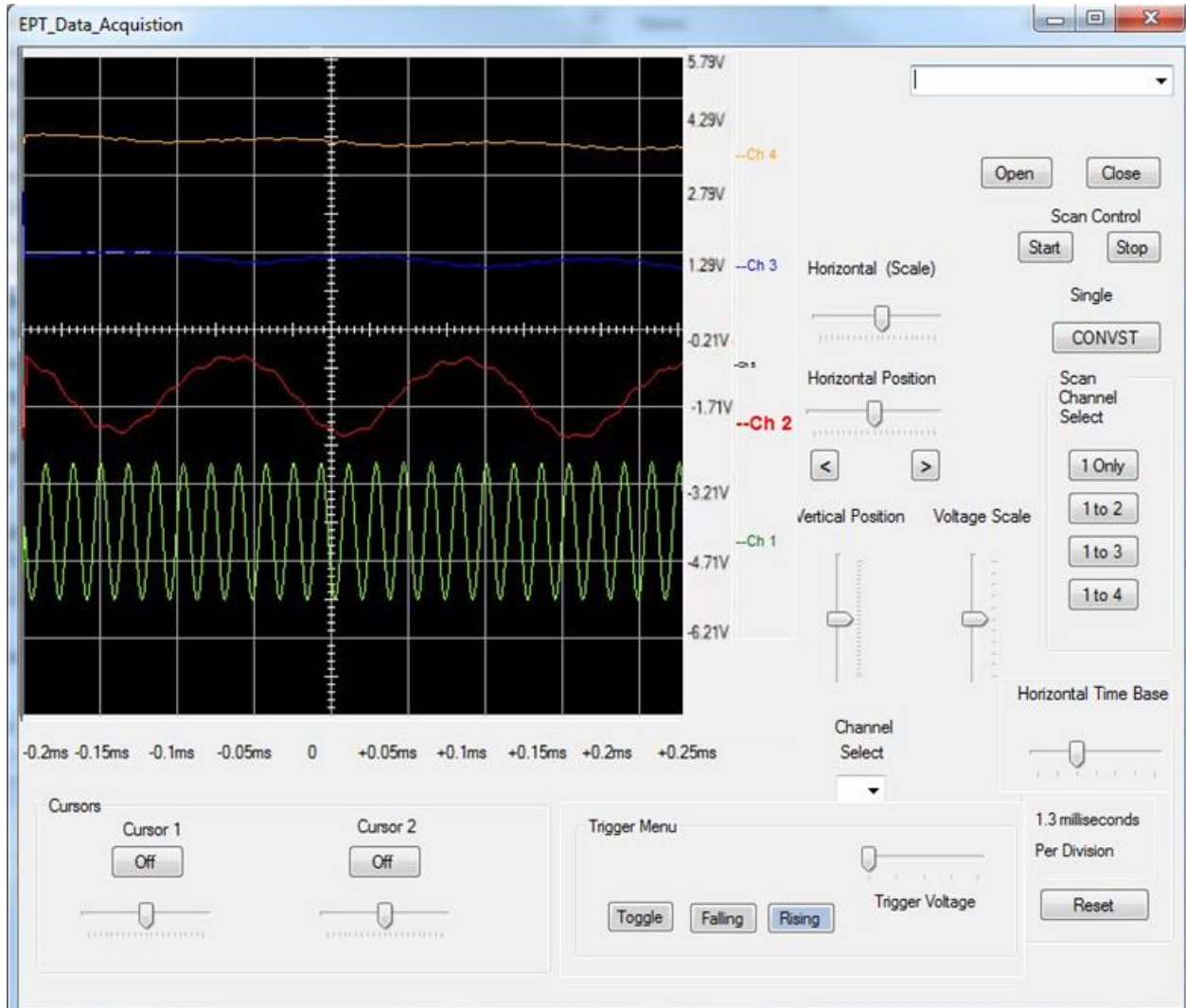
Table 5. Reset Register

BIT NAME	BIT	FUNCTION
—	7 (MSB)	Set to 0 to select reset register.
—	6	Set to 0 to select reset register.
—	5	Set to 0 to select reset register.
—	4	Set to 1 to select reset register.
RESET	3	Set to zero to reset all registers. Set to 1 to clear the FIFO only.
X	2	Don't care.
X	1	Don't care.
X	0 (LSB)	Don't care.

8 The UnoProLyzer Application

The source DVD for the UnoMax2 comes with the UnoProLyzer application project. This project allows the user to display 1 to 4 channels of analog input in a graphing application. The project utilizes the PC to perform all data storage and graphing. The PC sends commands to and receives the data from the UnoMax2 and stores each channel data in its own separate buffer in memory. The UnoProLyzer collects all samples from each channel by streaming across up to four dedicated communication “pipes”.

The UnoMax2 commands the ADC to start a conversion on the channels selected by the user. It then waits for the ADC to complete the conversion on all channels. It transfers the data for each channel across its own dedicated communication pipe. Then starts the process over again. The UnoProLyzer application will accept each data word and decode the pipe number it came across. It stores each word into a separate buffer for each channel. The UnoProLyzer then performs post processing on each data word. It performs trigger detection, smoothing, sorting, scaling and searching. It then displays the data set in 500 data point segments.

