



USB CPLD Development System User Manual

EARTH PEOPLE TECHNOLOGY, Inc

USB-CPLD DEVELOPMENT SYSTEM MEGAPROLOGIC User Manual

The EPT MegaProLogic development system provides an innovative method of developing and debugging programmable logic code. It also provides a high speed data transfer mechanism between an Arduino board and a host PC. The MegaProLogic development system provides a convenient, user-friendly work flow by connecting seamlessly with Intel's Quartus Prime software. The user will develop the code in the Quartus Prime environment on a Windows Personal Computer. The programmable logic code is loaded into the CPLD using only the Quartus Prime Programmer tool and a standard USB-C cable. The Active Host SDK provides a highly configurable communications interface between Arduino and host. It 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. No scheduling USB transfers, USB driver interface or inf file changes are needed. The EPT USB-CPLD development system is a unique combination of hardware and software.

Circuit designs, software and documentation are copyright © 2023, 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/>



USB CPLD Development System User Manual

Table of Contents

1 Introduction and General Description4

1.1 Test Driving the Active Host Test Application5

1.2 MEGAPROLOGIC 18

1.2.1 Serial USB Communications20

1.2.2 Host PC Connection22

1.2.3 PMOD Connectors22

1.2.4 Inputs and Outputs24

1.2.5 User LEDs25

1.2.6 User Push Buttons27

1.2.7 Slide Switches28

1.2.8 JTAG30

1.3 Active Host EndTerms31

1.4 Active Transfer EndTerms32

2 EPT Drivers.....32

2.1 USB Driver33

2.2 JTAG DLL Insert to Quartus Prime Lite.....36

2.2.1 Installing Quartus36

2.2.2 Downloading Quartus38

2.2.3 Quartus Installer43

2.2.4 Adding the EPT_Blaster to Quartus Prime51

2.3 Active Host Application DLL52

3 Active Transfer Library.....59

3.1 EPT Active Transfer System Overview59

3.2 Active Transfer Library60

3.2.1 Active Trigger EndTerm63

3.2.2 Active Transfer EndTerm66

3.2.3 Active Block EndTerm68

3.3 Timing Diagram for Active Transfer Methods71

3.3.1 Active Trigger EndTerm Timing71

3.3.2 Active Transfer EndTerm Timing.....71

3.3.3 Active Block EndTerm Timing.....72

4 Compiling, Synthesizing, and Programming CPLD73

4.1 Setting up the Project and Compiling.....73

4.1.1 Selecting Pins and Synthesizing.....83

4.1.2 Programming the CPLD.....93

5 Active Host Application.....100

5.1 Trigger EndTerm101

5.2 Transfer(Byte) EndTerm101

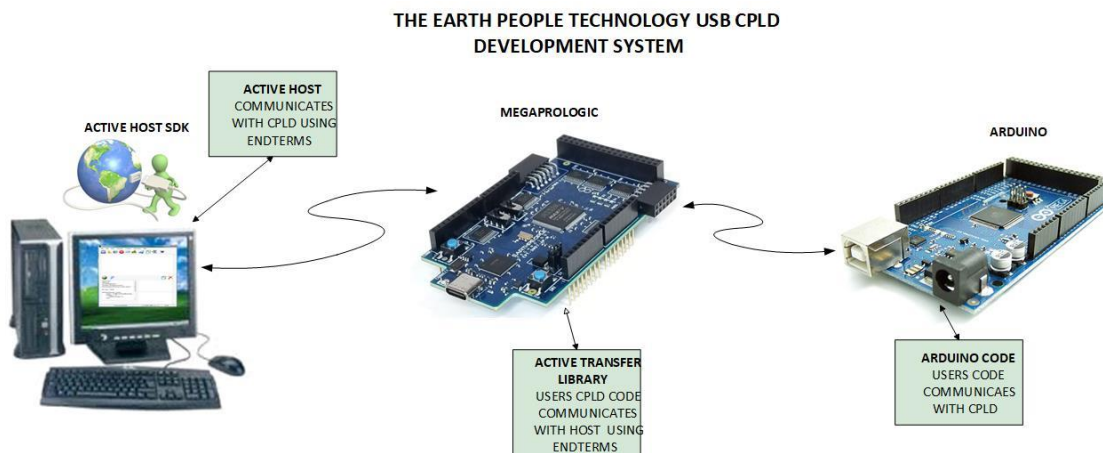


USB CPLD Development System User Manual

5.3	Block EndTerm	102
5.4	Active Host DLL	102
5.4.1	Active Host Open Device.....	103
5.4.2	Active Host Triggers	106
5.4.3	Active Host Byte Transfers	108
5.4.4	Active Host Block Transfers.....	110
6	Assembling, Building, and Executing a .NET Project on the PC	113
6.1	Creating a Project	114
6.1.1	Setting up the C# Express Environment for x64 bit	118
6.2	Assembling Files into the Project.....	124
6.2.1	Changing Project Name	125
6.2.2	Add Files to Project.....	127
6.2.3	Adding Controls to the Project.....	129
6.2.4	Adding the DLL's to the Project.....	134
6.2.5	Building the Project	134
6.2.6	Testing the Project.....	135
7	Designing a Simple Data Collection Sampler.....	148
7.1.1	The Arduino Microcontroller Board	149
7.1.2	Create Data Generator.....	149
7.1.3	Select I/O's for Fast Throughput on Arduino	150
7.1.4	Coding the Arduino Data Sampler.....	151
7.1.5	Building Arduino Project	154
7.1.6	Programming the Arduino.....	157
7.1.7	CPLD Active Transfer Coding and Initiation	160
7.1.8	CPLD: Define the User Design.....	160
7.1.9	CPLD: Compile/Synthesize the Project	175
7.1.10	CPLD: Program the CPLD.....	180
7.1.11	PC: Design the Project	182
7.1.12	PC: Coding the Project.....	183
7.1.13	PC: Compiling the Active Host Application.....	194
7.1.14	Connecting the Project Together.....	196
7.1.15	Testing the Project.....	199

1 Introduction and General Description

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



The user's Arduino code is developed to perform particular functions required by the user (such as reading a temperature sensor). The code is downloaded to the microcontroller using the Arduino IDE system provided as part of the microcontroller development system. The MegaProLogic USB-CPLD 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 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 MegaProLogic 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

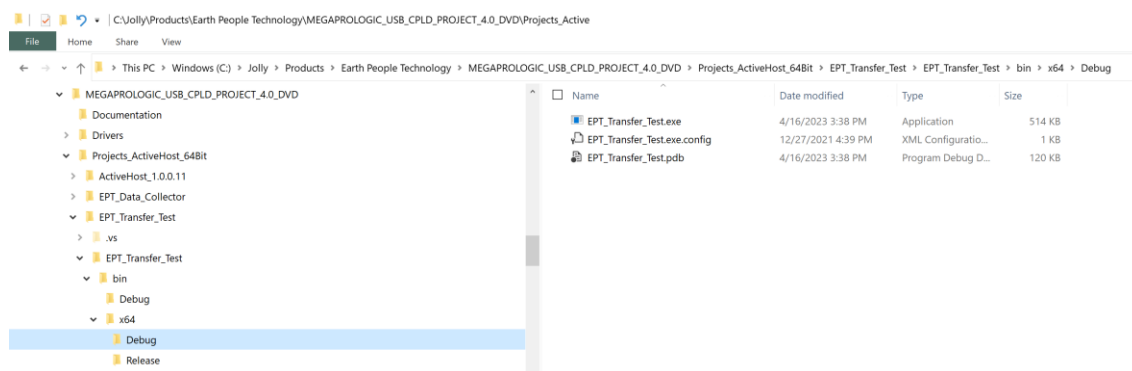


USB CPLD Development System User Manual

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

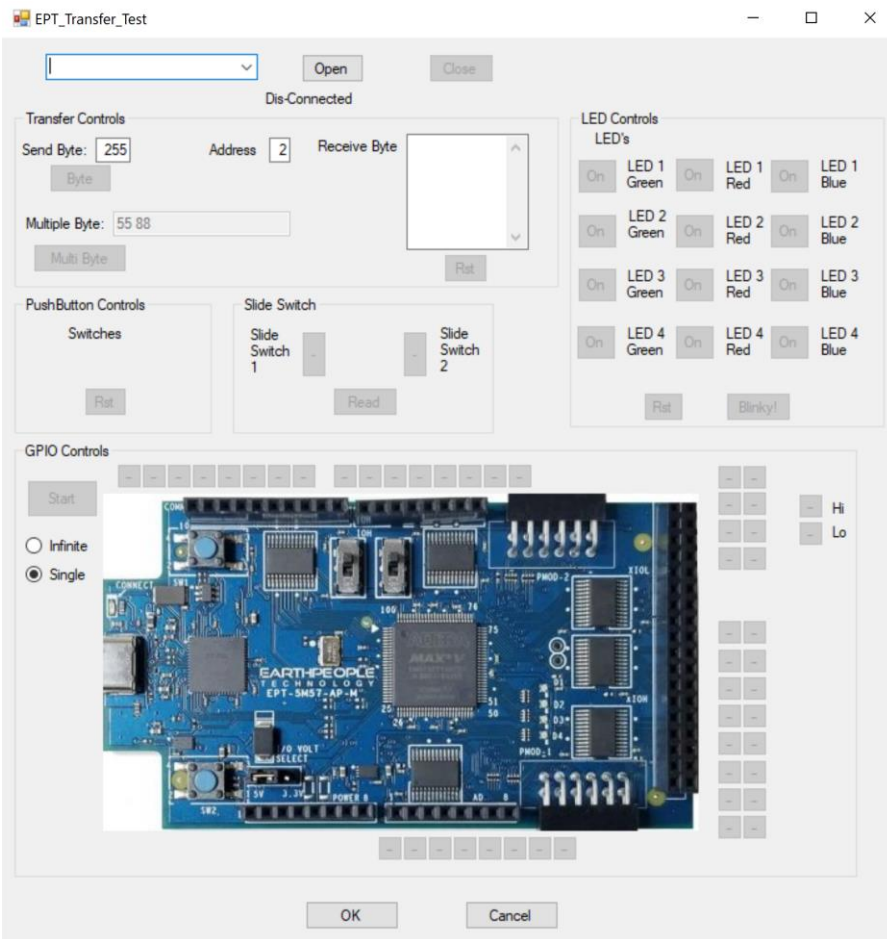
The MEGAPROLOGIC board comes pre-loaded with the EPT_Transfer_Test HDL project in the CPLD. This project allows the user to test out the functions of the Active Host API and the board hardware.



To test drive the application, connect the MegaProLogic to the Windows PC using Type A to USB-C cable. Load the driver for the board. See the section EPT Drivers for instructions on loading the MegaProLogic driver. If the USB driver fails to load, the Windows OS will indicate that no driver was loaded for the device.

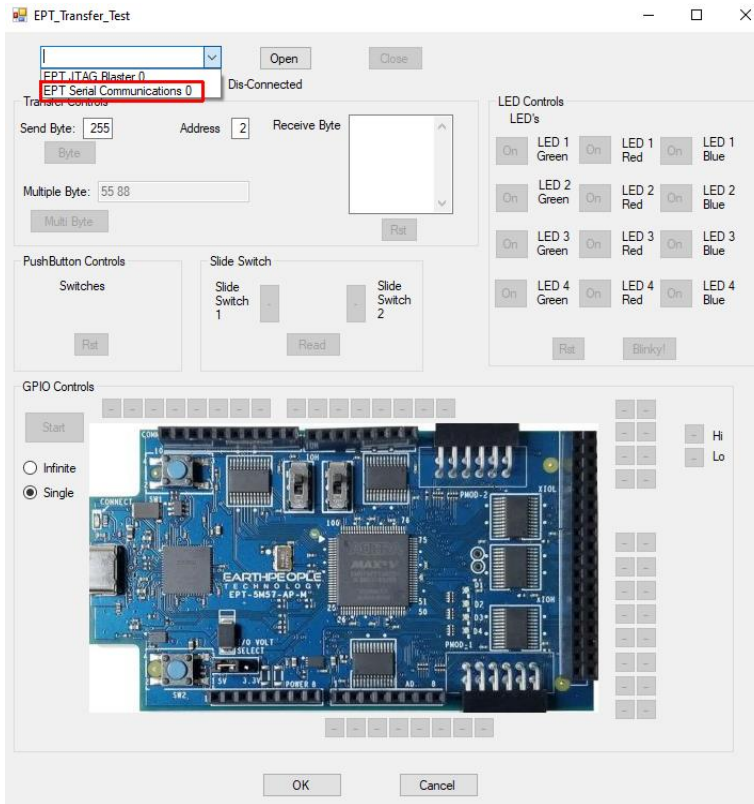
Next, open a Windows Explorer browser. Browse to the Projects_ActiveHost_64Bit\EPT_Transfer_Test\EPT_Transfer_Test\bin\X64\Debug\ folder on the MEGA_USB_CPLD_PROJECT_4.x_DVD. The application should load with a Windows form.

USB CPLD Development System User Manual



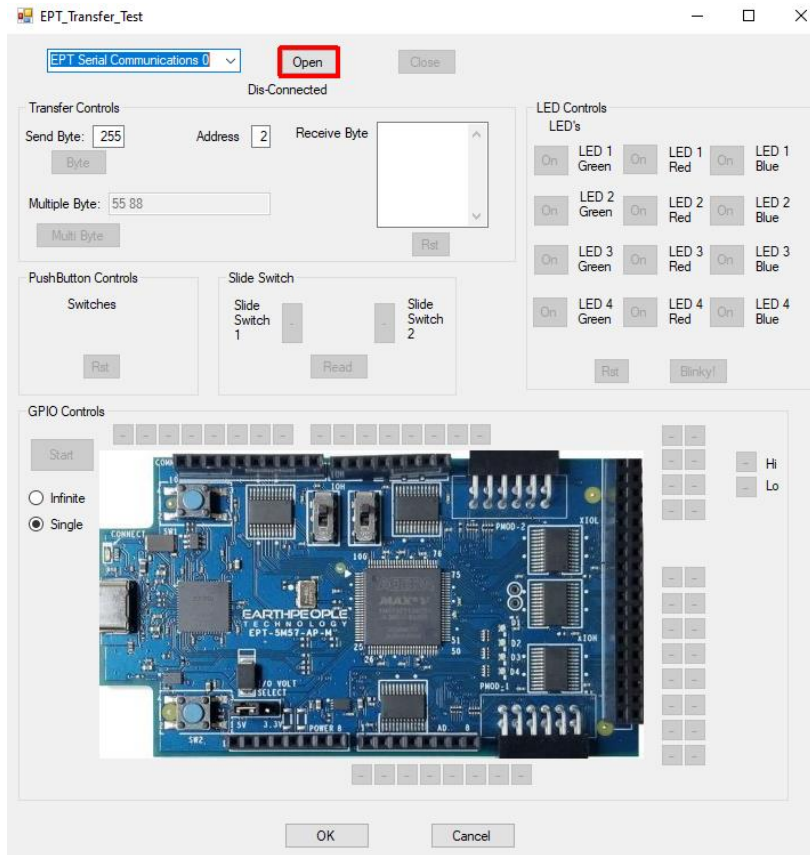
With the application loaded, select the EPT Serial Communications x board from the dropdown combo box.

USB CPLD Development System User Manual



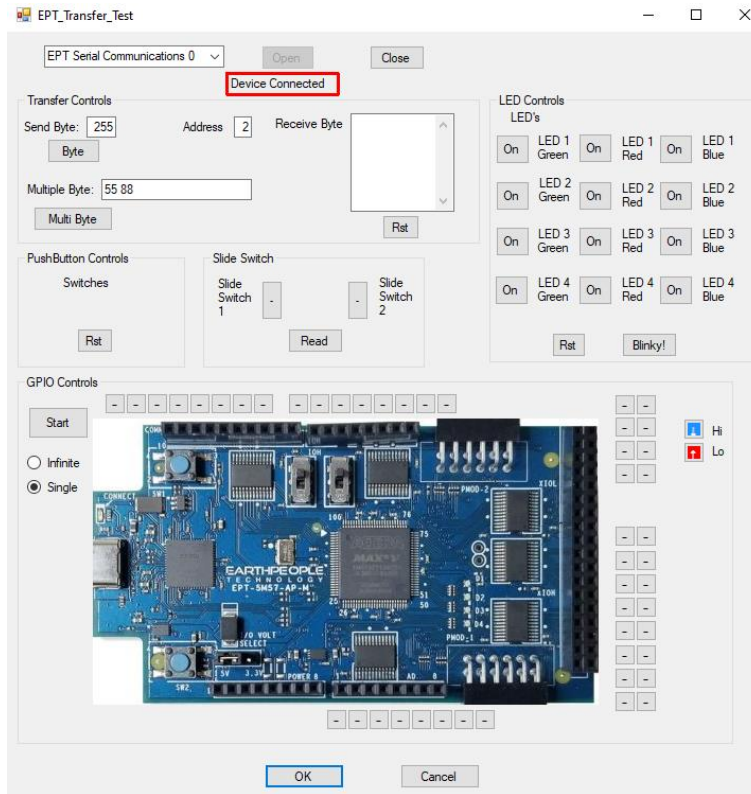
Next, click on the Open button.

USB CPLD Development System User Manual



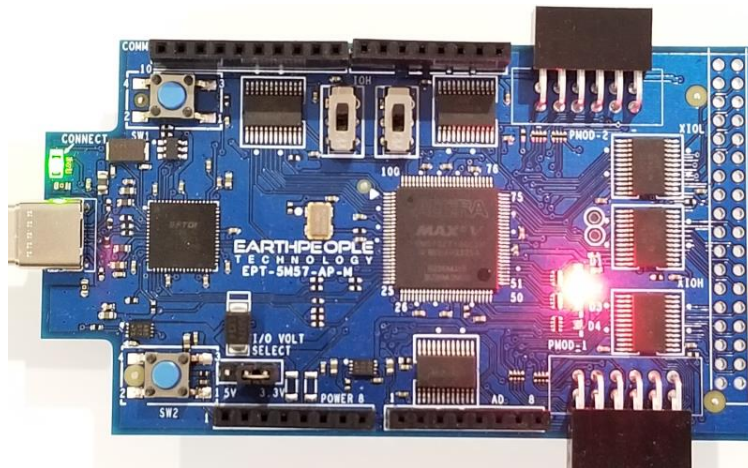
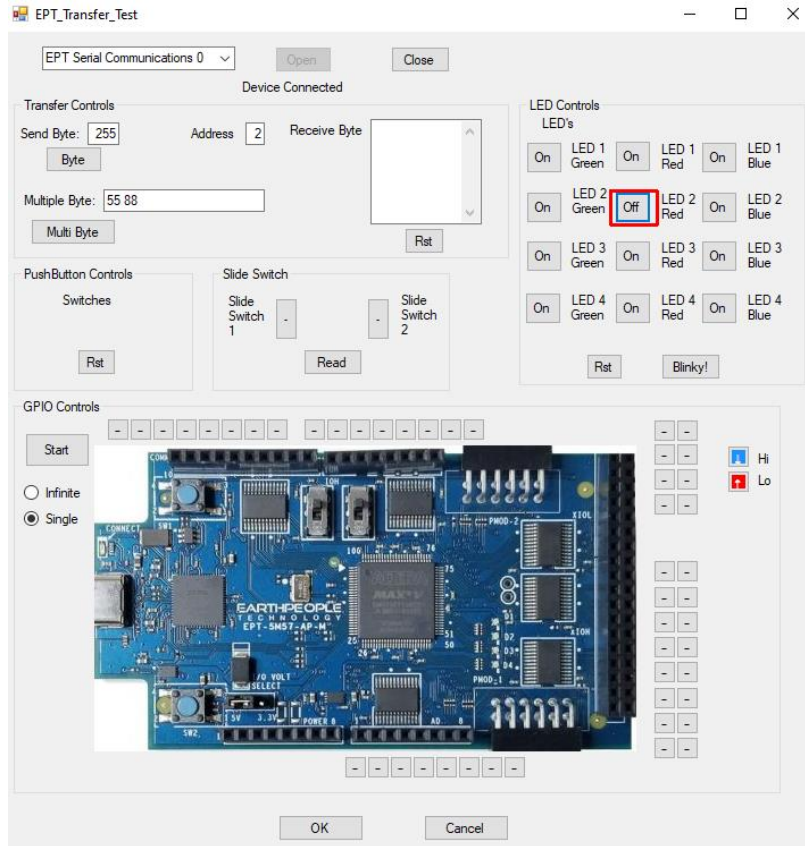
The “Device Connected” label should be seen to indicate the Windows App is now connected to the MegaProLogic.

USB CPLD Development System User Manual



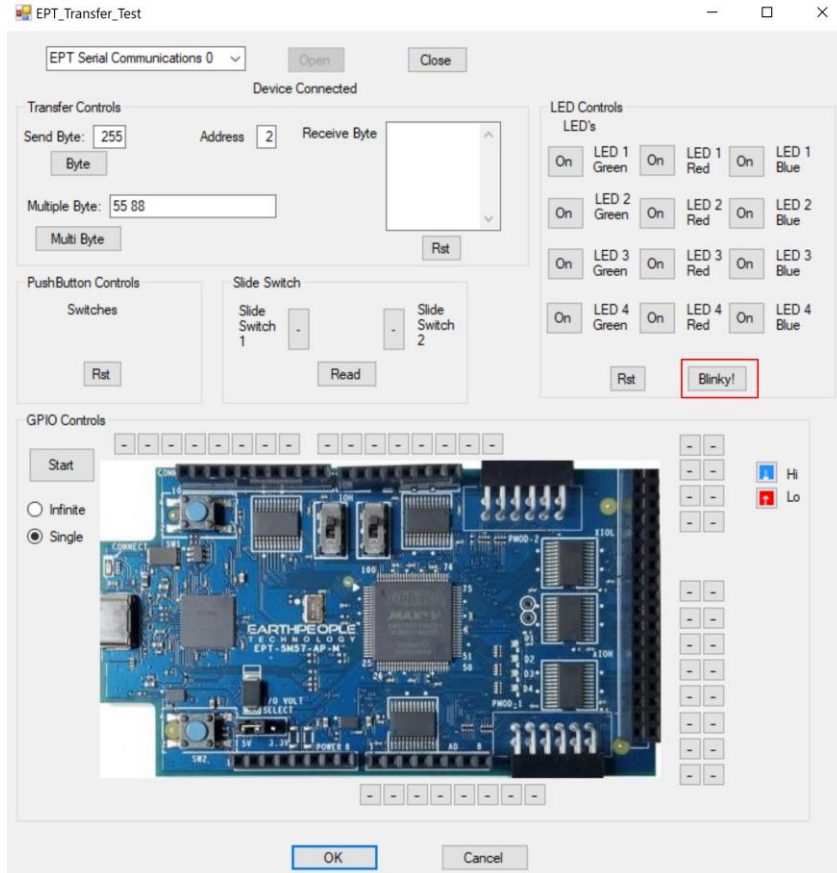
Click on one of the LED buttons in the middle of the window. The corresponding LED on the MegaProLogic board should light up.

USB CPLD Development System User Manual



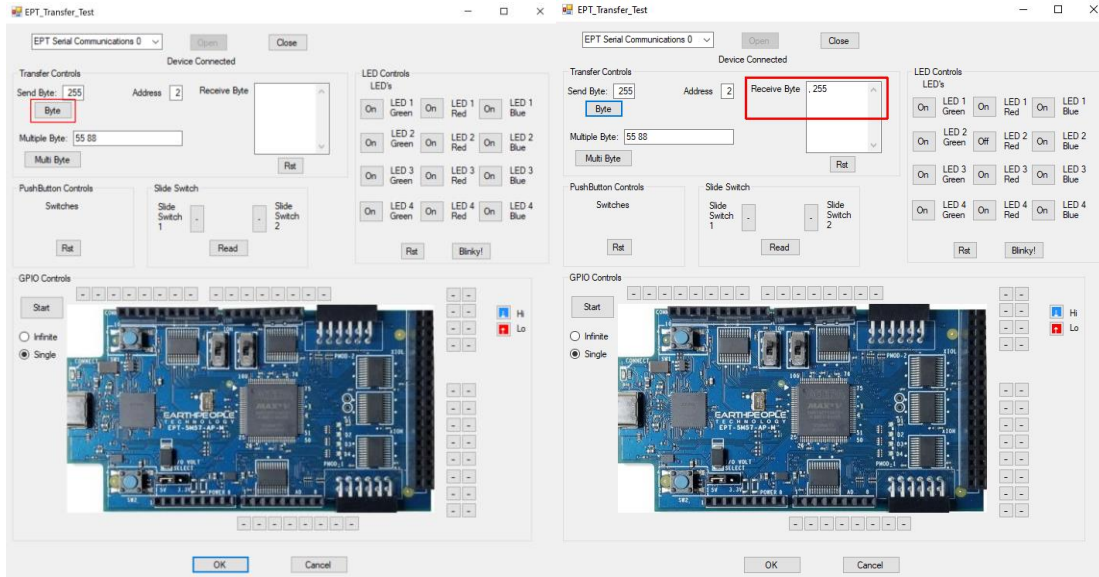
Click on the Blinky button for a light show.

USB CPLD Development System User Manual

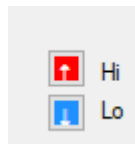


To exercise the Single Byte Transfer EndTerm, click the “Byte” button in the Transfer Controls group. Type in several numbers separated by a space and less 256 into the Multiple Byte textbox. Then hit the Multi Byte button. The numbers appear in the Receive Byte textbox.

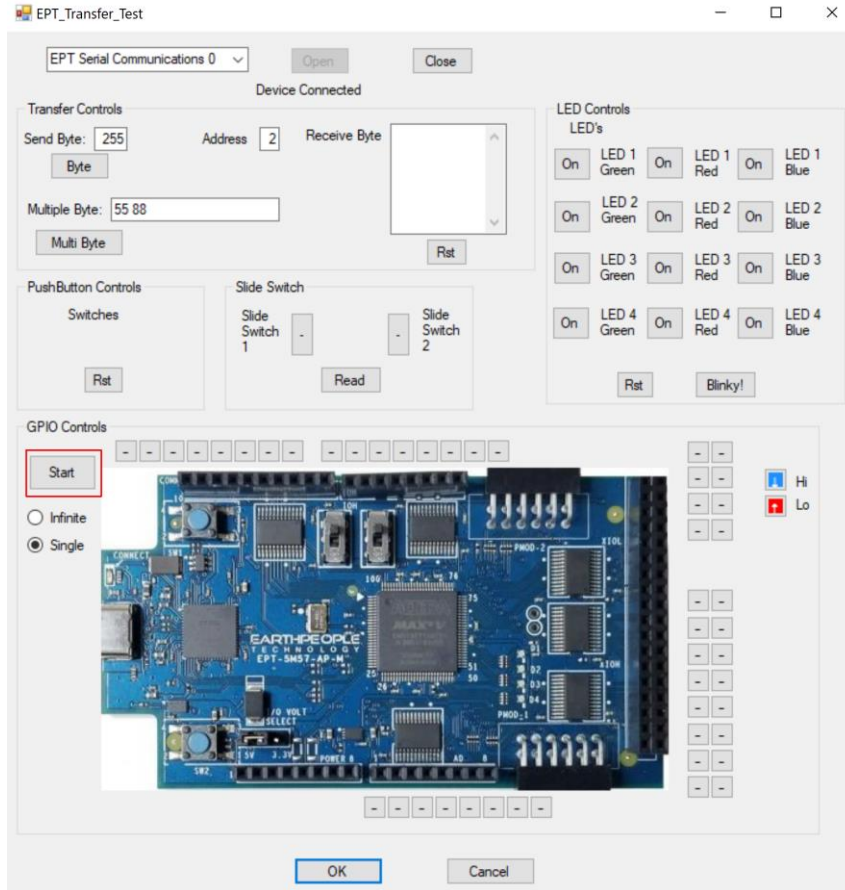
USB CPLD Development System User Manual



To exercise the Block Transfer EndTerm, click the “Start” button in the GPIO Controls group. The MegaProLogic will sample the state of each Input pin of the CPLD that is connected to a board edge connector. The Transfer Test Window will display the results of each pin, Hi or Lo

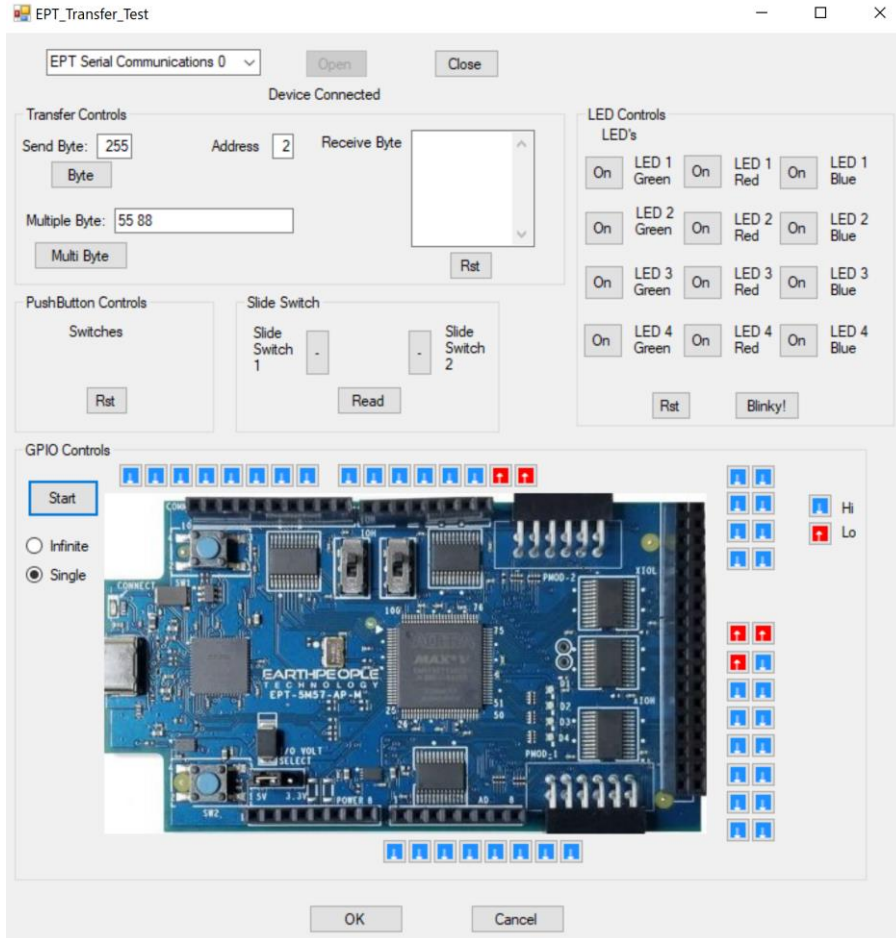


USB CPLD Development System User Manual



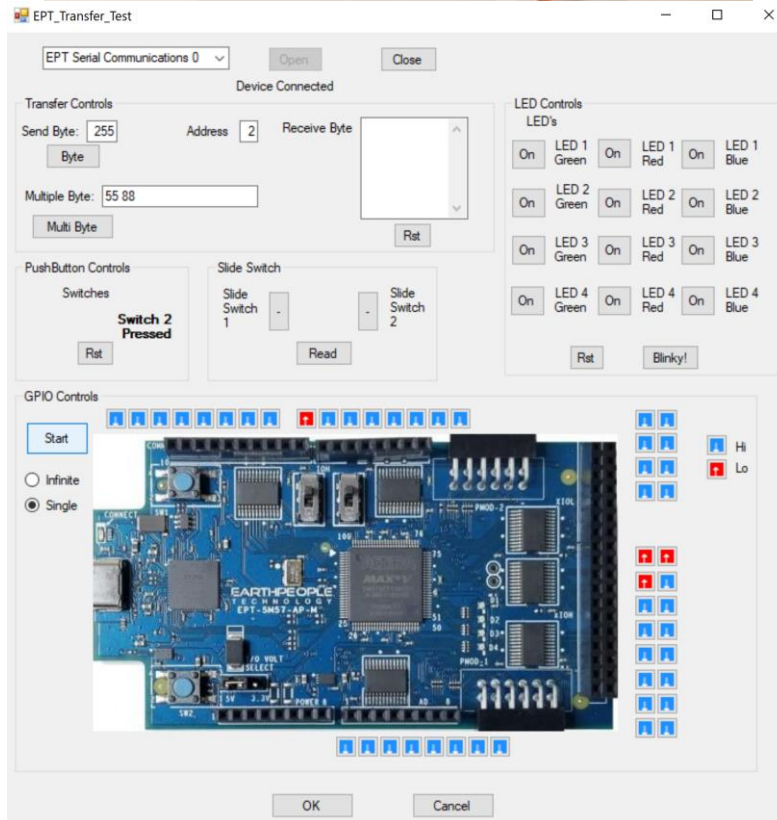
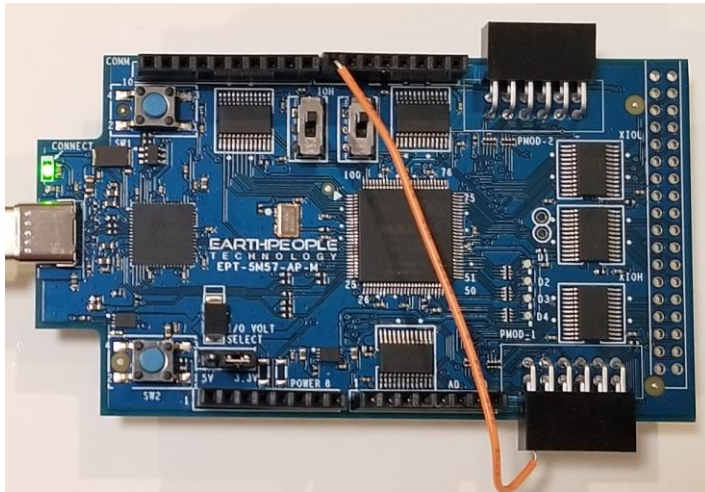
The results of each pin are displayed next to the image of the MegaProLogic in separate buttons.

USB CPLD Development System User Manual



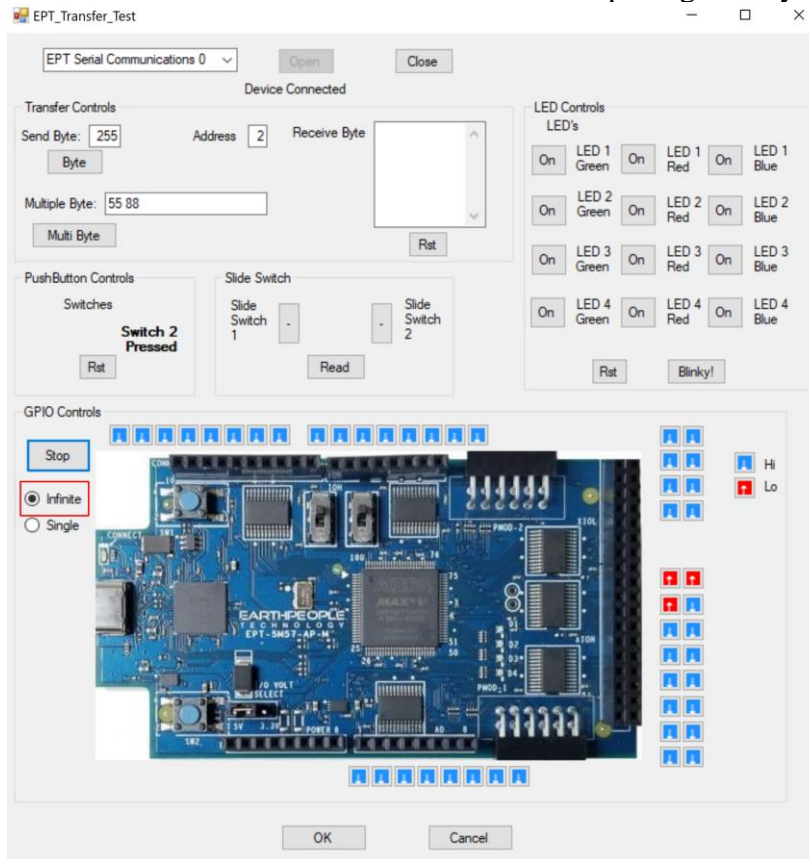
Use a jumper cable to connect between +3.3V and any of the Input pins to see the display detect a high on the selected pin.

USB CPLD Development System User Manual



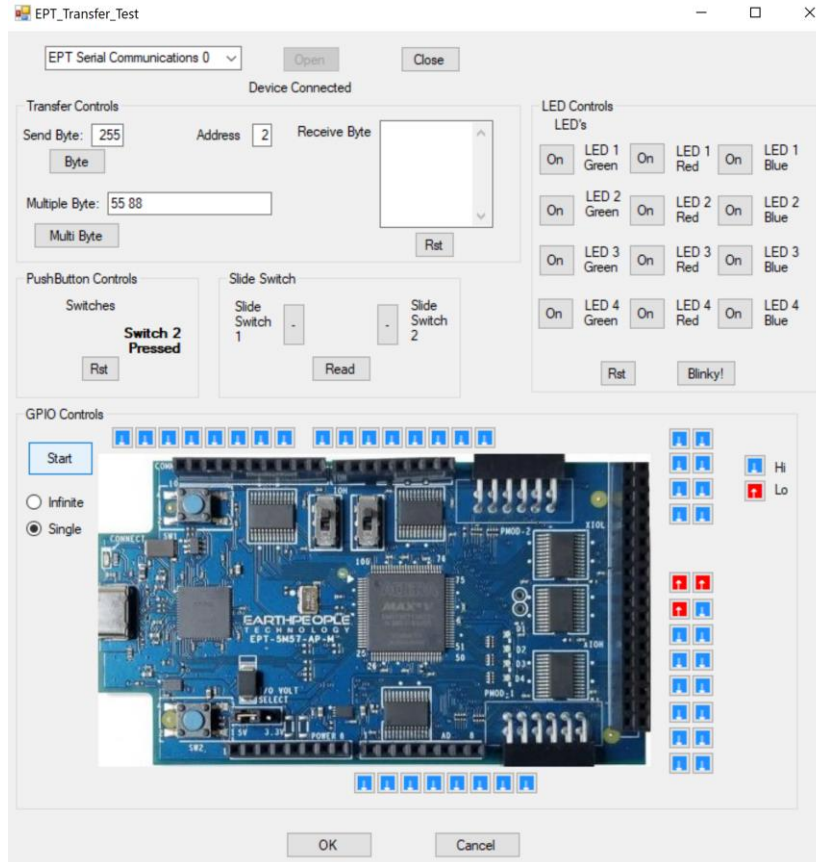
USB CPLD Development System User Manual

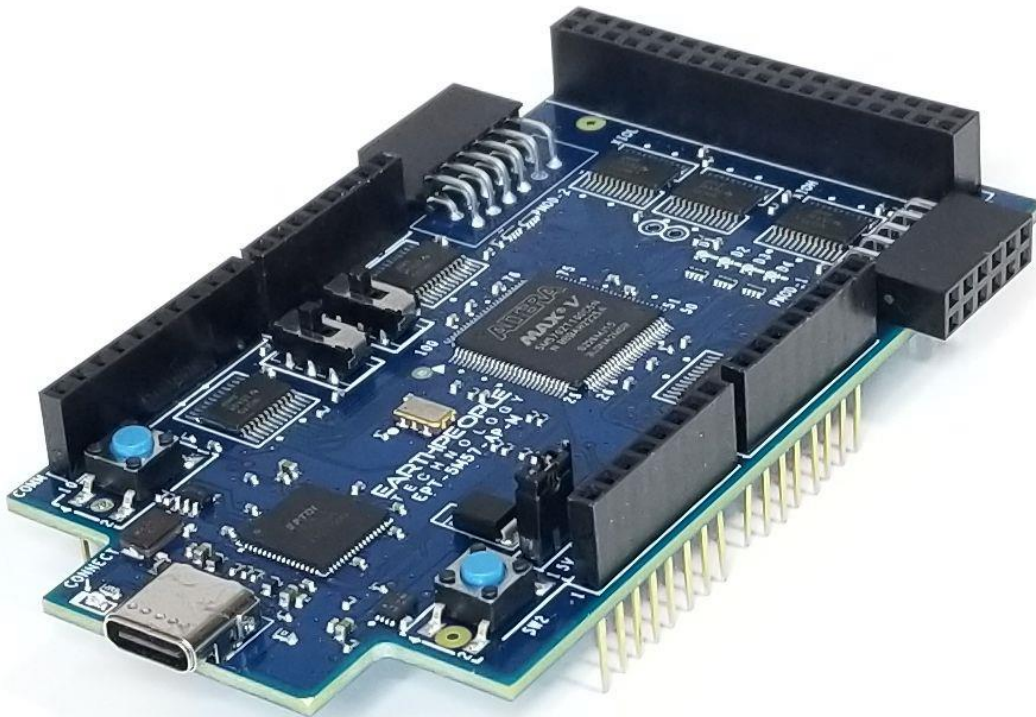
Click on the “Infinite” radio button and “Start” to see a simple logic analyzer.



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

USB CPLD Development System User Manual





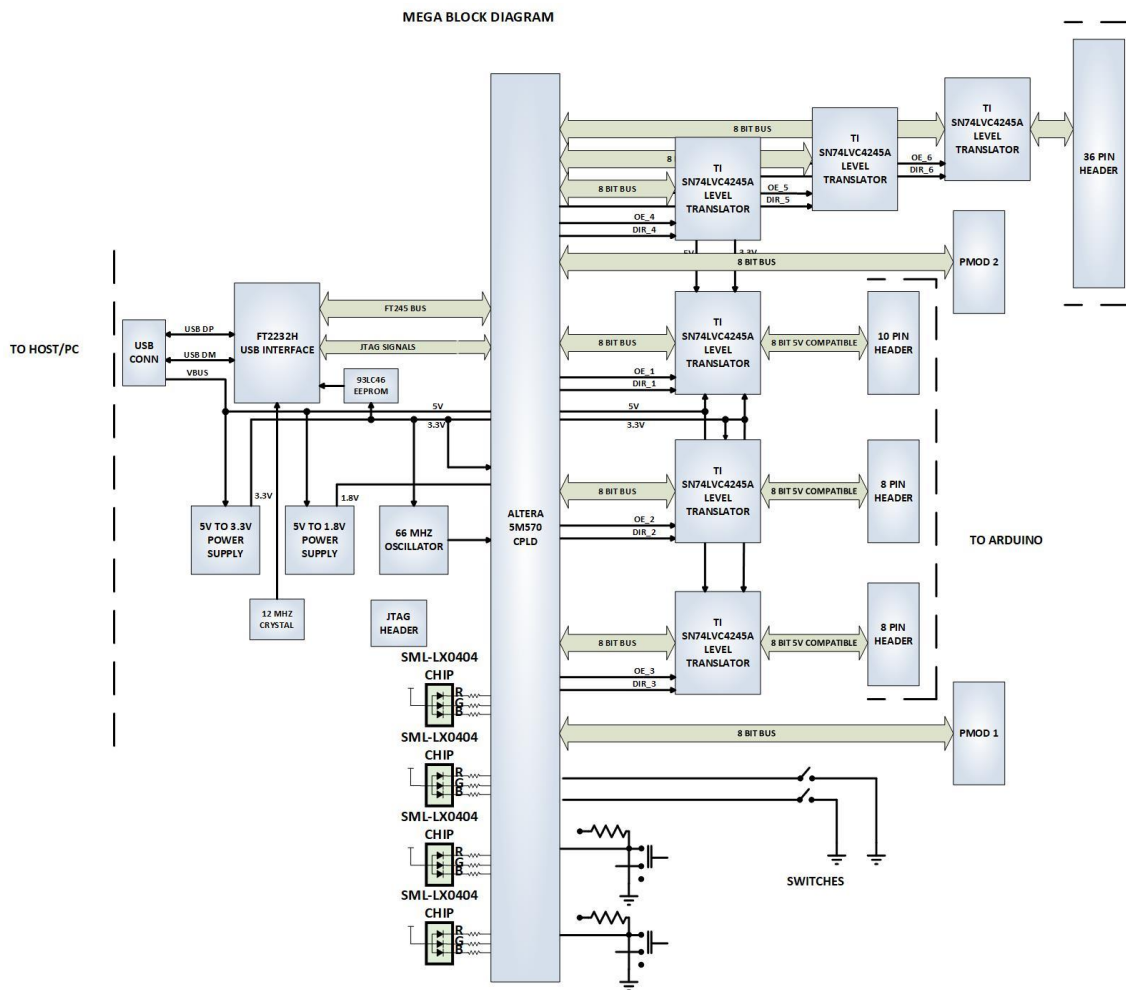
1.2 MEGAPROLOGIC

The MegaProLogic board is equipped with an Altera 5M570 CPLD; which is programmed using the Quartus Prime Lite 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 0.1 Mega Bytes per second. Thirty Two I/O's from the CPLD are attached to five 8 bit transceivers to provide 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 RGB LED's, two Slide Switches, two PMOD Connectors and two Push Buttons that are controllable by the user code. The hardware features are as follows.

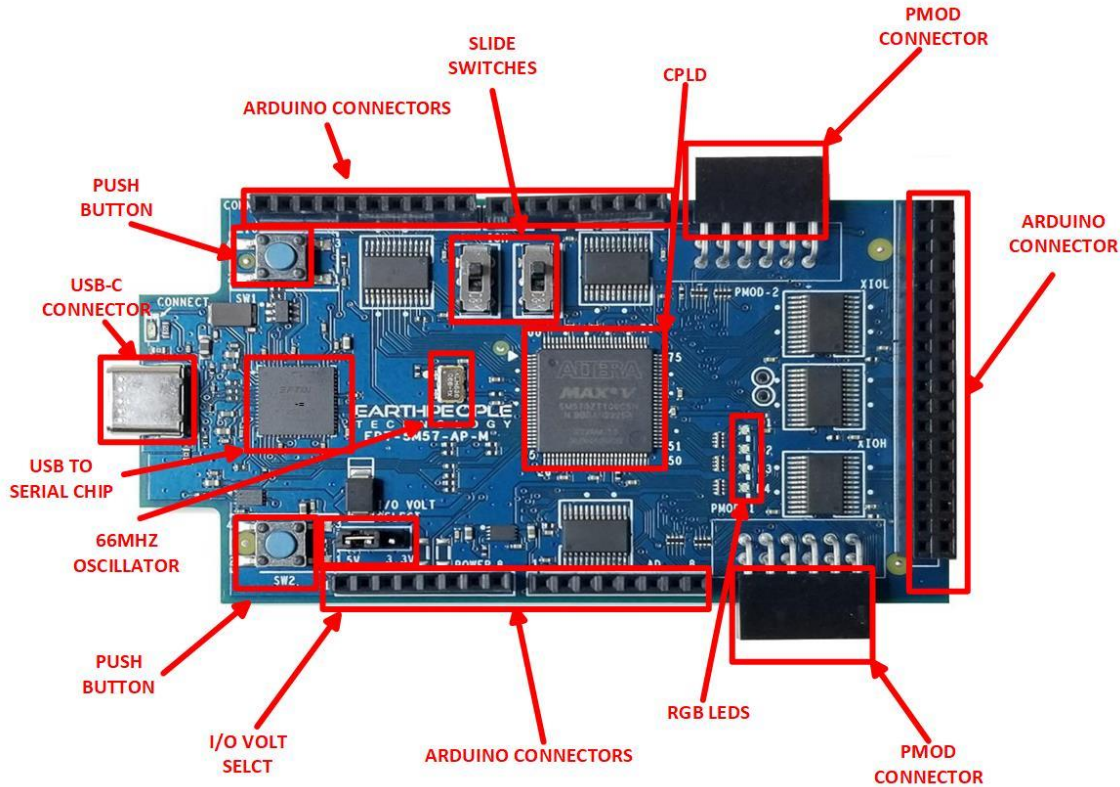
- Intel 5M570 in the TQFP 100 pin package
- FT2232H USB to Serial Interface chip
- 66 MHz oscillator for driving USB data transfers and users code
- Five 74LVC245 bidirectional voltage translator/bus transceiver
- 32 user Input/Outputs

USB CPLD Development System User Manual

- Four RGB LED's accessible by the user
- Two PCB switches accessible by the user
- Two Slide Switches
- Two PMOD Connectors
- All connectors to stack into the Arduino 2560 Mega



USB CPLD Development System User Manual

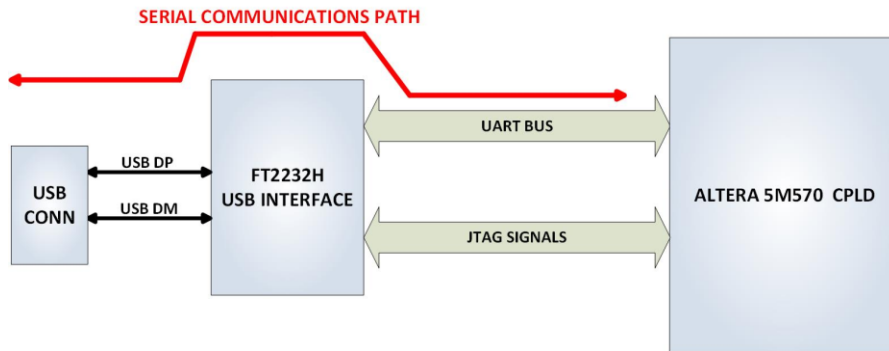


1.2.1 Serial USB Communications

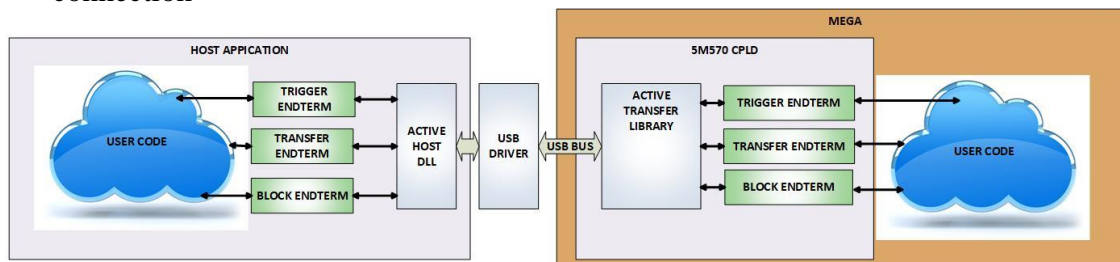
The MegaProLogic USB-CPLD Development system connects an FT2232H Dual High Speed USB (480 Mbits/sec) chip to the CPLD. The CPLD uses a dedicated channel on the FT2232H for high speed transfers to the PC. Using the EPT Active Transfer Library, sustained speeds of 0.1 Mbytes/sec can be achieved. The transfers 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 a single COM Port. 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.

USB CPLD Development System User Manual



The Serial Communications with the MegaProLogic is handled easily using the Active Host API. 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:

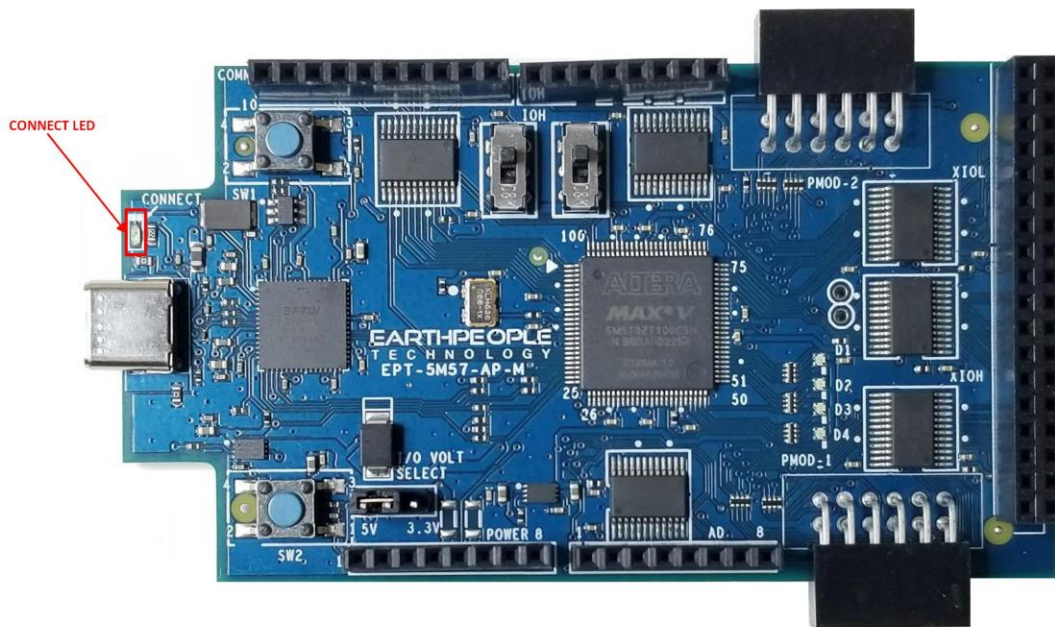
- Trigger Endterm
- Transfer Endterm
- Block Endterm

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.

1.2.2 Host PC Connection

The MegaProLogic includes an LED that signifies the connection of the board with the Host PC. The connect LED has the word “CONNECT” in silkscreen next to the LED. This LED will only light up once the Host PC has correctly enumerated the USB device (FT2232HQ chip). When this LED is lit up it can tell the user three things:

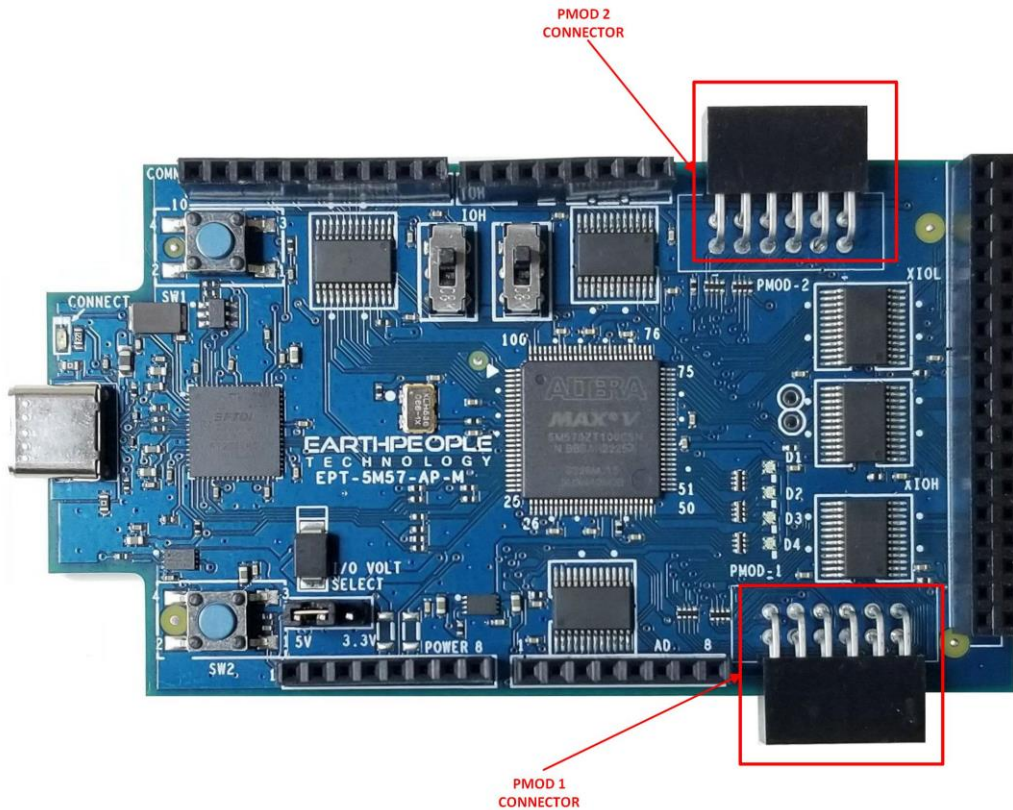
- Power has been applied to the MegaProLogic via USB
- The FT2232HQ chip is working properly
- The Host PC has found the appropriate driver and will communicate with the MegaProLogic



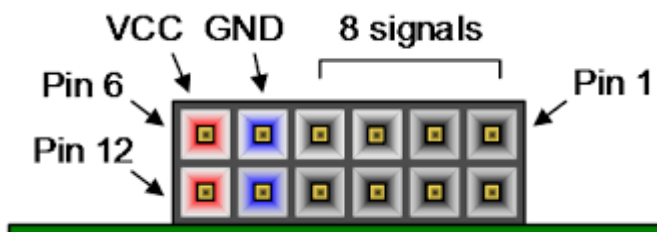
1.2.3 PMOD Connectors

The MegaProLogic includes two PMOD Connectors. These two connectors are located towards the rear of the board.

USB CPLD Development System User Manual

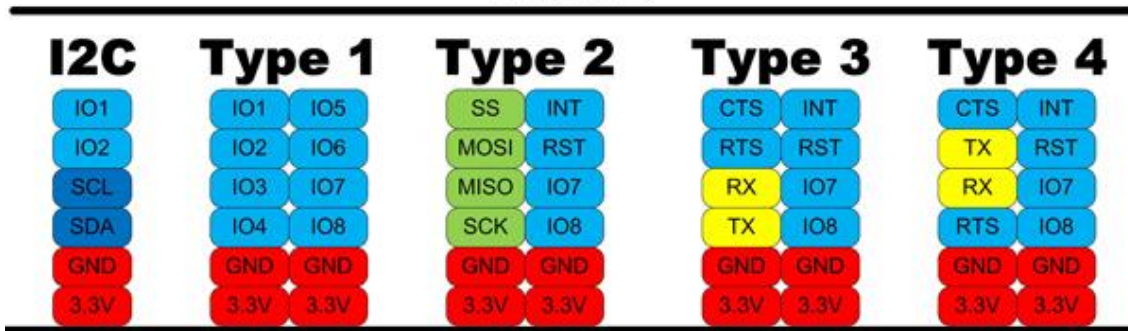


The PMOD pinouts follows the standard pinout. Pin 1 is located in the upper right when facing the connector.



VCC is +3.3V and the Inputs and Outputs of the 8 signals are +3.3V only. The eight I/O's are connected directly to FPGA pins and can be designated as any communications standard.

PMOD



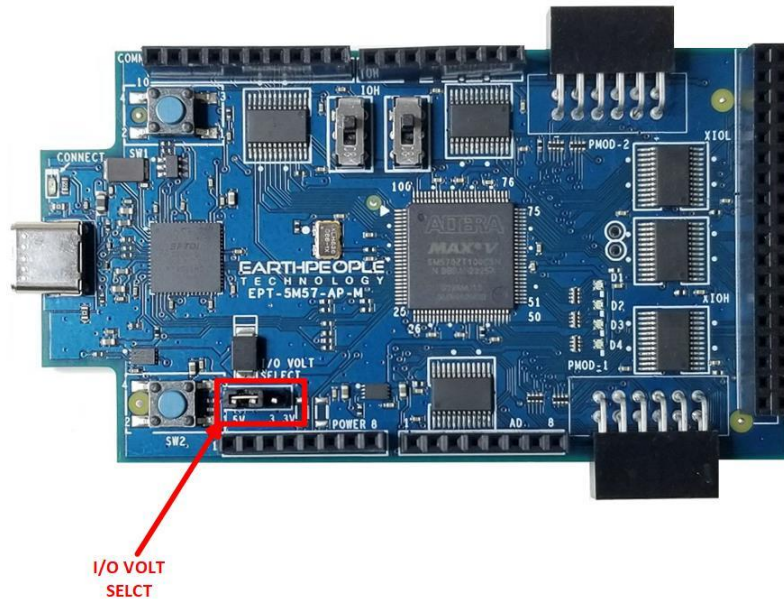
The PMOD have the following connections to the MAX V chip:

PMOD Pin Number	Signal Name	MAX V Pin Number
1-1	LB31	69
1-2	LB29	67
1-3	LB27	64
1-4	LB25	61
1-7	LB30	68
1-8	LB28	66
1-9	LB26	62
1-10	LB24	58
2-1	LB39	84
2-2	LB37	82
2-3	LB35	78
2-4	LB33	76
2-7	LB38	83
2-8	LB36	81
2-9	LB34	77
2-10	LB32	75

1.2.4 Inputs and Outputs

There are 32 Inputs/Outputs which are selectable between +3.3V and +5 Volt. JMP1 is used to select which voltage the 32 Inputs/Outputs are set to.

USB CPLD Development System User Manual

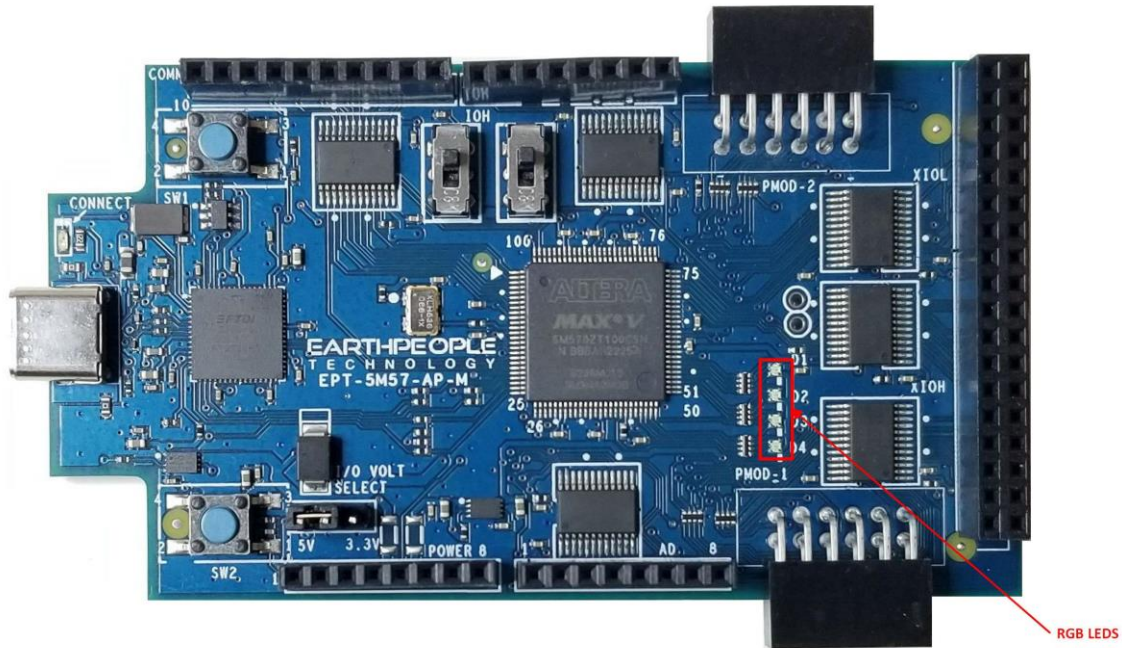


The I/O's are organized as three 8 bit directional ports. Each port must be defined as input or output. This means that all 8 bits of a port will point in the same direction, depending on the direction bit of the transceiver. The direction bit can be changed at any time, so that a port can change from 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”.

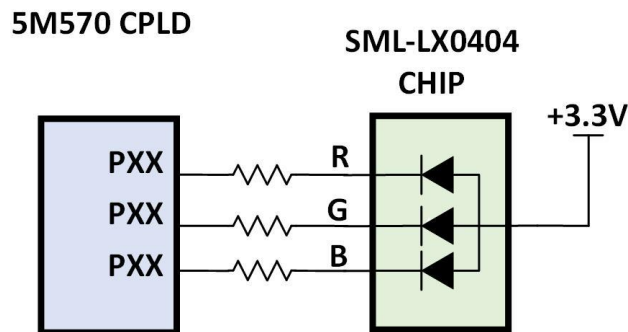
1.2.5 User LEDs

The User LEDs are four RGB LEDs. These LEDs are SML-LX0404 chips and are for use only with +3.3V.

USB CPLD Development System User Manual



The SML-LX0404 chip is a current sink and are connected to pins on the MAX V CPLD. The anode is connected to +3.3V. The series resistors are calculated for current limiting based on +3.3V.



Each series resistor uses a 220 Ohm in a resistor array. In order to light up the each LED, the user code must assert a zero on the associated signal for the LED. To turn off the LED, assert High Z on the signal.

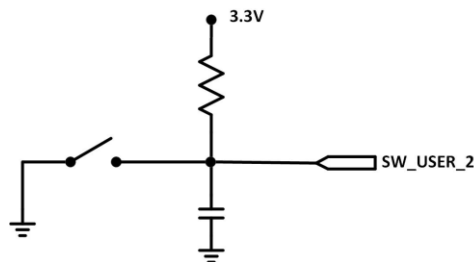
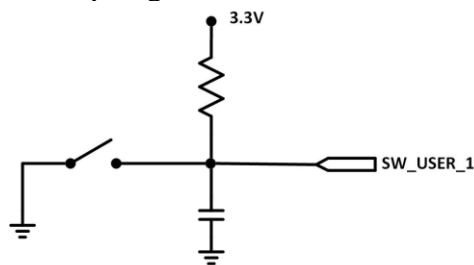
USB CPLD Development System User Manual

The LED RGB signals are organized on the following pins from the MAX V chip:

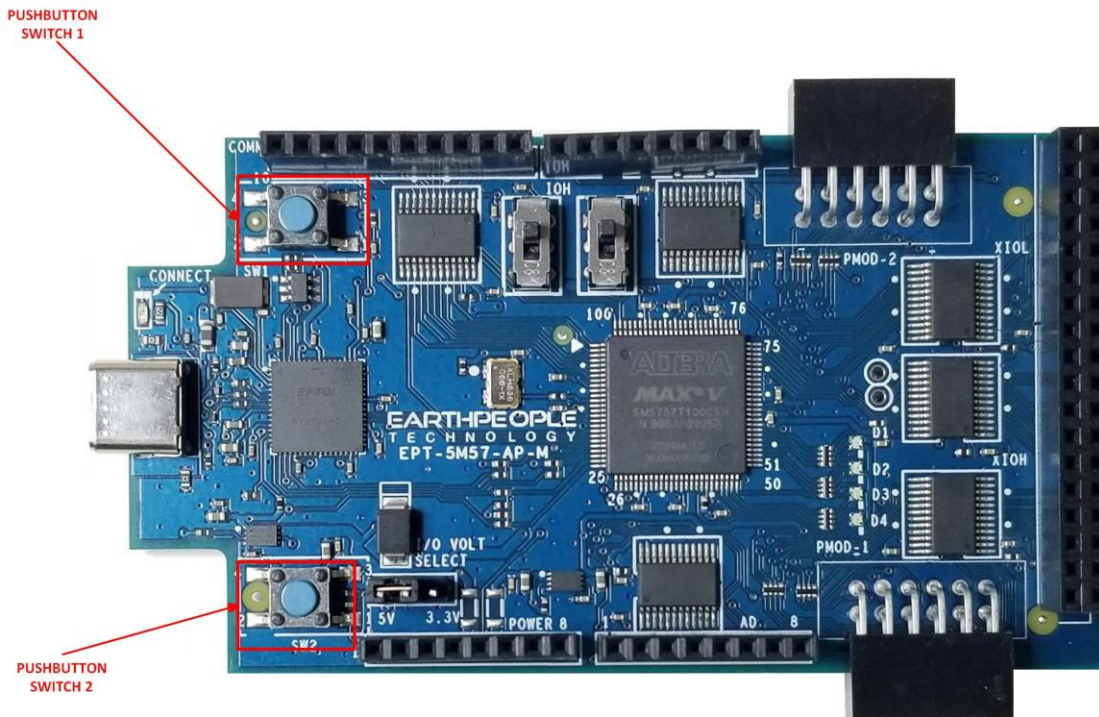
LED Number	Signal Name	MAX V Pin Number
D1	LED_GREEN_1_N	57
D1	LED_BLUE_1_N	56
D1	LED_RED_1_N	55
D2	LED_GREEN_2_N	54
D2	LED_BLUE_2_N	53
D2	LED_RED_2_N	52
D3	LED_GREEN_3_N	51
D3	LED_BLUE_3_N	50
D3	LED_RED_3_N	49
D4	LED_GREEN_4_N	48
D4	LED_BLUE_4_N	47
D4	LED_RED_4_N	43

1.2.6 User Push Buttons

The MegaProLogic includes two push button switches. Both are momentary contact switches. They include a 1uF cap to ground to debounce both switches.



USB CPLD Development System User Manual

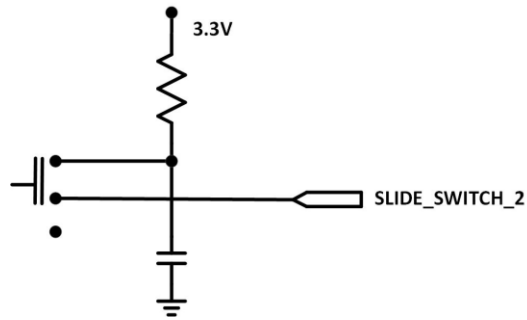
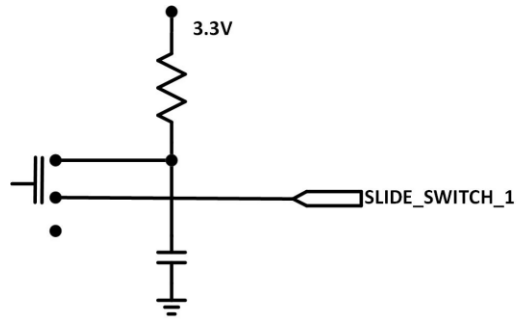


Component	Net Name	Pin on CPLD	Signal in EPT Project Pinout
SW1	SW_USER_1	20	SW_USER_1
SW2	SW_USER_2	21	SW_USER_2

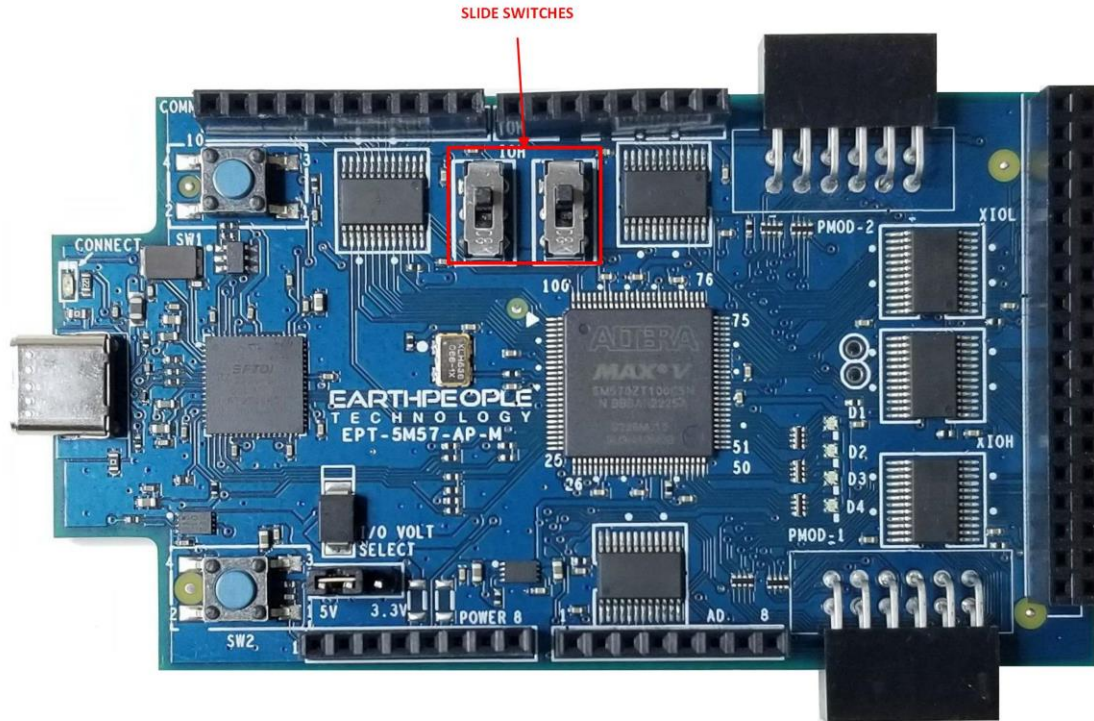
1.2.7 Slide Switches

The MegaProLogic includes two slide switches. Both are full contact switches. They include a 1uF cap to ground to debounce both switches.

USB CPLD Development System User Manual



USB CPLD Development System User Manual

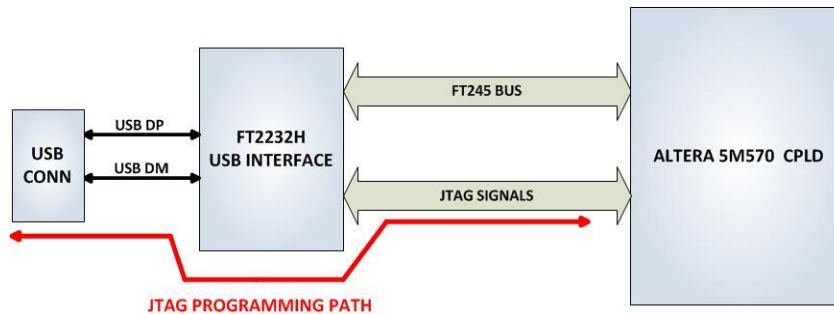


Component	Net Name	Pin on CPLD	Signal in EPT Project Pinout
SW3	SLIDE_SWITCH_1	16	SLIDE_SWITCH_1
SW4	SLIDE_SWITCH_2	17	SLIDE_SWITCH_2

1.2.8 JTAG

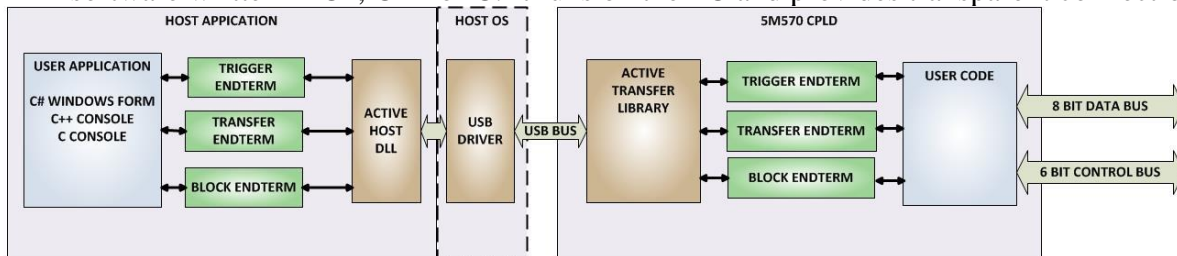
The MegaProLogic uses the second channel of the FT232H chip as a dedicated CPLD programming port. The CPLD must be programmed via JTAG signals and the FT232H has built in JTAG signals. The CPLD can be programmed directly from Quartus Prime Lite by using the “jtag_hw_mbfldi_blaster.dll”. Just click on the Programmer button and select the EPT-Blaster.

USB CPLD Development System User Manual



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

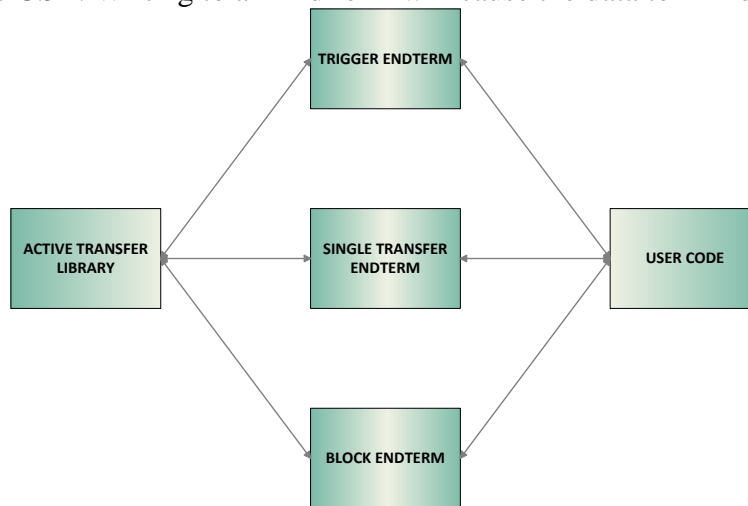
registered in the users application. This callback function provides a mechanism to transparently receive data from the CPLD. The user application does not need to schedule a read from the USB or call any blocking threads.

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
- 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 MegaProLogic Development system requires drivers for any interaction between PC and the board. The communication between the two consists of programming the CPLD and data transfer. In both cases, the USB Driver is required. This will allow



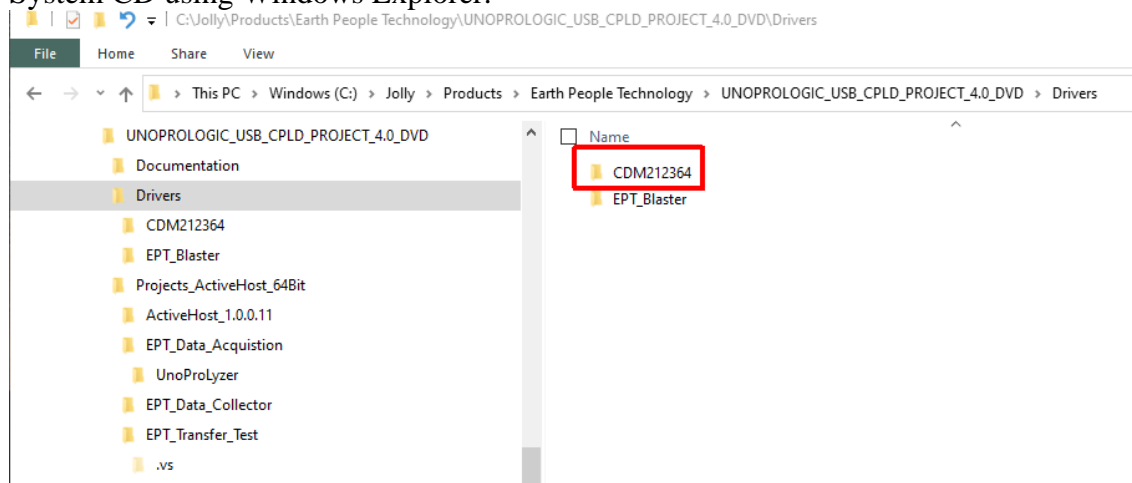
USB CPLD Development System User Manual

Windows to recognize the USB Chip and setup a pathway for Windows to communicate with the USB hardware.

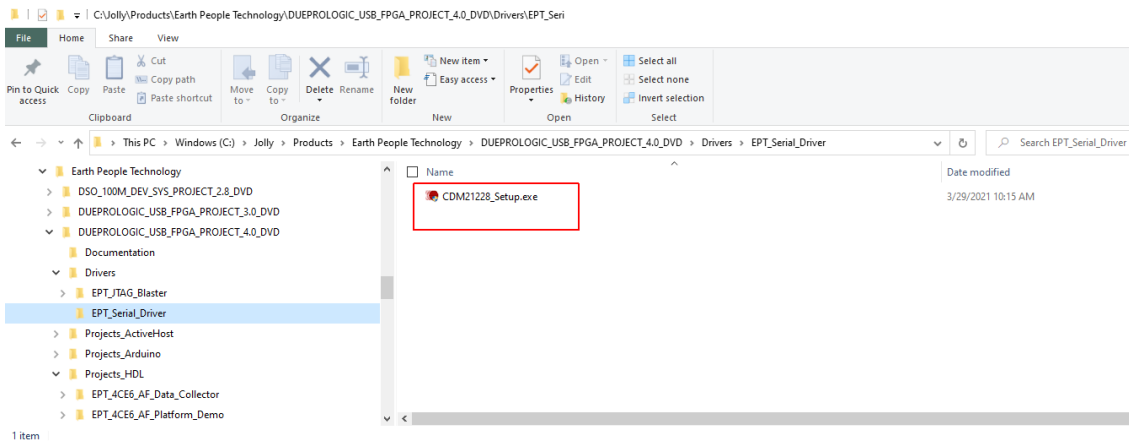
2.1 USB Driver

The MegaProLogic 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 CDM212xxx Folder. The installation of the FTDI 2.12.28 driver is easily accomplished by double clicking the CDM21228_Setup.exe.

Locate the CDM212xxx folder in the Drivers folder of the MegaProLogic Development System CD using Windows Explorer.



USB CPLD Development System User Manual



Double click on the *.exe file and select the default settings when the software tool queries the user.

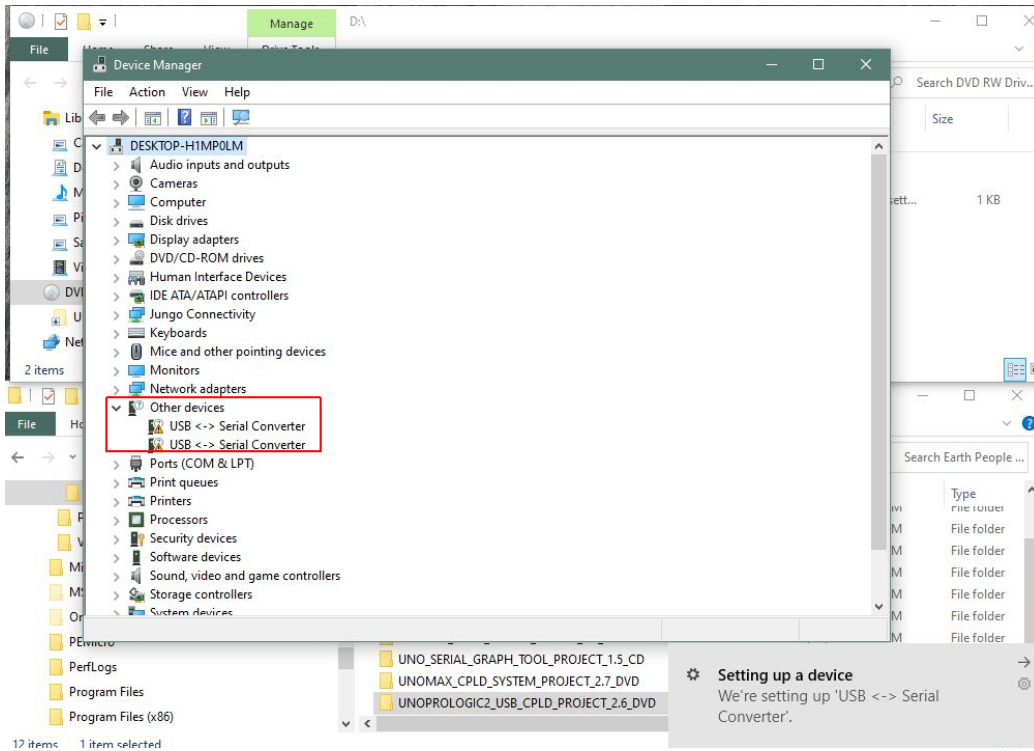
Plug in the MegaProLogic device into an available USB port.



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.

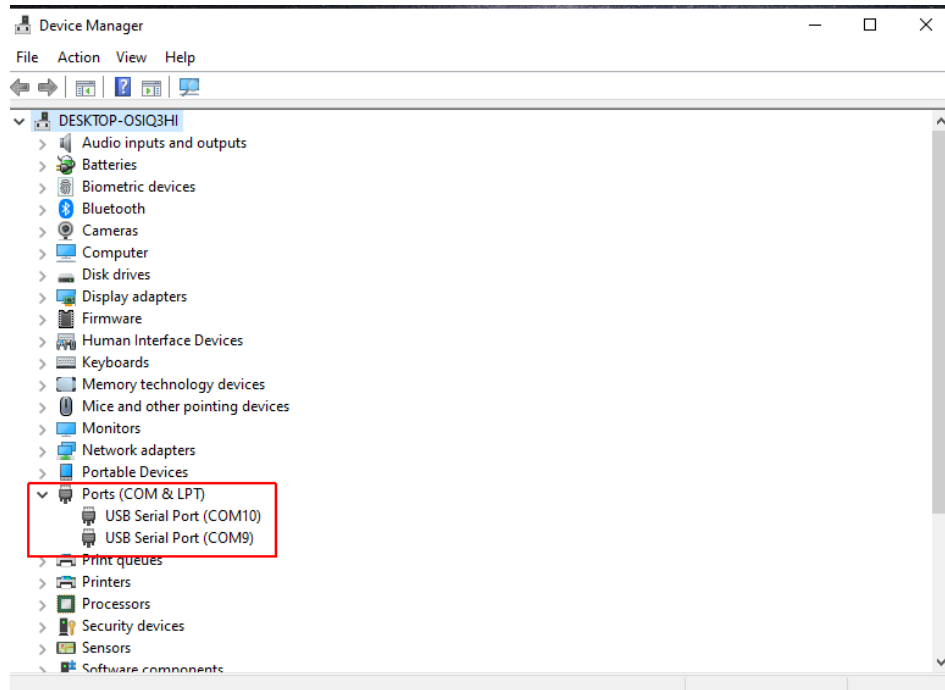
If Windows cannot load a driver for the MegaProLogic, a notification window will inform the user that the driver load has failed for the device.

USB CPLD Development System User Manual



If the driver is successfully installed, Windows will inform the user. The user can check Device Manager to ensure the correct driver was installed for the MegaProLogic. The MegaProLogic will show up as two COM Ports under the “Ports (COM & LPT)” under the Device Manager.

USB CPLD Development System User Manual



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

2.2 JTAG DLL Insert to Quartus Prime Lite

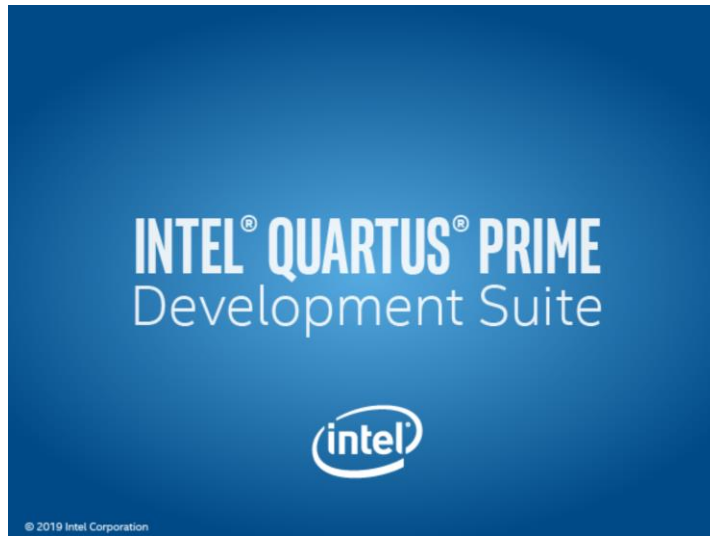
The JTAG DLL Insert to Quartus Prime Lite allows the Programmer Tool under Quartus to recognize the MegaProLogic. The MegaProLogic can then be selected and perform programming of the CPLD. The file, `jtag_hw_mbftdi_blaster.dll` must be placed into the folder that hosts the `jtag_server` for Quartus.

2.2.1 Installing Quartus

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 `QuartusLiteSetup-xxx.xxx.xxx-windows.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.

USB CPLD Development System User Manual



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

Next, skip the “Download Quartus” section. Go down to the “Quartus Installer” section to complete the Quartus installation.



USB CPLD Development System User Manual

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:

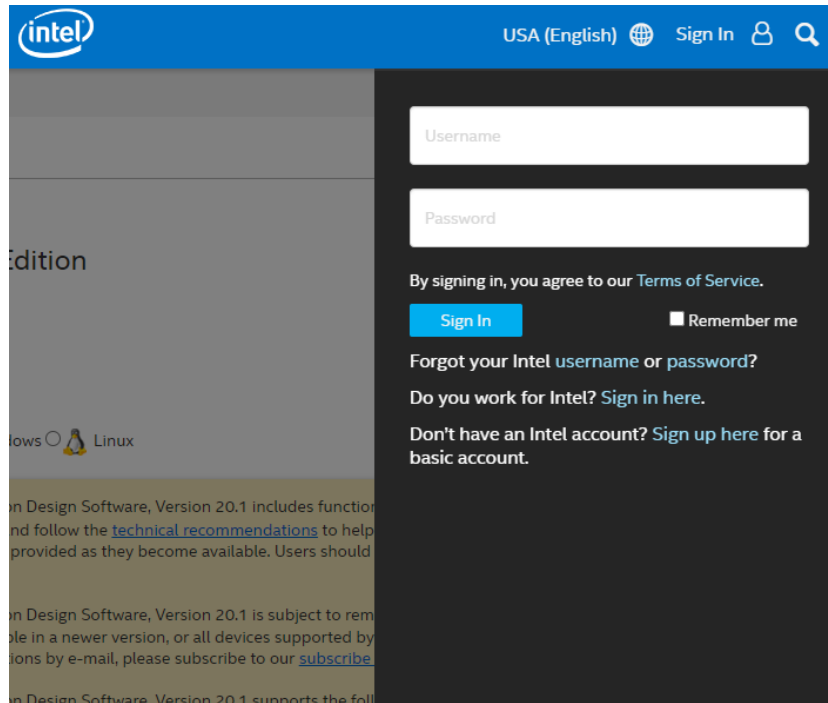
[Intel FPGA Quartus Prime Lite](#)

You will first need to apply for an account with Intel. Then use your login and password to access the download site. Click on the Download Windows Version.

The screenshot shows the Intel website's download center for FPGAs. At the top, there is a blue navigation bar with "Products", "Solutions", and "Support" on the left, the Intel logo in the center, and "USA (English)", a globe icon, a "Sign In" button with a user icon, and a search icon on the right. Below the navigation bar, the page title is "Download Center for FPGAs". On the left side, there is a vertical menu with buttons for "Design Software", "Embedded Software", "Archives", "Licensing", "Programming Software", "Drivers", "Board System Design", "Board Layout and Test", and "Legacy Software". The main content area is titled "Quartus Prime Lite Edition" and includes the following information: "Release date: June, 2020", "Latest Release: v20.1", "Select edition: Lite" (with a dropdown arrow), and "Select release: 20.1" (with a dropdown arrow). Below this, it says "Operating System" with icons for Windows and Linux. A yellow callout box contains two green checkmarks and text: "The Quartus Prime Lite Edition Design Software, Version 20.1 includes functional and security updates. Users should keep their software up-to-date and follow the technical recommendations to help improve security. Additional security updates are planned and will be provided as they become available. Users should promptly install the latest version upon release." and "The Quartus Prime Lite Edition Design Software, Version 20.1 is subject to removal from the web when support for all devices in this release are available in a newer version, or all devices supported by this version are obsolete. If you would like to receive customer notifications by e-mail, please subscribe to our subscribe to our customer notification mailing list."

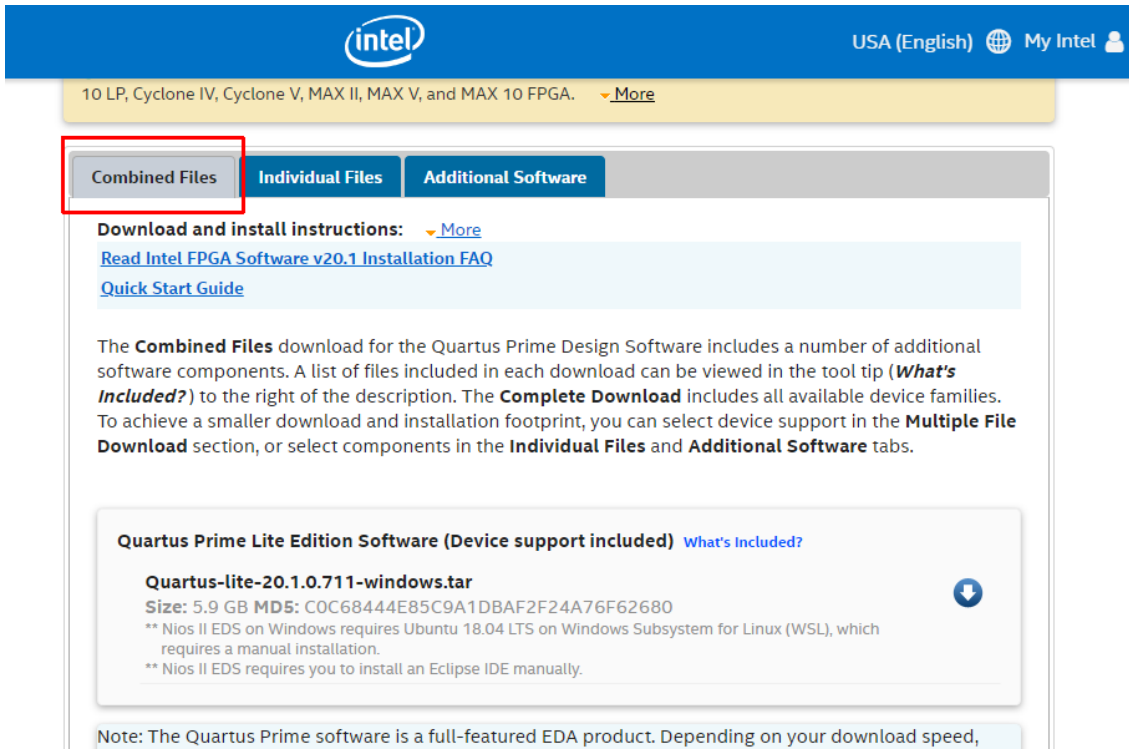
The next page will require you to sign into your “myAltera” account. If you do not have one, follow the directions under the box, “Don’t have an account?”

USB CPLD Development System User Manual



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.

USB CPLD Development System User Manual




10 LP, Cyclone IV, Cyclone V, MAX II, MAX V, and MAX 10 FPGA. [More](#)

Combined Files Individual Files Additional Software

Download and install instructions: [More](#)
[Read Intel FPGA Software v20.1 Installation FAQ](#)
[Quick Start Guide](#)

The **Combined Files** download for the Quartus Prime Design Software includes a number of additional software components. A list of files included in each download can be viewed in the tool tip (**What's Included?**) to the right of the description. The **Complete Download** includes all available device families. To achieve a smaller download and installation footprint, you can select device support in the **Multiple File Download** section, or select components in the **Individual Files** and **Additional Software** tabs.

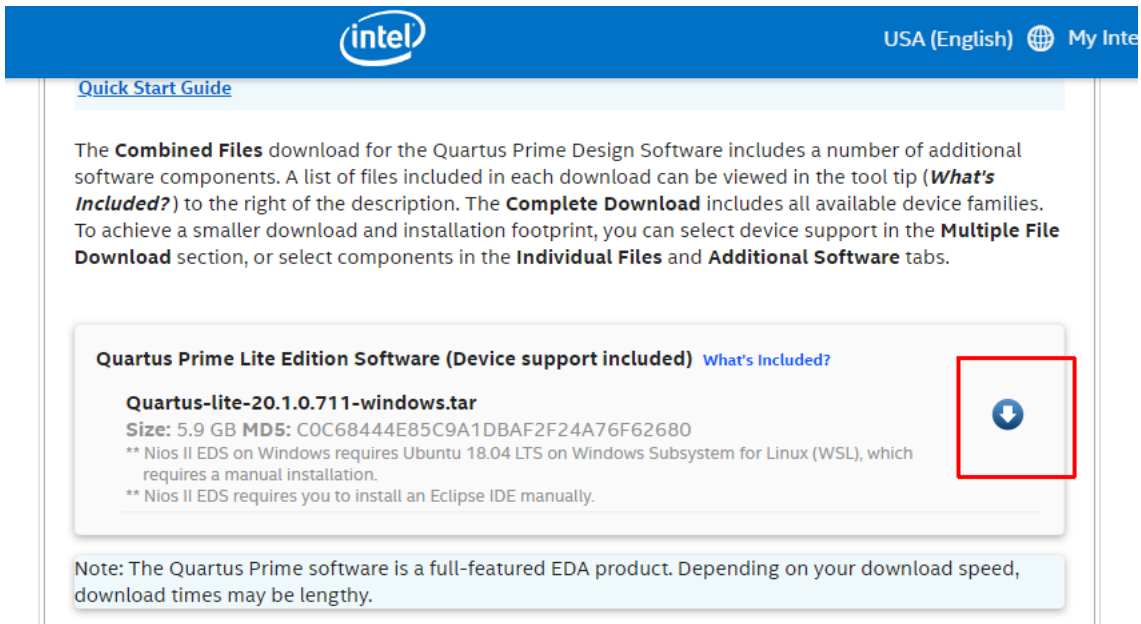
Quartus Prime Lite Edition Software (Device support included) [What's Included?](#)

Quartus-lite-20.1.0.711-windows.tar 

Size: 5.9 GB MD5: COC68444E85C9A1DBAF2F24A76F62680
** Nios II EDS on Windows requires Ubuntu 18.04 LTS on Windows Subsystem for Linux (WSL), which requires a manual installation.
** Nios II EDS requires you to install an Eclipse IDE manually.