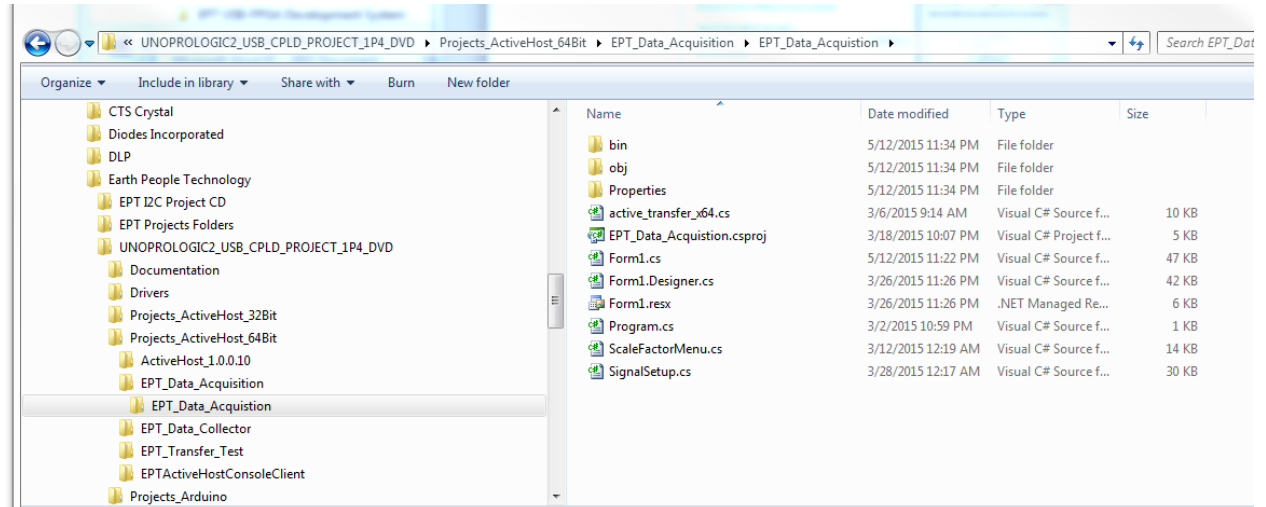


8.1 Accessing the UnoProLyzer Application

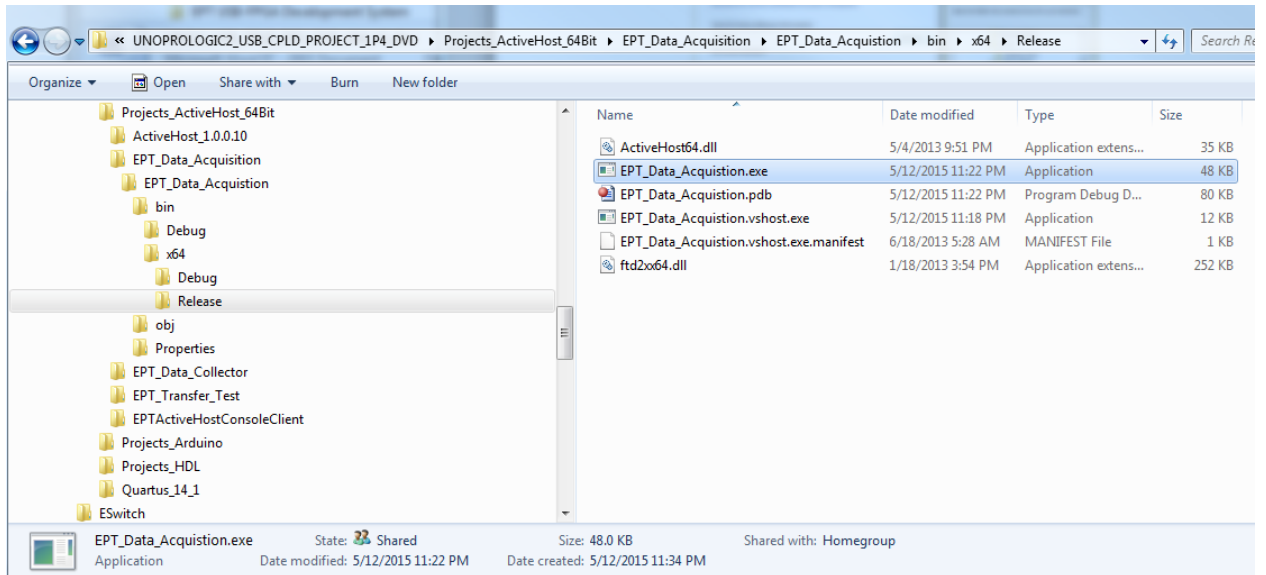
Locate the EPT_Data_Acquisition folder in the Drivers folder of the EPT USB-CPLD Development System DVD using Windows Explorer.

UnoMax CPLD Development System User

Manual



Go to the Release folder and locate the EPT_Data_Acquisition.exe file.