Note: The Quartus Prime software is a full-featured EDA product. Depending on your download speed,


Click on the download icon.



[Quick Start Guide](#)

The **Combined Files** download for the Quartus Prime Design Software includes a number of additional software components. A list of files included in each download can be viewed in the tool tip (**What's Included?**) to the right of the description. The **Complete Download** includes all available device families. To achieve a smaller download and installation footprint, you can select device support in the **Multiple File Download** section, or select components in the **Individual Files** and **Additional Software** tabs.

Quartus Prime Lite Edition Software (Device support included) [What's Included?](#)

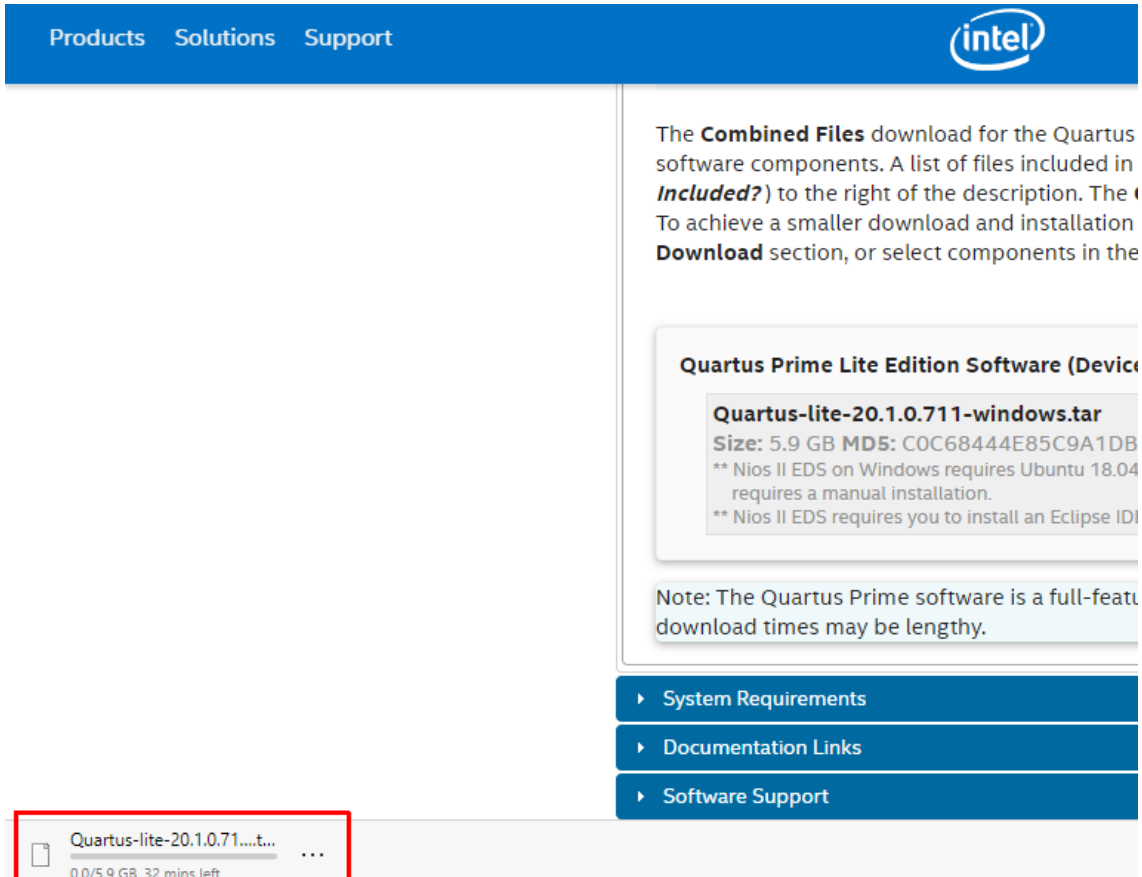
Quartus-lite-20.1.0.711-windows.tar 

Size: 5.9 GB MD5: COC68444E85C9A1DBAF2F24A76F62680
** Nios II EDS on Windows requires Ubuntu 18.04 LTS on Windows Subsystem for Linux (WSL), which requires a manual installation.
** Nios II EDS requires you to install an Eclipse IDE manually.


Note: The Quartus Prime software is a full-featured EDA product. Depending on your download speed, download times may be lengthy.

USB CPLD Development System User Manual

This will start the download.



The screenshot shows the Intel website's download page for Quartus Prime Lite Edition Software. The page has a blue header with 'Products Solutions Support' and the Intel logo. The main content area contains text about the 'Combined Files' download and a list of included files. A specific file, 'Quartus-lite-20.1.0.711-windows.tar', is highlighted with a red box. Below the file list, there are three blue buttons: 'System Requirements', 'Documentation Links', and 'Software Support'. At the bottom, a download progress bar shows the file name 'Quartus-lite-20.1.0.711...t...' and the progress '0.0/5.9 GB, 32 mins left'.

Products Solutions Support 

The **Combined Files** download for the Quartus software components. A list of files included in **Included?**) to the right of the description. The To achieve a smaller download and installation **Download** section, or select components in the

Quartus Prime Lite Edition Software (Device

Quartus-lite-20.1.0.711-windows.tar
Size: 5.9 GB MD5: C0C68444E85C9A1DB
** Nios II EDS on Windows requires Ubuntu 18.04 requires a manual installation.
** Nios II EDS requires you to install an Eclipse IDE

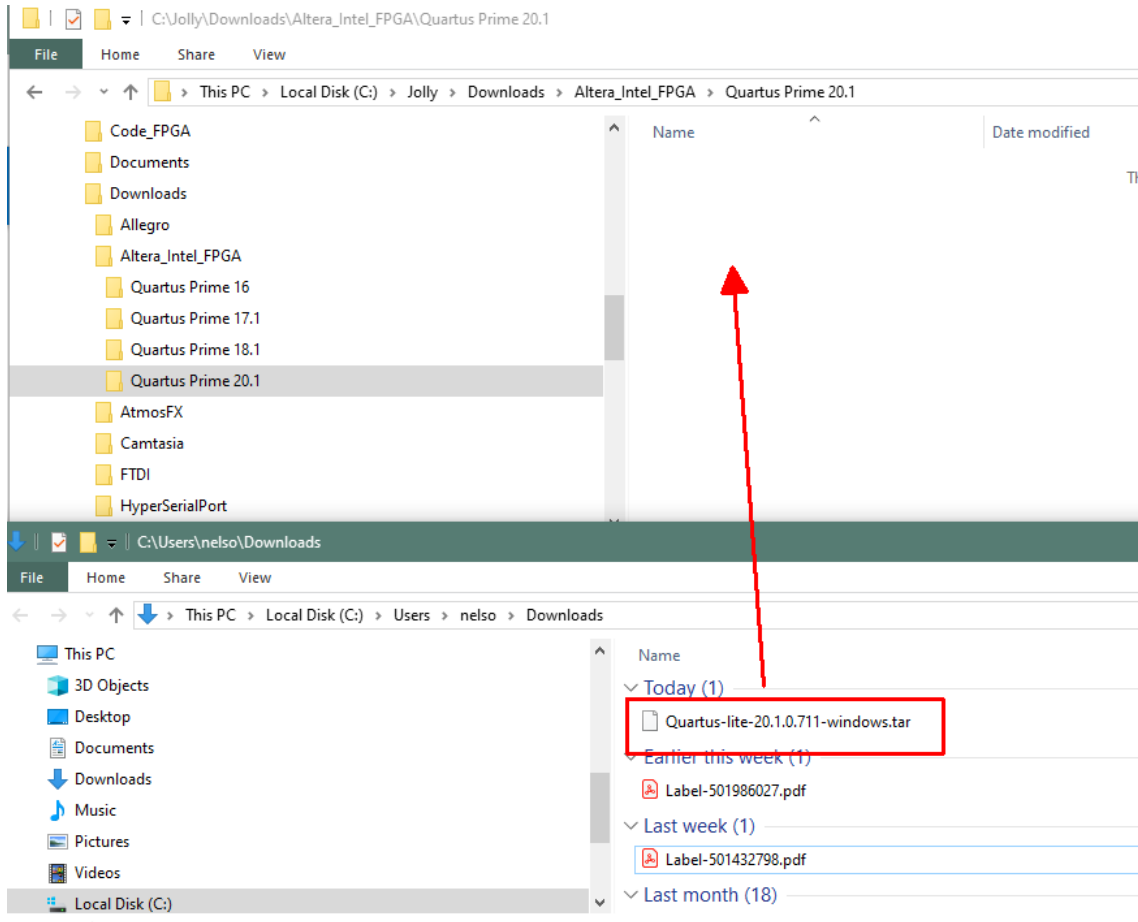
Note: The Quartus Prime software is a full-featured download times may be lengthy.

- ▶ System Requirements
- ▶ Documentation Links
- ▶ Software Support

Quartus-lite-20.1.0.711...t...
0.0/5.9 GB, 32 mins left

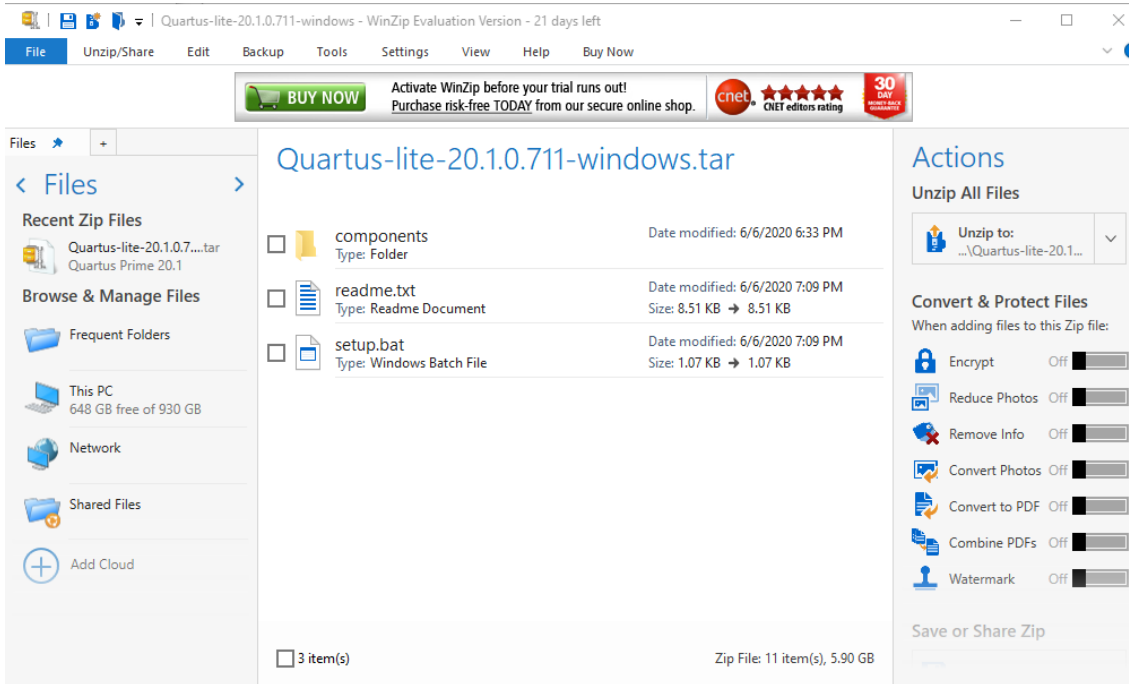
The file is 5.9 GB, so this could take a couple of hours depending on your internet connection. When download is complete, store the *.tar file in a directory on your PC.

USB CPLD Development System User Manual

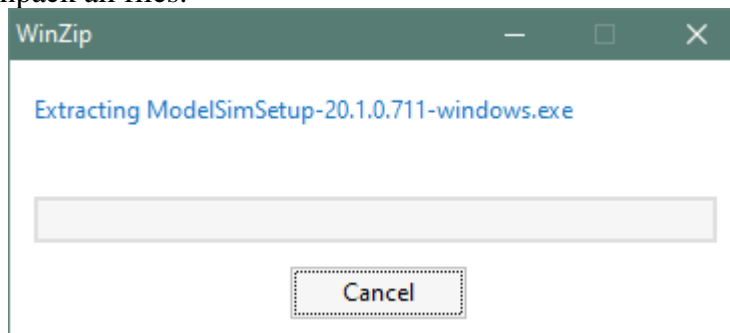


Use a tool such as WinZip to Extract the *.tar file.

USB CPLD Development System User Manual



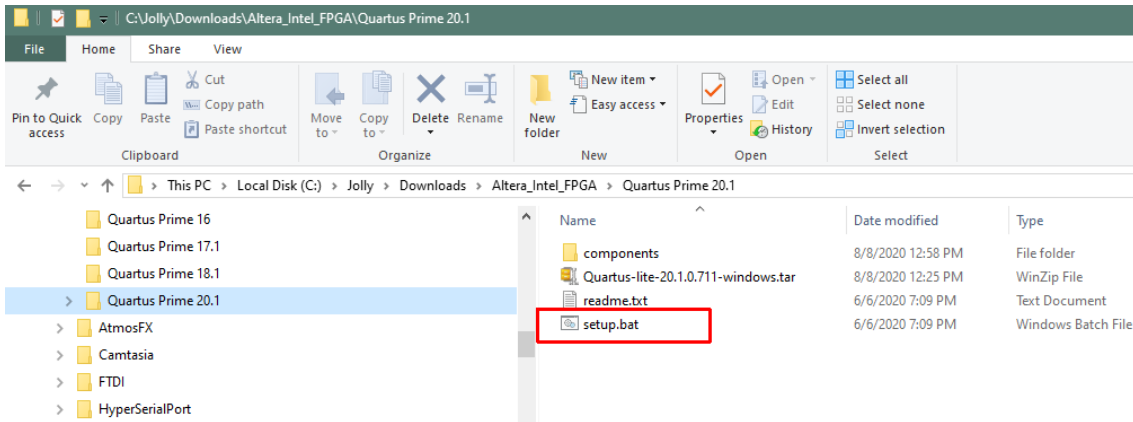
The tool will unpack all files.



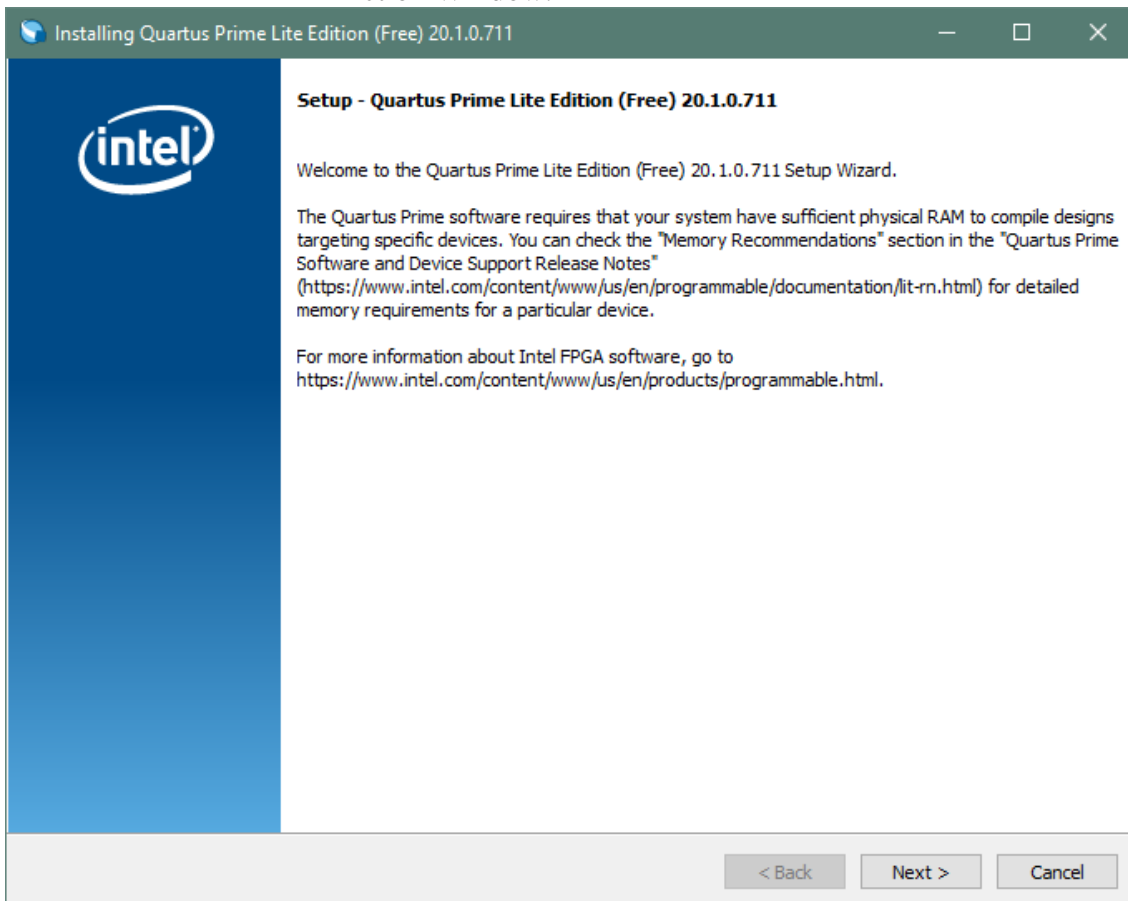
2.2.3 Quartus Installer

When the unpacking finishes from the previous section, double click the setup.bat file in the download folder.

USB CPLD Development System User Manual



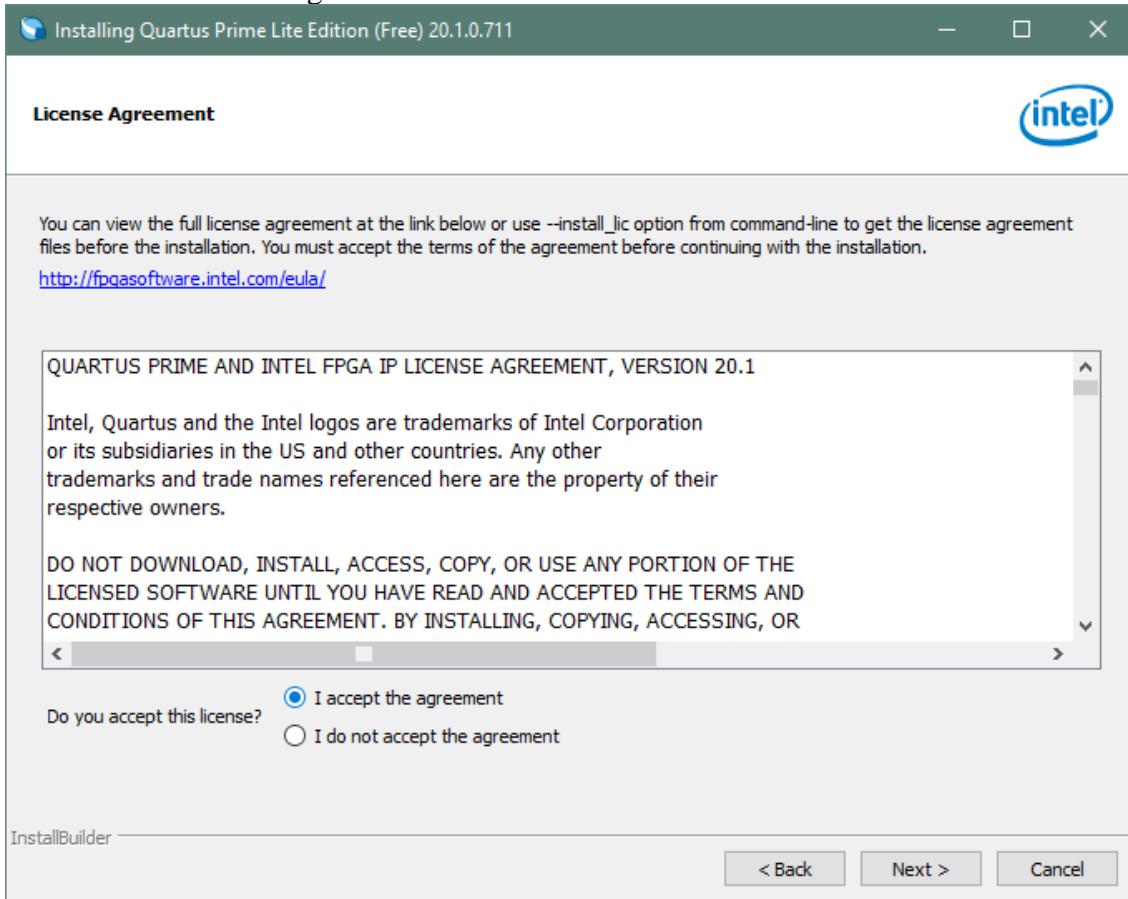
Click “Next” on the Introduction Window.





USB CPLD Development System User Manual

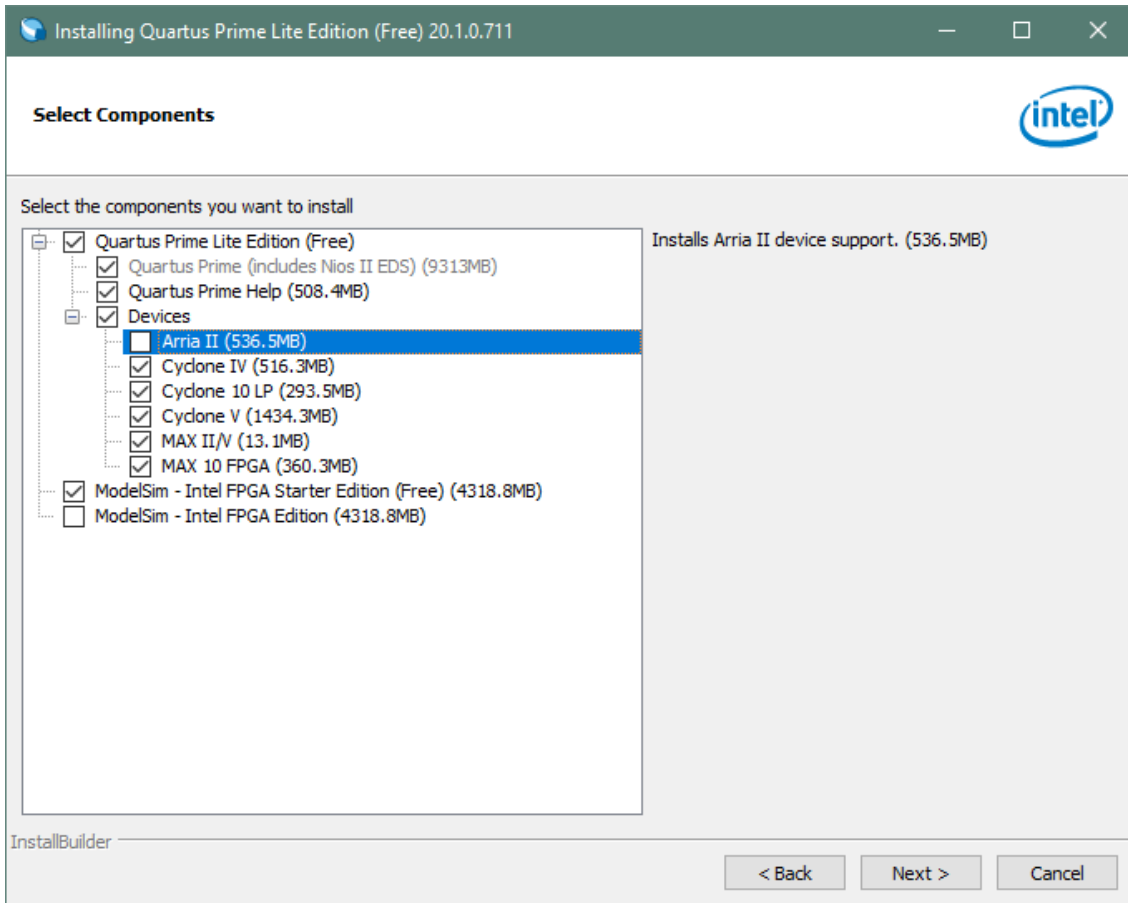
Click the checkbox to agree to the license terms. Then click “Next”.



Click “Next” and accept the defaults.

At the Select Products Window, de-select the Quartus Prime 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).

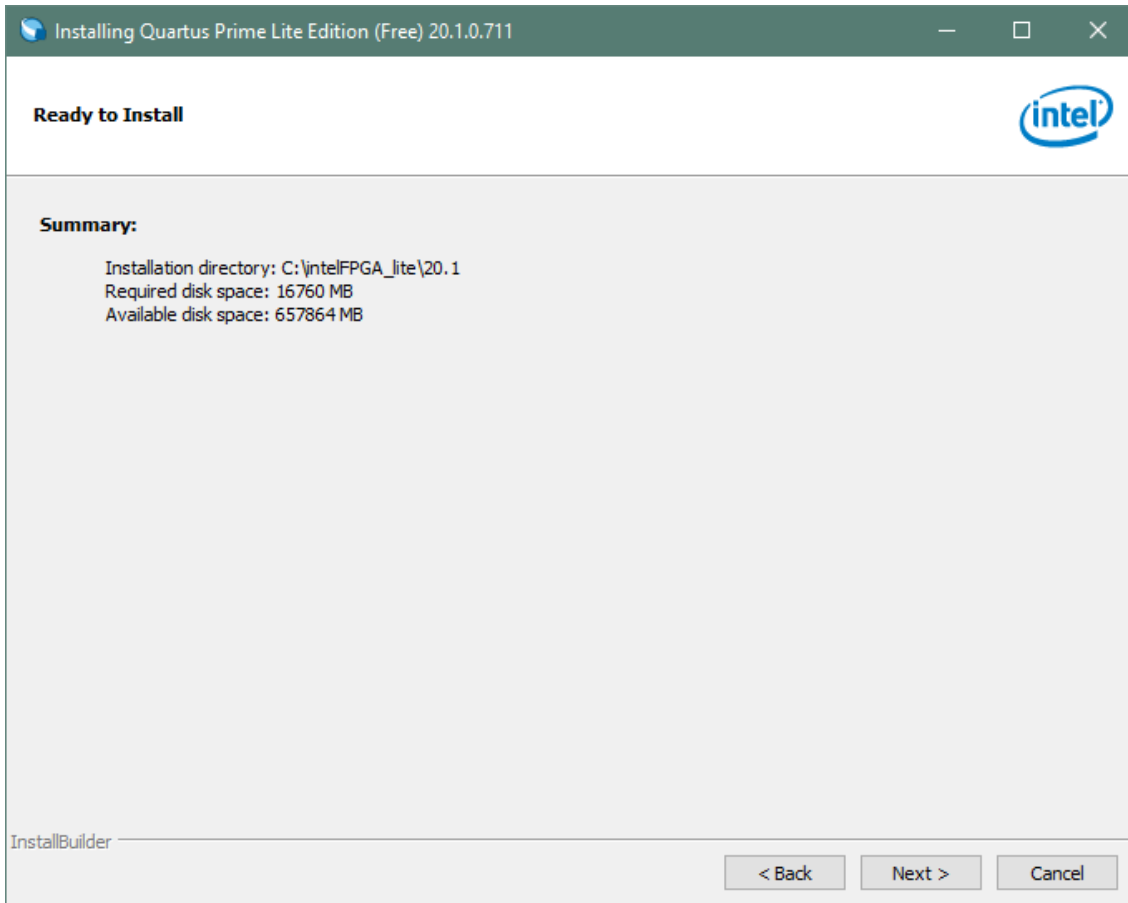
USB CPLD Development System User Manual



Click “Next” to accept the defaults



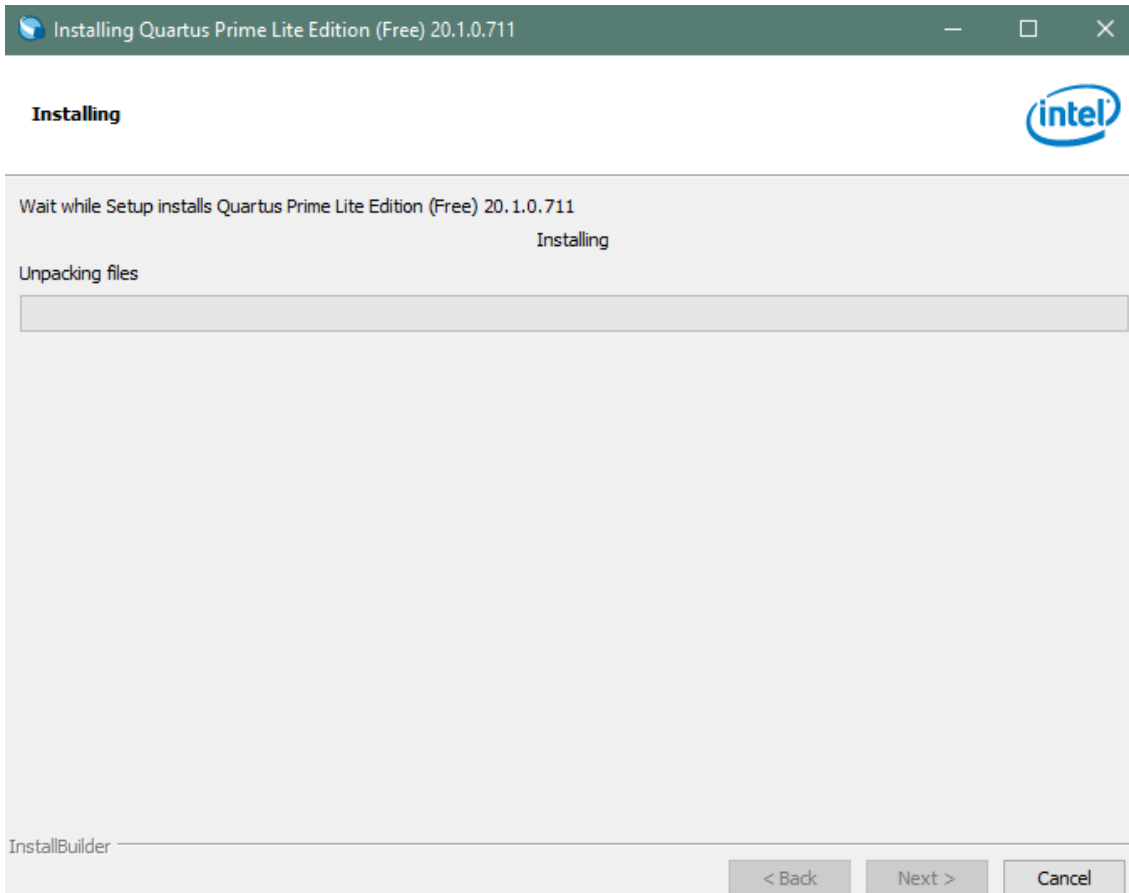
USB CPLD Development System User Manual



Click “Next” to accept the defaults

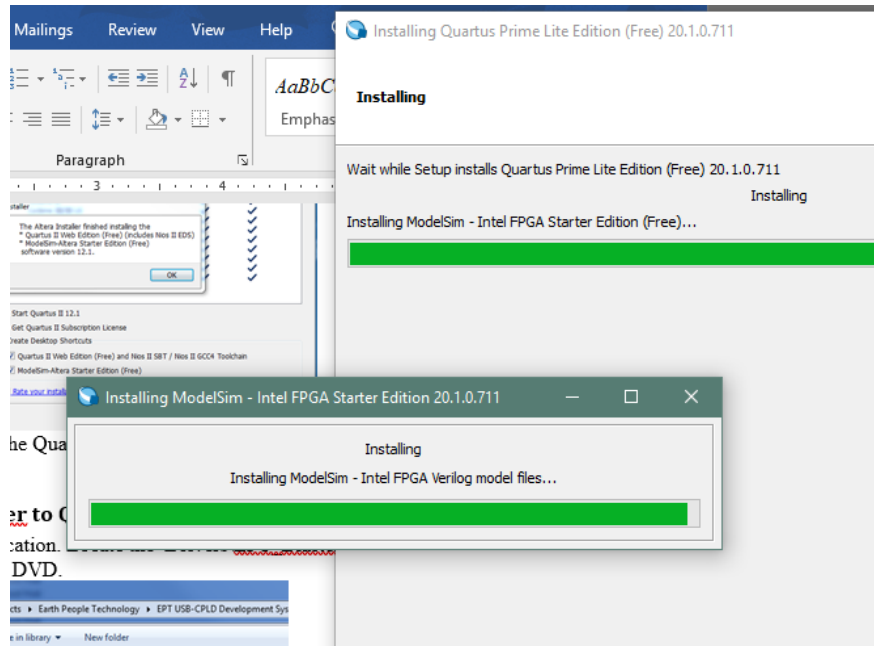


USB CPLD Development System User Manual

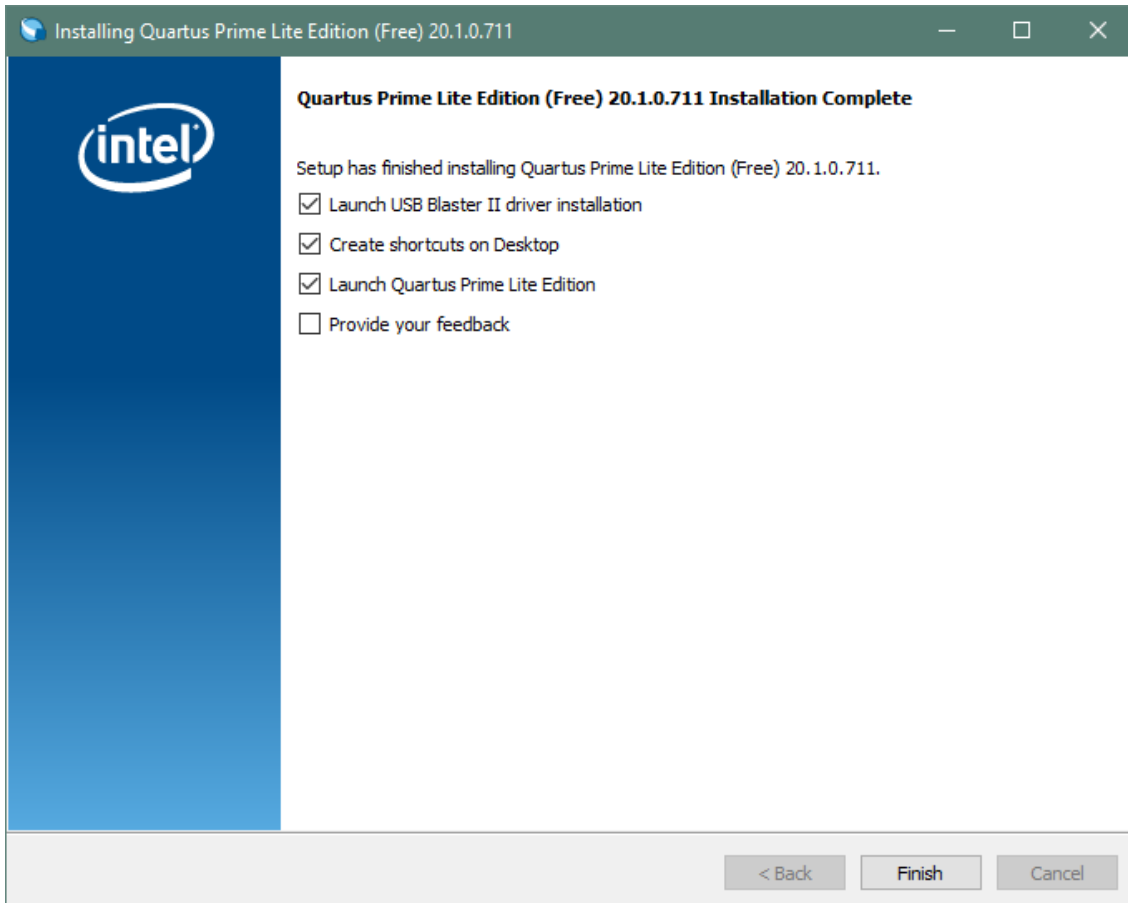


Wait for the installation to complete.

USB CPLD Development System User Manual

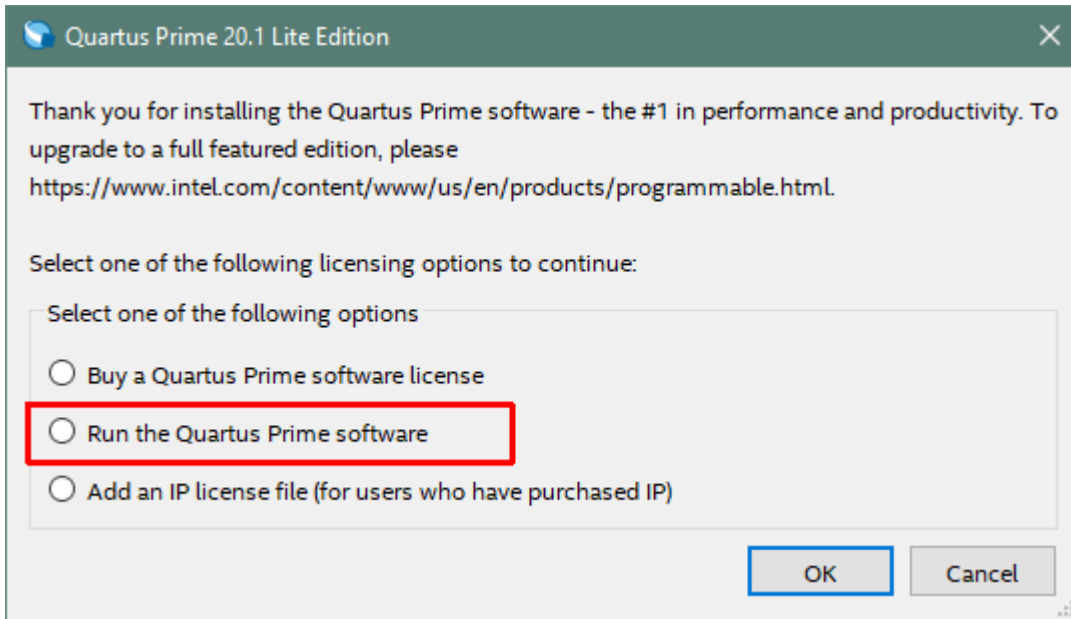


USB CPLD Development System User Manual



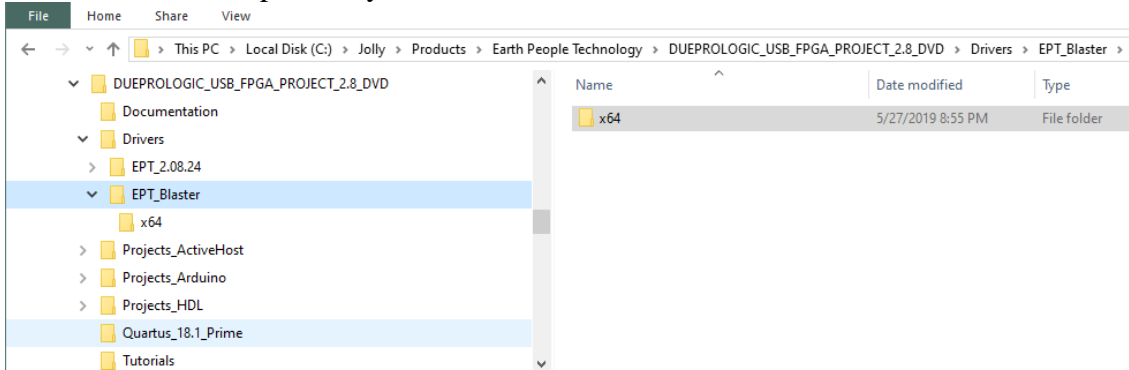
Click "Ok", then click "Finish". The Quartus Prime is now installed and ready to be used.

USB CPLD Development System User Manual



2.2.4 Adding the EPT_Blaster to Quartus Prime

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

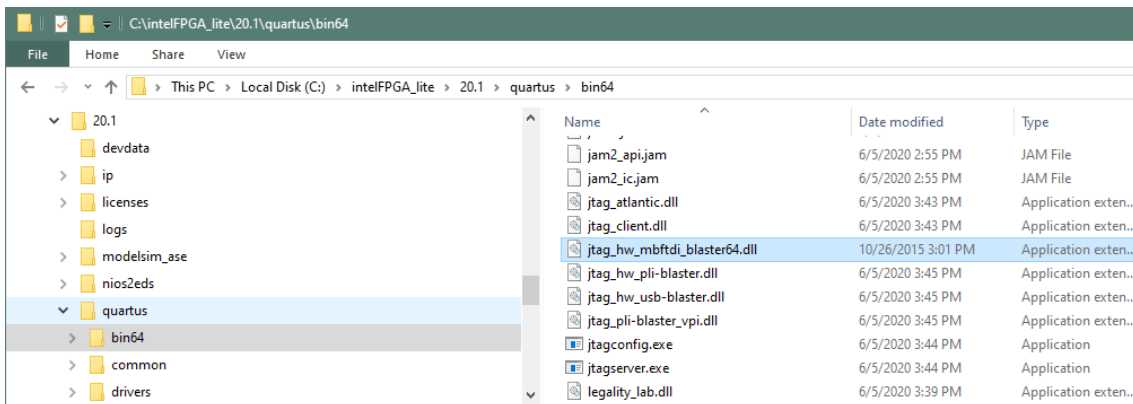


Follow these directions:

1. Open the
C:\..\MEGAPROLOGIC_USB_CPLD_PROJECT_x.x_DVD\Drivers\EPT_Blaster\x64 folder.
2. Select the file “jtag_hw_mbftdi_blaster.dll” and copy it.
3. Browse over to C:\intelFPGA_lite\xx.x\quartus\bin64.

USB CPLD Development System User Manual

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



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

2.3 Active Host Application DLL

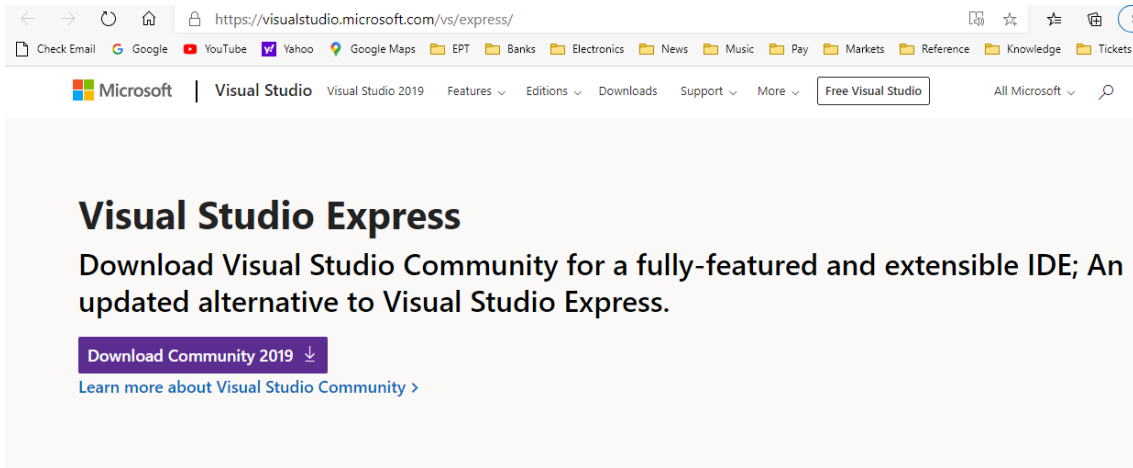
Download the latest version of Microsoft Visual C# Express environment from Microsoft. It’s a free download.

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

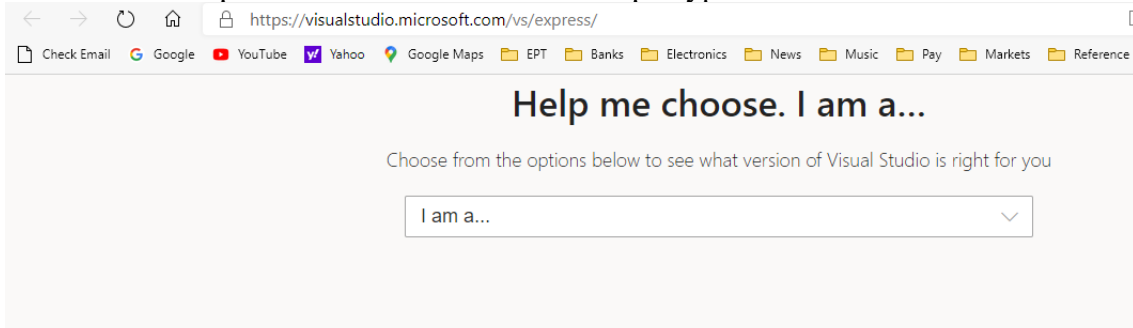
Go to the website and click on the “+” icon next to the Visual C# Express.



USB CPLD Development System User Manual



Click on the “Express 20xx for Windows Desktop” hypertext.



Still want Visual Studio Express?

[Express 2017 for Windows Desktop](#)

Supports building managed and native desktop applications.*

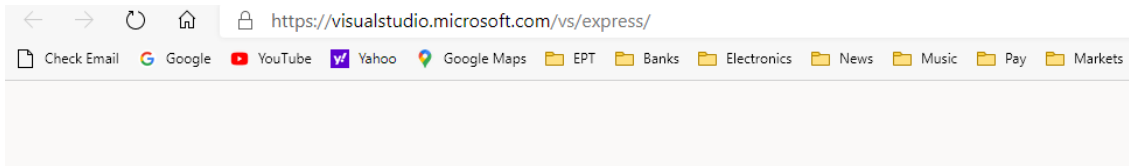
[Express 2015 for Windows Desktop](#)

Supports the creation of desktop applications for Windows.

The download manager file will download the “WDExpress.exe” file.



USB CPLD Development System User Manual



Still want Visual Studio Express?

[Express 2017 for Windows Desktop](#)

Supports building managed and native desktop applications.*

[Express 2015 for Windows Desktop](#)

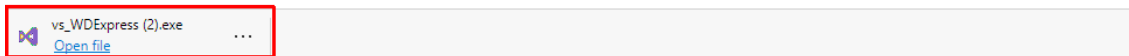
Supports the creation of desktop applications for Windows.

[Express 2015 for Web](#)

Create standards-based, responsive websites, web APIs, or real-time online experiences using ASP.NET.

[Express 2015 for Windows 10](#)

Provides the core tools for building compelling, innovative apps for Universal Windows Platform. Windows is required.



Right click on the WDEpress.exe.

Still want Visual Studio Express?

[Express 2017 for Windows Desktop](#)

Supports building managed and native desktop applications.*

[Express 2015 for Windows Desktop](#)

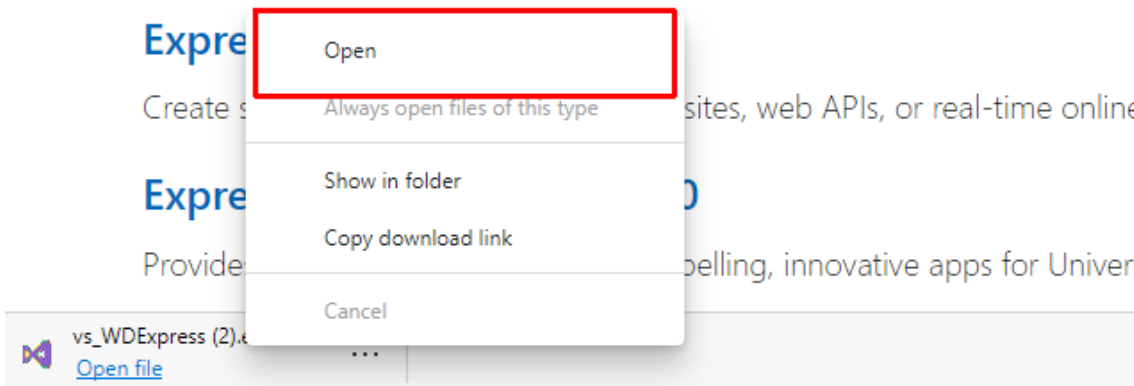
Supports the creation of desktop applications for Windows.