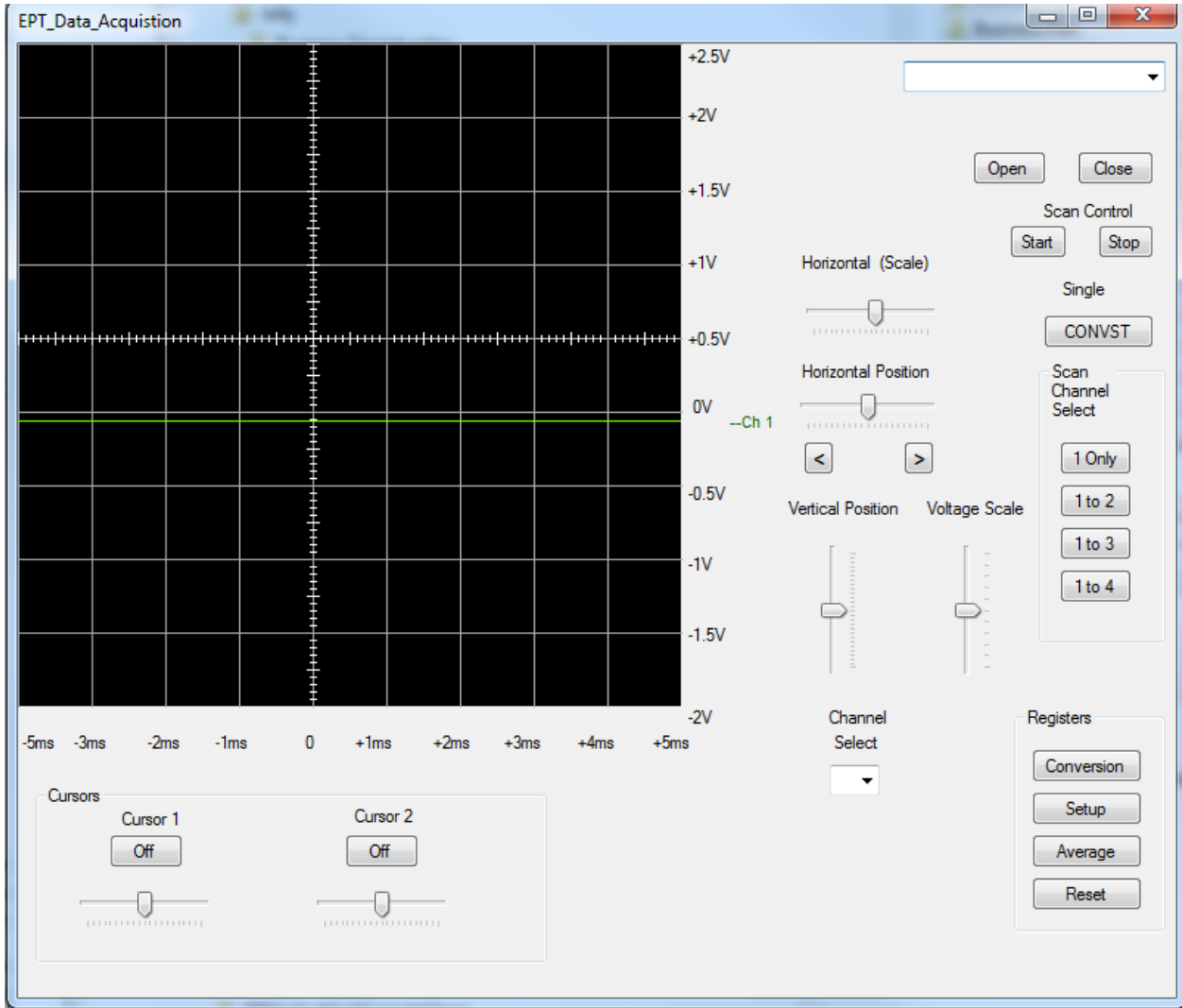


Make sure the UnoMax2 is installed and the USB driver has been loaded.

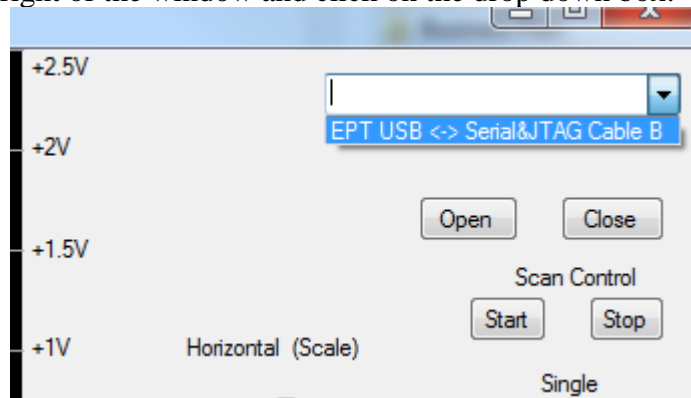


Double click on the EPT_Data_Acquisition.exe file and the UnoProLyzer Application will open.