[Express](#)

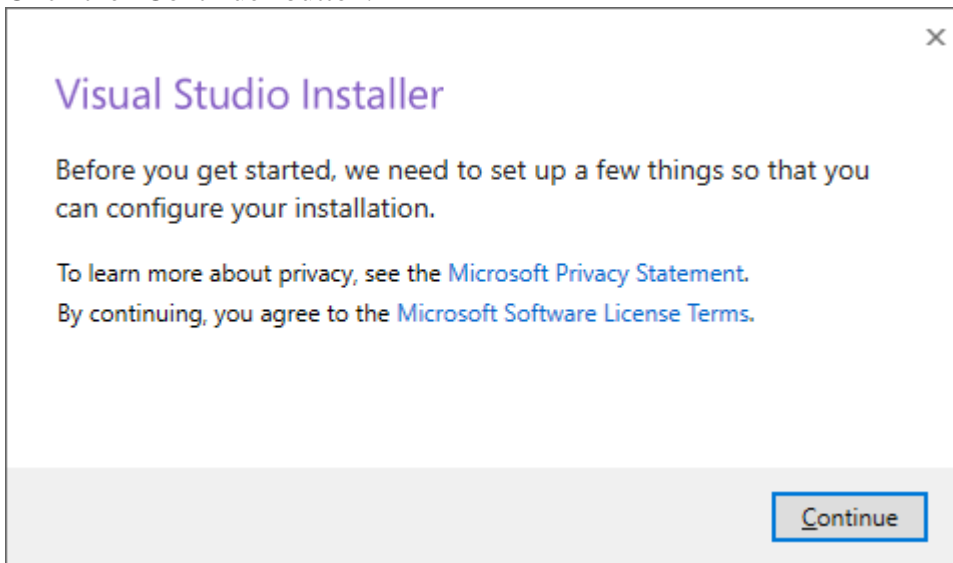
Create s... Always open files of this type... sites, web APIs, or real-time online

[Express](#)

Provide... Copy download link... Selling, innovative apps for Univer:



Click the “Continue” button.

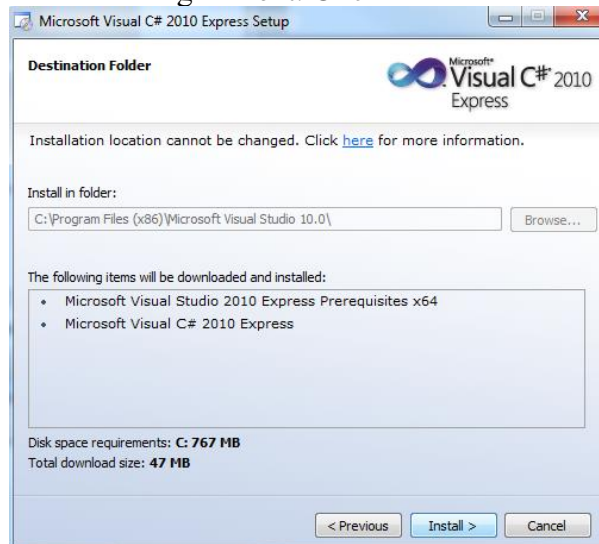


Next, follow the on screen windows and accept the default answers.

USB CPLD Development System User Manual



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



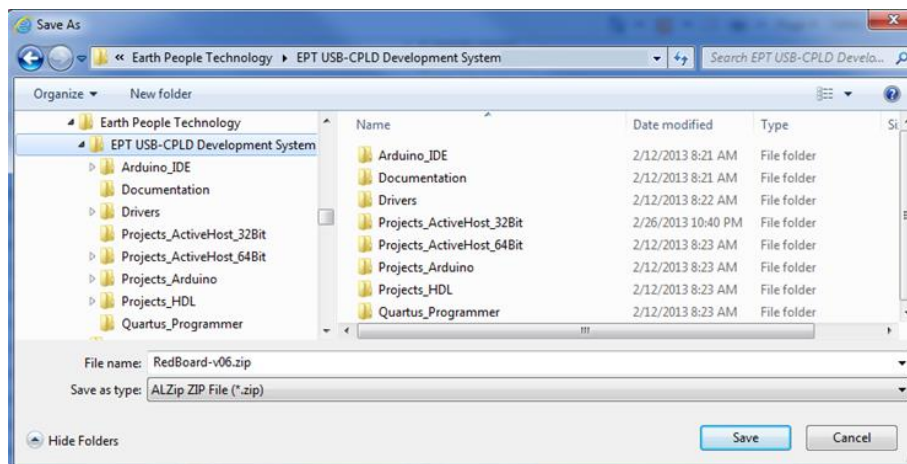
Visual C# 2010 Express will install. This may take up to twenty minutes depending on your internet connection.

USB CPLD Development System User Manual



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

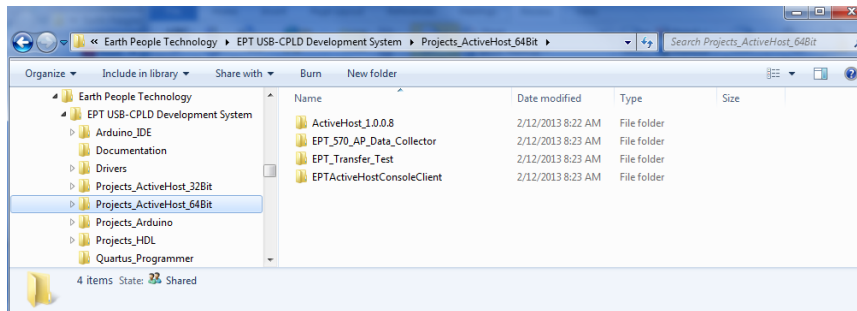
To use the Active Host Application Software, the Active Host DLL and the ftd2xx DLL must be included in the Microsoft Visual project. The Active Host Application Software will allow the user to create a custom applications on the PC using the EndTerms to perform Triggers and Data Transfer to/from the MegaProLogic. 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 MegaProLogic Development System CD.



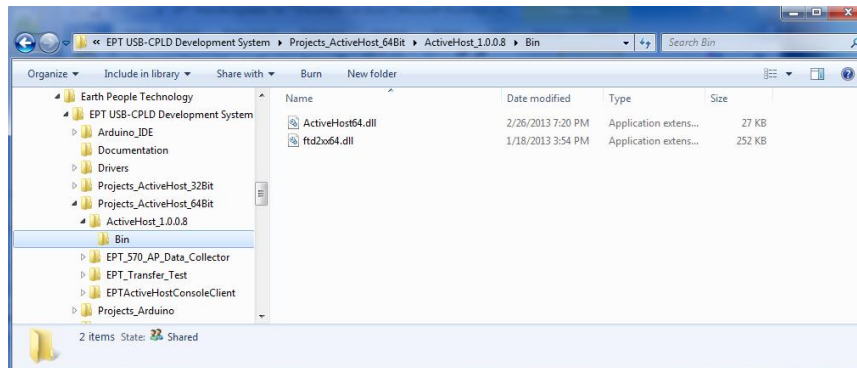


USB CPLD Development System User Manual

Locate the Projects_ActiveHost_64Bit in the MegaProLogic Development System using Windows Explorer.

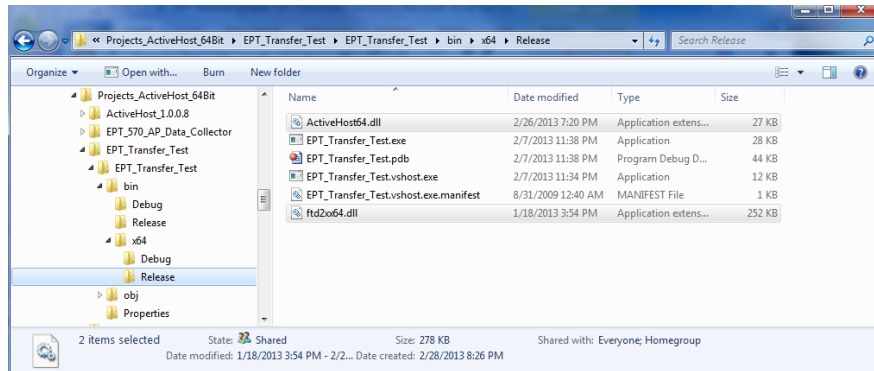


Locate the Projects_ActiveHost_64Bit \ActiveHost_1.0.0.8\Bin folder and copy the ActiveHost64.dll and the ftd2xx64.dll.



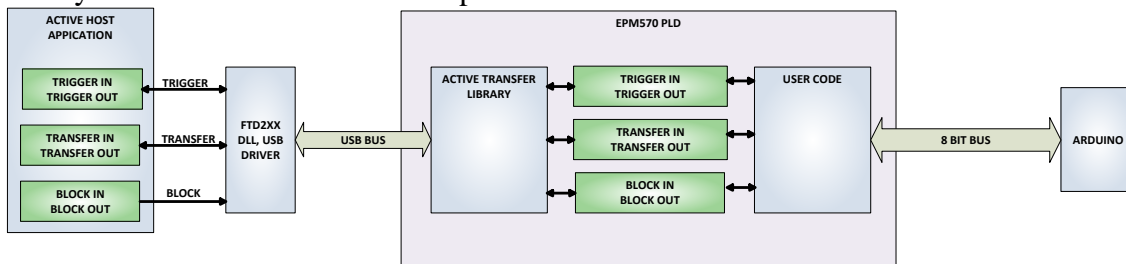
Save the DLL's in the bin\x64\Release folder of the user project under the Microsoft C# Express project. See the Active Host Application section of the MegaProLogic Development System User Manuals for instructions on how to add the dll to the Microsoft C# Express project.

USB CPLD Development System User Manual



3 Active Transfer Library

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



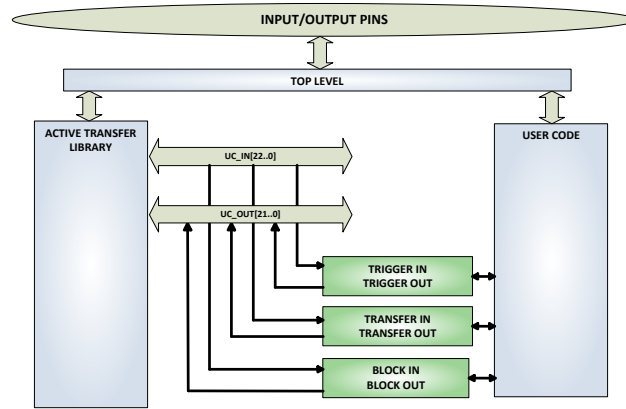
3.1 EPT Active Transfer System Overview

The Active Transfer System components consist of the following:

- active_serial_library.v
- ft_245_state_machine.v
- endpoint_registers.vqm
- active_trigger.v
- active_transfer.v
- active_block.v
- active_transfer_uart.v
- uart_receiver.v
- uart_top.v
- uart_transmitter.v

USB CPLD Development System User Manual

The Active_Serial_Library provides the communication to the USB hardware. While separate Input and Output buses provide bi-directional communications with the plug in modules.

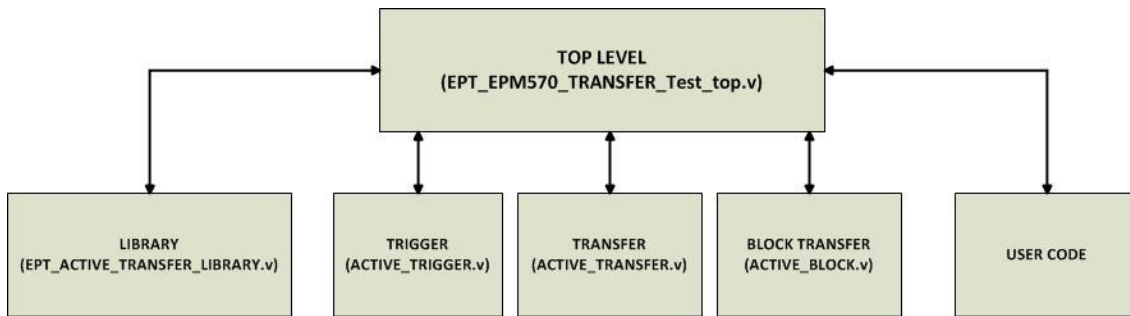


The figure above shows how the modules of the EPT Active Transfer Library attach to the overall user project. The EPT Active_Transfer_Library.vqm, Active_Trigger.v, Active_Transfer.v and Active_Block.v modules are instantiated in the top level of the user project. The User_Code.v module is also instantiated 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.

USB CPLD Development System User Manual



```

1  //#####
2  //#
3  //# Copyright   Earth People Technology Inc. 2023
4  //#
5  //#
6  //# File Name:   EPT_5M57_AP_M4_Top.v
7  //# Author:     R. Jolly
8  //# Date:       January 14, 2023
9  //# Revision:   A
10 //#
11 //# Development: USB Test Tool Interface board
12 //# Application: Altera MAX V CPLD
13 //# Description: This file contains verilog code which will allow access
14 //#               to Active Transfer Library. The MAX V receives its commands via
15 //#               USB.
16 //#
17 //#
18 //#
19 //#*****
20 //#
21 //# Revision History:
22 //#   DATE       VERSION   DETAILS
23 //#   1/14/23    A         Created      RJJ
24 //#
25 //#
26 //#
27 //#
28 //#####
29
30 `timescale 1ns/1ps
31
32
  
```

USB CPLD Development System User Manual

```

33
34 //*****
35 /* Module Declaration
36 //*****
37
38 module EPT_5M57_AP_M4_Top (
39
40
41     input wire          CLK_66MHZ,
42     input wire          RST,
43
44     //UART COMMAND BUS
45     output wire         UART_OUT, //To external
46     input wire          UART_IN, //From external
47
48     input wire [7:0]    LB_XIOLA,    //XIOH -- J16
49     input wire [2:0]    LB_XIOLC,    //XIOH -- J16
50     input wire [7:0]    LB_XIOH,    //XIOL -- J16
51     input wire [7:0]    LB_COMM,    //COMM -- J10
52     input wire [7:0]    LB_AD,      //XIOH -- J8
53     input wire [7:0]    LB_IOH,     //XIOH -- J9
54
55     //Transceiver Control Signals
56     output wire         TR_DIR_1,
57     output wire         TR_OE_1,
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
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153 //-----
154 // Instantiate the EPT Library
155 //-----
156
157 active_serial_library ACTIVE_SERIAL_LIBRARY_INST
158 (
159     .aa          (aa),
160     .UART_OUT    (UART_OUT),
161     .UART_IN     (uart_txd),
162
163     .UC_IN       (UC_IN),
164     .UC_OUT      (UC_OUT)//,
165
166     //.STATE_OUT  (active_serial_state_out),
167     //.TEST_OUT   (active_transfer_test_out)
168
169 );
170

```

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



USB CPLD Development System User Manual

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 bus UART_IN and UART_OUT 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.

USB CPLD Development System User 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.



USB CPLD Development System User 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
```

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.

USB CPLD Development System User 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

```

USB CPLD Development System User Manual

To receive a byte, the Active Host will send a byte using it's dll. The user code must monitor the transfer_received port. The transfer_received port will assert high for one clock cycle (66 MHz) when a byte is ready for reading on the transfer_to_device port. User code should use an asynchronous always block to detect when the transfer_received port is asserted. Upon assertion, the user code should read the byte from the transfer_to_device port into a local register.

```

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

```

3.2.3 Active Block EndTerm

The Active Block module is designed to transfer blocks of data between Host and User Code and vice versa. This allows buffers of data to be transferred with a minimal amount of code. The Active Block module is addressable, so up to eight individual



USB CPLD Development System User Manual

modules can be instantiated and separately addressed. The length of the block to be transferred must also be specified in the `uc_length` port.