Manual

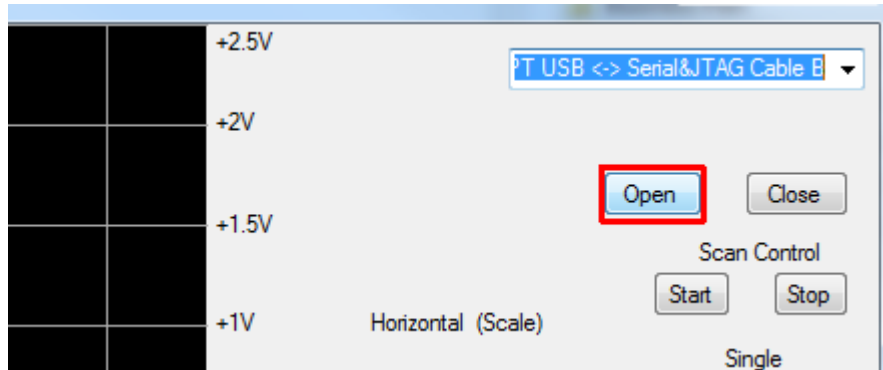


Go to the upper right of the window and click on the drop down box.

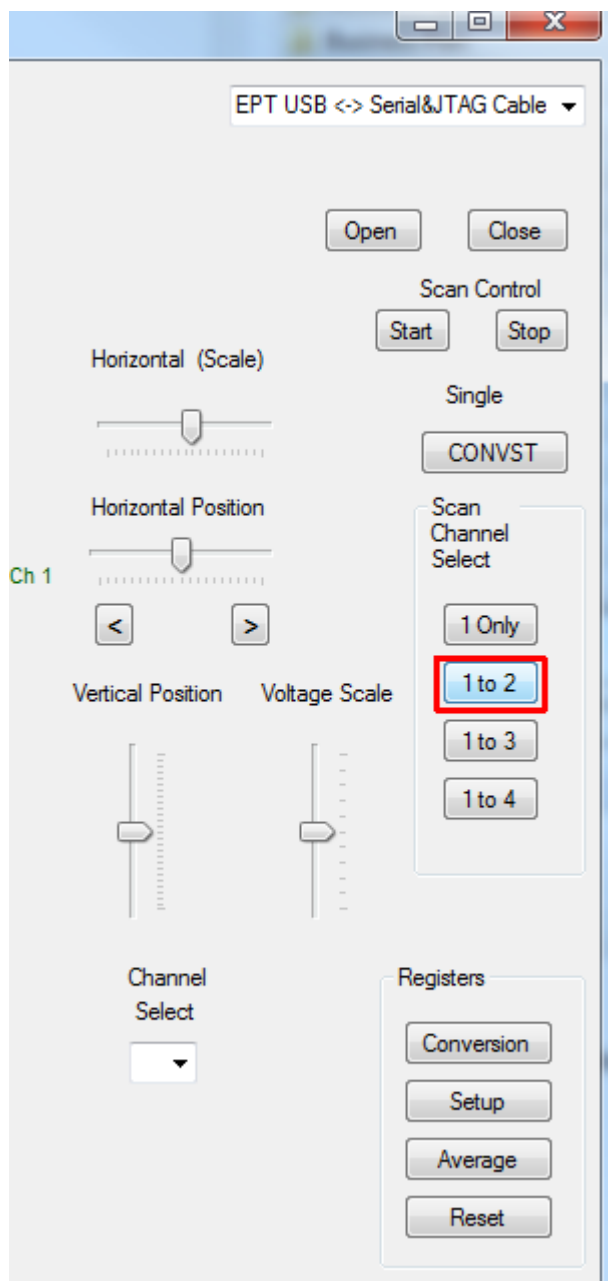


Select the “EPT USB<-> Serial&JTAG Cable B. Then click on the Open button

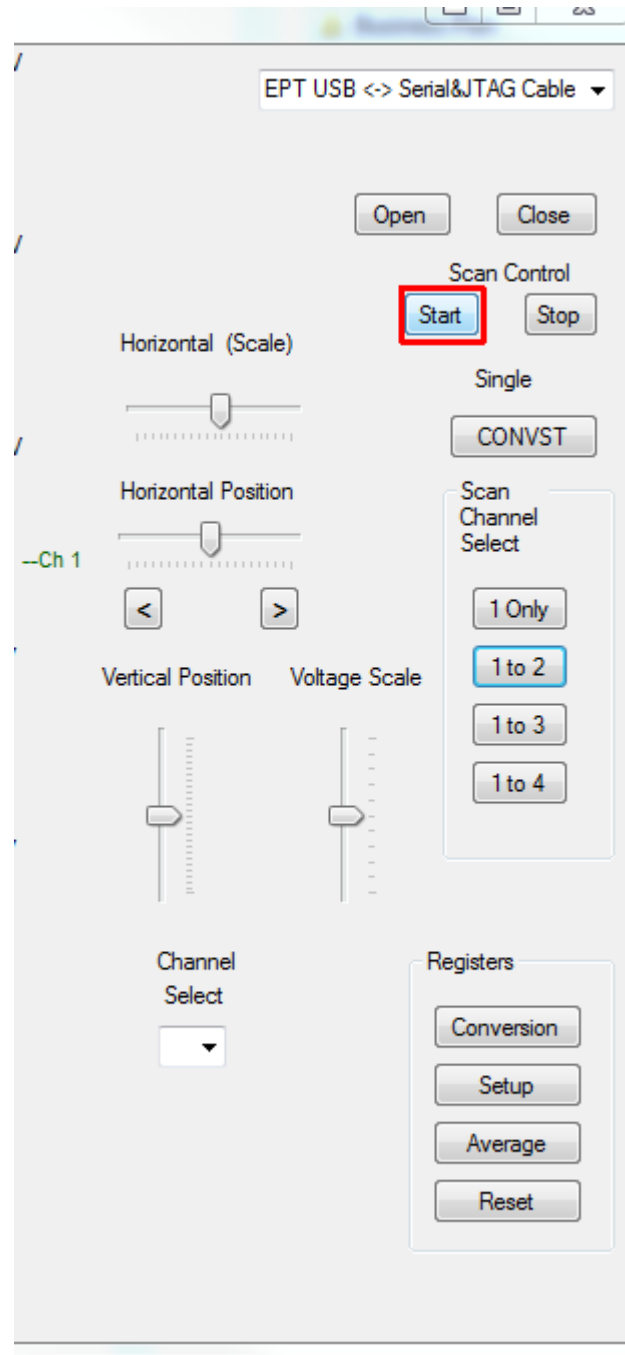
Manual