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

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

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



USB CPLD Development System User 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
```


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 Methods

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

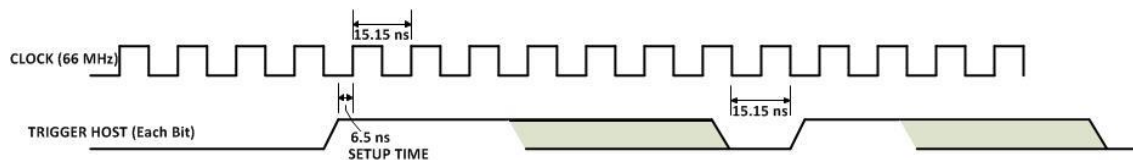


Figure xx Active Trigger to Host Timing

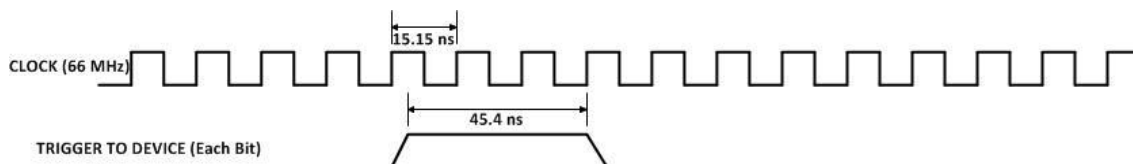


Figure xx Active Trigger to Device Timing

3.3.2 Active Transfer EndTerm Timing

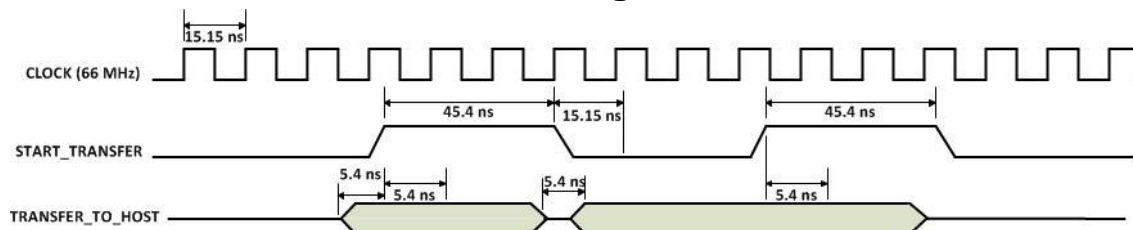


Figure xx Active Transfer To Host Timing

USB CPLD Development System User Manual

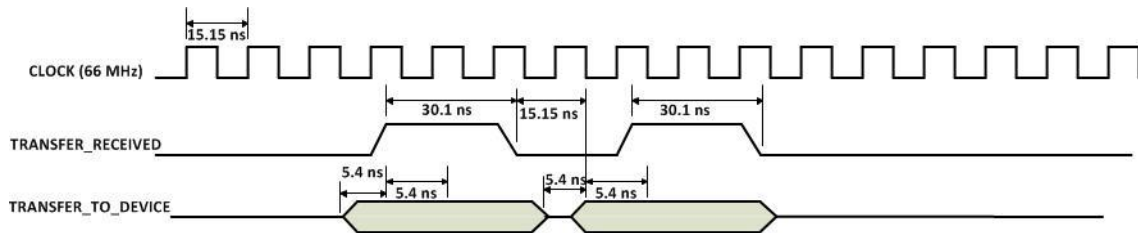


Figure xx Active Transfer To Device Timing

3.3.3 Active Block EndTerm Timing

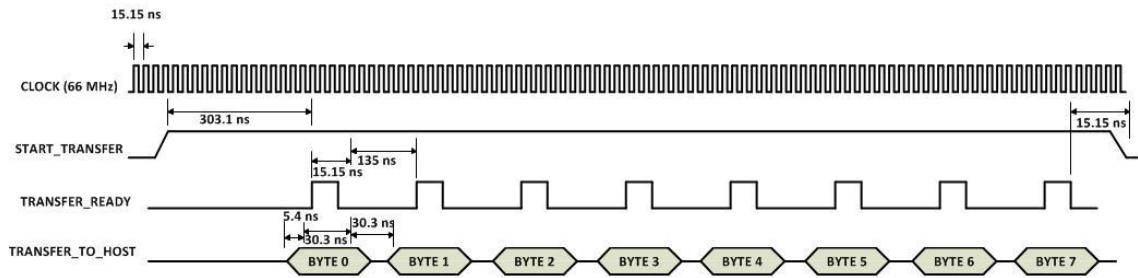


Figure xx Active Block To Host Timing

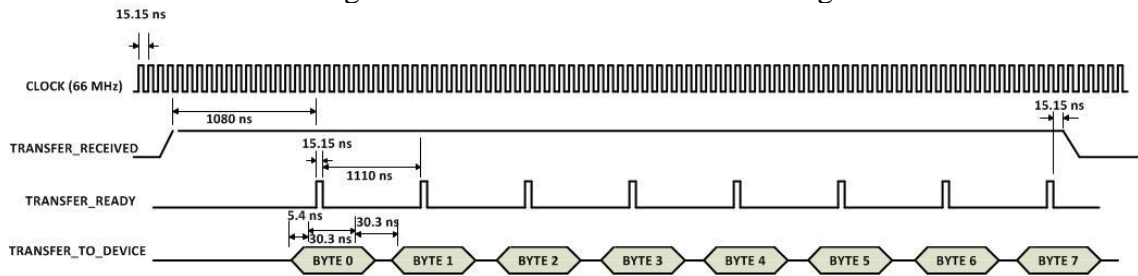
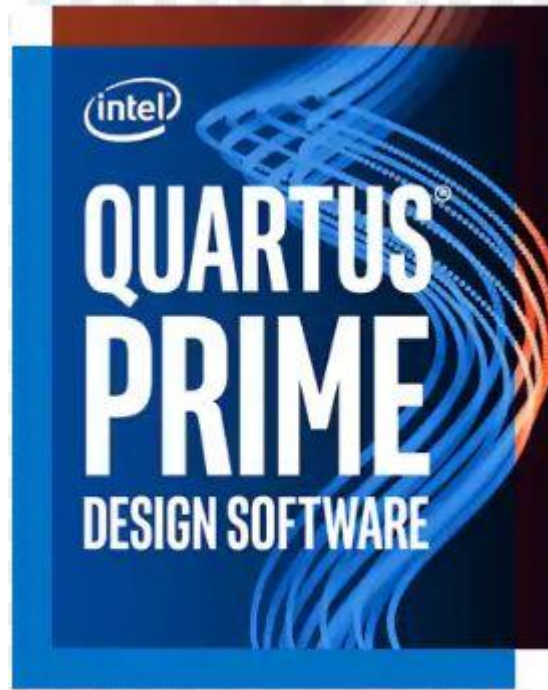


Figure xx Active Block To Device Timing

4 Compiling, Synthesizing, and Programming CPLD

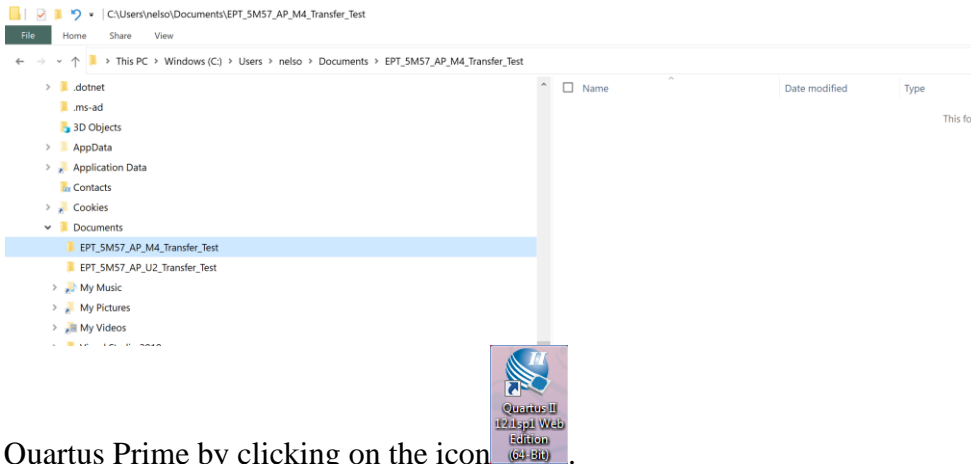


The CPLD on the MEGAPROLOGIC can be programmed with the Active Transfer Library and custom HDL code created by the user. Programming the CPLD requires the use of the Quartus Prime software and a standard USB cable. There are no extra parts to buy, just plug in the USB cable. Once the user HDL code is written according to the syntax rules of the language (Verilog and VHDL) it can be compiled and synthesized using the Quartus Prime software. This manual will not focus on HDL coding or proper coding techniques, instead it will use the example code to compile, synthesize and program the CPLD.

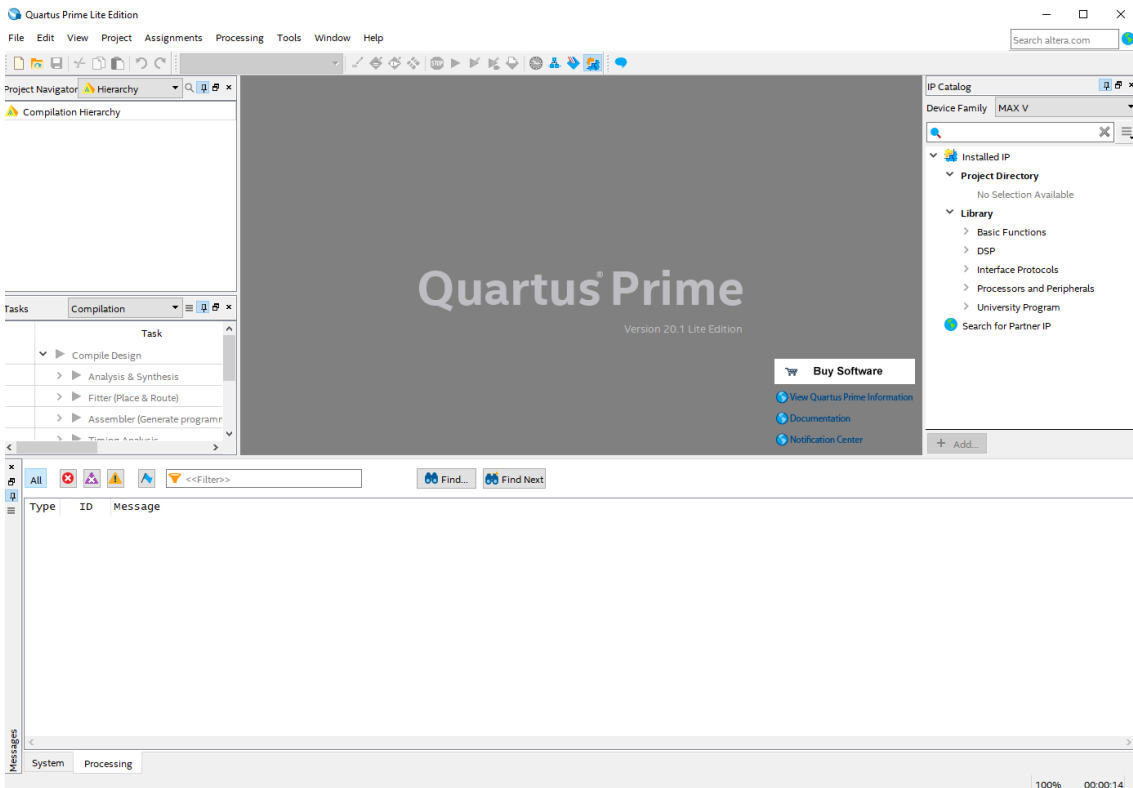
4.1 *Setting up the Project and Compiling*

Once the HDL code (Verilog or VHDL) is written and verified using a simulator, a project can be created using Quartus Prime. Writing the HDL code and simulating it will be covered in later sections. Bring up Quartus Prime, then use Windows Explorer to browse to C:\C:\Users\nelso\Documents\EPT_5M57_AP_M4_Transfer_Test to create a new directory called: "EPT_Transfer_Test".

USB CPLD Development System User Manual

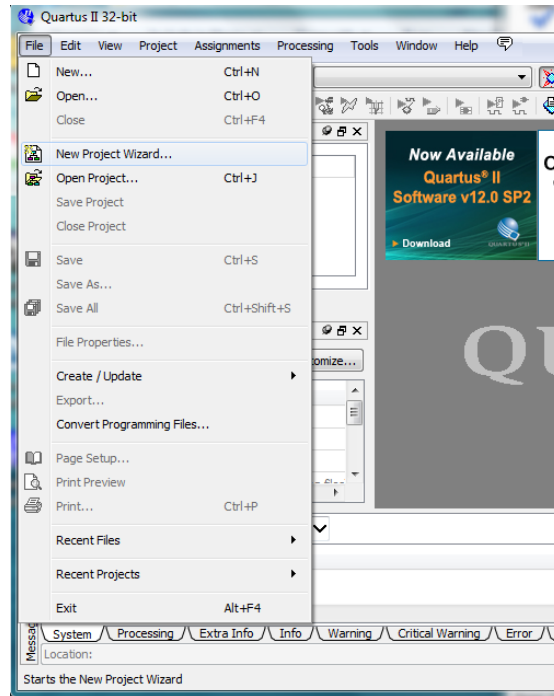


Open Quartus Prime by clicking on the icon .



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

USB CPLD Development System User Manual



At the Top-Level Entity page, browse to the C:\Users\



USB CPLD Development System User Manual

New Project Wizard

Directory, Name, Top-Level Entity

What is the working directory for this project?

 ...

What is the name of this project?

 ...

What is the name of the top-level design entity for this project? This name is case sensitive and must exactly match the entity name in the design file.

 ...

< Back Next > Finish Cancel Help

New Project Wizard

Project Type

Select the type of project to create.

Empty project
Create new project by specifying project files and libraries, target device family and device, and EDA tool settings.

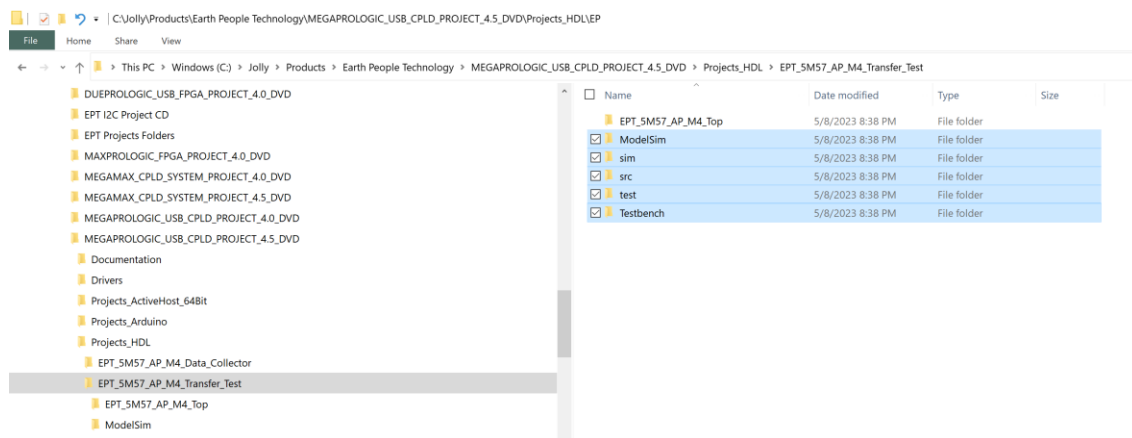
Project template
Create a project from an existing design template. You can choose from design templates installed with the Quartus Prime software, or download design templates from the [Design Store](#).

< Back Next > Finish Cancel Help

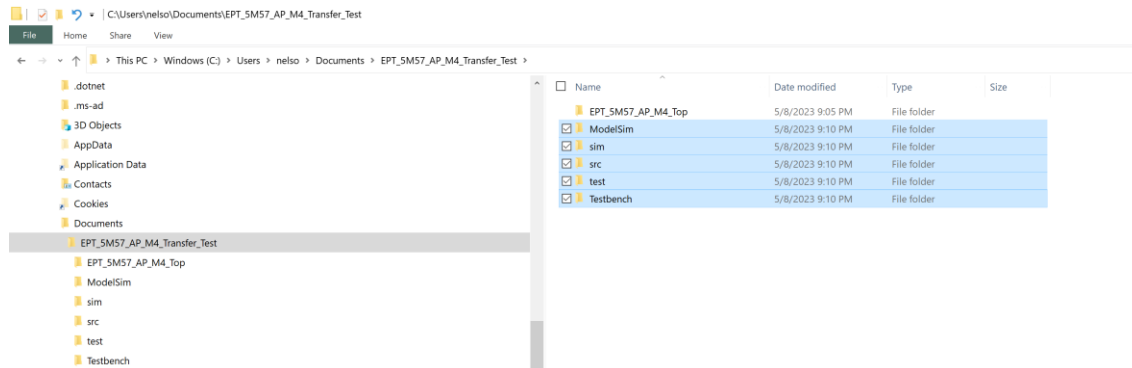


USB CPLD Development System User Manual

Select Next. At the Add Files window: Browse to the \Projects_HDL\EPT_Transfer_Test\src folder of the MegaProLogic Development System DVD. Copy the files from the following directory.

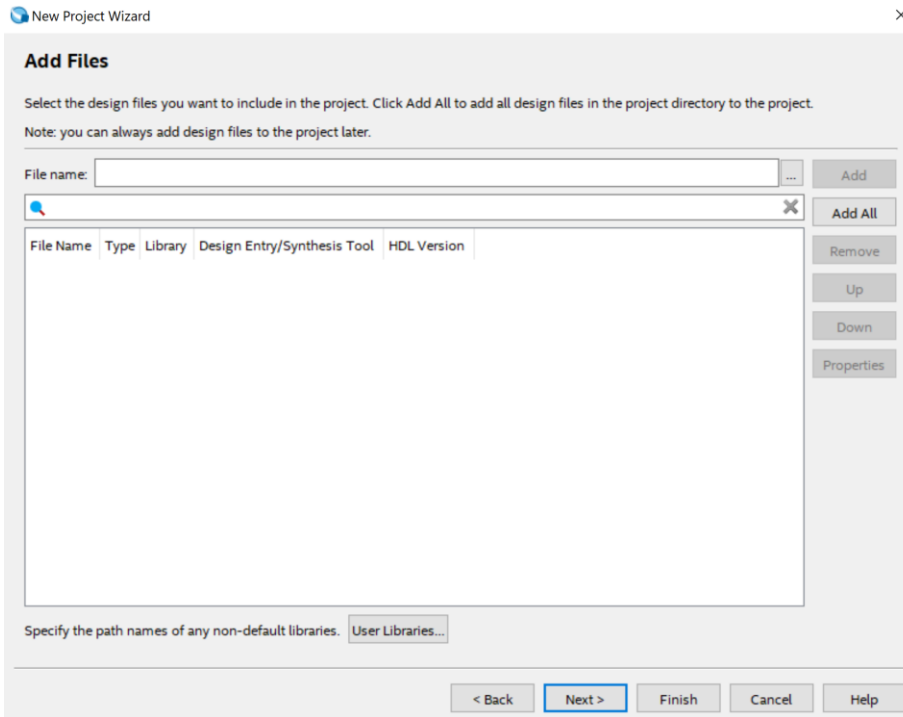


Then paste them in the C:/users/<your name>/Documents/EPT_5M57_AP_M4_Tranfer_Test\EPT_5M57_AP_M4_Top folder:

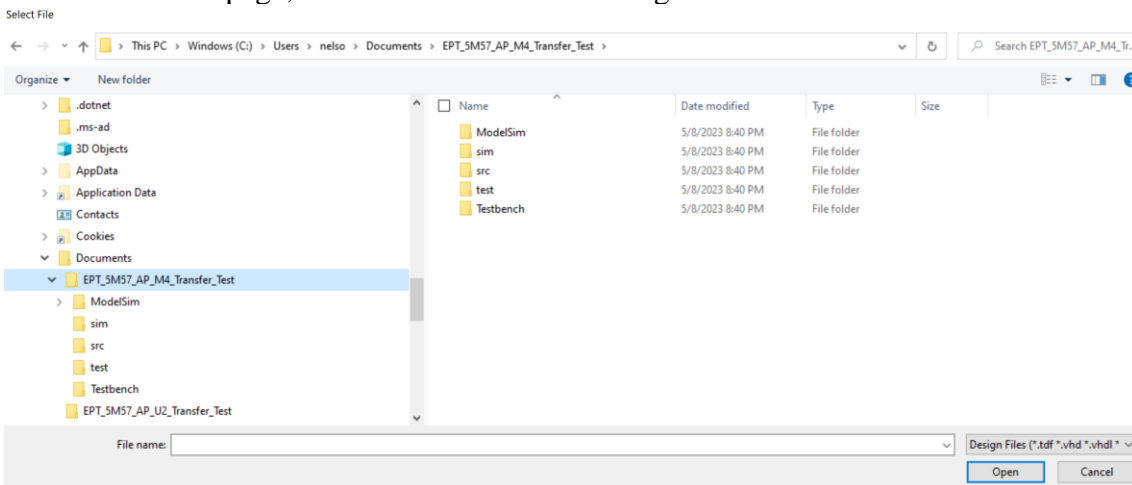


Then go to the “Add Files” page:

USB CPLD Development System User Manual



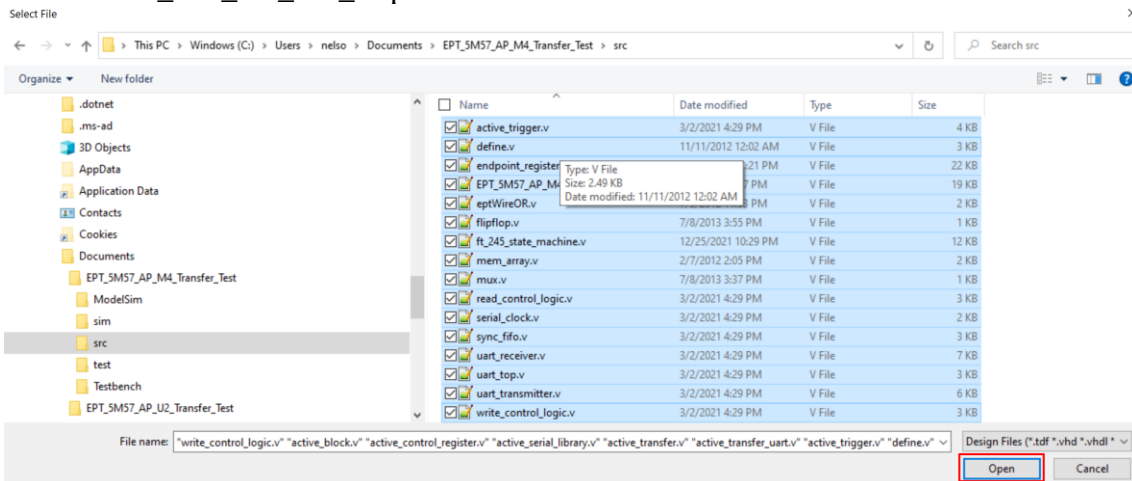
On the Resource page, click “Select File” and navigate to the folder:



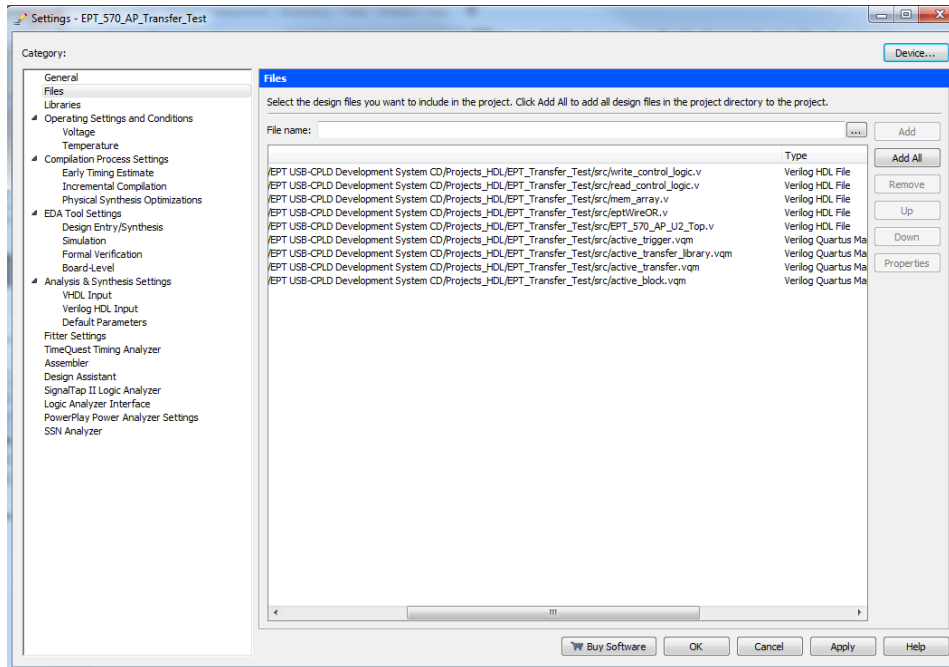
- Active_block.v
- Active_transfer.v
- Active_trigger.v
- Active_Serial_library.v
- Endpoint_registers.v

USB CPLD Development System User Manual

- FT_245_state_machine.v
- eptWireOr.v
- mem_array.v
- read_control_logic.v
- Serial_clock.v
- Uart_receiver.v
- Uart_transmitter.v
- write_control_logic.v
- EPT_570_AP_M4_Top.v



USB CPLD Development System User Manual



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

USB CPLD Development System User Manual

New Project Wizard

Family, Device & Board Settings

Device Board

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

To determine the version of the Quartus Prime software in which your target device is supported, refer to the [Device Support List](#) webpage.

Device family

Family: MAX V

Device: 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

Core speed grade: Any

Name filter:

Show advanced devices

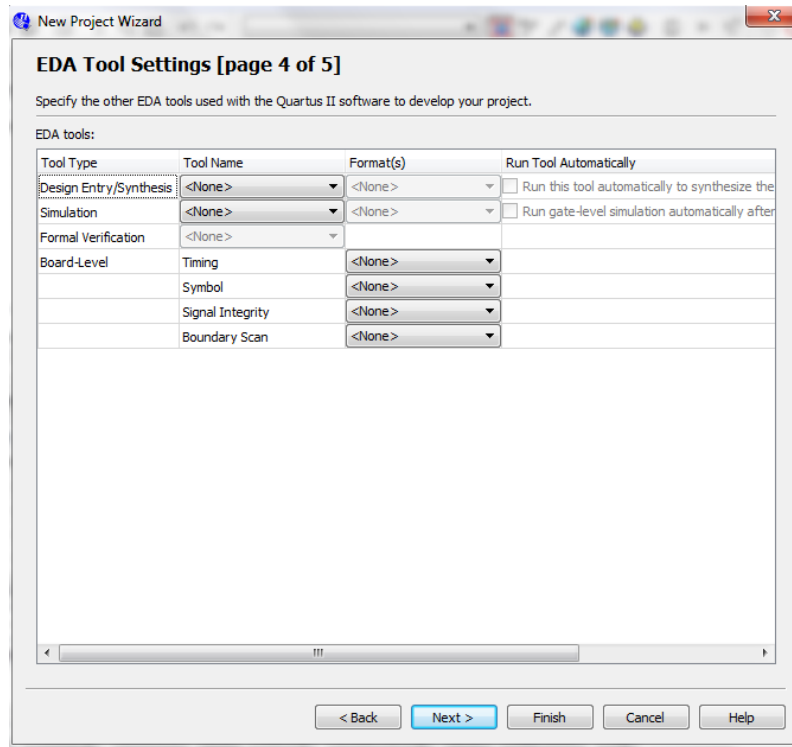
Available devices:

Name	Core Voltage	LEs	UFM blocks
5M570ZM100C4	1.8V	570	1
5M570ZM100C5	1.8V	570	1
5M570ZM100I5	1.8V	570	1
5M570ZT100A5	1.8V	570	1
5M570ZT100C4	1.8V	570	1
5M570ZT100C5	1.8V	570	1
5M570ZT100I5	1.8V	570	1
5M570ZT144C4	1.8V	570	1
5M570ZT144C5	1.8V	570	1
5M570ZT144I5	1.8V	570	1
5M1270ZF256A5	1.8V	1270	1
5M1270ZF256C4	1.8V	1270	1
5M1270ZF256C5	1.8V	1270	1

< Back Next > Finish Cancel Help

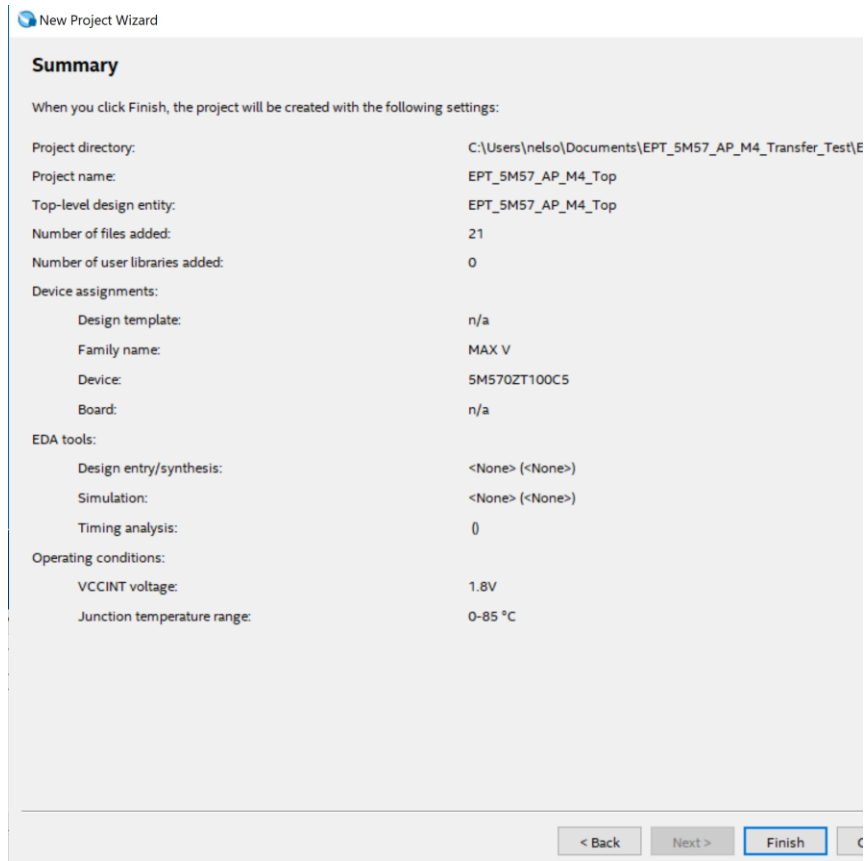
Select Next, leave defaults for the EDA Tool Settings.

USB CPLD Development System User Manual



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

USB CPLD Development System User Manual

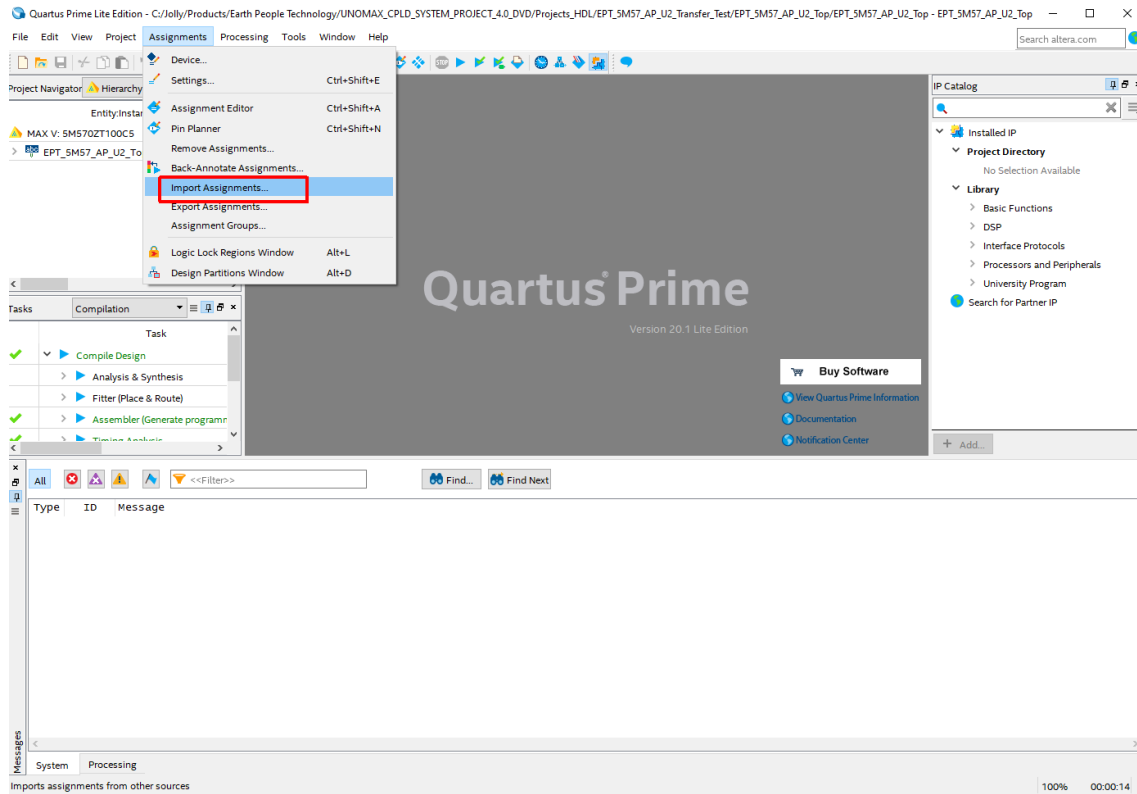


Next, we will select the pins and synthesize the project.

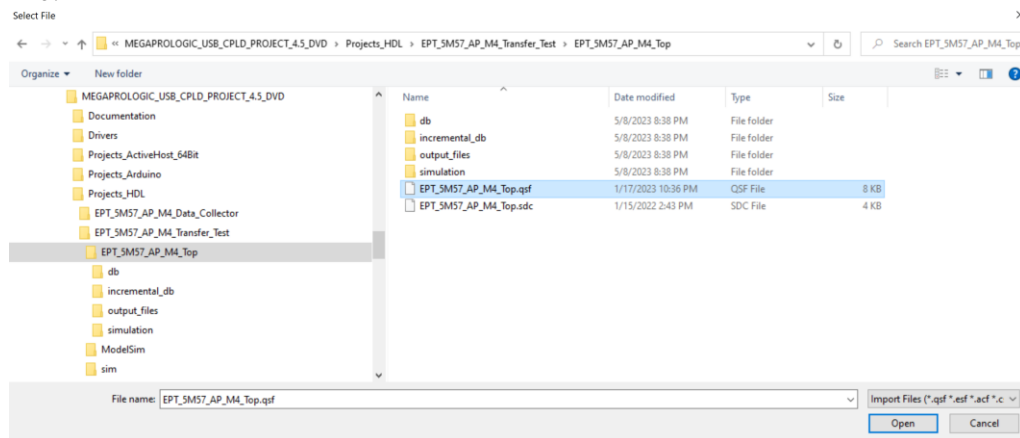
4.1.1 Selecting Pins and Synthesizing

With the project created, we need to assign pins to the project. The signals defined in the top level file (in this case: EPT_5M57_AP_M4_Top.v) will connect directly to pins on the CPLD. The Pin Planner Tool from Quartus Prime will add the pins and check to verify that our pin selections do not violate any restrictions of the device. In the case of this example we will import pin assignments that created at an earlier time. Under Assignments, Select Import Assignments.

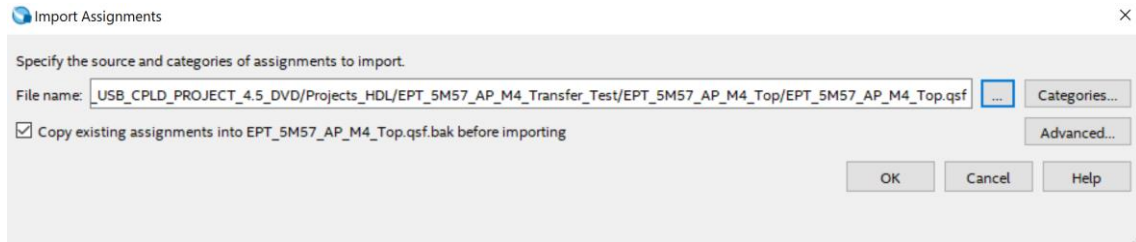
USB CPLD Development System User Manual



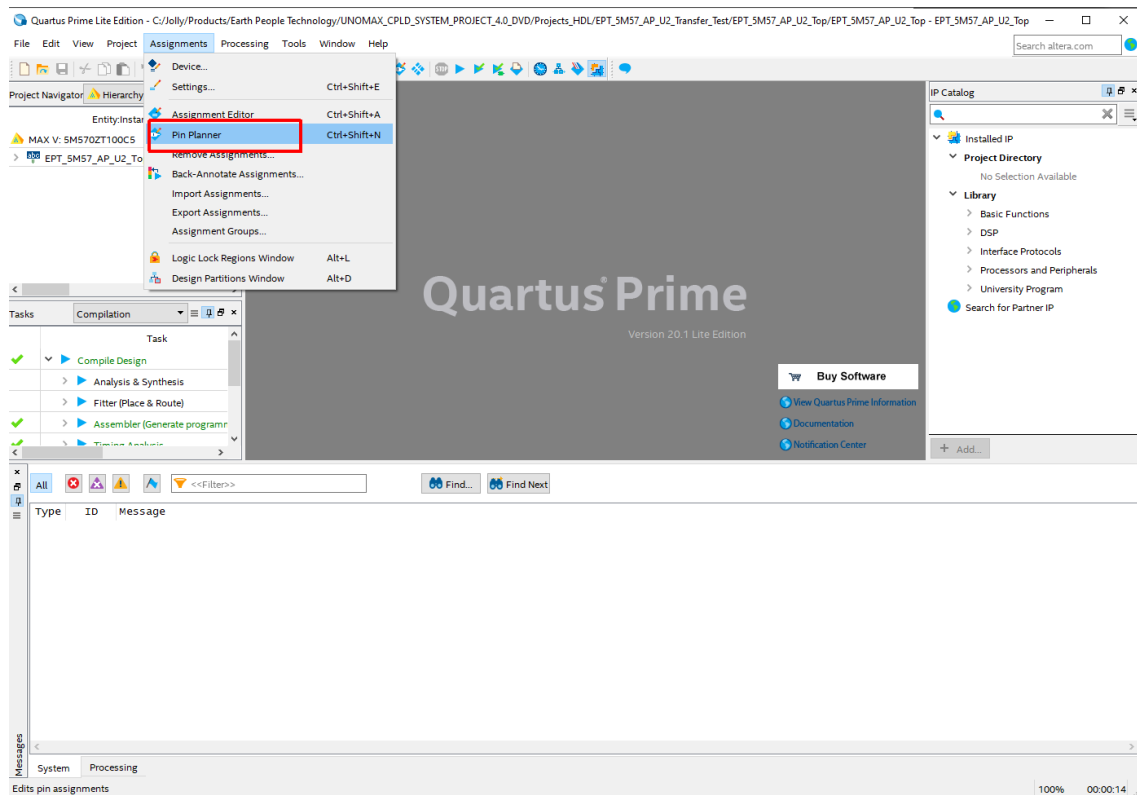
At the Import Assignment dialog box, Browse to the \Projects_HDL\EPT_Transfer_Test\EPT_MegaProLogic_TOP folder of the MegaProLogic Development System DVD. Select the “EPT_5M57_AP_M4_Top.qsf” file.



USB CPLD Development System User Manual

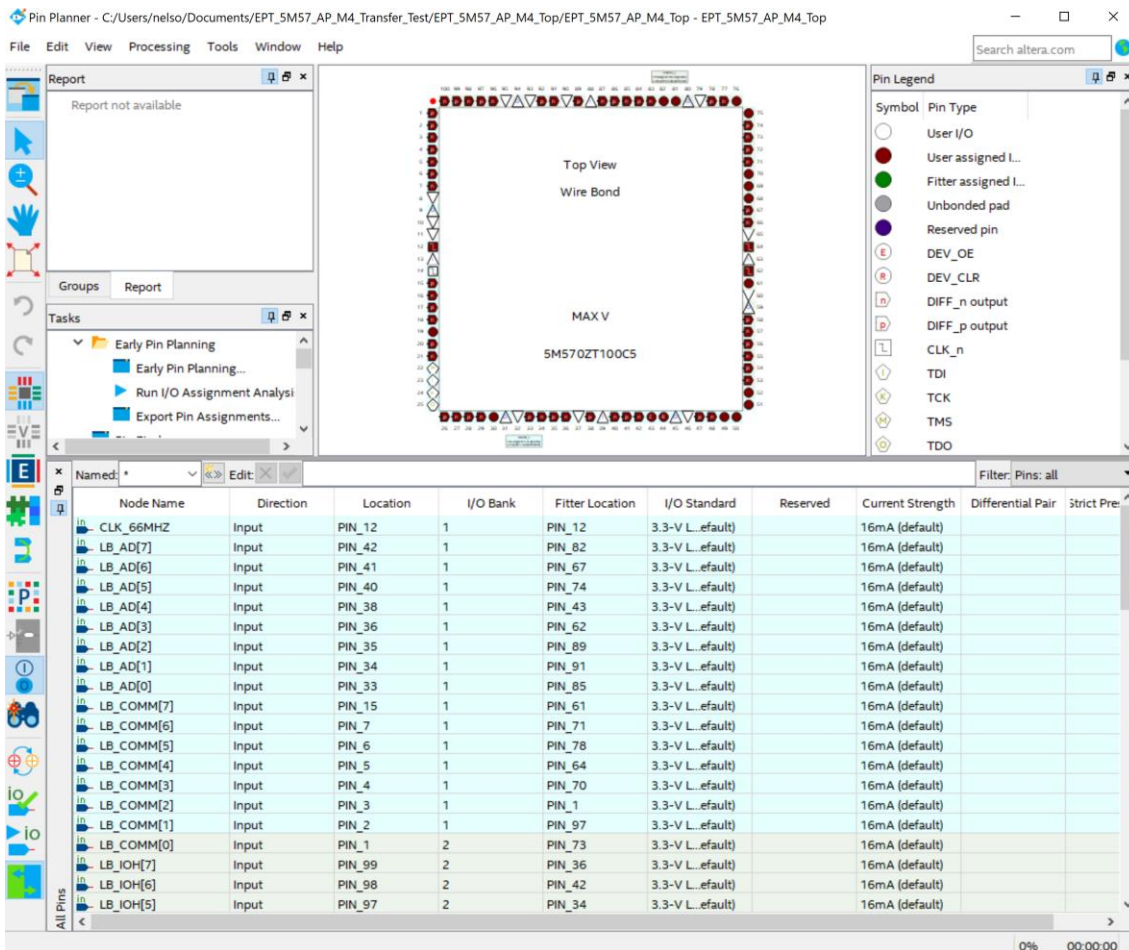


Click Ok. Under Assignments, Select Pin Planner. Verify the pins have been imported correctly.



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

USB CPLD Development System User Manual



The screenshot shows the Pin Planner software interface. The main window displays a top view of a device with wire bonds. The device is labeled "MAX V" and "5M570ZT100C5". The wire bonds are shown as red and green lines connecting the device pins to external components. The interface includes a menu bar (File, Edit, View, Processing, Tools, Window, Help), a search bar, and a Pin Legend on the right. The Pin Legend lists various pin types and their symbols, including User I/O, User assigned I/O, Fitter assigned I/O, Unbonded pad, Reserved pin, DEV_OE, DEV_CLR, DIFF_n output, DIFF_p output, CLK_n, TDI, TCK, TMS, and TDO.

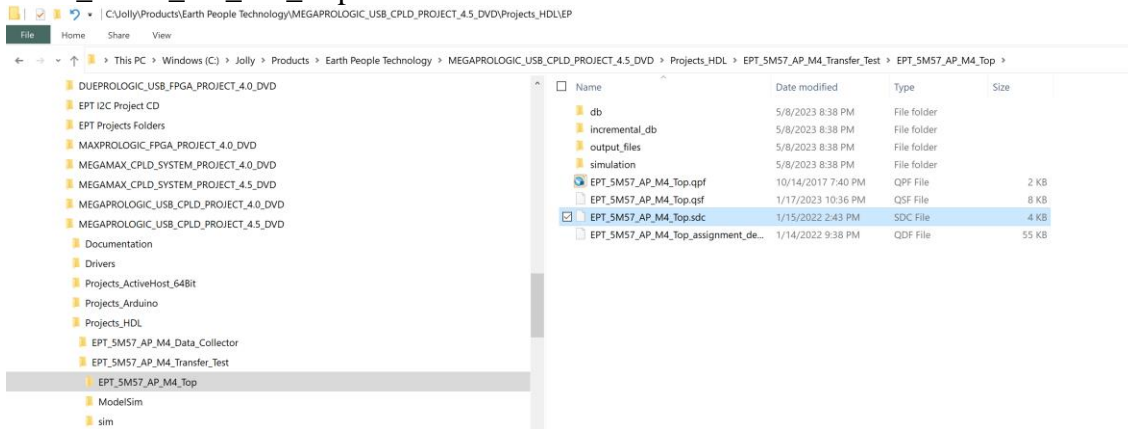
Node Name	Direction	Location	I/O Bank	Fitter Location	I/O Standard	Reserved	Current Strength	Differential Pair	Strict Pre
CLK_66MHZ	Input	PIN_12	1	PIN_12	3.3-V L_default		16mA (default)		
LB_AD[7]	Input	PIN_42	1	PIN_82	3.3-V L_default		16mA (default)		
LB_AD[6]	Input	PIN_41	1	PIN_67	3.3-V L_default		16mA (default)		
LB_AD[5]	Input	PIN_40	1	PIN_74	3.3-V L_default		16mA (default)		
LB_AD[4]	Input	PIN_38	1	PIN_43	3.3-V L_default		16mA (default)		
LB_AD[3]	Input	PIN_36	1	PIN_62	3.3-V L_default		16mA (default)		
LB_AD[2]	Input	PIN_35	1	PIN_89	3.3-V L_default		16mA (default)		
LB_AD[1]	Input	PIN_34	1	PIN_91	3.3-V L_default		16mA (default)		
LB_AD[0]	Input	PIN_33	1	PIN_85	3.3-V L_default		16mA (default)		
LB_COMM[7]	Input	PIN_15	1	PIN_61	3.3-V L_default		16mA (default)		
LB_COMM[5]	Input	PIN_7	1	PIN_71	3.3-V L_default		16mA (default)		
LB_COMM[5]	Input	PIN_6	1	PIN_78	3.3-V L_default		16mA (default)		
LB_COMM[4]	Input	PIN_5	1	PIN_64	3.3-V L_default		16mA (default)		
LB_COMM[3]	Input	PIN_4	1	PIN_70	3.3-V L_default		16mA (default)		
LB_COMM[2]	Input	PIN_3	1	PIN_1	3.3-V L_default		16mA (default)		
LB_COMM[1]	Input	PIN_2	1	PIN_97	3.3-V L_default		16mA (default)		
LB_COMM[0]	Input	PIN_1	2	PIN_73	3.3-V L_default		16mA (default)		
LB_IOH[7]	Input	PIN_99	2	PIN_36	3.3-V L_default		16mA (default)		
LB_IOH[6]	Input	PIN_98	2	PIN_42	3.3-V L_default		16mA (default)		
LB_IOH[5]	Input	PIN_97	2	PIN_34	3.3-V L_default		16mA (default)		

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

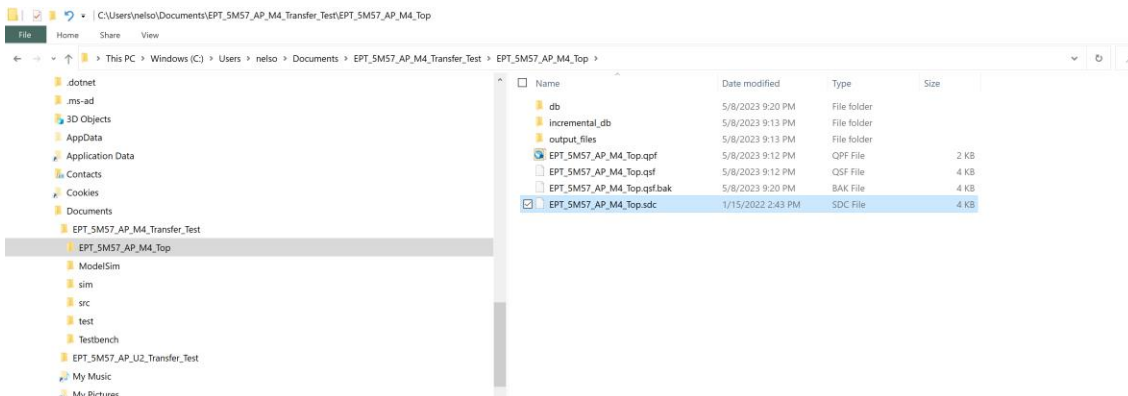
https://ftp.intel.com/Public/Pub/fpgaup/pub/Teaching_Materials/current/Tutorials/VHDL/Timequest.pdf

USB CPLD Development System User Manual

Browse to the \Projects_HDL\EPT_Transfer_Test \EPT_MegaProLogic_TOP folder of the MegaProLogic Development System DVD. Select the “EPT_5M57_AP_M4_Top.sdc” file.

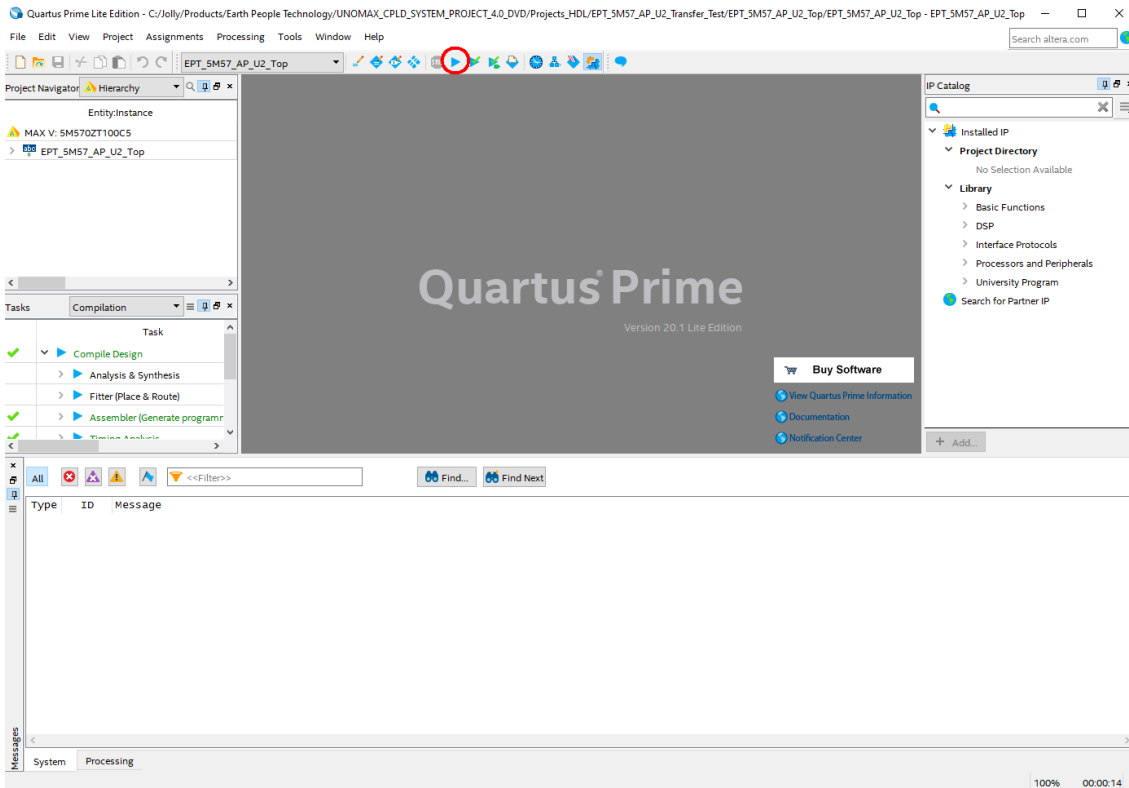


Copy the file and browse to C:\Users\nelso\Documents\EPT_5M57_AP_M4_Transfer_Test\EPT_5M57_AP_M4_Top directory. Paste the file.

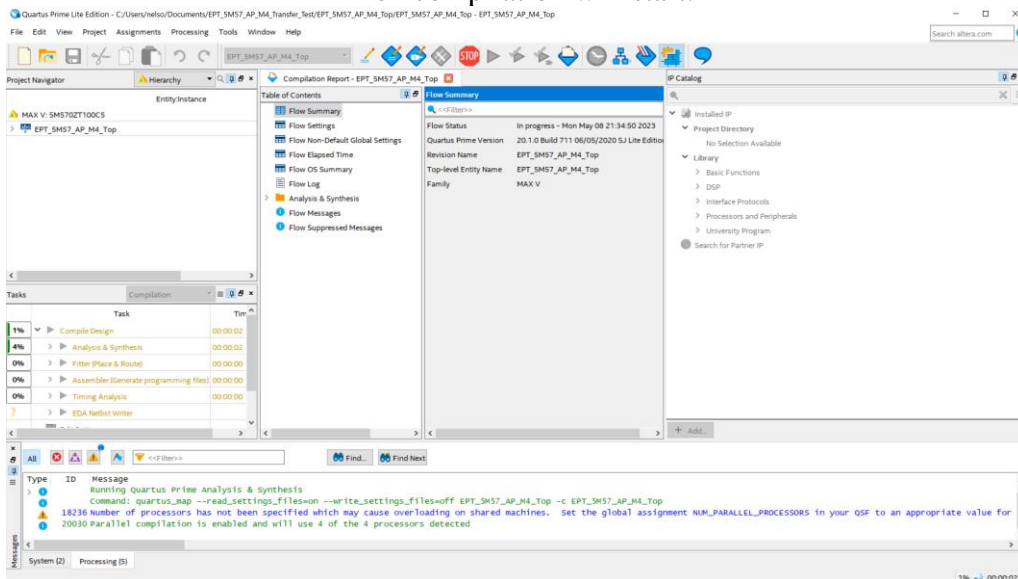


Select the Start Compilation button.

USB CPLD Development System User Manual

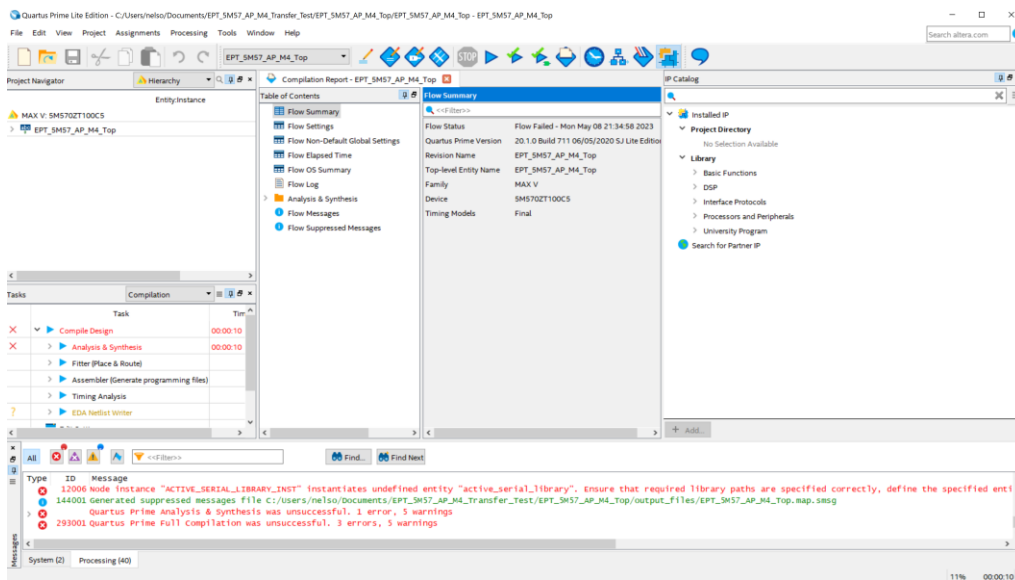


Then compilation will start.



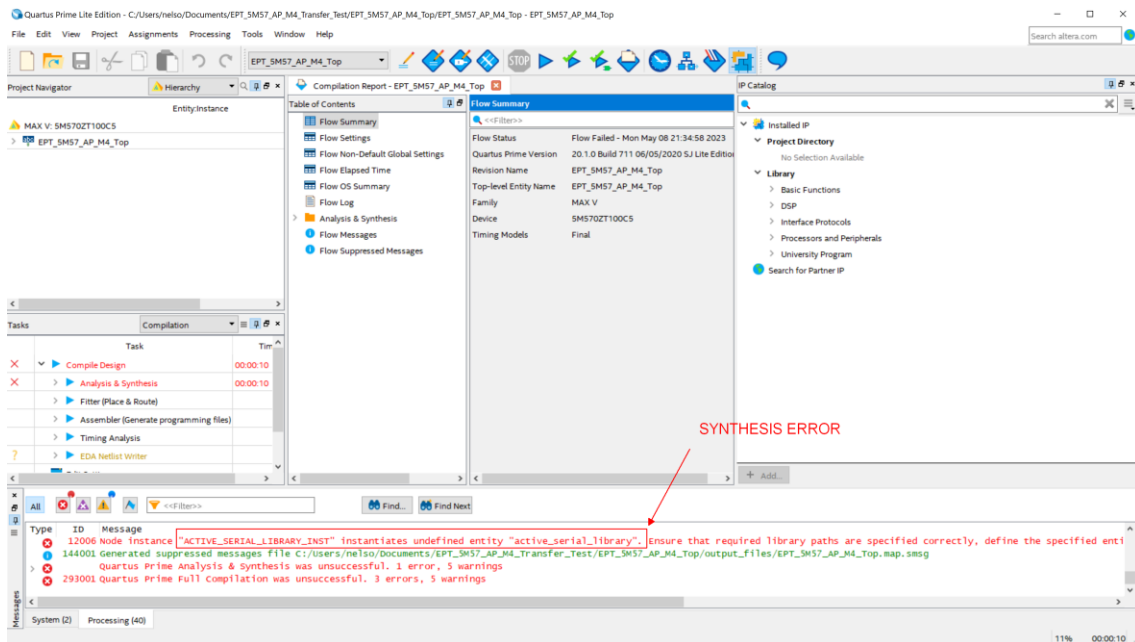
USB CPLD Development System User Manual

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



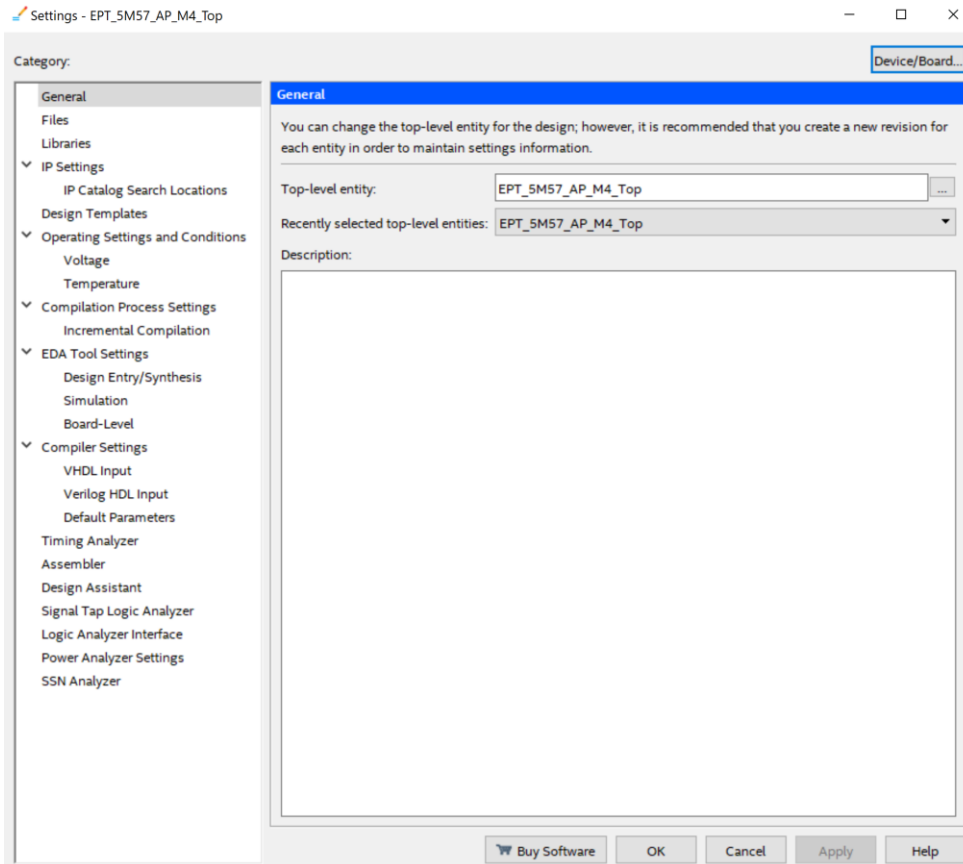
Click on the “Processing” tab at bottom to see the error.

USB CPLD Development System User Manual



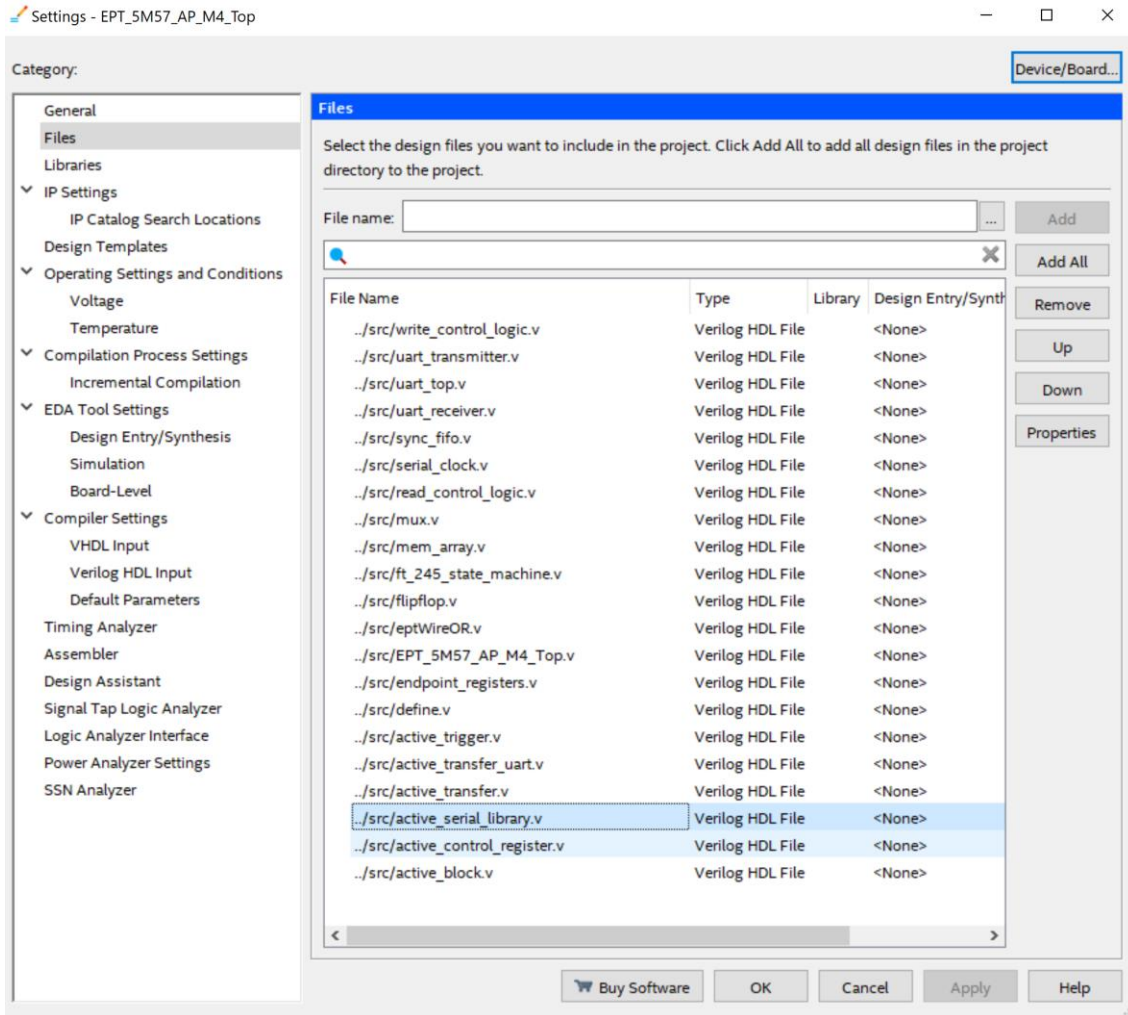
The error in this case is the missing file “active_serial_library.v”. Click on the Assignment menu,

USB CPLD Development System User Manual



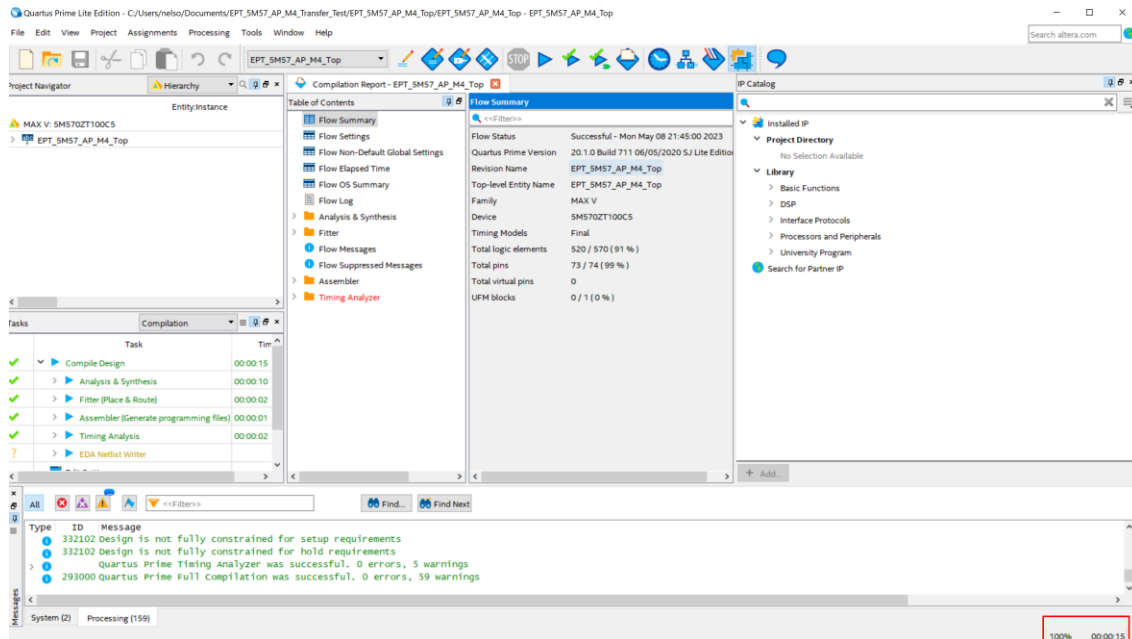
then select Settings, then select Files. Add the “active_serial_library.v” file from the database.

USB CPLD Development System User Manual



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

USB CPLD Development System User Manual



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.

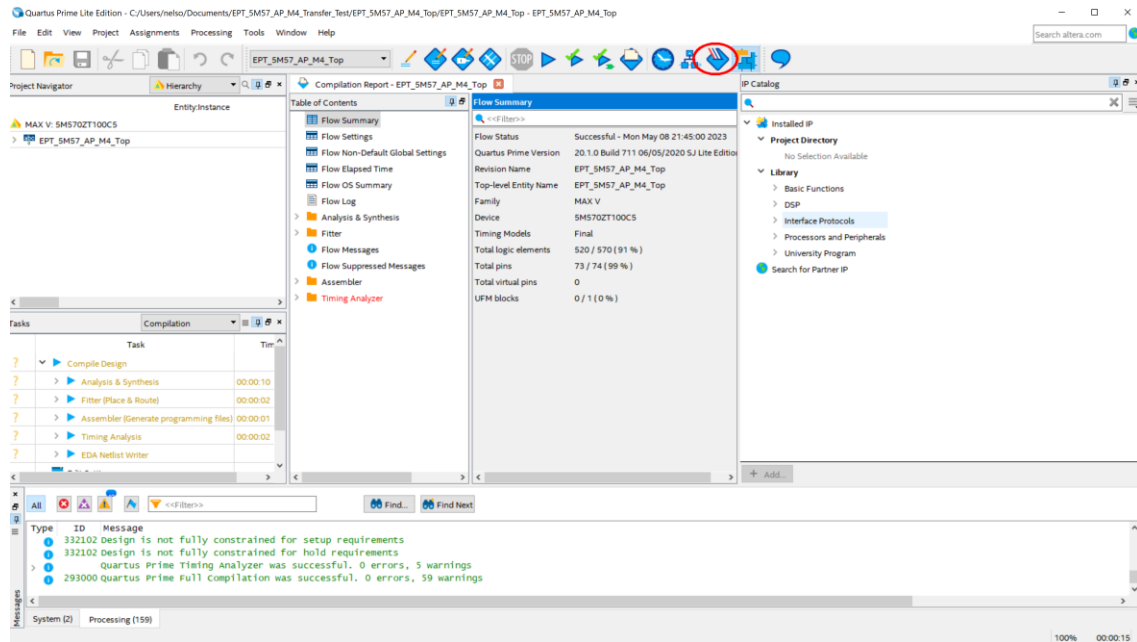
4.1.2 Programming the CPLD

Programming the CPLD is quick and easy. All that is required is a standard USB-C cable. Connect the MegaProLogic to the PC, open up Quartus Prime, 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.

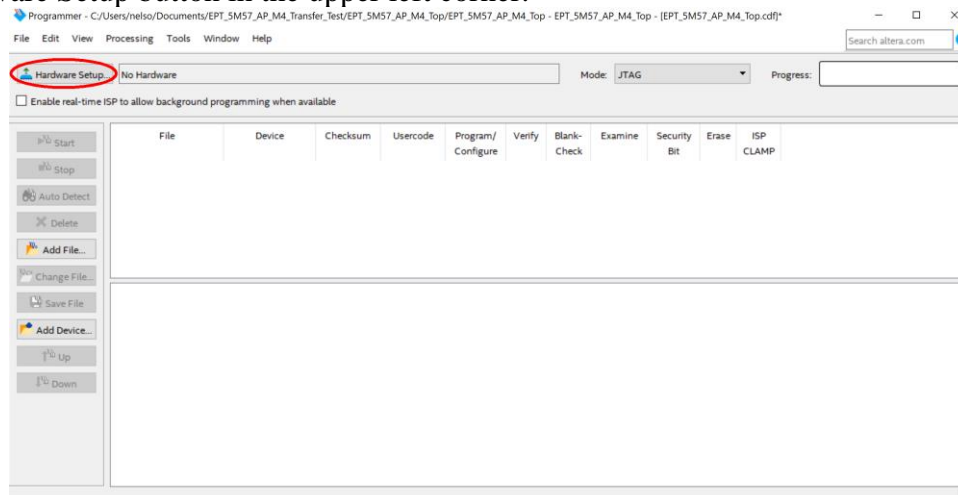


USB CPLD Development System User Manual

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

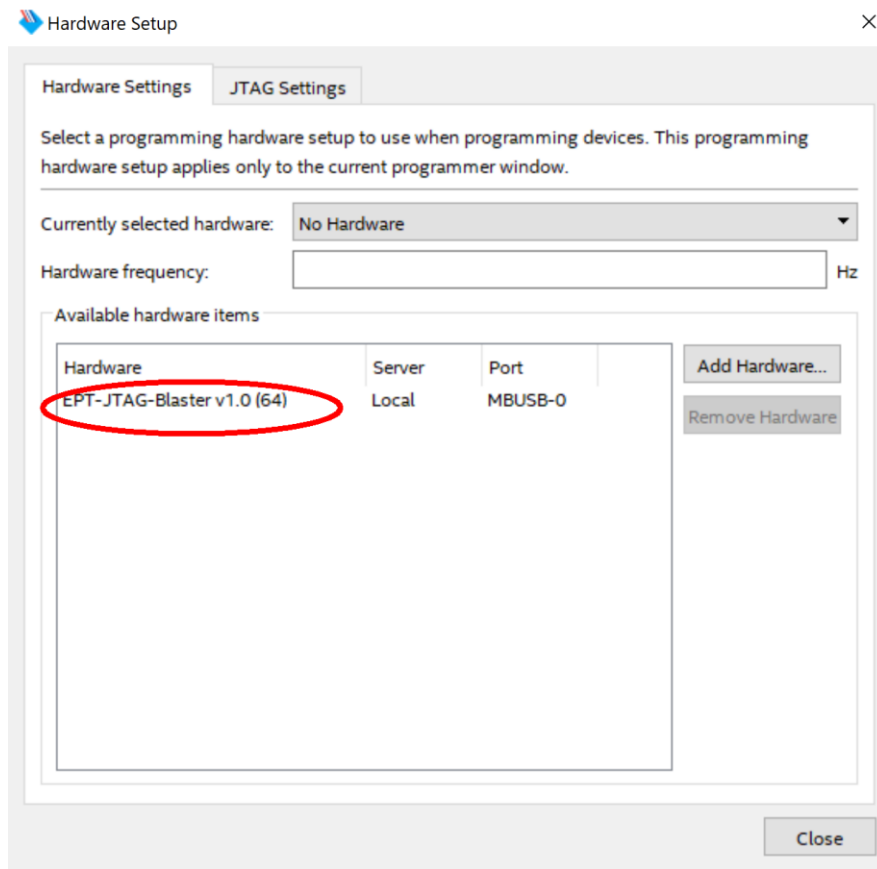


The Programmer Window will open up with the programming file selected. Click on the Hardware Setup button in the upper left corner.



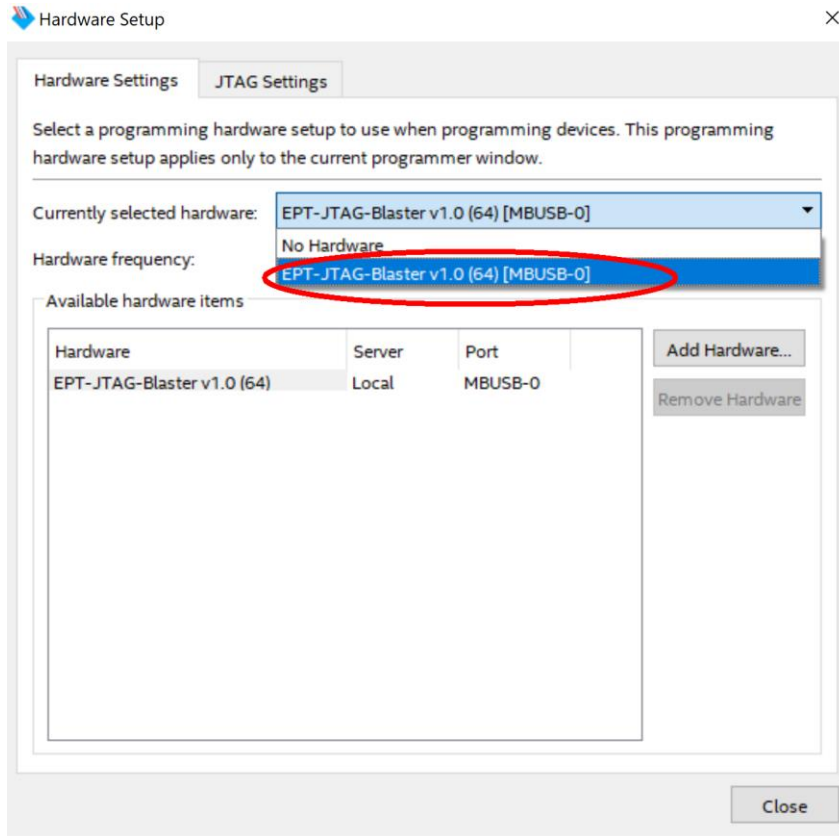
USB CPLD Development System User Manual

The Hardware Setup Window will open. In the “Available hardware items”, double click on “EPT-Blaster v1.0”.

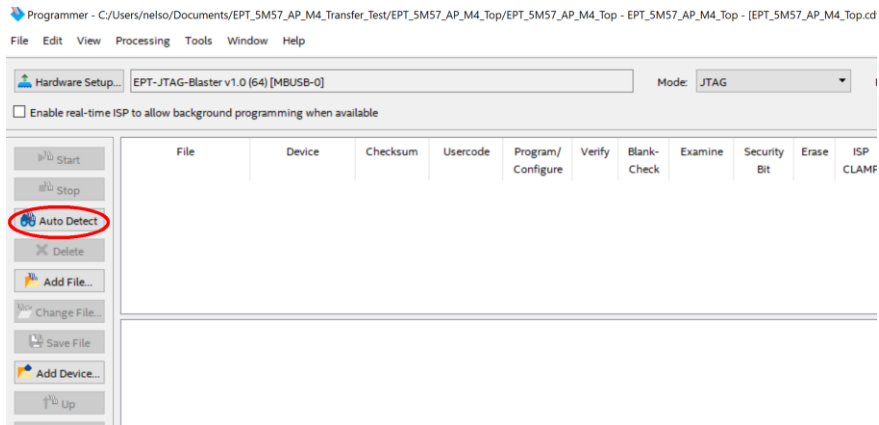


If you successfully double clicked, the “Currently selected hardware:” dropdown box will show the “EPT-Blaster v1.0 (64)”.

USB CPLD Development System User Manual

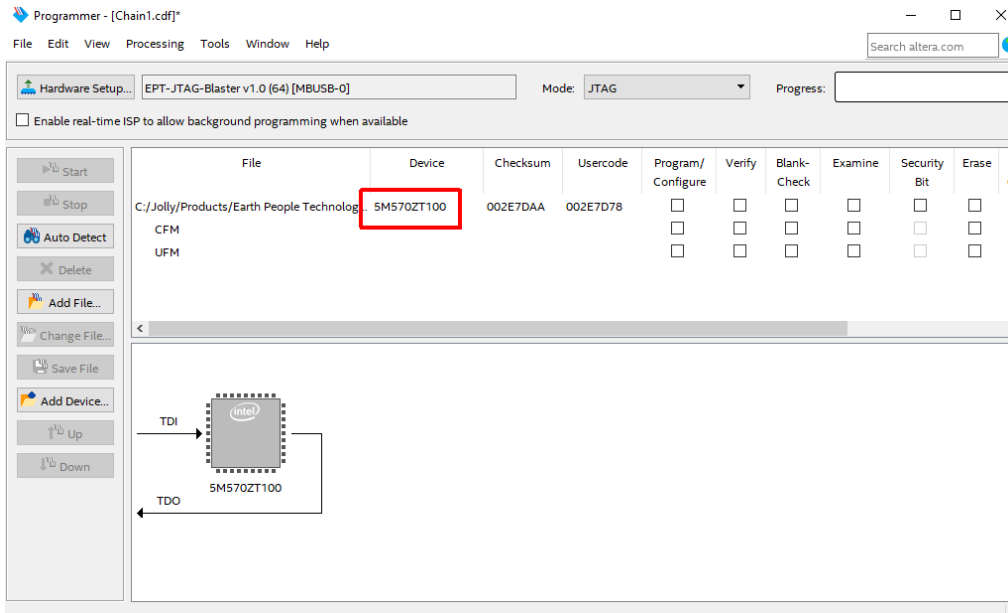


Click on the Auto-Detect button. This will verify that the EPT-Blaster driver can connect with the MegaProLogic device.

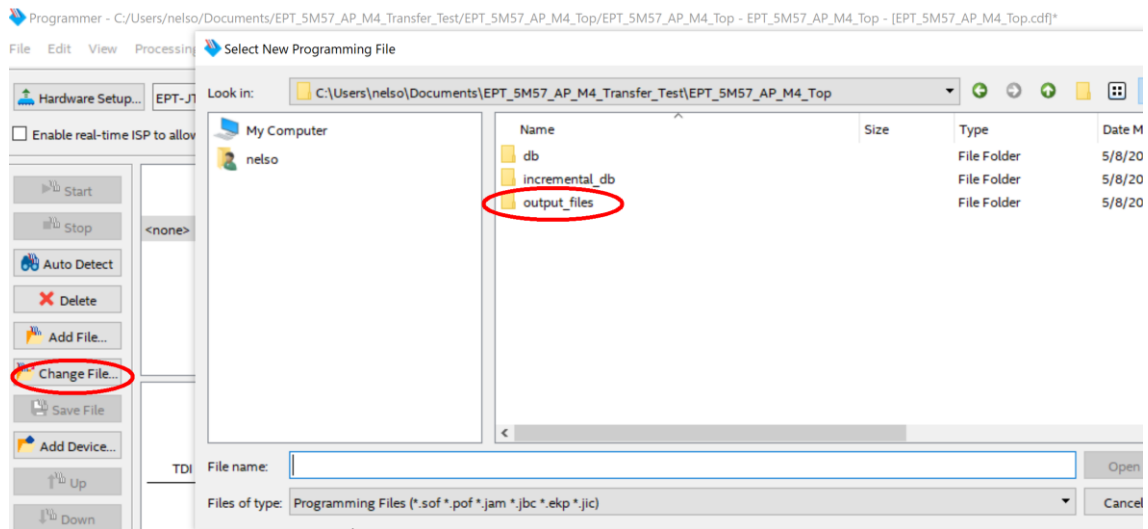


USB CPLD Development System User Manual

Select the 5M570 under “Device”.

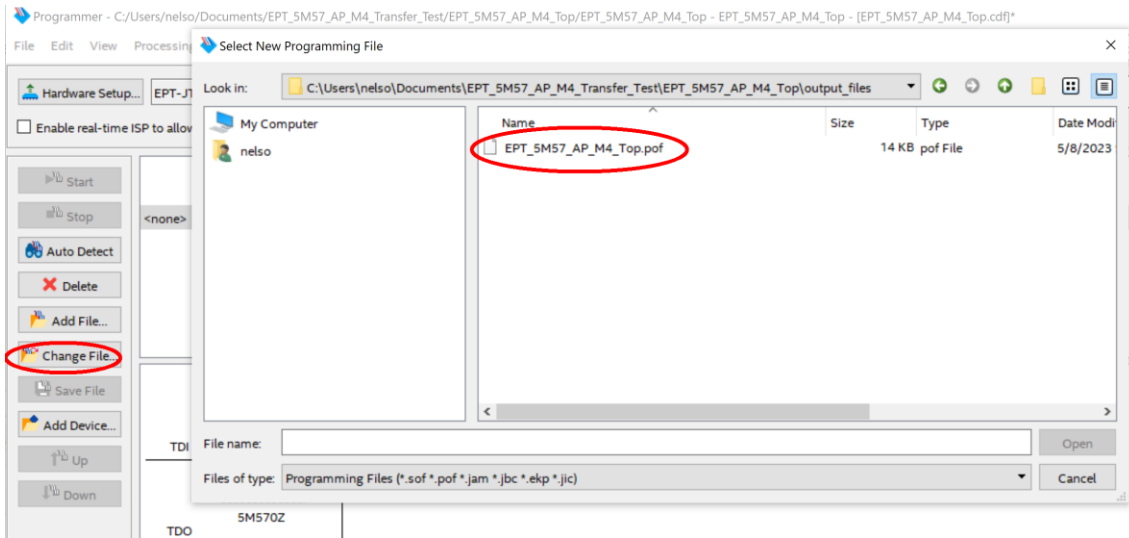


Click on the “Change File” button and browse to the output_files folder.



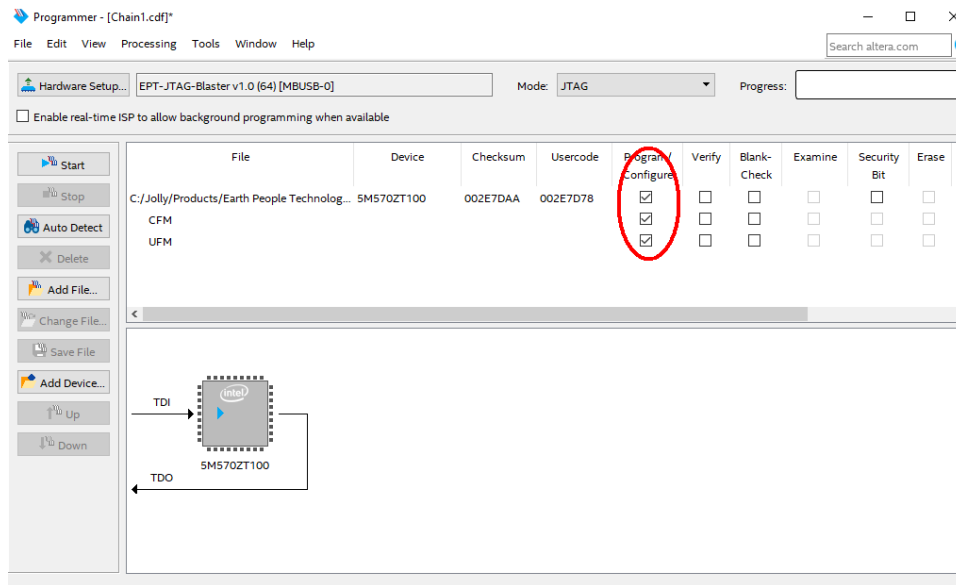
Click on the EPT_5M57_AP_M4_Top.pof file to select it.

USB CPLD Development System User Manual



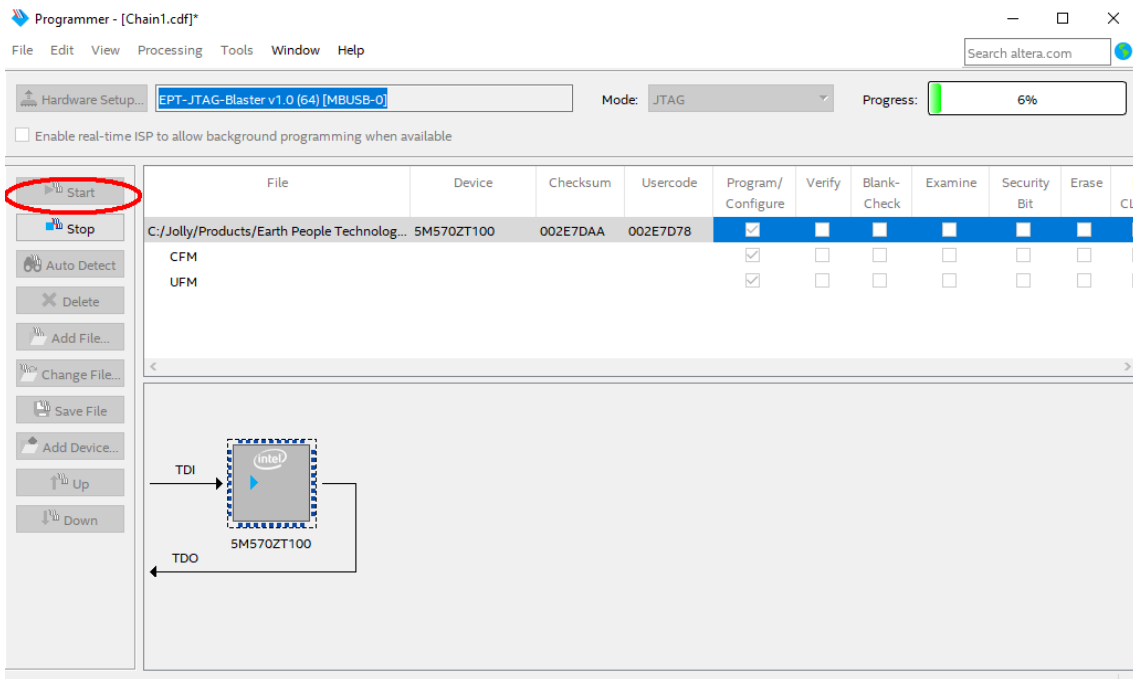
Click the Open button in the lower right corner.

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



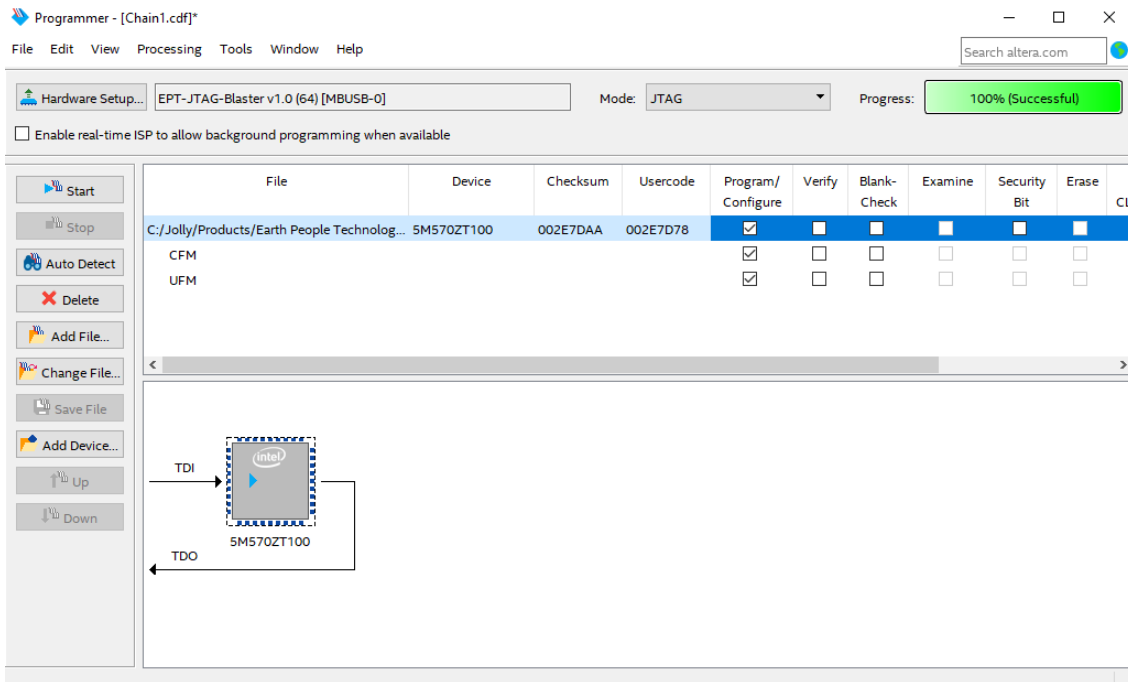
Click on the Start button to start programming the CPLD. The Progress bar will indicate the progress of programming.

USB CPLD Development System User Manual



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

USB CPLD Development System User Manual



At this point, the MegaProLogic 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 connect to the Arduino Uno and write some code to transfer data between microcontroller and PC.

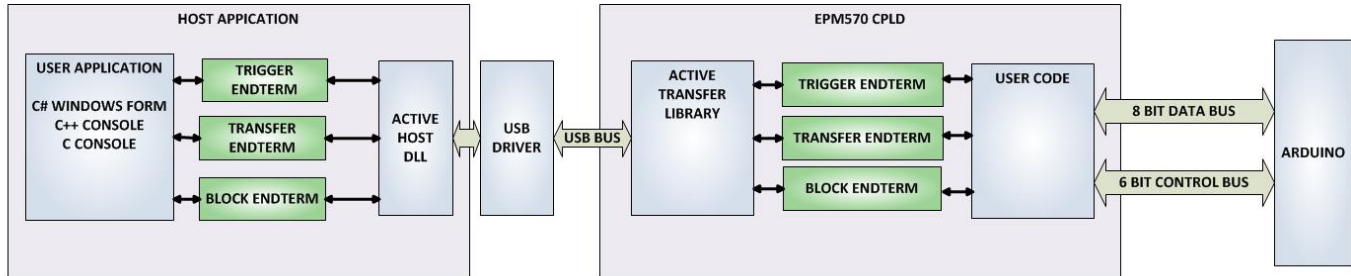
5 Active Host Application

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

USB CPLD Development System User Manual

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

USB CPLD Development System User Manual

circular buffer in memory. This allows the user code to fetch the transfers with out losing bytes.

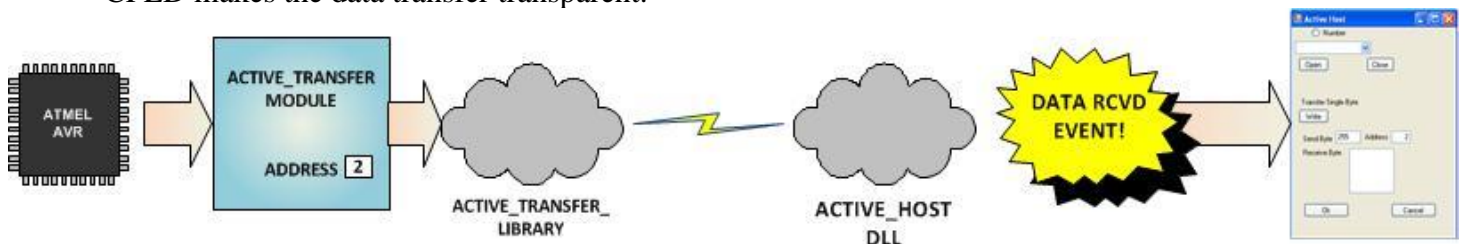
5.3 Block EndTerm

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

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



USB CPLD Development System User Manual

- EPT_AH_SendTrigger ()
- EPT_AH_SendByte ()
- EPT_AH_SendBlock ()
- EPT_AH_SendTransferControlByte()

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.

USB CPLD Development System User Manual