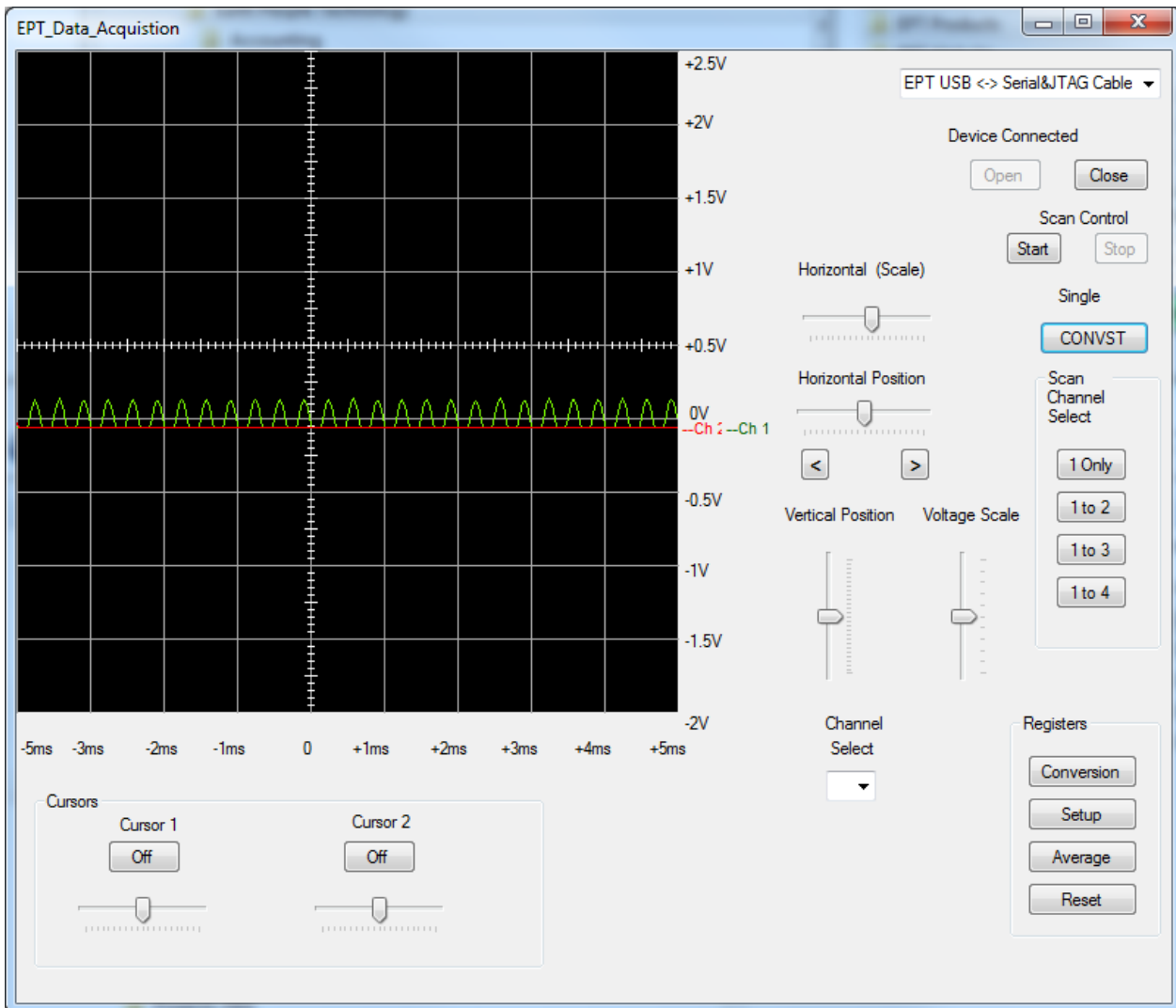
Next, select the number of channels to display. The channels have to be selected in sequential order, you cannot pick out single channel (except for channel 1). So for two channels, click on the “1 to 2” button. This will display the data from both channel 1 and 2.