```
private void EPT_Transfer_Test_Load(object sender, System.EventArgs e)
{
    //String buffer
    String PortText = "";

    //Index registers
    int Index = 0, EPTgroupNumber = 0;

    // Call the List Devices function
    List<string> names = ComPortNames("0403", "6010");

    // Get a list of serial port names.
    string[] ports;
    ports = SerialPort.GetPortNames();

    if (names.Count > 0)
    {
        foreach (String port in ports)
        {
            //Compare port name with the found VID/PID
            //combinations. Add them to Matching port list
            //and comboDevList
            if (names.Contains(port))
            {
                MatchingComPortList[Index] = port;

                if (Index == 0)
                {
                    PortText = "EPT JTAG Blaster " + EPTgroupNumber;
                    Index++;
                }
                else
                {
                    PortText = "EPT Serial Communications " + EPTgroupNumber++;
                    Index++;
                }
                cmbDevList.Items.Add(PortText);
            }
        }
    }
    else
        MessageBox.Show("No EPT Devices found!");

    //SetButtonEnables_Close();
}
```

The ListDevices() function calls the

```
ports = SerialPort.GetPortNames();
```

USB CPLD Development System User Manual

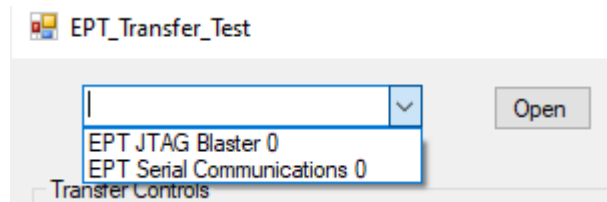
to determine the Serial devices attached to the PC. Next,

```
if (names.Contains(port))
```

is called inside a for loop to return the ASCII name of each Serial device attached to the PC. It will automatically populate the combo box, cmbDevList with all the EPT devices it finds.

```
cmbDevList.Items.Add(PortText);
```

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

```
OpenSerialPort1()
```

function. The device_index is passed into the function. If the function is successful, the device name is displayed in the label, labelDeviceCnt. Next, the Open button is grayed out and the Close button is made active.

USB CPLD Development System User Manual

```

1 reference
public bool OpenSerialPort1()
{
    try
    {
        //Set the serial port parameters
        serialPort_AH.PortName = PortName;
        serialPort_AH.BaudRate = Convert.ToInt32(BaudRate);
        serialPort_AH.Parity = (Parity)Enum.Parse(typeof(Parity), vParity);
        serialPort_AH.DataBits = Convert.ToInt16(DataBits);
        serialPort_AH.StopBits = (StopBits)Enum.Parse(typeof(StopBits), StopBits);
        serialPort_AH.Handshake = (Handshake)Enum.Parse(typeof(Handshake), pHandshake);

        if (!serialPort_AH.IsOpen)
        {
            serialPort_AH.Open();
            btnOpenDevice.Enabled = false;
            btnCloseDevice.Enabled = true;
            //textBox1.ReadOnly = false;
            return true;
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    return false;
}

```

5.4.2 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 OpenSerialPort1 (). 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



USB CPLD Development System User Manual

- TRANSFER_IN
- BLOCK_IN

```
public void EPT_AH_Receive(byte[] receiveBytes)
{
    uint c, a, p, l, index;
    //Compare first byte to the incoming message code
    string s = String.Empty;
    foreach (byte b in receiveBytes)
    {
        s += String.Format("{0:x2}", (int)System.Convert.ToUInt32(b.ToString()));
        s += "\r\n";
    }
    //tbBlockRcv.AppendText(s);
    //this.Invoke(new MethodInvoker(delegate () { textBox1.AppendText(s); }));

    //Write the command into the EPTReceiveDevice
    c = (uint)receiveBytes[0];
    c = c & 0xf8;

    //Write the address into EPTReceiveDevice
    a = (uint)receiveBytes[0];
    a = a & 0x07;
    EPTReceiveDevice.Address = a;
    //Display Address to text box
    string r = String.Empty;
    r += String.Format("EPTReceiveDevice Address= {0:x2}", EPTReceiveDevice.Address);
    r += "\r\n";
    //this.Invoke(new MethodInvoker(delegate () { textBox1.AppendText(r); }));

    switch (c)
    {
        case 0xc8:
            //this.Invoke(new MethodInvoker(delegate () { textBox1.AppendText("Trigger Recieved\r\n"); }));
            EPTReceiveDevice.Command = TRIGGER_IN_COMMAND;
            break;
        case 0xd0:
            //this.Invoke(new MethodInvoker(delegate () { textBox1.AppendText("Transfer Byte Recieved\r\n"); }));
            EPTReceiveDevice.Command = TRANSFER_IN_COMMAND;
            break;
        case 0xe0:
            //this.Invoke(new MethodInvoker(delegate () { textBox1.AppendText("Block Recieved\r\n"); }));
            EPTReceiveDevice.Command = BLOCK_IN_COMMAND;
    }
}
```




USB CPLD Development System User Manual

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

The 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.

5.4.3 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



USB CPLD Development System User Manual

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 `OpenSerialPort1()`. 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 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.

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.



USB CPLD Development System User Manual

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

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

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

5.4.4 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.



USB CPLD Development System User Manual

To send a block, first, open the EPT device to be used with `EPT_AH_OpenDeviceByIndex()`. Next, use the `EPT_AH_SendBlock()` function to send the block. Add the address of the transfer module as the first parameter. Next, place the pointer to the buffer in the second parameter of `EPT_AH_SendBlock()`. Add the length of the buffer as the third parameter. Then execute the function, the entire buffer will be transferred to the USB chip. The data is available at the port of the Active Block module in the user code on the 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



USB CPLD Development System User Manual

received data is then stored into local memory and register bits are set that will indicate data has been received from a particular address. The receive callback function is then called from the Active Host dll. This function start a thread to do something with the block data.

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

```
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.



USB CPLD Development System User Manual

```
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.

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 MEGAPROLOGIC. A standalone project can be range from a simple program to display and send data from the user to/from the Arduino MEGA. Or it can more complex to include receiving data, processing it, and start or end a process on the Arduino. This section will outline the procedures to take an example project and Assemble it, Build it, and Execute it. This guide will focus on writing a Windows Forms application using the C# language for the Microsoft Visual Studio with .NET Framework. This is due to the idea that beginners can write effective Windows applications with the C# .NET Framework. They can focus on a subset of the language which is very similar to the C language. Anything that deviates from the subset of the C language, presented as in the Arduino

USB CPLD Development System User Manual

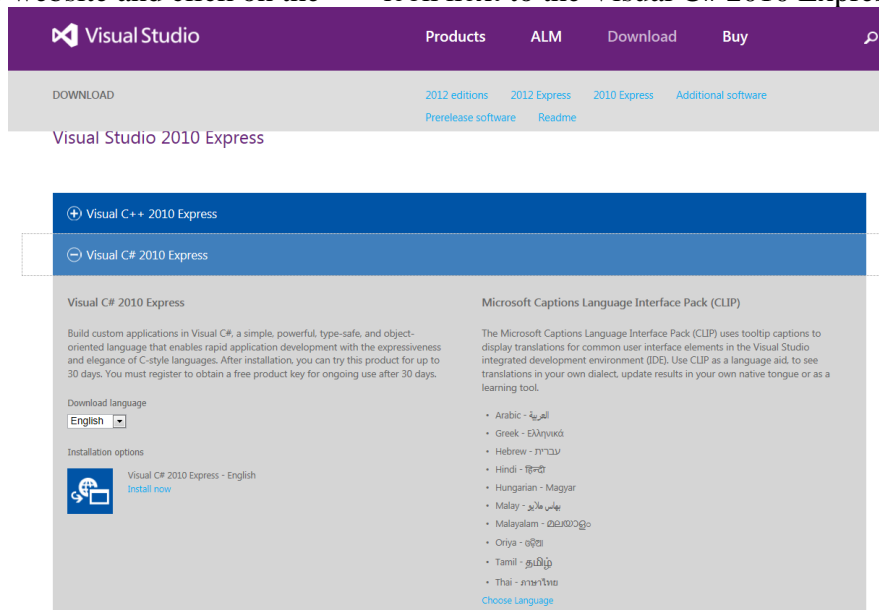
implication (such as events and controls), will be explained as the explanation progresses. Any language can be used with the Active Host Application DLL.

6.1 Creating a Project

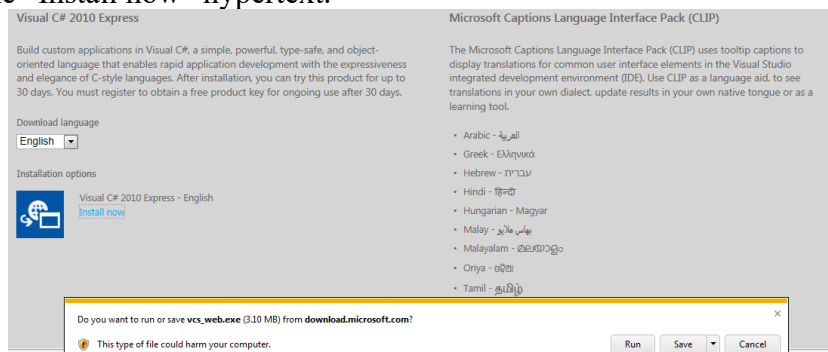
Download the latest version of Microsoft Visual C# 2010 Express environment from Microsoft. It's a free download.

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

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



Click on the “Install now” hypertext.

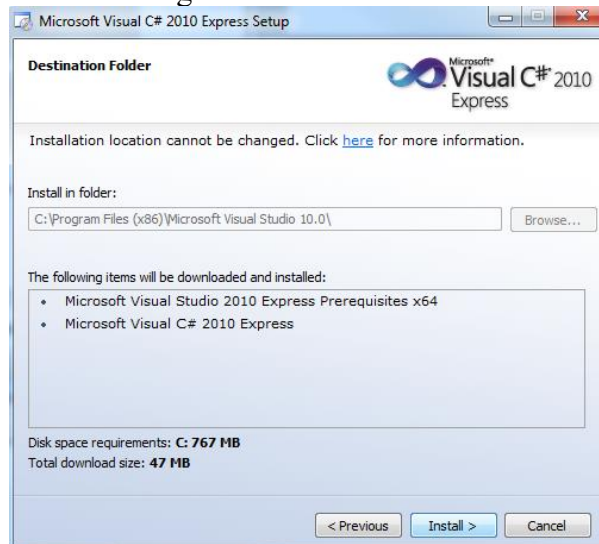


Click the “Run” button.

USB CPLD Development System User Manual



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



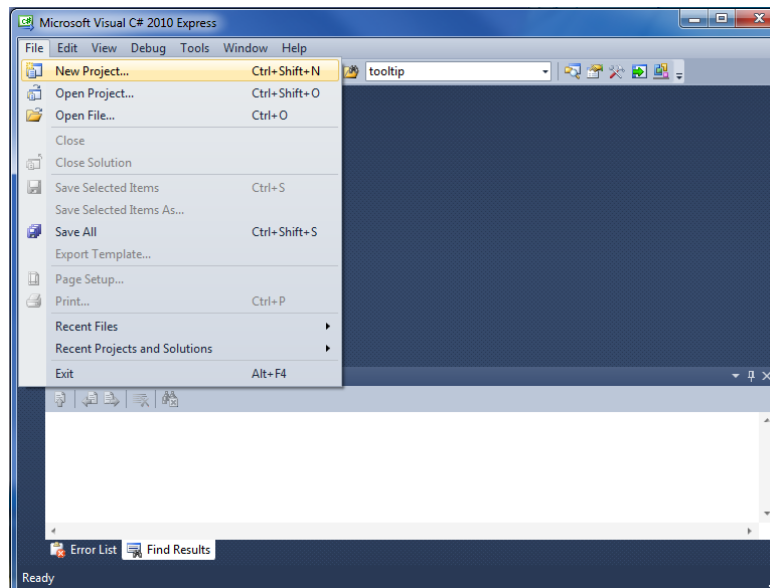
Visual C# 2010 Express will install. This may take up to twenty minutes depending on your internet connection.

USB CPLD Development System User Manual



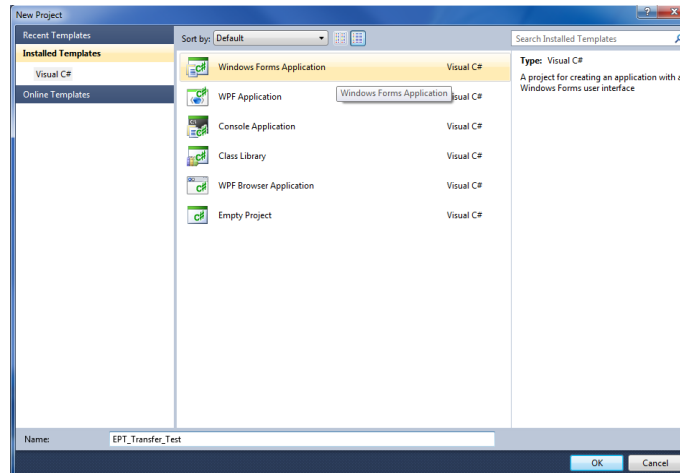
The installed successfully window will be displayed when Visual C# Express is ready to use.

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

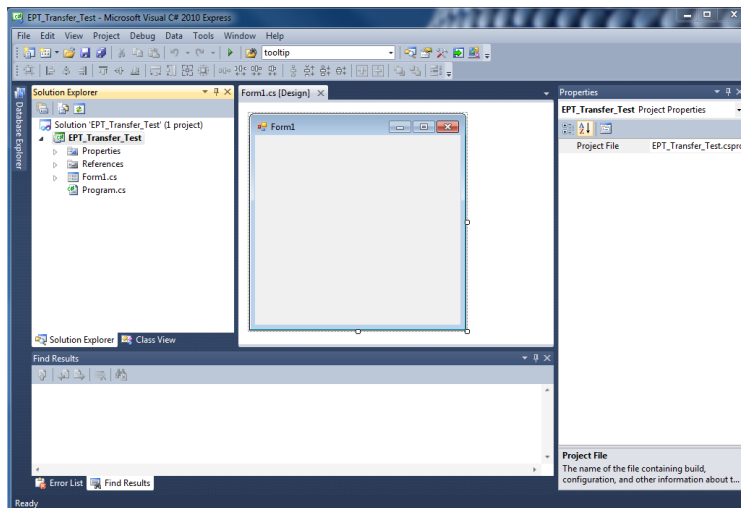


At the New Project window, select the Windows Forms Application. Then, at the Name: box, type in EPT_Transfer_Test

USB CPLD Development System User Manual

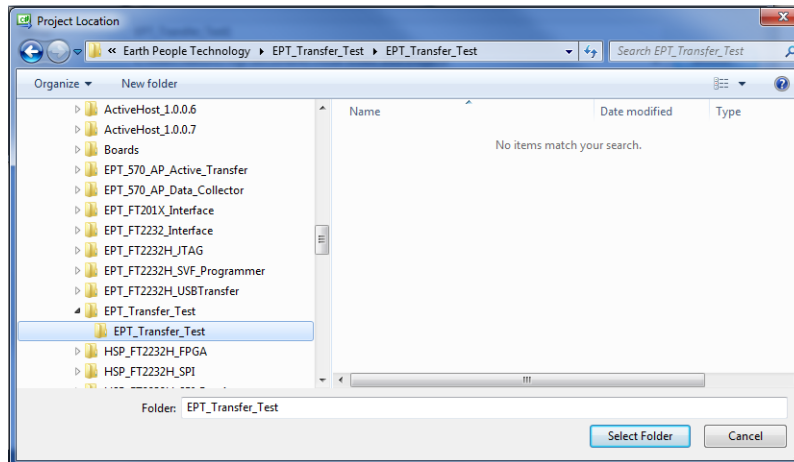


The project creation is complete.



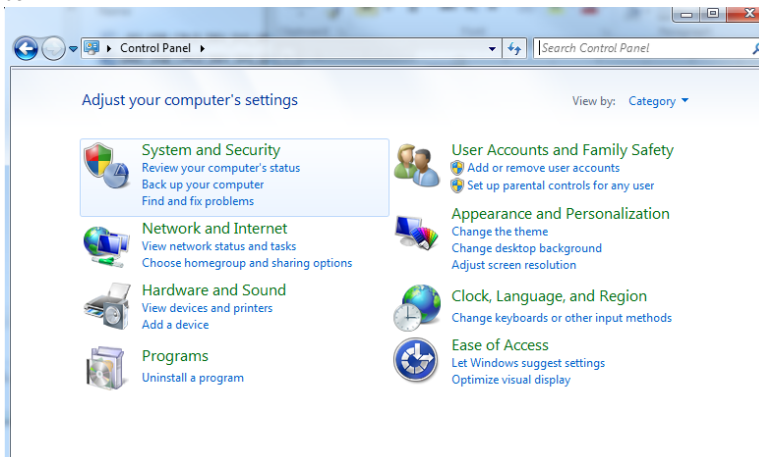
Save the project, go to File->Save as, browse to a folder to create EPT_Transfer_Test folder. The default location is c:\Users\

USB CPLD Development System User Manual



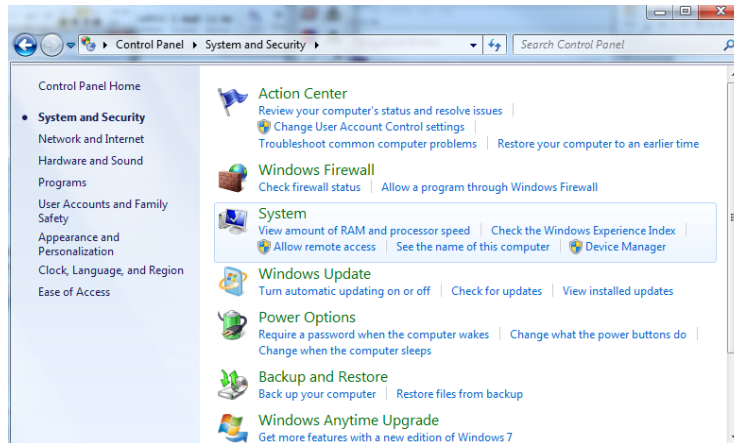
6.1.1 Setting up the C# Express Environment for x64 bit

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

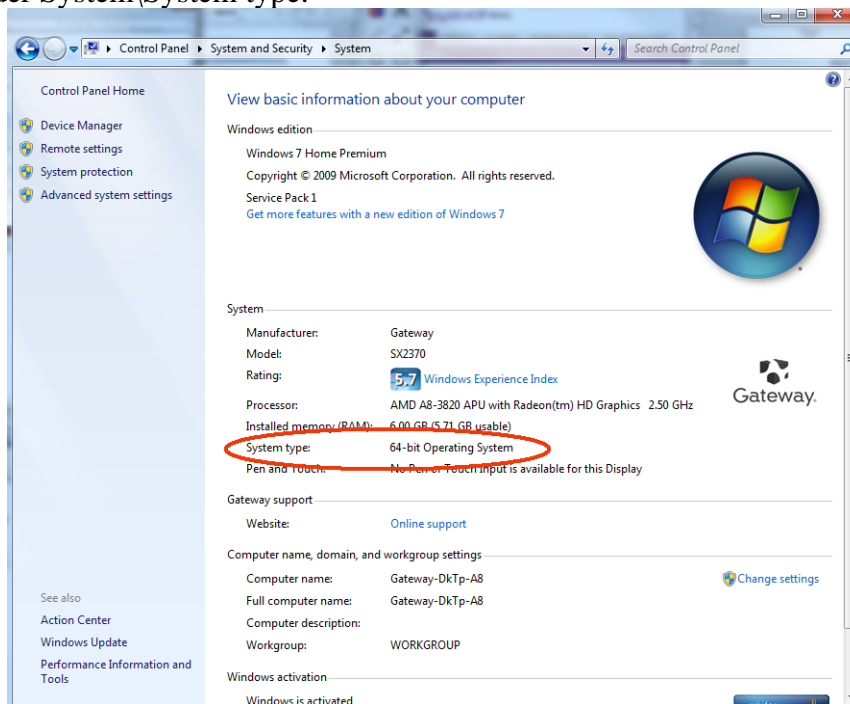


Click on System.

USB CPLD Development System User Manual

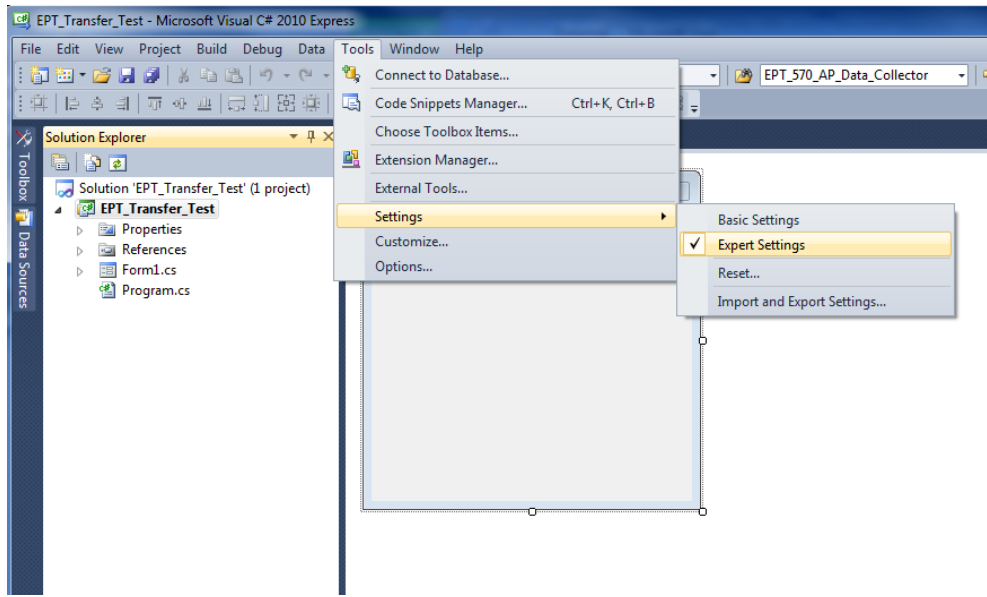


Check under System\System type:

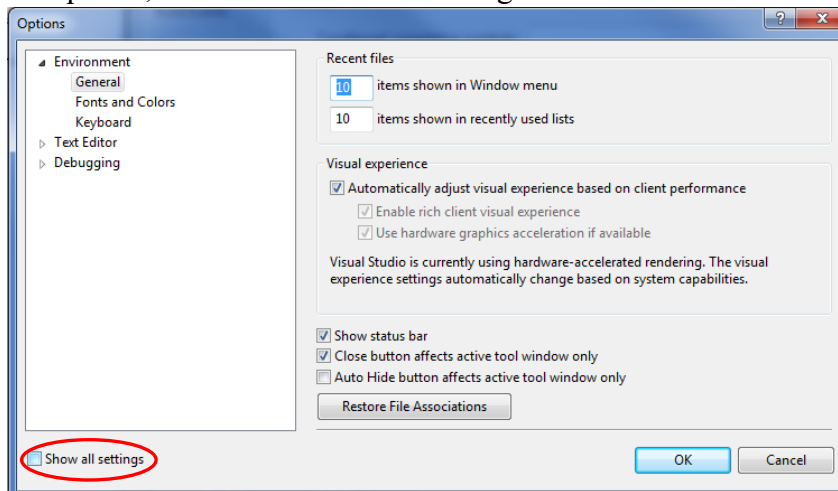


First, we need tell C# Express to produce 64 bit code if we are running on a x64 platform. Go to Tools->Settings and select Expert Settings

USB CPLD Development System User Manual

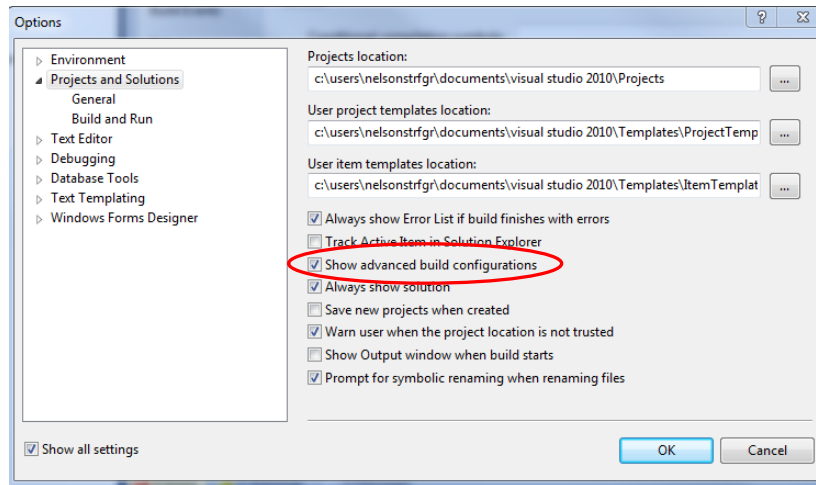


Go to Tools->Options, locate the “Show all settings” check box. Check the box.

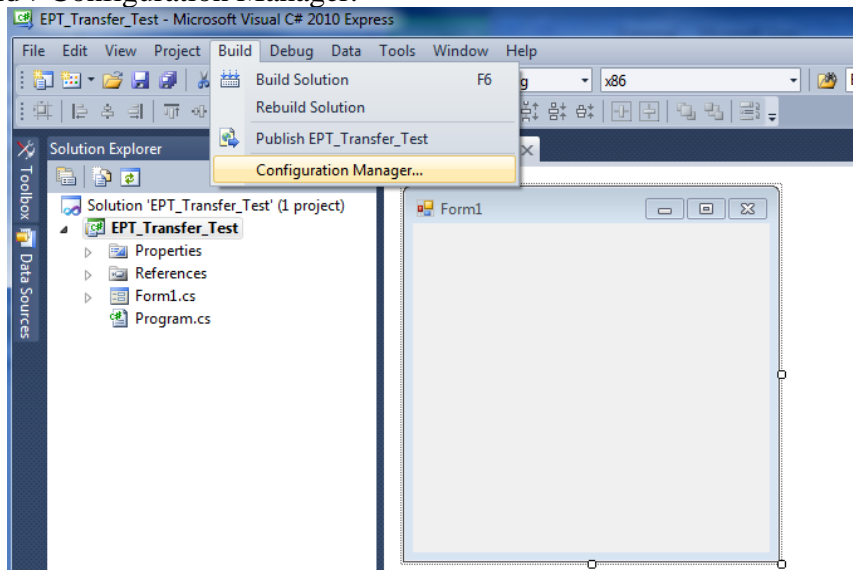


In the window on the left, go to “Projects and Solutions”. Locate the “Show advanced build configurations” check box. Check the box.

USB CPLD Development System User Manual

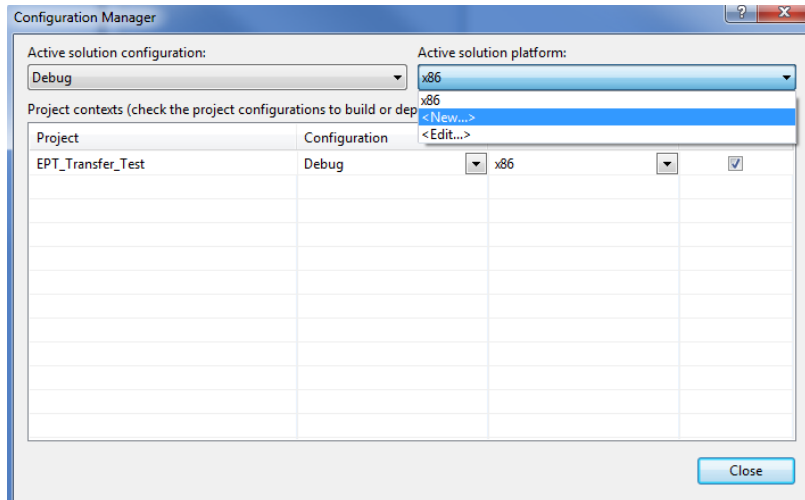


Go to Build->Configuration Manager.

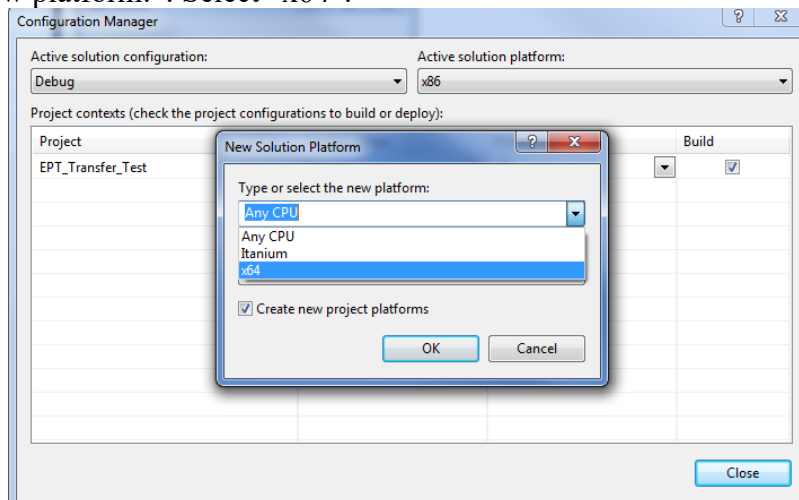


In the Configuration Manager window, locate the “Active solution platform:” label, select “New” from the drop down box.

USB CPLD Development System User Manual

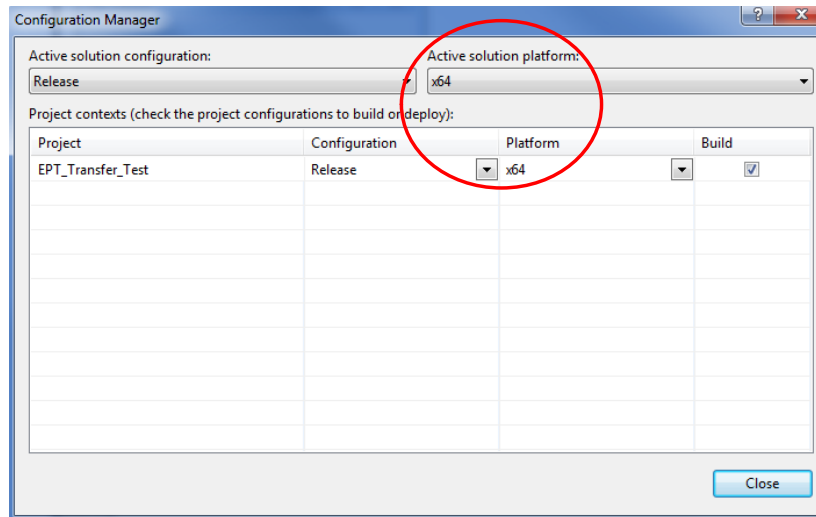


In the New Solution Platform window, click on the drop down box under “Type or select the new platform:”. Select “x64”.

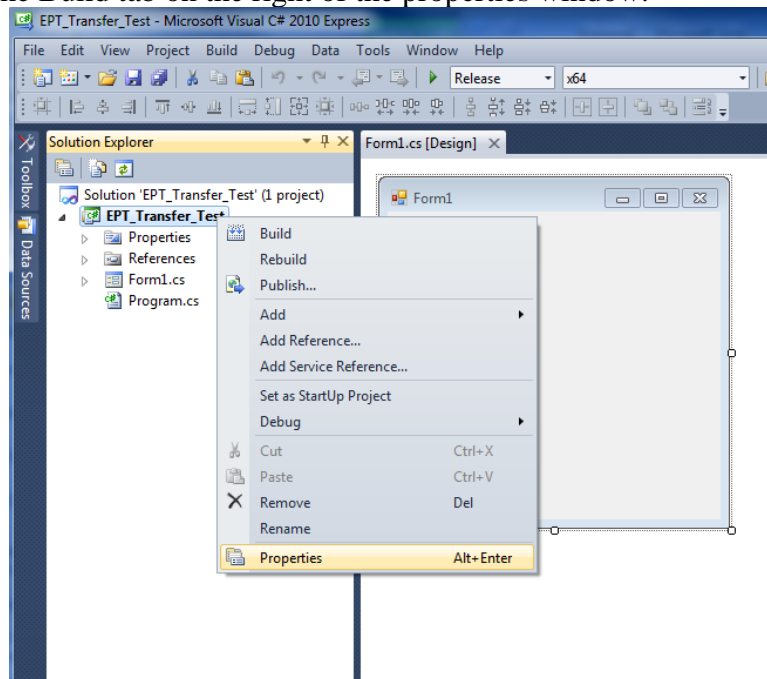


Click the Ok button. Verify that the “Active Solution Platform” and the “Platform” tab are both showing “x64”.

USB CPLD Development System User Manual

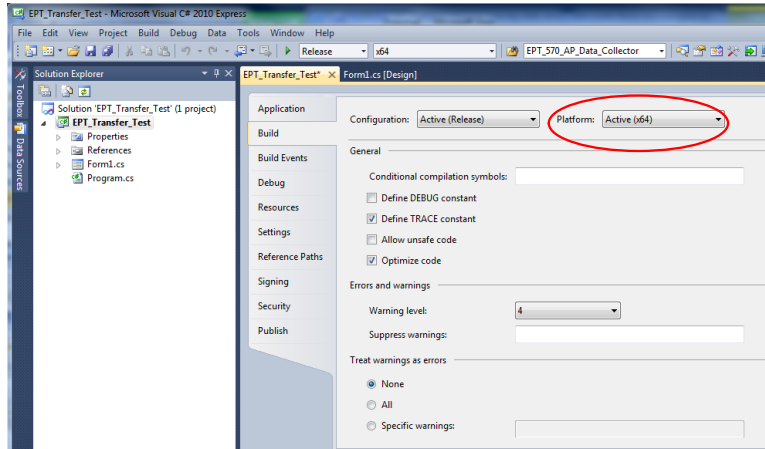


Also, select “Release” under “Active solution configuration”. Click Close. Then, using the Solution Explorer, you can right click on the project, select Properties and click on the Build tab on the right of the properties window.



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

USB CPLD Development System User Manual



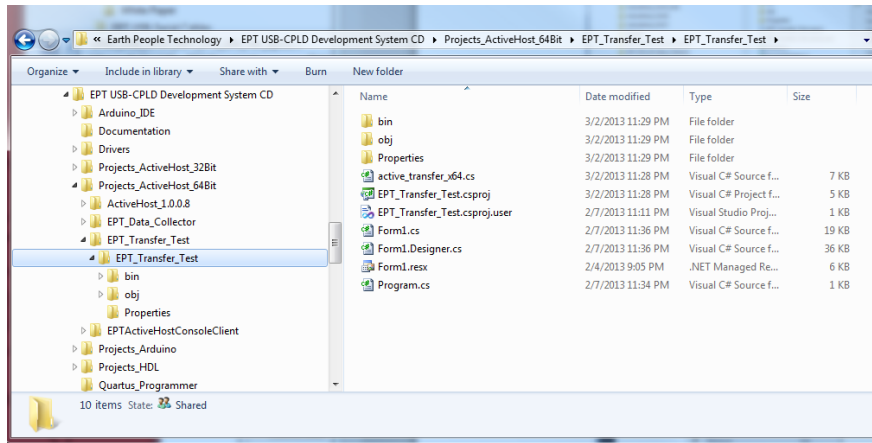
Click on the Save All button on the tool bar. The project environment is now setup and ready for the project files. Close the Project.

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. These files are:

- Active_transfer_xxx.cs
- Form1.cs
- Form1.Designer.cs
- Program.cs

USB CPLD Development System User Manual



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.