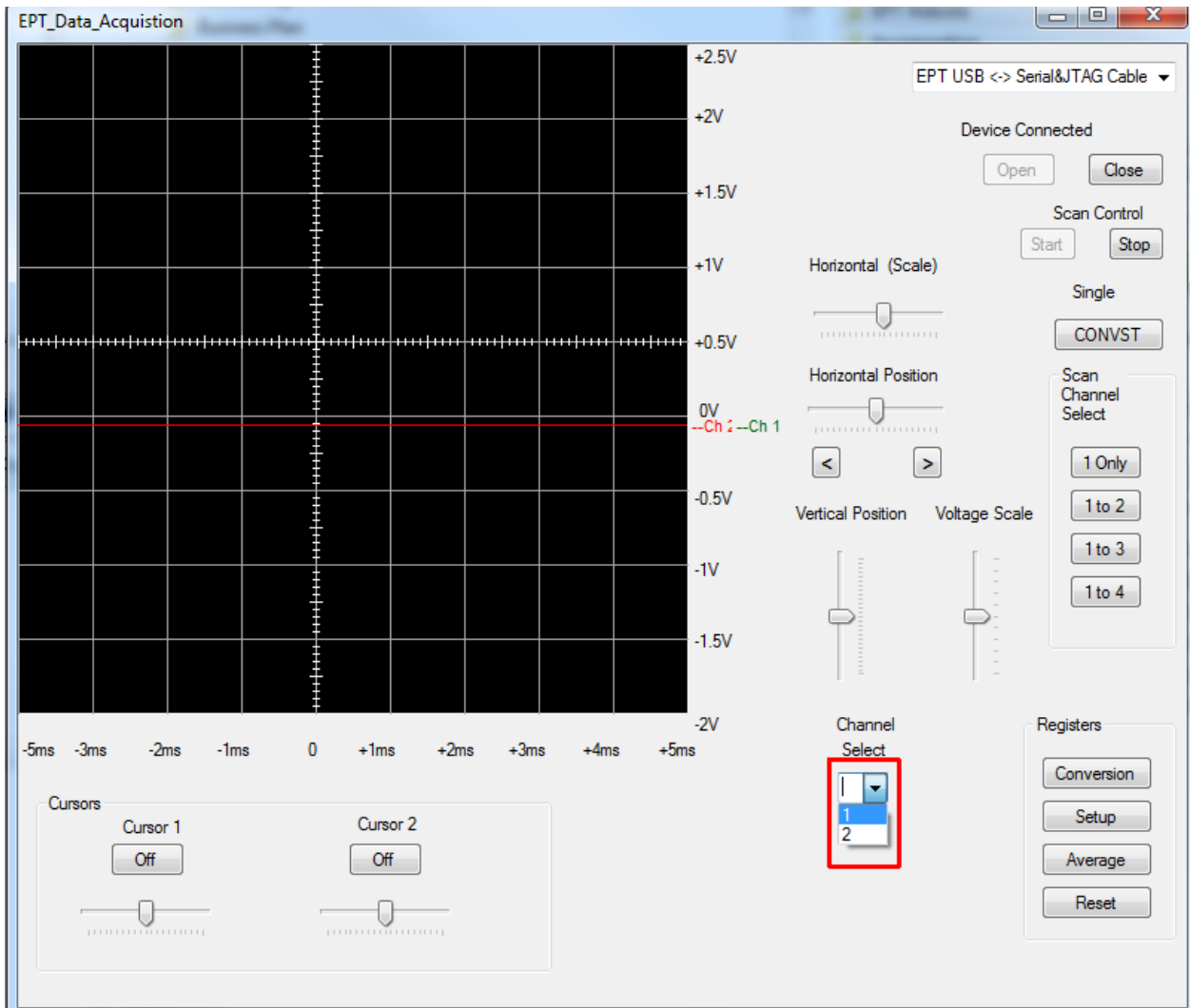
Next, click on the Start button.