```

EPT_FT2232_Interface.EPT_FT2232_Interface
using System;
using System.Drawing;
using System.Collections;
using System.Windows.Forms;
using System.Data;
using System.Threading;
using System.Runtime.InteropServices;
using System.Diagnostics;

1
namespace EPT_FT2232_Interface
{
2
    public partial class EPT_FT2232_Interface : System.Windows.Forms.Form
    {
3
        public EPT_FT2232_Interface()
        {
            InitializeComponent();

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

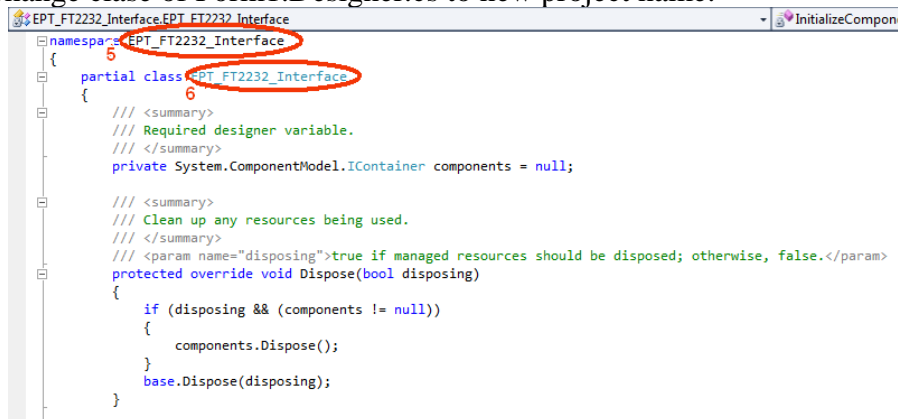
```

USB CPLD Development System User Manual

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

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

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



```
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.

USB CPLD Development System User Manual

```

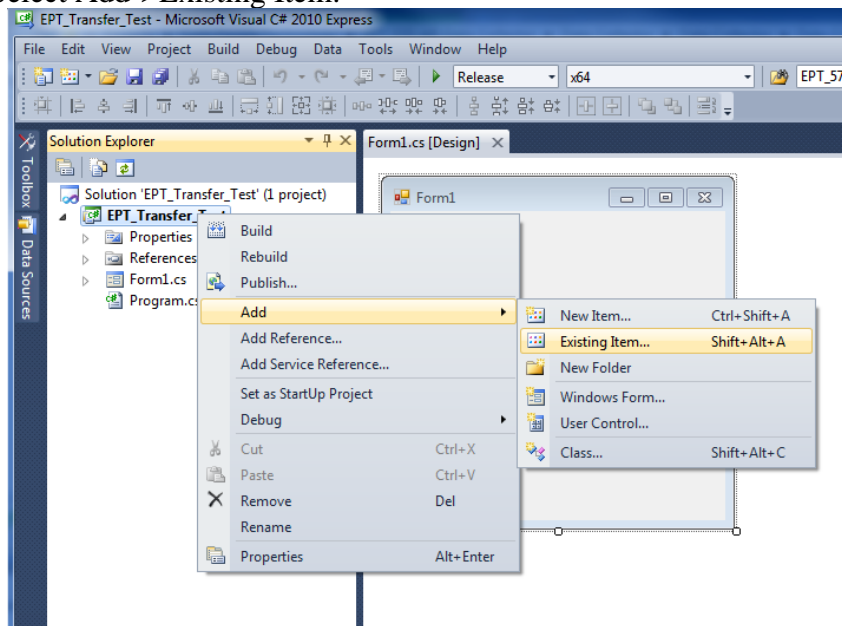
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;

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

```

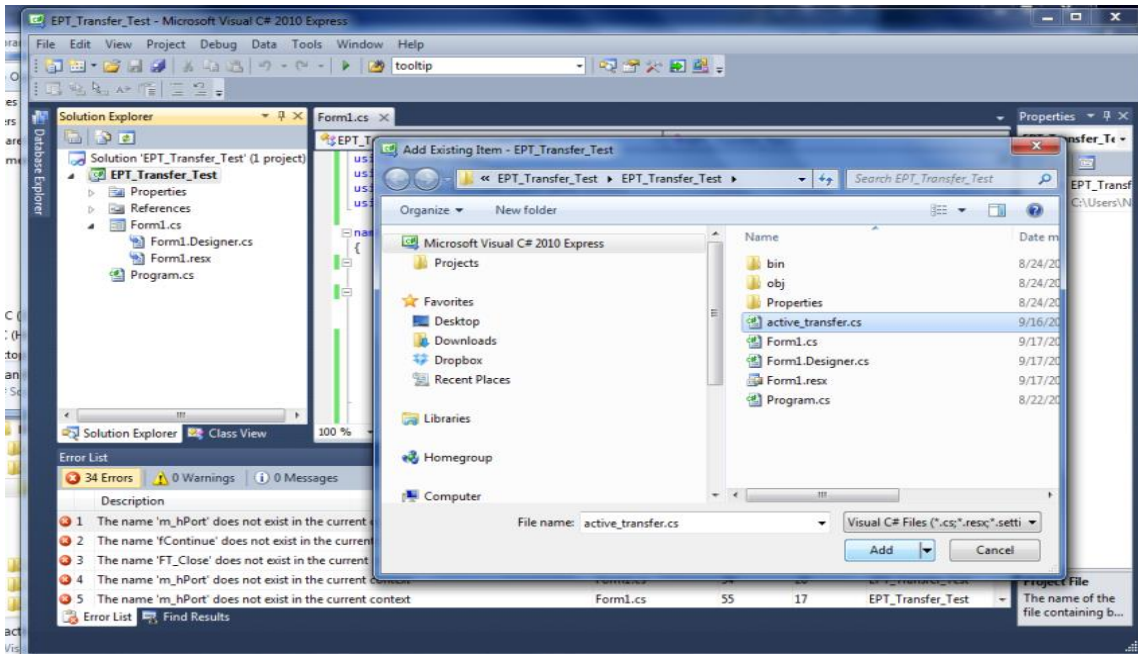
6.2.2 Add Files to Project

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

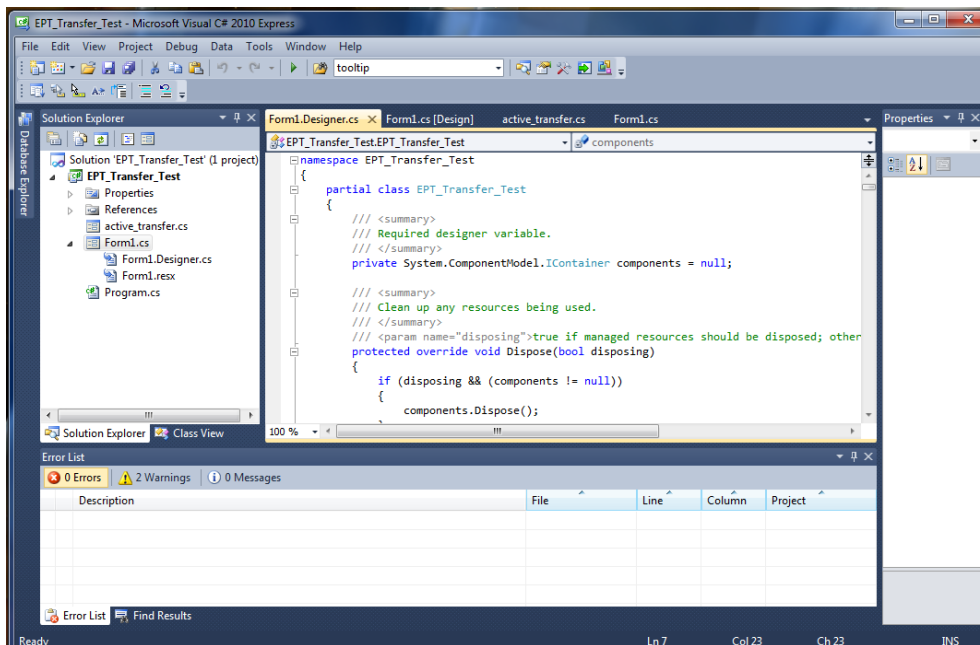


Browse to the EPT_Transfer_Test project folder and select the active_transfer_xx.cs file (choose either the 32 bit or 64 bit version, depending on whether your OS is 32 or 64 bit). Click Add.

USB CPLD Development System User 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.






USB CPLD Development System User Manual

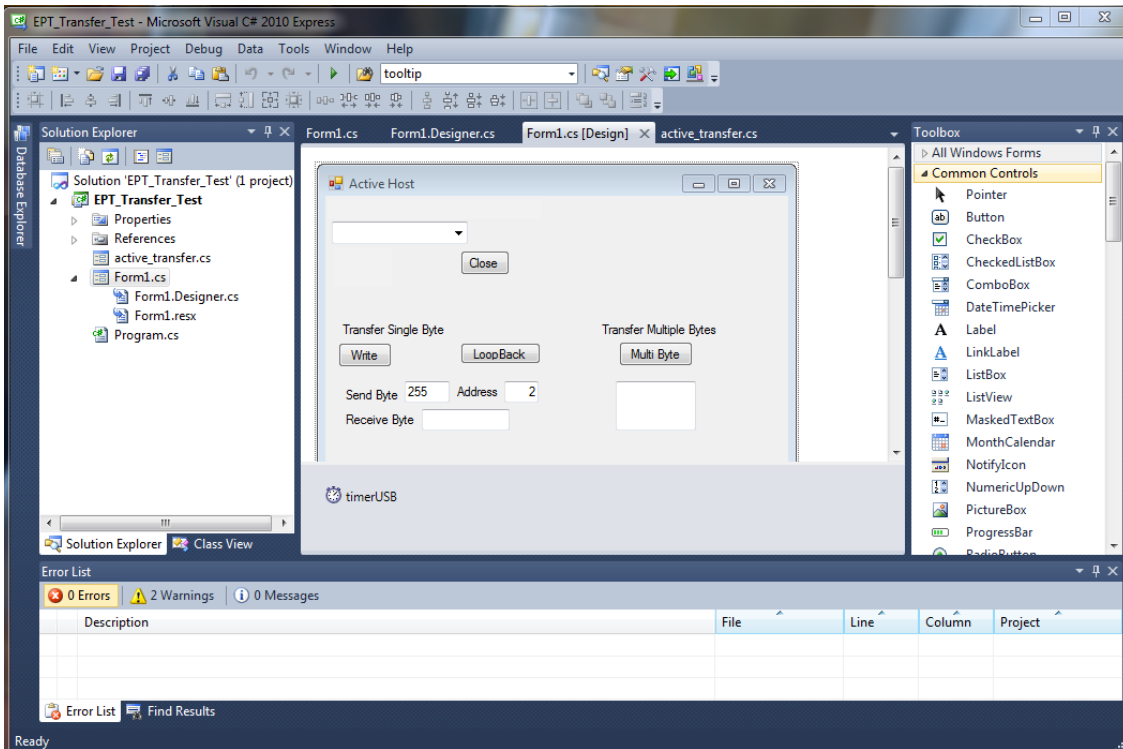
6.2.3 Adding Controls to the Project

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.

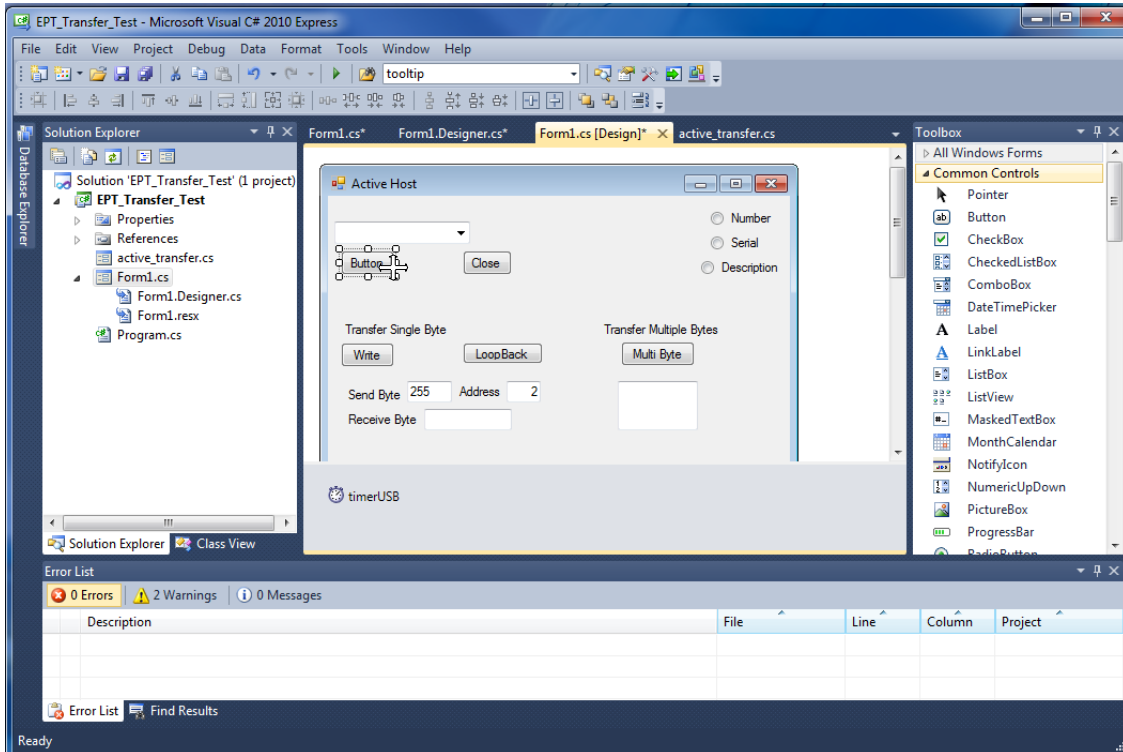
The easiest way to add a button to a form is to double click the Form1.cs in the Solution Explorer. Click on the  button to launch the Toolbox.

USB CPLD Development System User Manual



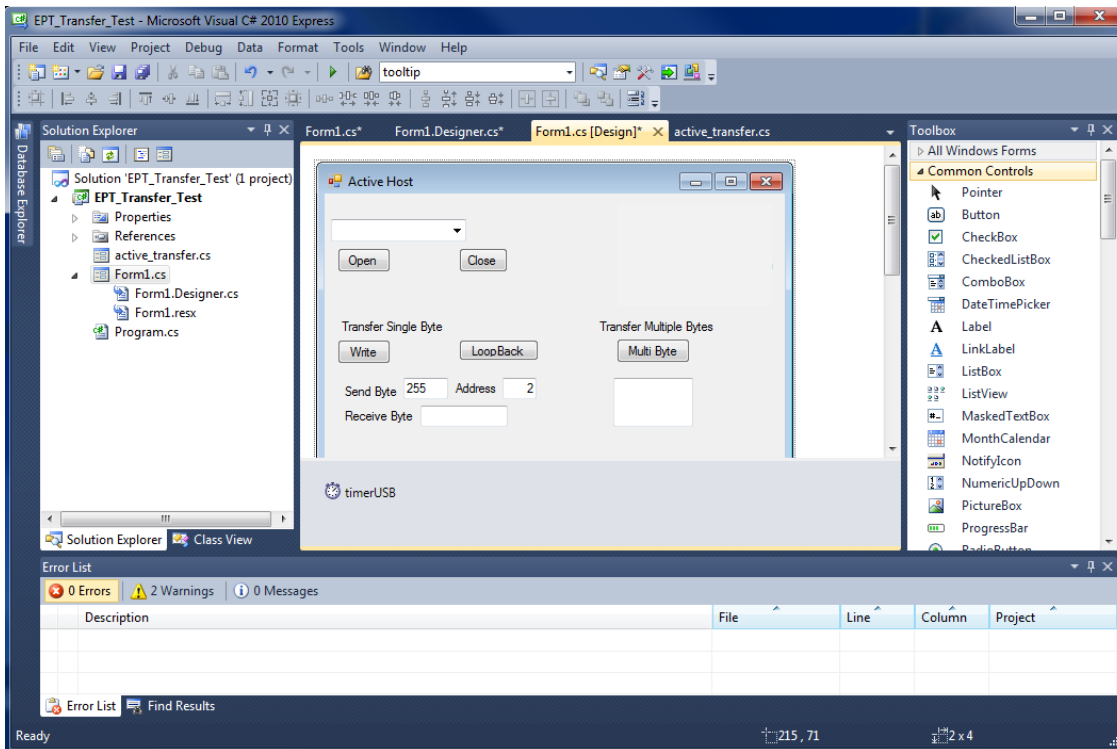
Locate the button on the Toolbox, grab and drag the button onto the Form1.cs [Design] and drop it near the top.

USB CPLD Development System User 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.

USB CPLD Development System User 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.



USB CPLD Development System User Manual

```
//  
// 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

USB CPLD Development System User Manual

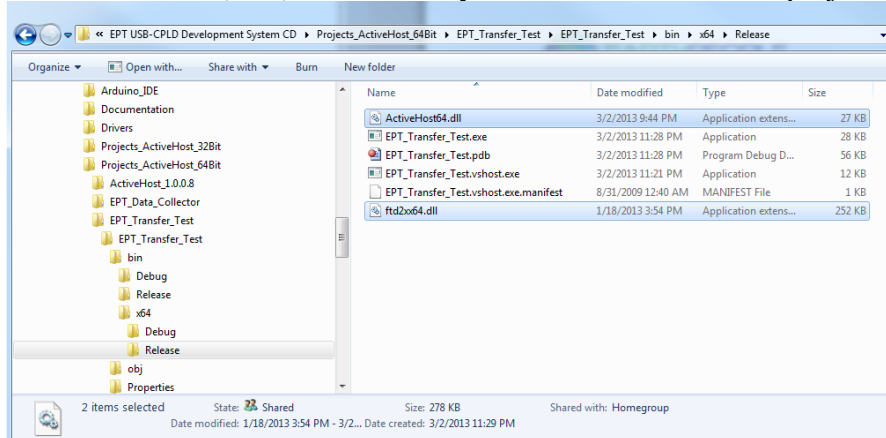
People Technology to perform some simple reads and writes to the EPT USB-CPLD Development System.

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 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.

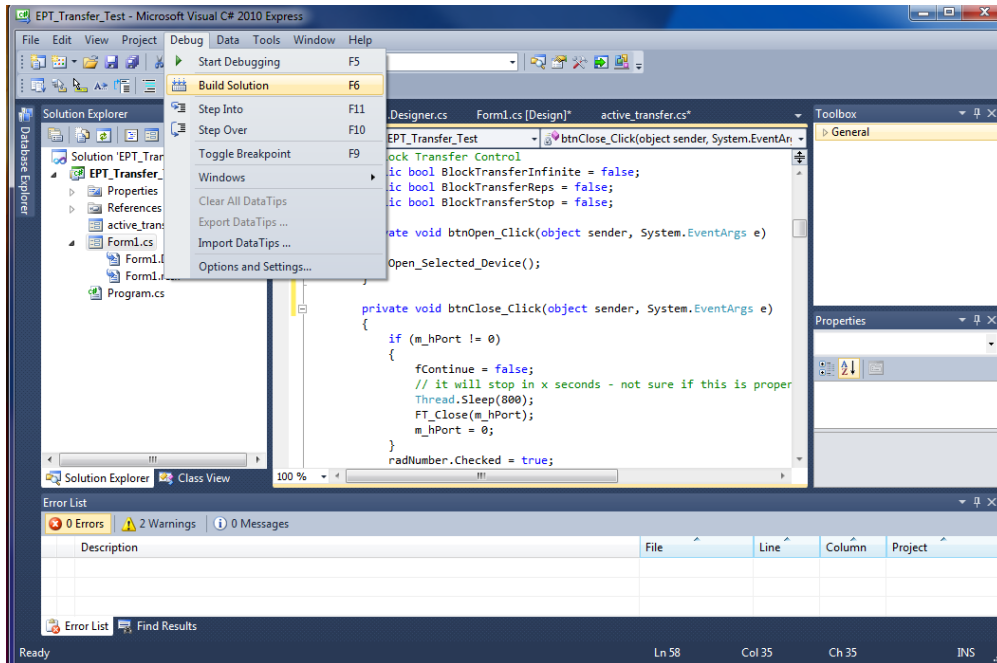


Save the project.

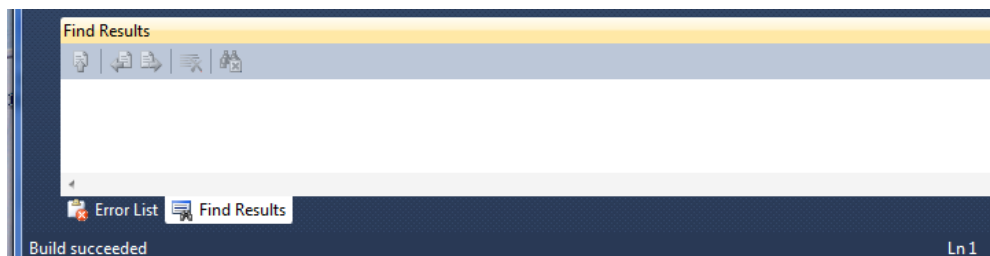
6.2.5 Building the Project

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.

USB CPLD Development System User Manual



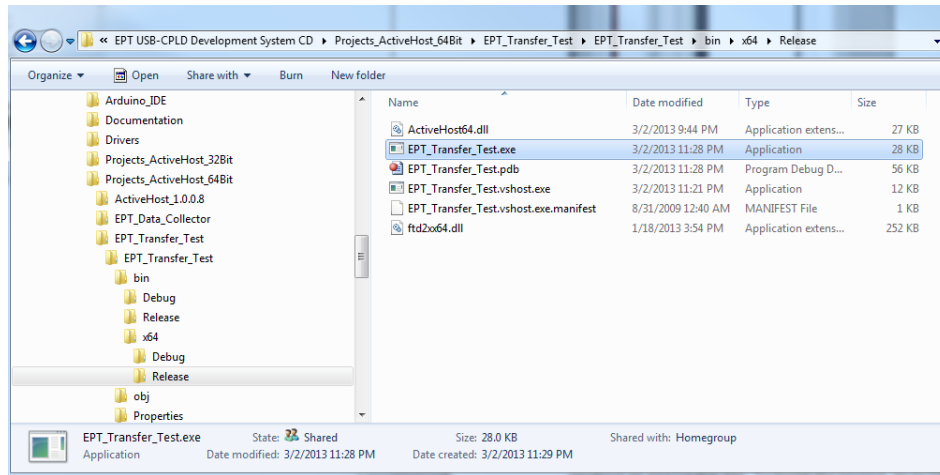
The C# Express compiler will start the building process. If there are no errors with code syntax, function usage, or linking, then the environment responds with “Build Succeeded”.



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.

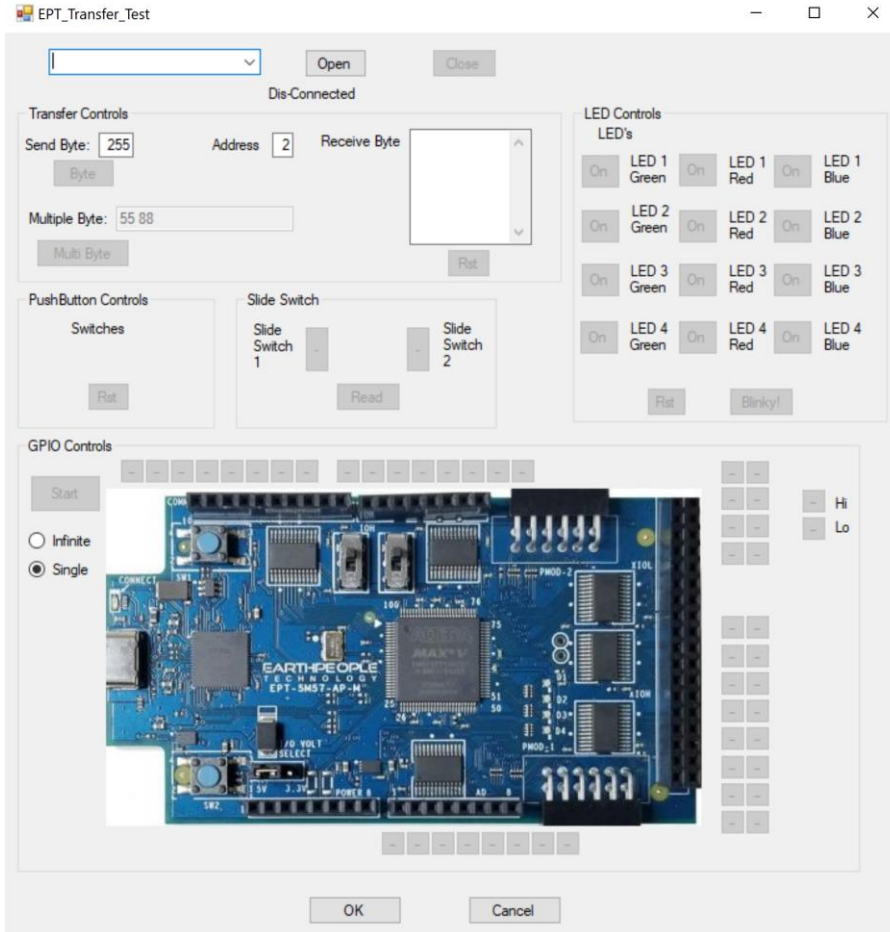
USB CPLD Development System User Manual



The EPT_Transfer_Text.exe file can now be tested using the MEGAPROLOGIC board. To test the file, connect the MEGAPROLOGIC to the Windows PC using Type A to USB-C 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_Test.exe file resides, and double click on the file. The application should load with a Windows form.

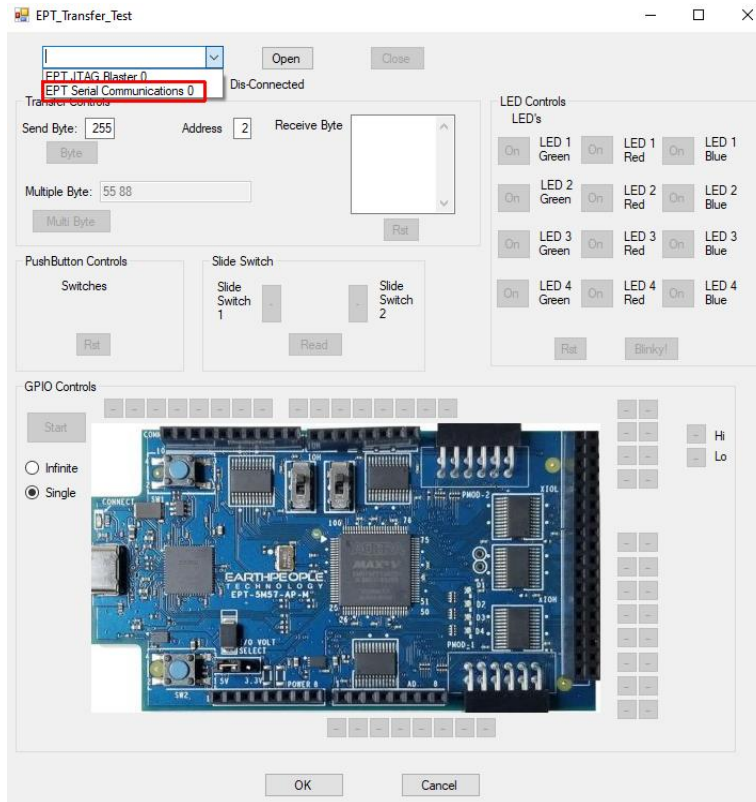


USB CPLD Development System User Manual



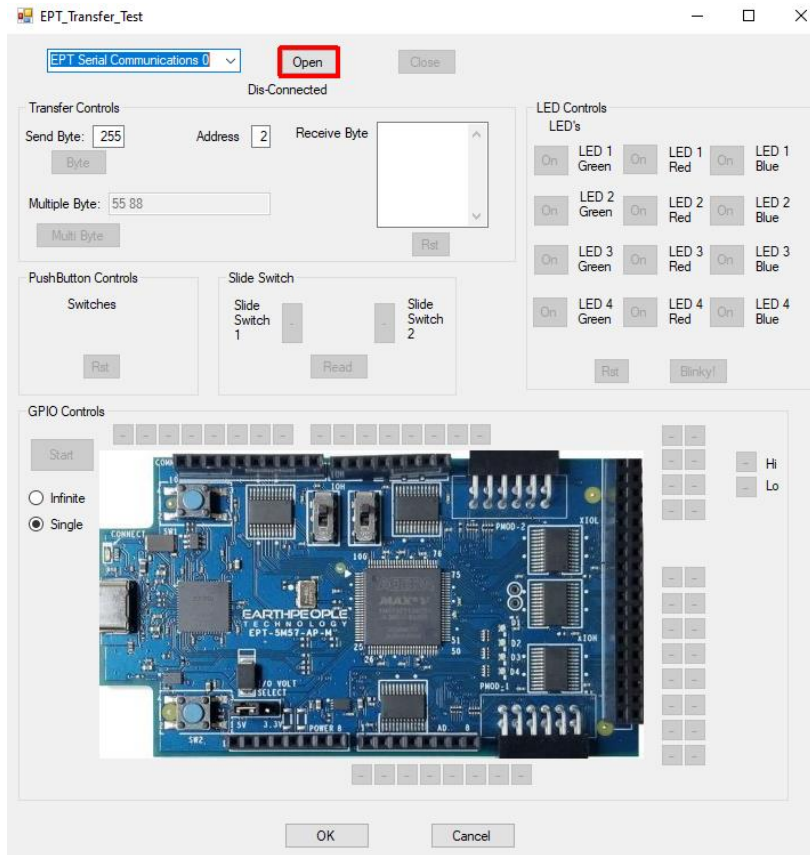
With the application loaded, select the EPT Serial Communications x board from the dropdown combo box.

USB CPLD Development System User Manual



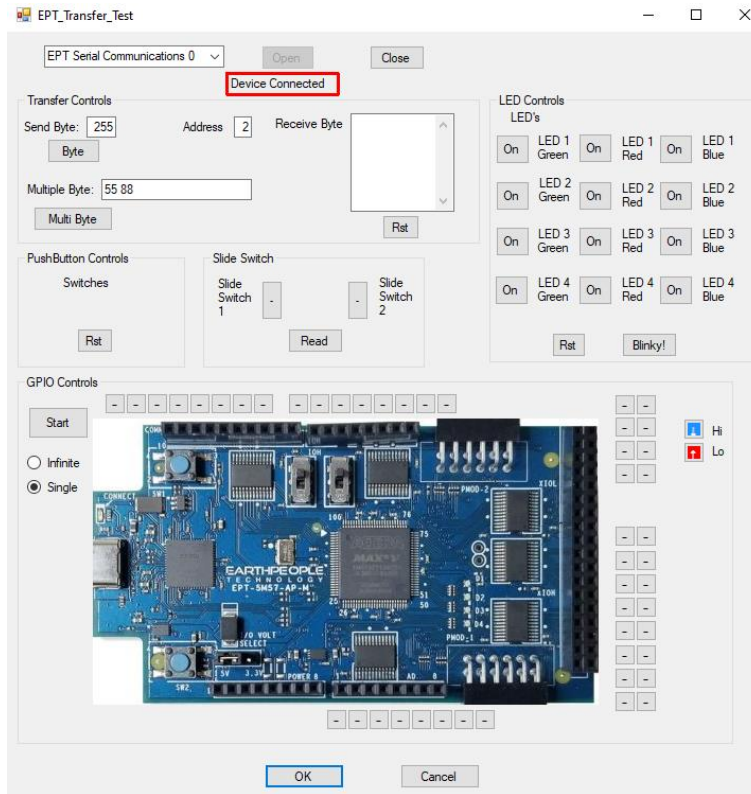
Next, click on the Open button.

USB CPLD Development System User Manual



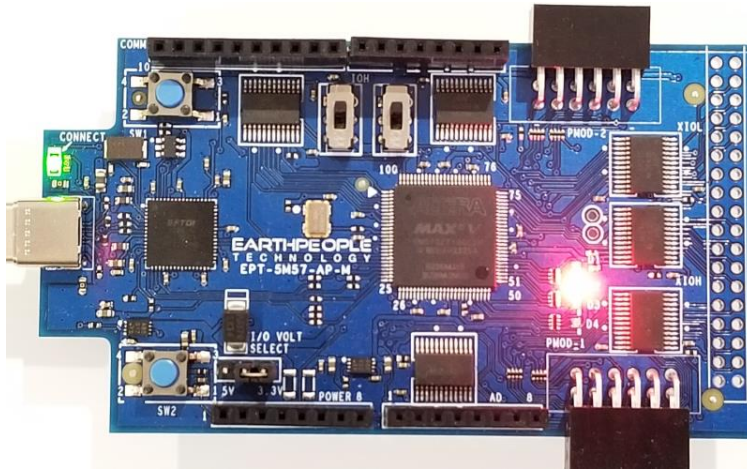
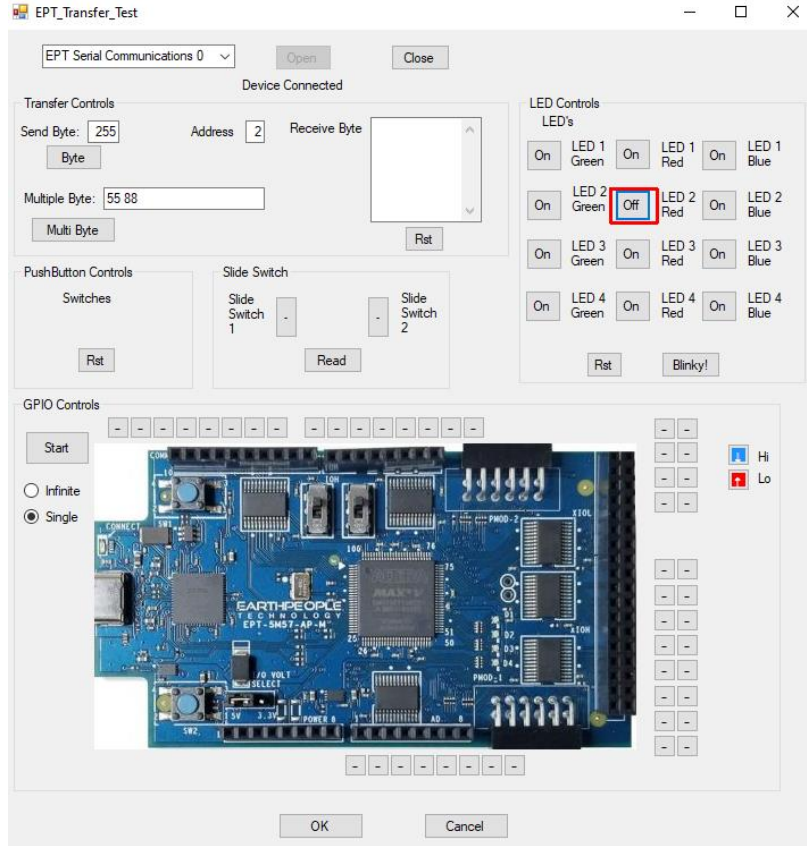
The “Device Connected” label should be seen to indicate the Windows App is now connected to the MegaProLogic.

USB CPLD Development System User Manual



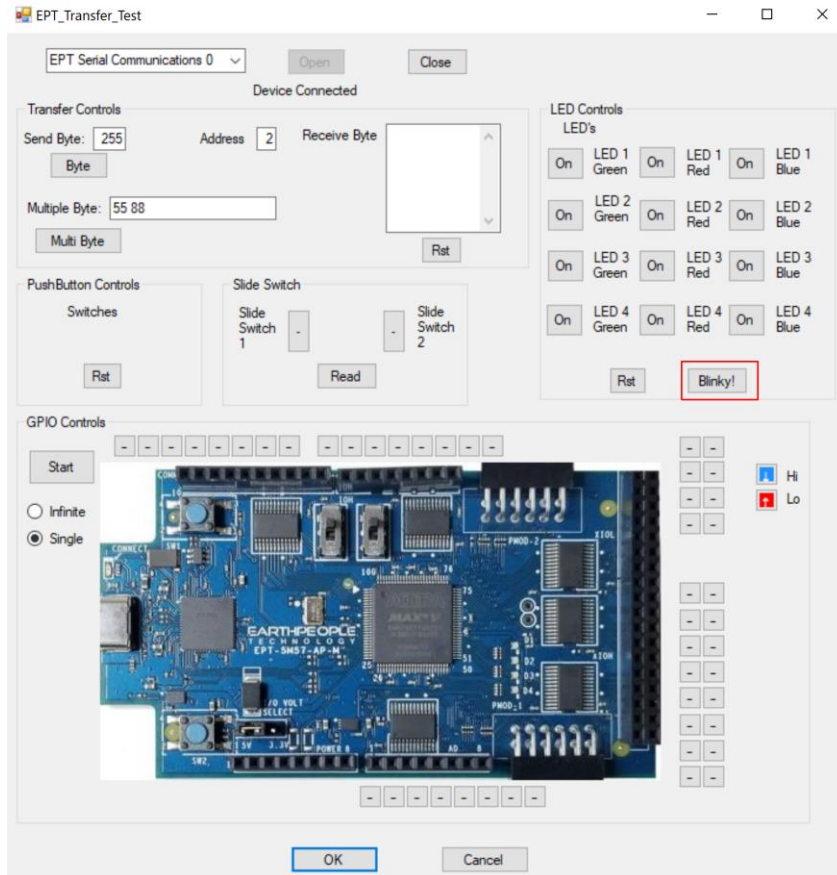
Click on one of the LED buttons in the middle of the window. The corresponding LED on the MegaProLogic board should light up.

USB CPLD Development System User Manual



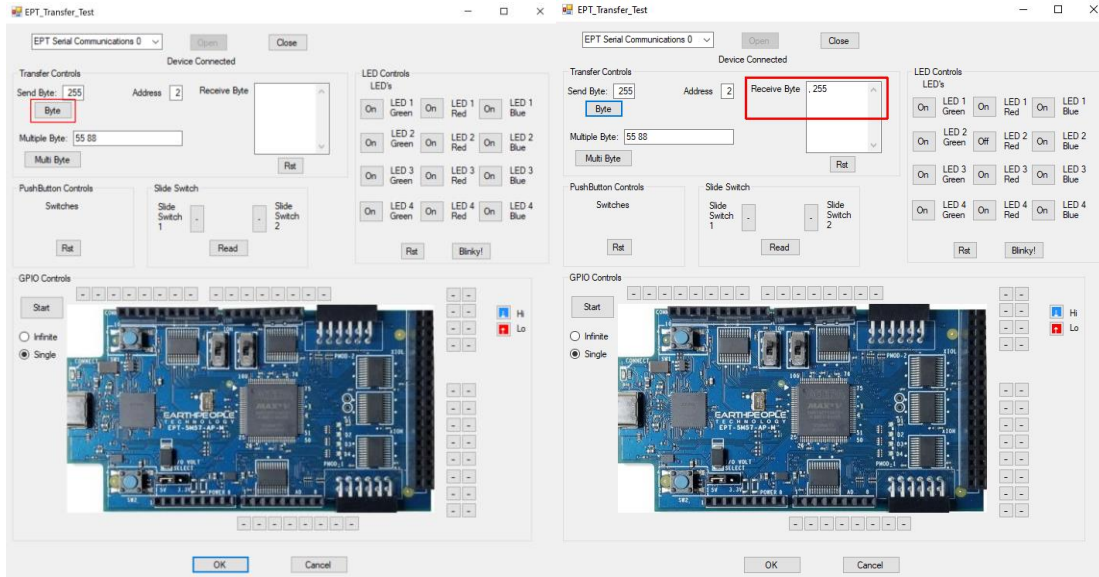
Click on the Blinky button for a light show.

USB CPLD Development System User Manual

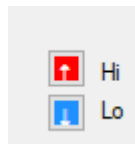


To exercise the Single Byte Transfer EndTerm, click the “Byte” button in the Transfer Controls group. Type in several numbers separated by a space and less 256 into the Multiple Byte textbox. Then hit the Multi Byte button. The numbers appear in the Receive Byte textbox.

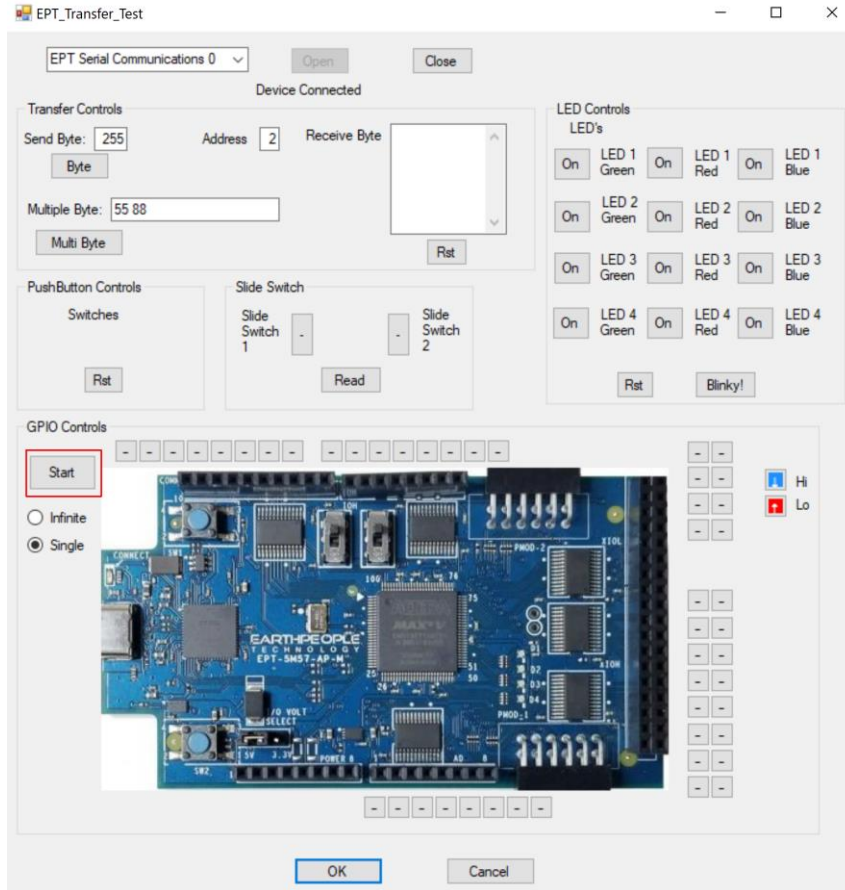
USB CPLD Development System User Manual



To exercise the Block Transfer EndTerm, click the “Start” button in the GPIO Controls group. The MegaProLogic will sample the state of each Input pin of the CPLD that is connected to a board edge connector. The Transfer Test Window will display the results of each pin, Hi or Lo

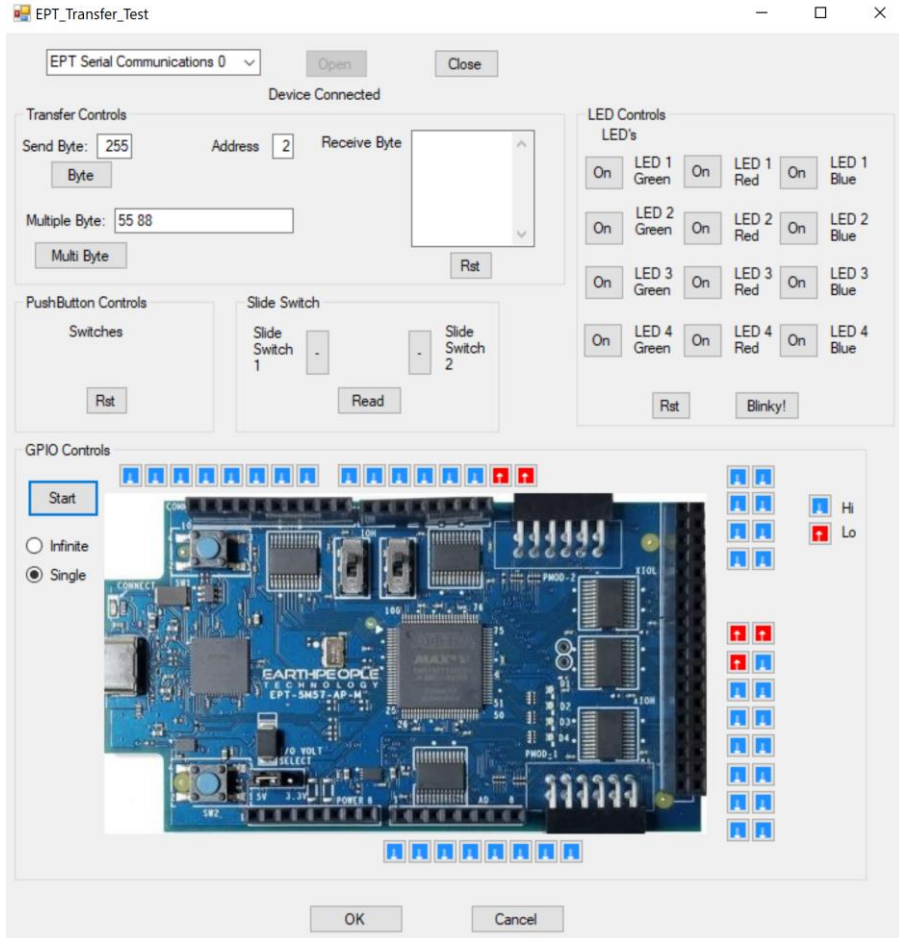


USB CPLD Development System User Manual



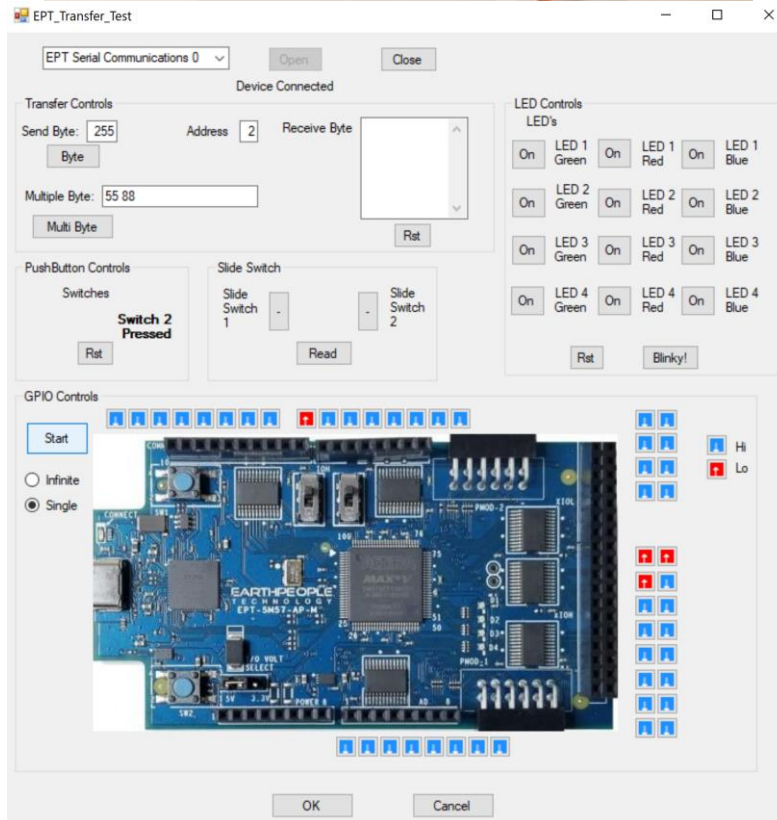
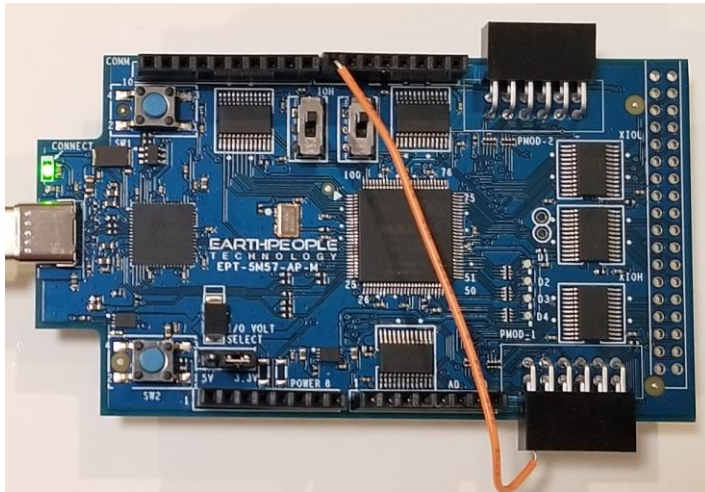
The results of each pin are displayed next to the image of the MegaProLogic in separate buttons.

USB CPLD Development System User Manual



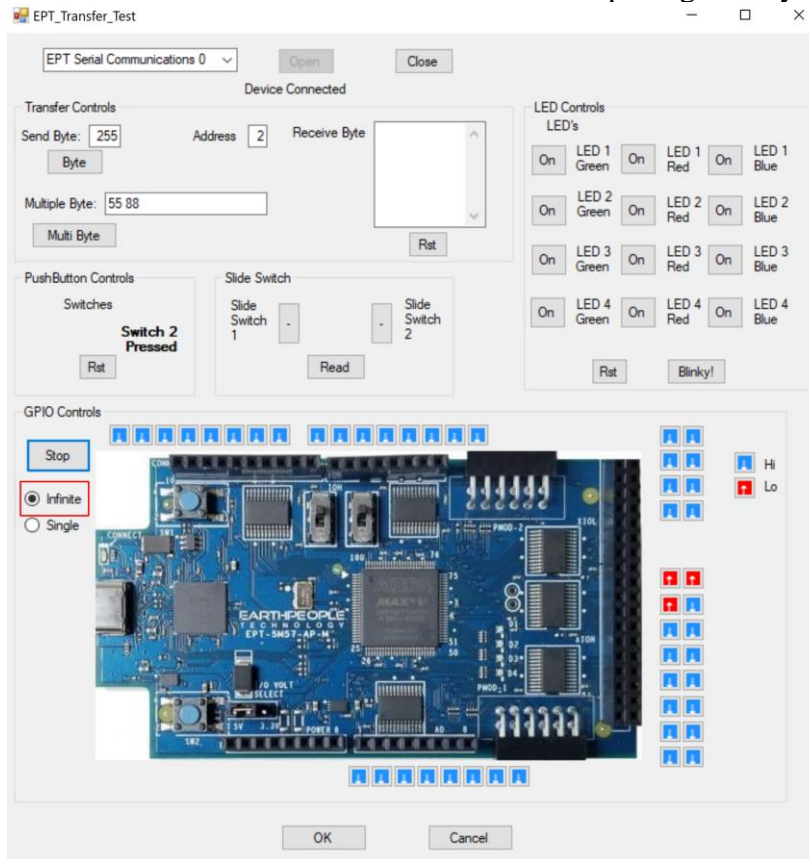
Use a jumper cable to connect between +3.3V and any of the Input pins to see the display detect a high on the selected pin.

USB CPLD Development System User Manual



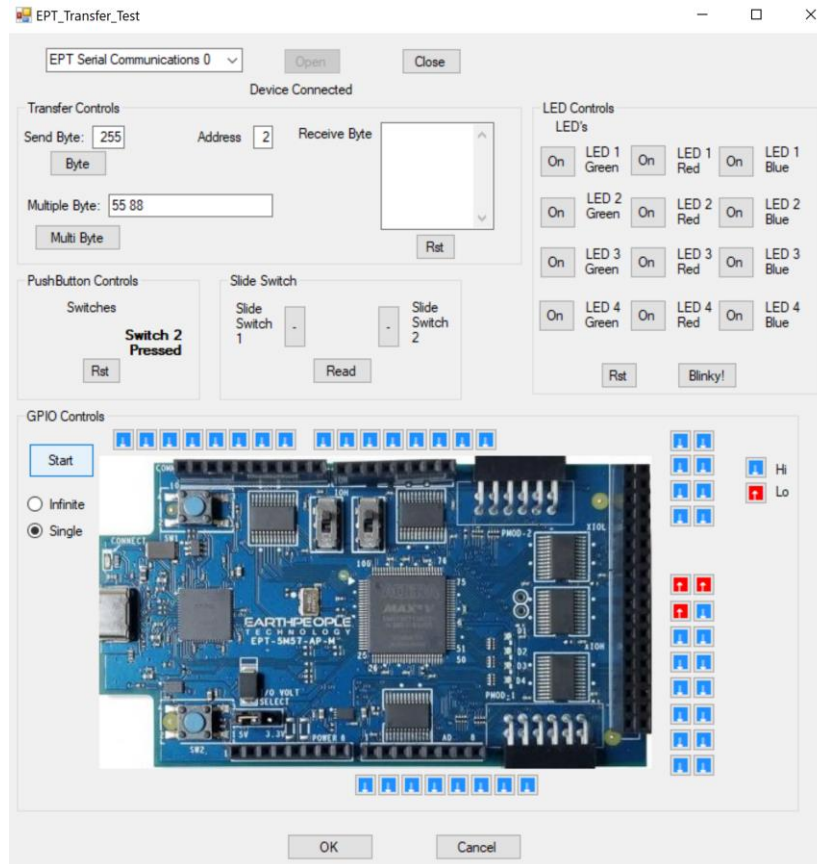
USB CPLD Development System User Manual

Click on the “Infinite” radio button and “Start” to see a simple logic analyzer.



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

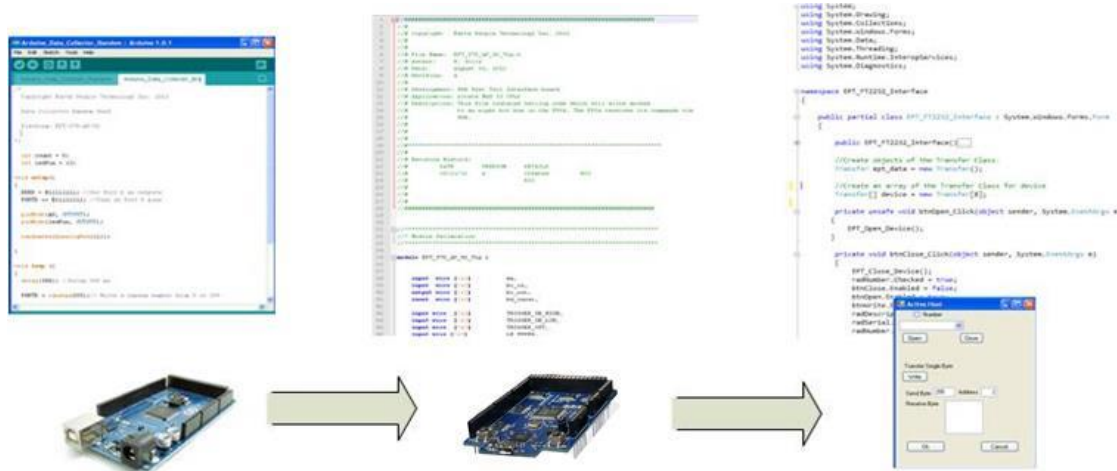
USB CPLD Development System User Manual



7 Designing a Simple Data Collection Sampler

The Data Collection Sampler is a very simple introductory project that will guide the user in the creation of an overall design using the Arduino Programming Language, Verilog HDL, and C# Language. These elements will run on the Arduino Platform, MEGAPROLOGIC-M4 CPLD, and a Windows PC respectively.

USB CPLD Development System User Manual



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

7.1.1 The Arduino Microcontroller Board

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

7.1.2 Create Data Generator

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

The randomSeed() function must be called during the setup() function. It takes as input parameter the output of the Analog Pin 1. The output of this Pin 1 will have a small

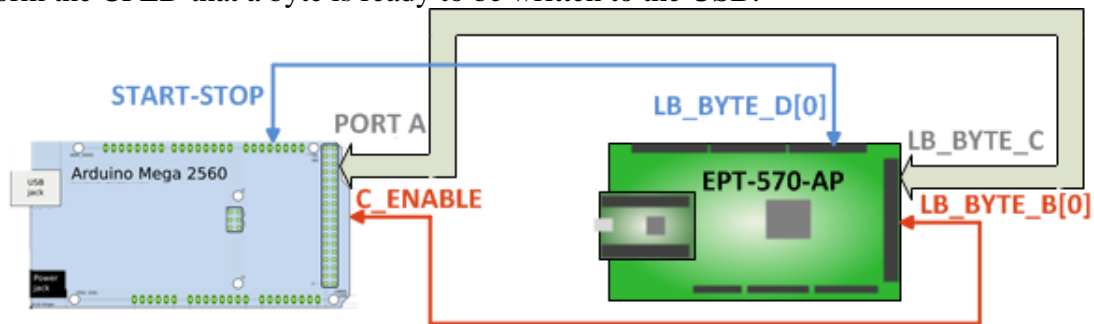
USB CPLD Development System User Manual

amount of random noise on it. Because of this noise, the randomSeed() function will produce a different seed every time the sketch is initialized.

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

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

An 8 bit port is used to connect the 8 bit byte from the random function output to the input of the MegaProLogic. There is also a one bit control line which will be used to inform the CPLD that a byte is ready to be written to the USB.



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

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

PORTA maps to Arduino digital pins 0 to 7

DDRA - The Port A Data Direction Register - read/write
 PORTA - The Port A Data Register - read/write
 PINA - The Port A Input Pins Register - read only



USB CPLD Development System User Manual

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