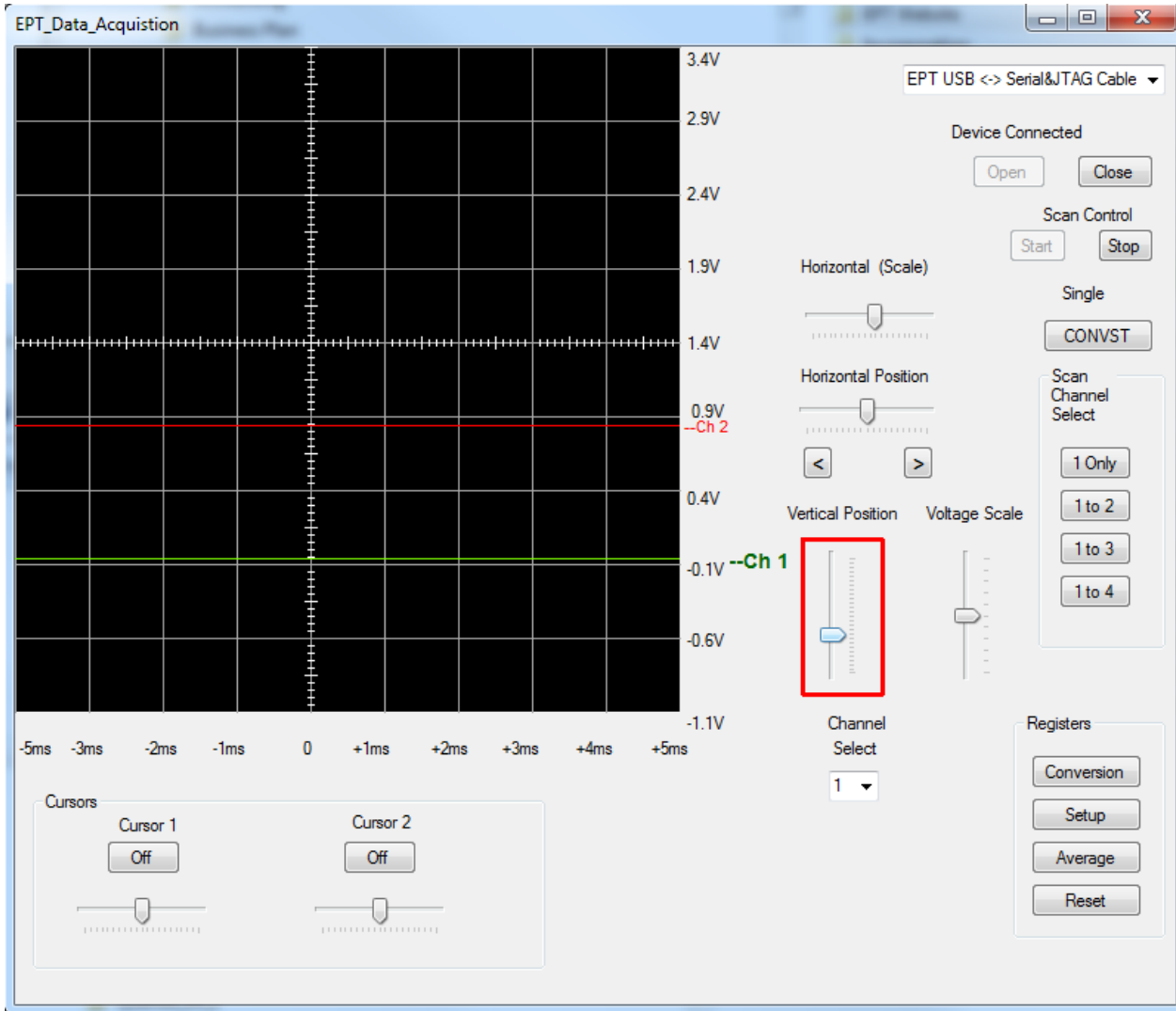
The data from the two channels will appear at the same latitude on the graph.



Next, locate the Channel Select drop down box and click on channel 1.



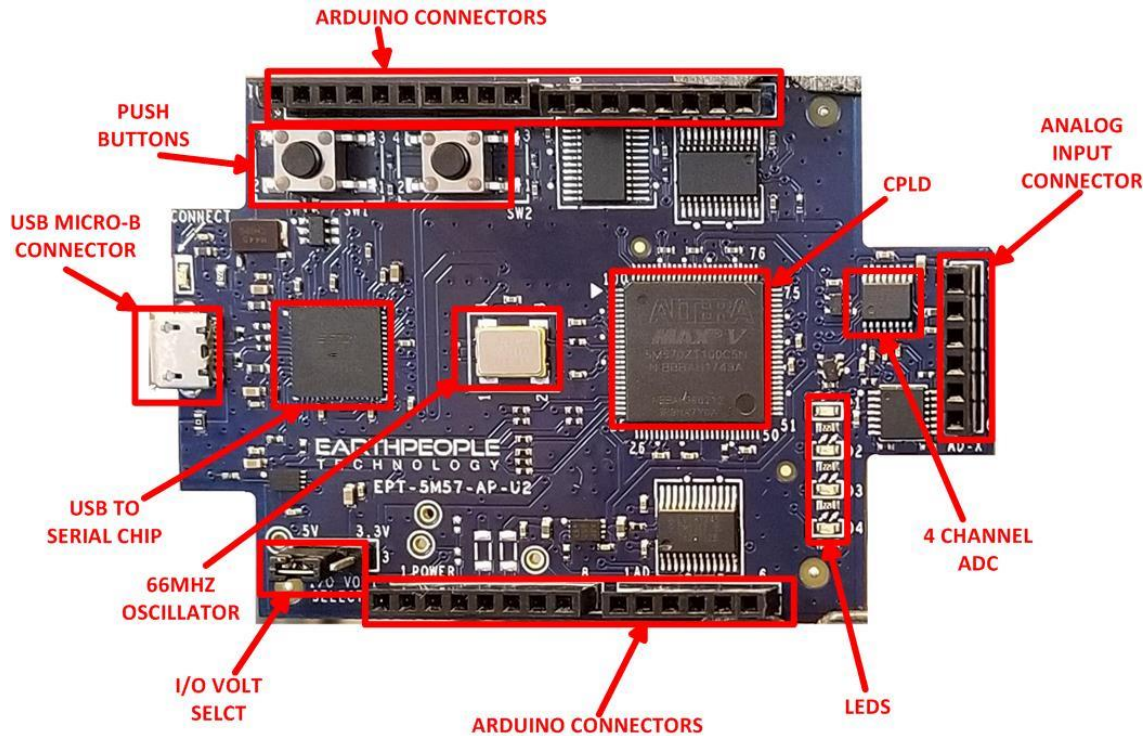
Locate the Vertical Position slider and pull it down. The channel 1 data will change position in the graph depending on where you move the slider. The voltage magnitude data also adjusts to indicate the magnitude of the data relative to the position of channel 1 data.



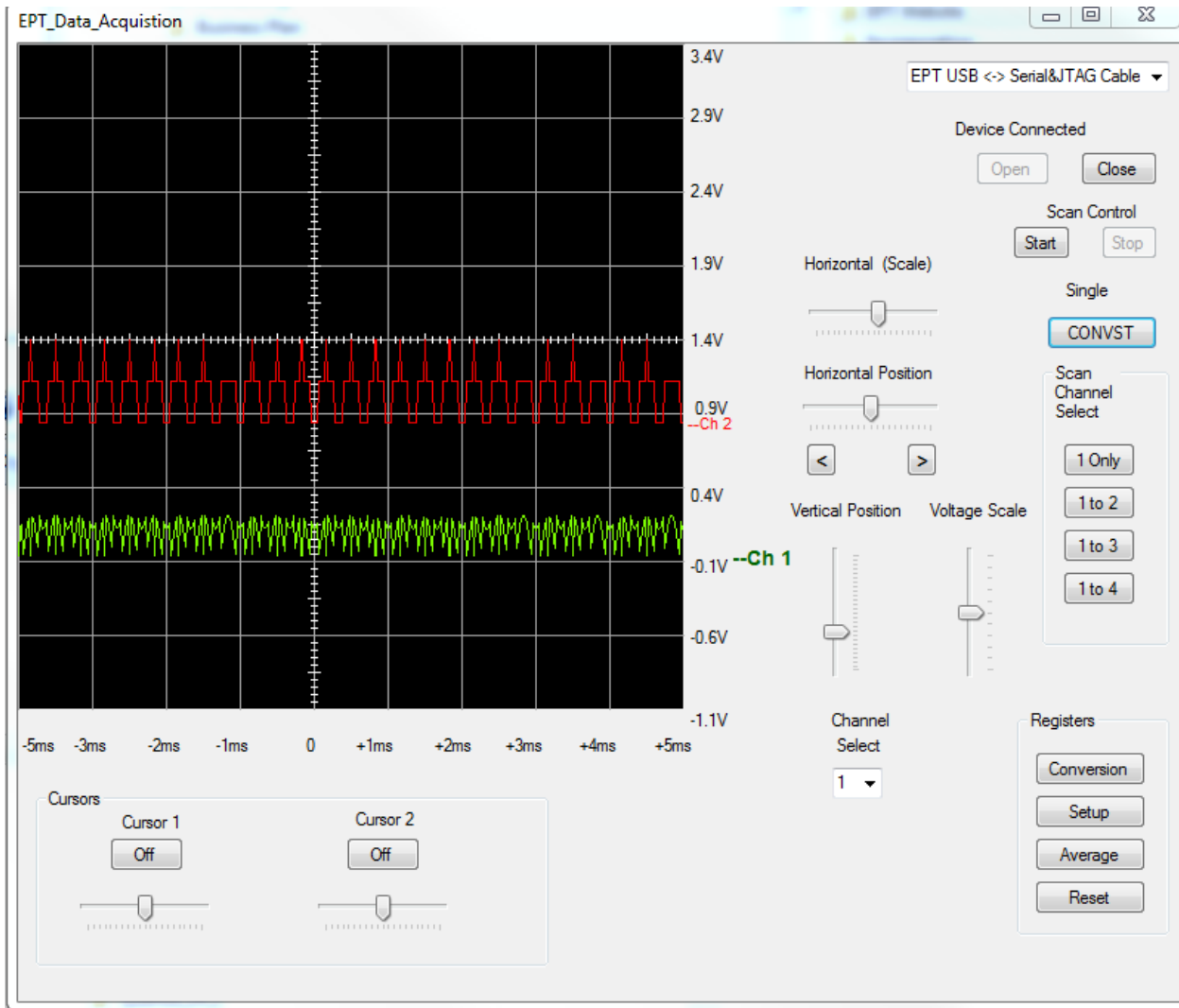
The selected channel will show up as a large icon. Its position indicates the zero position of the data. The magnitude information along the y-axis is only for the selected channel.

Manual

Then connect a signal to the channel 1 input on the UnoMax2.



If you don't have a 0-5 Volt signal to connect to the UnoMax2, you can use your finger and touch it to the bottom of the Analog Input Connector. The ambient electricity from your body has just enough current to give the Analog inputs a deflection from zero.



Now the UnoMax and UnoProLyzer are ready to measure an 0-5VDC signals.

9 Developing Verilog Applications

The UnoMax CPLD development system includes tutorials for getting started with Verilog. The DVD includes a Verilog tutorial and a Modelsim tutorial. These two tutorials will assist with learning Verilog and getting up to speed with simulating Verilog designs in software. The Verilog simulation is

10 APPENDIX I

List of Abbreviations and Acronyms



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

APPENDIX II

Details of the Altera EPM570 CPLD