```
void setup()
{
  DDRA = B11111111; //Set Port A as outputs
  PORTA &= B11111111; //Turn on Port A pins
```

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

7.1.4 Coding the Arduino Data Sampler

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

```
void loop ()
{

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

  delay(500); //Delay 500 ms

  if(startStopBit)
  {
```

This code will sample the `Start/Stop` switch which is an output from the MEGAPROLOGIC on `J5 PIN 3`. On the Arduino, this is `PIN 19` of the Digital pins. Each iteration of the `loop()` function, the `startStopBit` variable stores the state of `DigitalPin19`. Then, a delay of 500 milliseconds is added. The `delay()` function pauses the program for the amount of time (in milliseconds) specified as parameter. Next, the `startStopBit` is checked with a conditional switch. If the bit is set, the conditional branch is entered and the random number is sent to the MEGAPROLOGIC. If the bit is not set, the end of the `loop()` function is reached and it branches to the top of the `loop()`.



USB CPLD Development System User Manual

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

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

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

So, the code looks like this:



USB CPLD Development System User Manual

```
/*
  Copyright Earth People Technology Inc. 2013

  Data Collector Random Seed

  Platform: EPT-570-AP-M4
*/

int startStopBit = 0;
int count = 0;
int ledPin = 13;
int inPin8 = 19;
int C_Enable = 37;

void setup()
{
  DDRA = B11111111; //Set Port A as outputs
  PORTA &= B11111111; //Turn on Port A pins

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

  randomSeed(analogRead(1));
}

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

  delay(500); //Delay 500 ms

  if(startStopBit)
  {
    // Write a random number from 0 to 299
    //to the input of the EPT-570-AP
    PORTA = random(255);
  }
}
```

USB CPLD Development System User Manual

```

//Set the Write Enable Pin High
digitalWrite(C_Enable, HIGH);

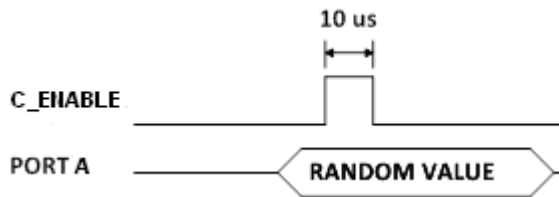
//Set the LED Pin High
digitalWrite(ledPin, HIGH);

delay(500); //Delay 500 ms

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

```

Notice that PORTA equals the return of random(255). The parameter passed to the random() function is the maximum decimal value of the return value. In our case we want the maximum value to be an 8 bit value, B11111111 = 0xff = 255(decimal). Also, note that the C_Enable write enable signal for the CPLD has back to back instructions turning it on then off immediately. Because the ATmega328 chip takes approximately 160 clock cycles to execute the digitalWrite() function and affect the Pin at C_Enable, this produces a write enable pulse of 10 microseconds.



The RANDOM VALUE will be stable before the C_Enable asserts thus guaranteeing a successful transfer of data from Arduino to CPLD.

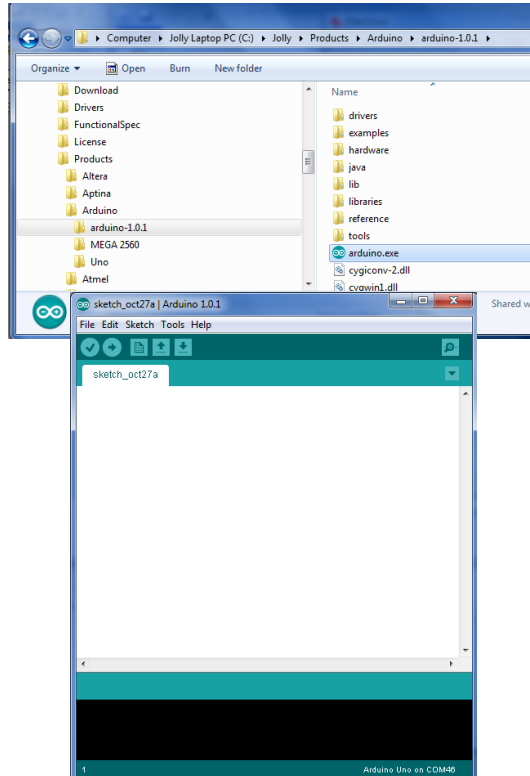
7.1.5 Building Arduino Project

Building the Arduino project is the process of converting (compiling) the code you just wrote into machine level code that the processor can understand. The Arduino IDE is the software tool that does the compiling. The machine level code is a set of basic instructions that the processor uses to perform the functions the user code. Browse to the \Projects_Arduino\Arduino_Data_Collector_Code\ folder of the EPT USB-CPLD Development System CD. Copy Arduino_Data_Collector_Code_M4.ino .

USB CPLD Development System User Manual

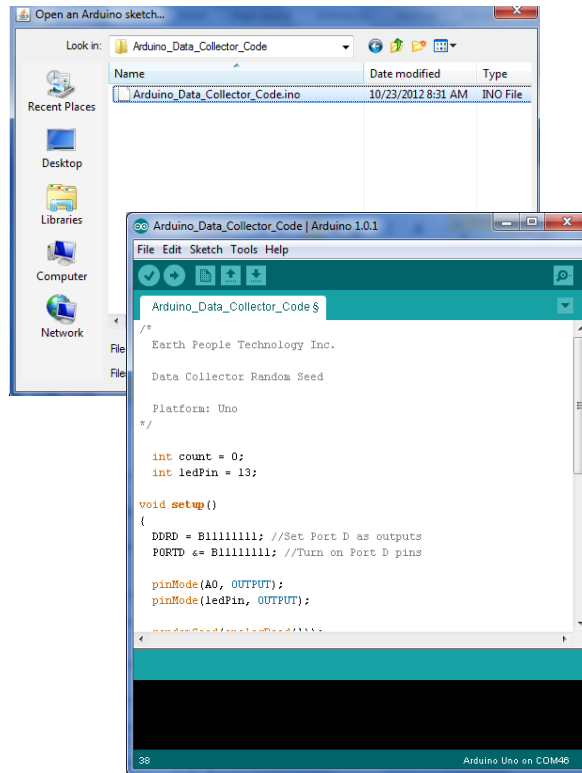
To compile your code,

- Open up the Arduino IDE

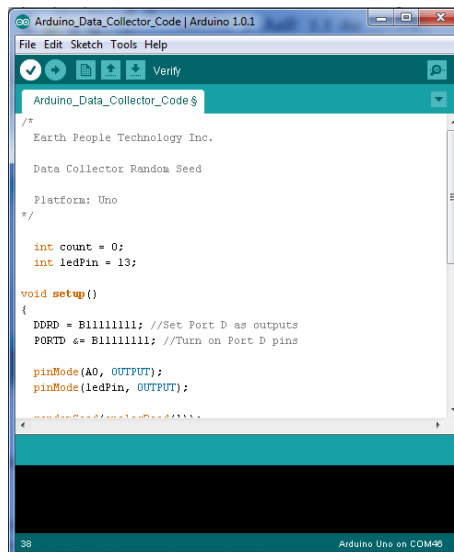


- Paste your code into the Sketch.

USB CPLD Development System User Manual

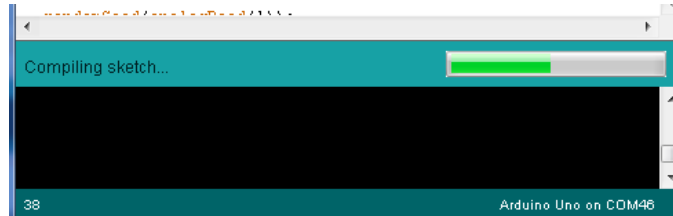


- Click the Verify button

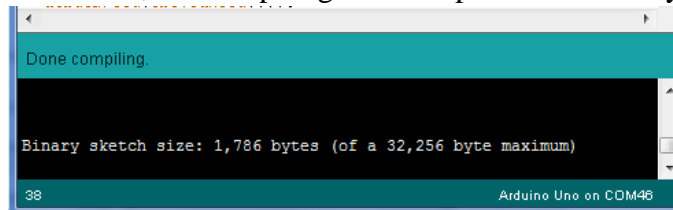


- The sketch will compile

USB CPLD Development System User Manual



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



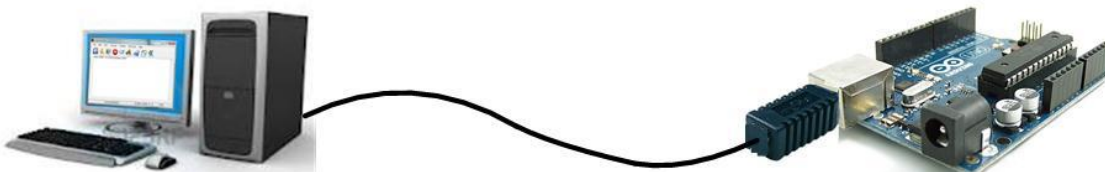
Now we are done with compiling and ready to program the Arduino

7.1.6 Programming the Arduino

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

To program the Arduino

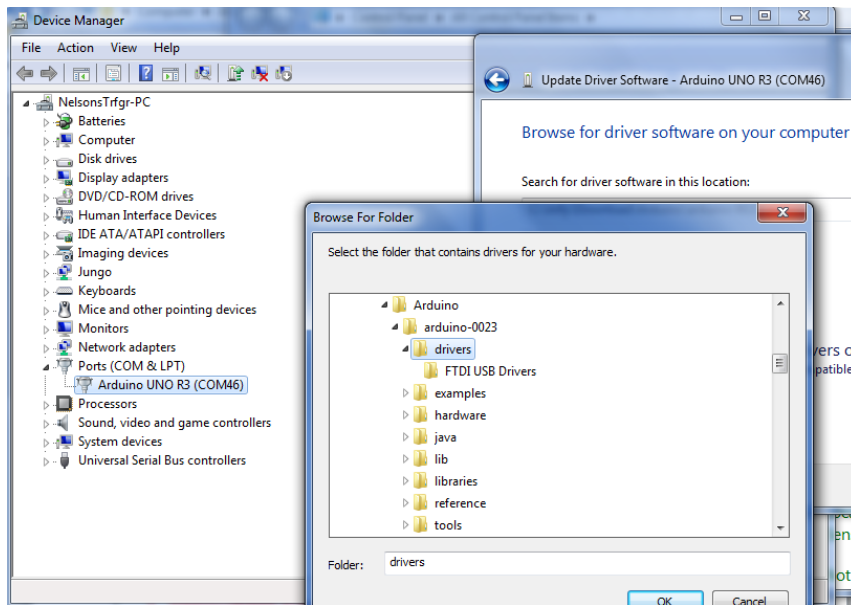
- Connect the USB cable from PC to Arduino



- Load the Arduino USB driver according to the manual
- Plug in your board and wait for Windows to begin its driver installation process. After a few moments, the process will fail, despite its best efforts
- Click on the Start Menu, and open up the Control Panel.
- While in the Control Panel, navigate to System and Security. Next, click on System. Once the System window is up, open the Device Manager.

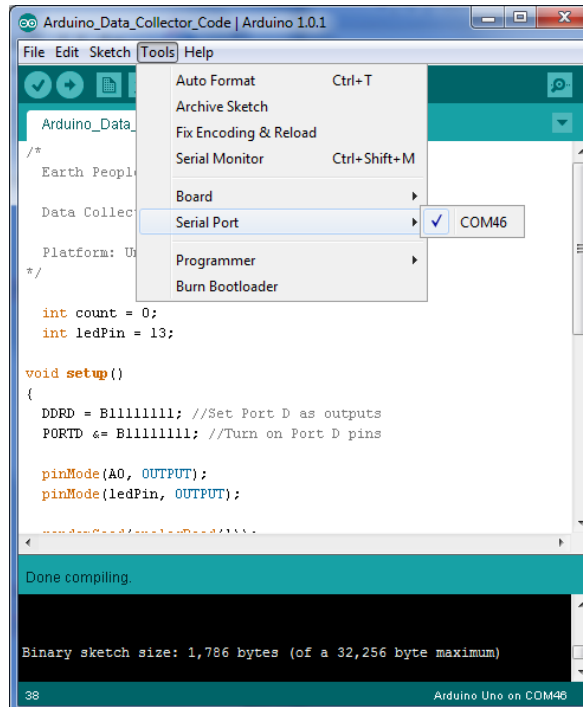
USB CPLD Development System User Manual

- Look under Ports (COM & LPT). You should see an open port named "Arduino MEGA (COMxx)"
- Right click on the "Arduino MEGA (COMxx)" port and choose the "Update Driver Software" option.
- Next, choose the "Browse my computer for Driver software" option.
- Finally, navigate to and select the MEGA's driver file, named "**ArduinoMEGA.inf**", located in the "Drivers" folder of the Arduino Software download (not the "FTDI USB Drivers" sub-directory).
- Windows will finish up the driver installation from there.

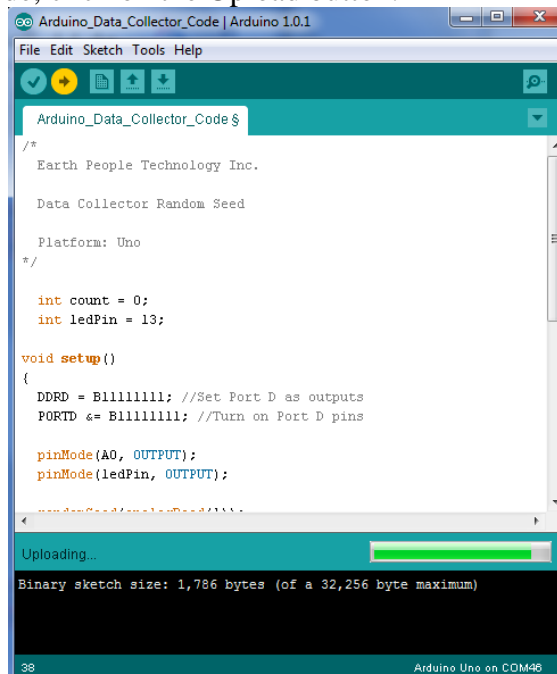


- Once the driver is loaded, we can set the COM Port. Click on Tools and select Serial Port, then click the available port.

USB CPLD Development System User Manual

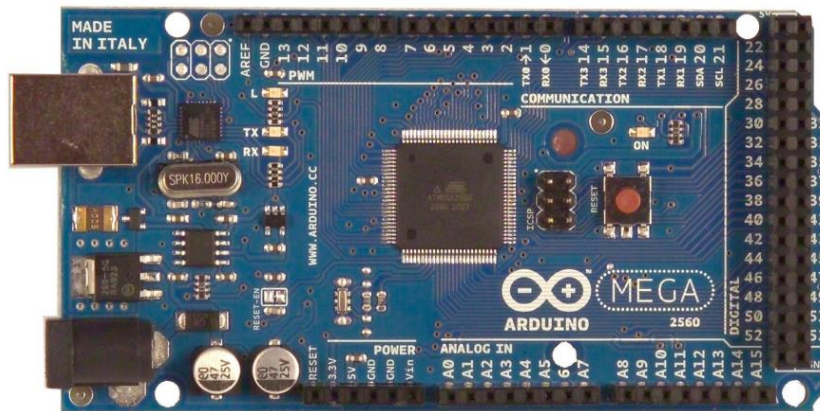


- To load the code, click on the Upload button.



USB CPLD Development System User Manual

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



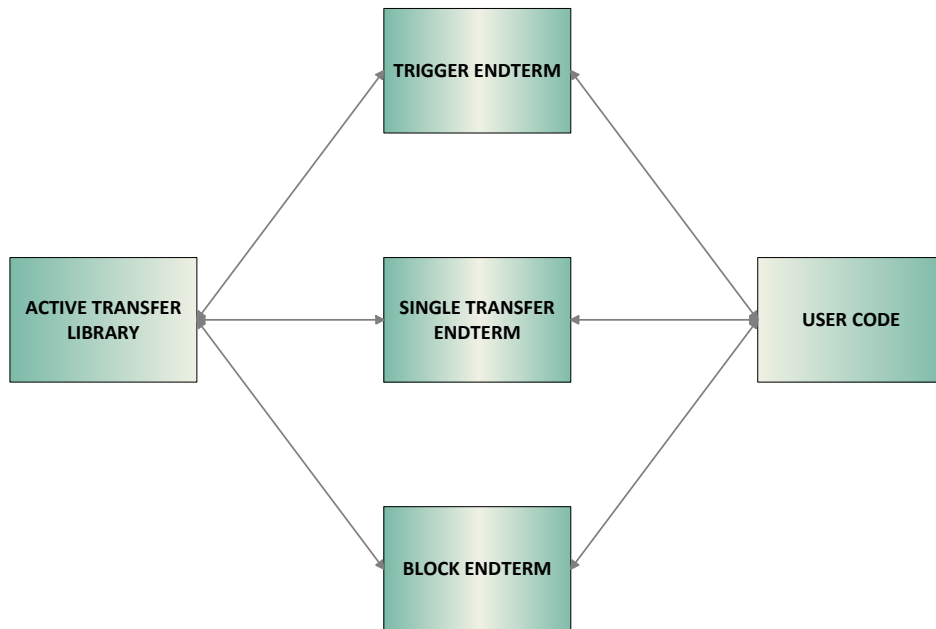
7.1.7 CPLD Active Transfer Coding and Initiation

The MEGAPROLOGIC will accept the data collected by the Arduino and transfer it to the PC. It is designed to plug directly into the Arduino MEGA and there is no need for external wires to be added. The Active Transfer Library can be used to send the data to the PC. This library has been designed to make it easy to transfer data to and from the PC via the USB. The user must create some interface code between the incoming data and the Active Transfer Library. We will now go through exercise of creating the CPLD code for the Data Collector Sampler.

7.1.8 CPLD: Define the User Design.

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

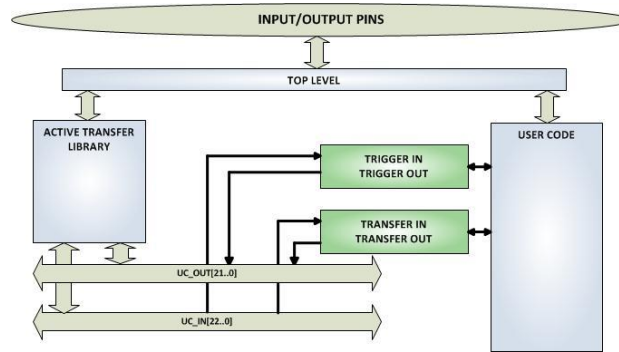
USB CPLD Development System User Manual



We will build our CPLD project using Quartus Prime software from Altera. The primary file defining the user's CPLD project is named "EPT_570_AP_M4_Top.v". It defines the user code and connects the `active_transfer_library` and `active_transfer` logic functions. In order to route the pins of the Arduino to the CPLD, the Pin Planner tool is used. This tool allows the user to match internal net names to the pins of the CPLD.

Our project needs to accept an 8 bit value on the J3 connector and a write enable on Pin 3 of J5. For this, we can use the `active_transfer.vqm` module as the interface to the `active_transfer_library`. It accepts a single byte and latches it with a single enable net. Because the `active_transfer_library` runs at 66 MHz we will need to write some code ensure that the slower `C_Enable` signal from the Arduino can latch the data into the `active_transfer` module.

USB CPLD Development System User Manual



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

www.asic-world.com/verilog/

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



USB CPLD Development System User Manual

```
module EPT_5M57_AP_M4_Top (  
  
    input wire          CLK_66MHZ,  
    input wire          RST,  
  
    //UART COMMAND BUS  
    output wire         UART_OUT, //To external  
    input wire          UART_IN, //From external  
  
    input wire [7:0]    LB_XIOLA,    //XIOH -- J16  
    input wire [2:0]    LB_XIOLC,    //XIOH -- J16  
    input wire [7:0]    LB_XIOH,    //XIOL -- J16  
    input wire [7:0]    LB_COMM,    //COMM -- J10  
    input wire [7:0]    LB_AD,      //XIOH -- J8  
    input wire [7:0]    LB_IOH,     //XIOH -- J9  
  
    //Transceiver Control Signals  
    output wire         TR_DIR_1,  
    output wire         TR_OE_1,  
  
    output wire         TR_DIR_2,  
    output wire         TR_OE_2,  
  
    output wire         TR_DIR_3,  
    output wire         TR_OE_3,  
  
    output wire         TR_DIR_4,  
    output wire         TR_OE_4,  
  
    output wire         TR_DIR_5,  
    output wire         TR_OE_5,  

```




USB CPLD Development System User Manual

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

```
//-----  
// Parameters  
//-----  
  
//Header Bytes for the Transfer Loopback detection  
parameter          TRANSFER_CONTROL_BYTE1 = 8'h5A;  
parameter          TRANSFER_CONTROL_BYTE2 = 8'hC3;  
parameter          TRANSFER_CONTROL_BYTE3 = 8'h7E;  
  
//State Machine Transfer Loopback detection  
parameter          TRANSFER_CONTROL_IDLE = 0,  
                   TRANSFER_CONTROL_HDR1 = 1,  
                   TRANSFER_CONTROL_HDR2 = 2,  
                   TRANSFER_DECODE_BYTE = 3,  
                   TRANSFER_CONTROL_SET  = 4;  
  
parameter          GLOBAL_RESET_COUNT = 12'h09c8;
```

USB CPLD Development System User Manual

Next is the Internal Signal and Register Declarations.

```

//*****
//* Internal Signals and Registers Declarations
//*****

wire          CLK_66;
wire          RST;

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

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

//LED registers
reg          led_reset;

//Switch registers
reg          switch_reset;

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

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

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

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

//Input/Output Signals
reg          start_stop_cntrl;

```



USB CPLD Development System User Manual

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

```
/**
 * Signal Assignments
 */
assign      LB_BYTE_A[7:0] = 8'h0;

assign      TR_DIR_1  = 1'b1; //1 = A to B; 0 = B to A
assign      TR_OE_1   = 1'b1;

assign      TR_DIR_2  = 1'b1; //1 = A to B; 0 = B to A
assign      TR_OE_2   = 1'b0;

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

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

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

//LED3 is used to signify to the user that the Start
//switch is enabled
assign      LED3 = start_stop_cntrl ? 1'b0 : 1'bz;
```

The reset signal is generated by a counter that starts counting upon power up. When the counter reaches GLOBAL_RESET_COUNT.



USB CPLD Development System User Manual

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

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

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

USB CPLD Development System User Manual

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

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



USB CPLD Development System User Manual

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

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

- transfer_write_reg –This is a latch register to hold the state of the C_Enable.
- transfer_write –This register is used to start the active_transfer single byte write to the PC.
- transfer_write_byte –This is an 8 bit register to hold the value of the Data Collection output.

This block will compare the input signal on LB_BYTE_B[0] to a high. The LB_BYTE_B [0] pin is routed to Pin 3 of J5 which is routed to the C_Enable of the Arduino Data Collector. When this bit goes high, the priority encoder goes into statement 1 and sets transfer_write_reg and transfer_write high and latches the value on the LB_BYTE_C[7:0] pins to the transfer_write_byte register. By setting transfer_write_reg high, the priority encoder goes into statement 2 which will set transfer_write register to low and stay in statement 2 of the priority encoder. The back to back high and low on the transfer_write register will cause the active_transfer module to latch the value of transfer_write_byte into the active_transfer_library module and sets up the byte transfer to the PC. When the LB_BYTE_B[0] - C_Enable pin goes low, the encoder will reset transfer_write_reg and transfer_write to low. The encoder goes back to waiting for the LB_BYTE_B[0] - C_Enable to assert high.

USB CPLD Development System User Manual

```

//-----
// Detect Transfer From Arduino
//-----
always @(posedge CLK_66 or negedge RST)
begin
  if (!RST)
  begin
    transfer_write_reg <= 1'b0;
    transfer_write <= 1'b0;
    transfer_write_byte <= 0;
    TR_DIR_4 <= 1'b0; //1 = A to B; 0 = B to A
    TR_OE_4 <= 1'b0;
    LB_BYTE_D[0] <= start_stop_cntrl;
    LB_BYTE_D[7:1] <= 7'b0000000;
  end
  else
  begin
    if(LB_BYTE_B[0] & !transfer_write_reg)
    begin
      TR_OE_4 <= 1'b1;
      LB_BYTE_D <= 8'hz;
      transfer_write_reg <= 1'b1;
      transfer_write <= 1'b1;
      transfer_write_byte <= LB_BYTE_C;
    end
    else if(LB_BYTE_B[0] & transfer_write_reg)
    begin
      transfer_write_reg <= 1'b1;
      transfer_write <= 1'b0;
    end
    else if(!LB_BYTE_B[0] & transfer_write_reg)
    begin
      transfer_write_reg <= 1'b0;
      transfer_write <= 1'b0;
      transfer_write_byte <= 0;
    end
    else
    begin
      TR_DIR_4 <= 1'b0; //1 = A to B; 0 = B to A
      TR_OE_4 <= 1'b0;
      LB_BYTE_D[0] <= start_stop_cntrl;
      LB_BYTE_D[7:1] <= 7'b0000000;
    end
  end
end
end

```

The 8 bit transceivers are used to connect the +3.3Volt Input/Outputs of the CPLD to the +5Volt Input/Outputs of the Arduino. These 8 bit transceivers incorporate a Direction bit and an Output Enable bit. To guide signals from the CPLD to the Arduino, we turn the Direction bit to “B to A”

```
TR_DIR_1 <= 1'b0;
```




USB CPLD Development System User Manual

and the Output Enable on.

$TR_OE_1 \leq 1'b0$; (Output Enables are asserted with a zero)

The start_stop_cntrl signal is set by using the TRANSFER_CONTROL state machine in the following section.

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

- The TRANSFER_CONTROL state machine will stay in the idle state of the parallel encoder until a byte from the active_transfer transfer_to_device register receives a 0x5a.
- This will cause the transfer_control_state to be changed to TRANSFER_CONTROL_HDR1.
- The state machine will stay in the TRANSFER_CONTROL_HDR1 state until the next byte is read from the active_transfer.
- If the byte from transfer_to_device is a 0xc3, the transfer_control_state will be changed to TRANSFER_CONTROL_HDR2.
- If the byte from transfer_to_device is not a 0xc3, the transfer_control_state will go back to idle.
- In the TRANSFER_CONTROL_HDR2 state, the state machine will stay in this state until the next byte from the active_transfer is received.
- If the byte from transfer_to_device is a 0x7e, the transfer_control_state will be changed to TRANSFER_DECODE_BYTE.
- If the byte from transfer_to_device is not a 0x7e, the transfer_control_state will go back to idle.
- In the TRANSFER_DECODE_BYTE state, the state machine will stay in this state until the next byte from the active_transfer.
- The next byte transferred from active_transfer will be decoded as the Control Register.

The bits of the Control Register are defined below.

Register	Bits	Description	Assertion
Control	0	Start Stop Cntrl	High
	1	Not Used	
	2	LED Reset	High

USB CPLD Development System User Manual

	3	Switch Reset	High
	4	Transfer In Loop Back	High
	5	Not Used	
	6	Not Used	
	7	Not Used	
	7	Not Used	

```

//-----
// State Machine: Control Register from Transfer In
//-----
always @(posedge CLK_66 or negedge RST)
begin
    if (!RST)
    begin
        transfer_in_received_reg <= 1'b0;
        transfer_control_state <= TRANSFER_CONTROL_IDLE;
        transfer_in_loop_back <= 1'b0;
        led_reset <= 1'b0;
        switch_reset <= 1'b0;
        start_stop_cntrl <= 1'b0;
    end
    else
    begin
        if(transfer_in_received & !transfer_in_received_reg)
        begin
            transfer_in_received_reg <= 1'b1;
            case(transfer_control_state)
            TRANSFER_CONTROL_IDLE:
                if((transfer_in_byte == TRANSFER_CONTROL_BYTE1))
                    transfer_control_state <= TRANSFER_CONTROL_HDR1;
                else if((transfer_in_byte != TRANSFER_CONTROL_BYTE1))
                    transfer_control_state <= TRANSFER_CONTROL_IDLE;
                else
                    transfer_control_state <= TRANSFER_CONTROL_IDLE;
            TRANSFER_CONTROL_HDR1:
                if((transfer_in_byte == TRANSFER_CONTROL_BYTE2))
                    transfer_control_state <= TRANSFER_CONTROL_HDR2;
                else if((transfer_in_byte != TRANSFER_CONTROL_BYTE2))
                    transfer_control_state <= TRANSFER_CONTROL_IDLE;
                else
                    transfer_control_state <= TRANSFER_CONTROL_HDR1;
            endcase
        end
    end
end

```

USB CPLD Development System User Manual

```

TRANSFER_CONTROL_HDR2:
    if((transfer_in_byte == TRANSFER_CONTROL_BYTE3))
        transfer_control_state <= TRANSFER_DECODE_BYTE;
    else if((transfer_in_byte != TRANSFER_CONTROL_BYTE3))
        transfer_control_state <= TRANSFER_CONTROL_IDLE;
    else
        transfer_control_state <= TRANSFER_CONTROL_HDR2;
TRANSFER_DECODE_BYTE:
begin
    start_stop_cntrl <= transfer_in_byte[0];
    led_reset <= transfer_in_byte[2];
    switch_reset <= transfer_in_byte[3];
    transfer_in_loop_back <= transfer_in_byte[4];
    transfer_control_state <= TRANSFER_CONTROL_SET;
end
TRANSFER_CONTROL_SET:
begin
    transfer_control_state <= TRANSFER_CONTROL_IDLE;
end
endcase
end
else if(!transfer_in_received & transfer_in_received_reg)
    transfer_in_received_reg <= 1'b0;
end
end

```

Next, up is the instantiation for the active_transfer_library. The ports include the input and output pins and the two buses that connect the active modules. These buses are the input UC_IN[23:0] and output UC_OUT[21:0].

```

//-----
// Instantiate the EPT Library
//-----

active_serial_library      ACTIVE_SERIAL_LIBRARY_INST
(
    .aa                    (aa),
    .UART_OUT              (UART_OUT),
    .UART_IN               (uart_txd),

    .UC_IN                 (UC_IN),
    .UC_OUT                (UC_OUT) //,

    // .STATE_OUT          (active_serial_state_out),
    // .TEST_OUT           (active_transfer_test_out)

);

```



USB CPLD Development System User Manual

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

```
//-----  
// Instantiate the EPT Active Modules  
//-----  
wire [22*2-1:0] uc_out_m;  
eptWireOR # (.N(2)) wireOR (UC_OUT, uc_out_m);  
    active_trigger          ACTIVE_TRIGGER_INST  
    (  
        .uc_clk              (CLK_IN),  
        .uc_reset            (RST),  
        .uc_in               (UC_IN),  
        // .uc_out            (UC_OUT),  
        .uc_out              (uc_out_m[ 0*22 +: 22 ]),  
  
        .trigger_to_host     (trigger_out),  
        .trigger_to_device   (trigger_in_byte)  
    );  
  
    active_transfer         ACTIVE_TRANSFER_INST  
    (  
        .uc_clk              (CLK_IN),  
        .uc_reset            (RST),  
        .uc_in               (UC_IN),  
        // .uc_out            (UC_OUT),  
        .uc_out              (uc_out_m[ 1*22 +: 22 ]),  
  
        .start_transfer      (transfer_out),  
        .transfer_received    (transfer_in_received),  
  
        .uc_addr             (3'h2),  
  
        .transfer_to_host    (transfer_out_byte),  
        .transfer_to_device  (transfer_in_byte)  
    );
```

Next, we are ready to compile and synthesize.

USB CPLD Development System User Manual

7.1.9 **CPLD:** Compile/Synthesize the Project

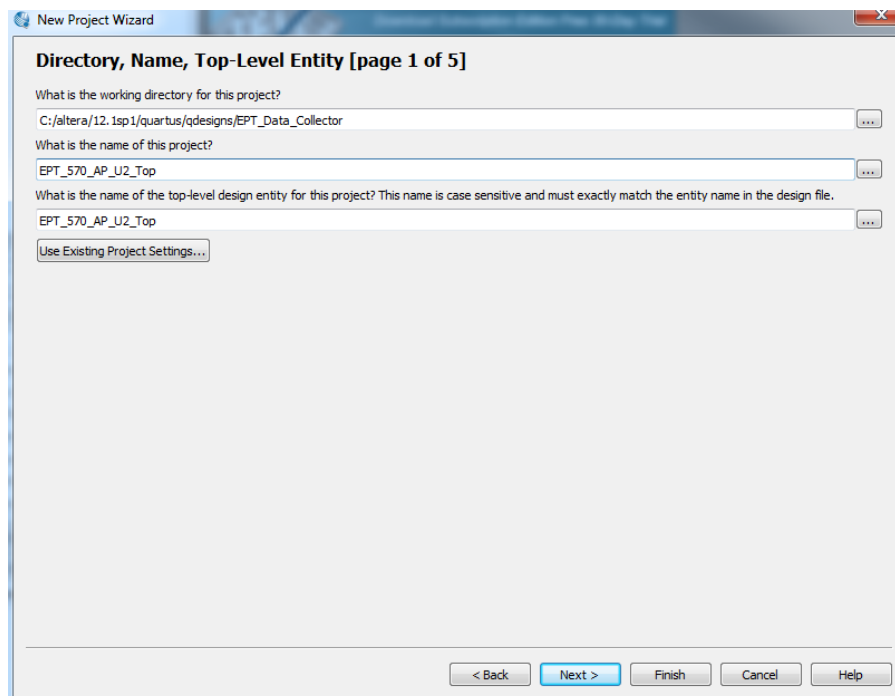
The Quartus Prime application will compile/ synthesize the user code, active_transfer_library, and the active EndTerms. The result of this step is a file containing the CPLD code with “*.pof”. First, we need to create a project in the Quartus Prime environment. Follow the directions in the section: “Compiling,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

Bring up Quartus Prime, then use Windows Explorer to browse to c:/altera/xxx/quartus/qdesigns create a new directory called: “EPT_Data_Collector”.



Open Quartus Prime by clicking on the icon .

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





USB CPLD Development System User Manual

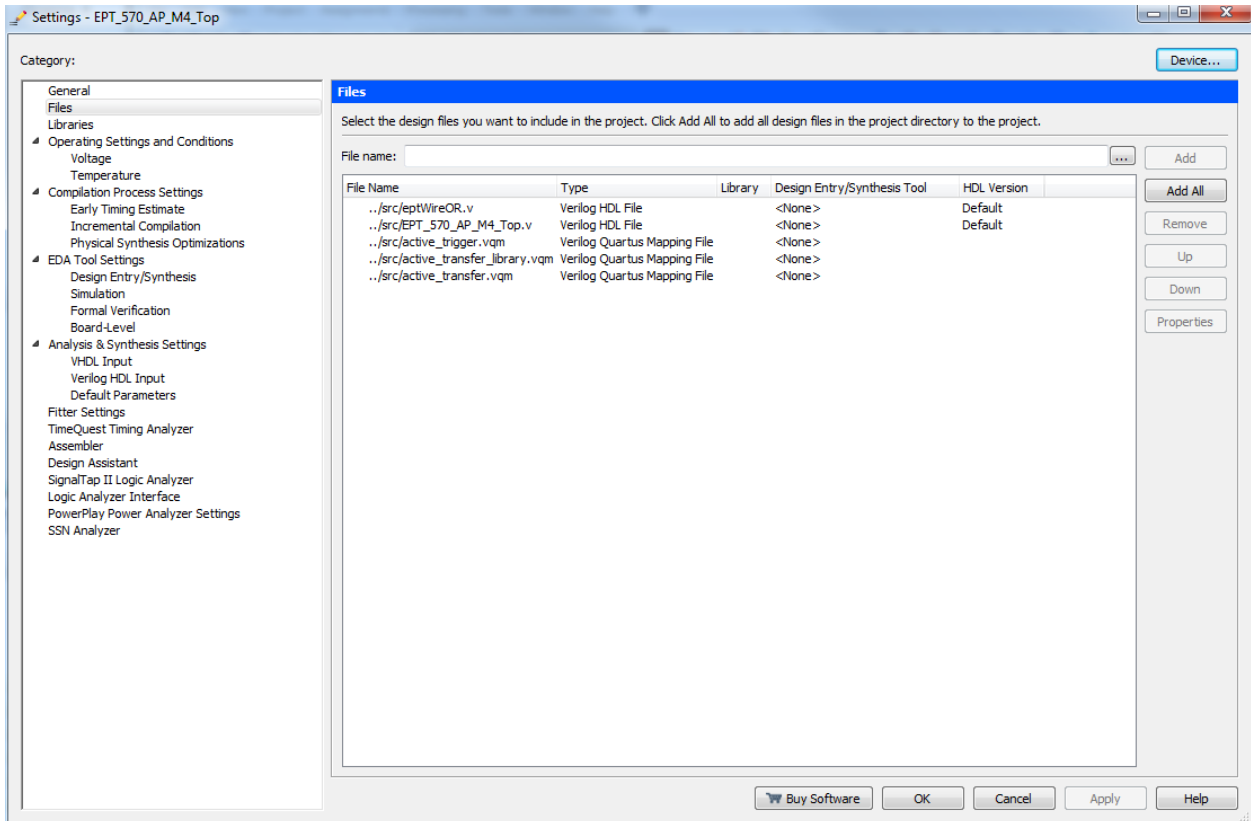
At the Top-Level Entity page, browse to the `c:\altera\xxx\quartus\qdesigns\EPT_Data_Collector` directory to store your project. Type in a name for your project “EPT_570_AP_M4_Top”.

Follow the steps up to Add Files. At the Add Files box, click on the Browse button and navigate to the project Data Collector install folder in the dialog box. Browse to the `\Projects_HDL\EPT_570_AP_Data_Collector\EPT_570_AP_M4_Top` folder of the EPT USB-CPLD Development System CD. Copy the files from the `\src` directory.

- Active_transfer.vqm
- Active_trigger.vqm
- Active_transfer_library.vqm
- eptWireOr.v
- EPT_570_AP_M4_Top.v

Add the files:

USB CPLD Development System User Manual



Continue following the instructions by adding a device and finishing the project instantiation. Then, bring up the Pin Planner.

- Under Assignments, Select Import Assignments.
- At the Import Assignment dialog box, browse to the \Projects_HDL\EPT_Data_Collector\EPT_5M57_AP_M4_Top folder of the EPT USB-CPLD Development System CD. Select the Quartus Specification file, “EPT_5M57_AP_M4_Top.qsf” file.
- Click Ok. Under Assignments, Select Pin Planner. Verify the pins have been imported correctly.

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

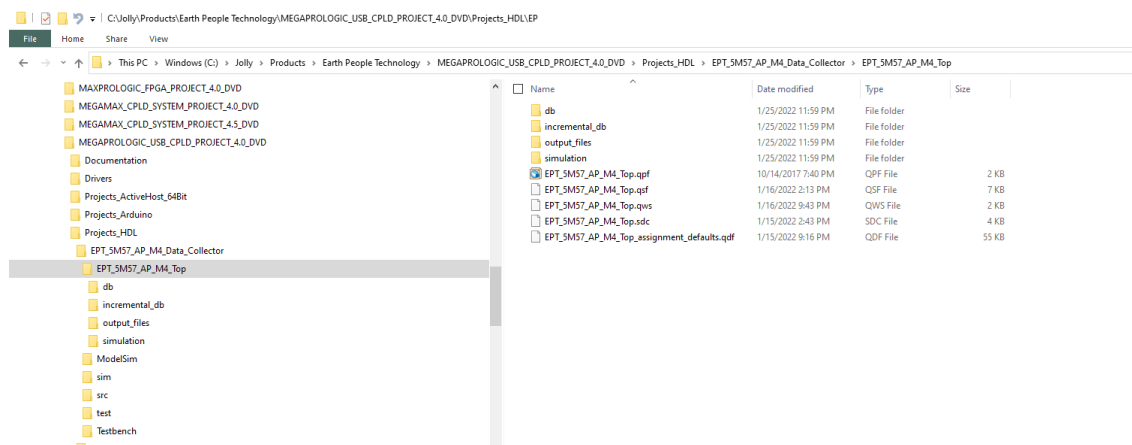


USB CPLD Development System User Manual

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

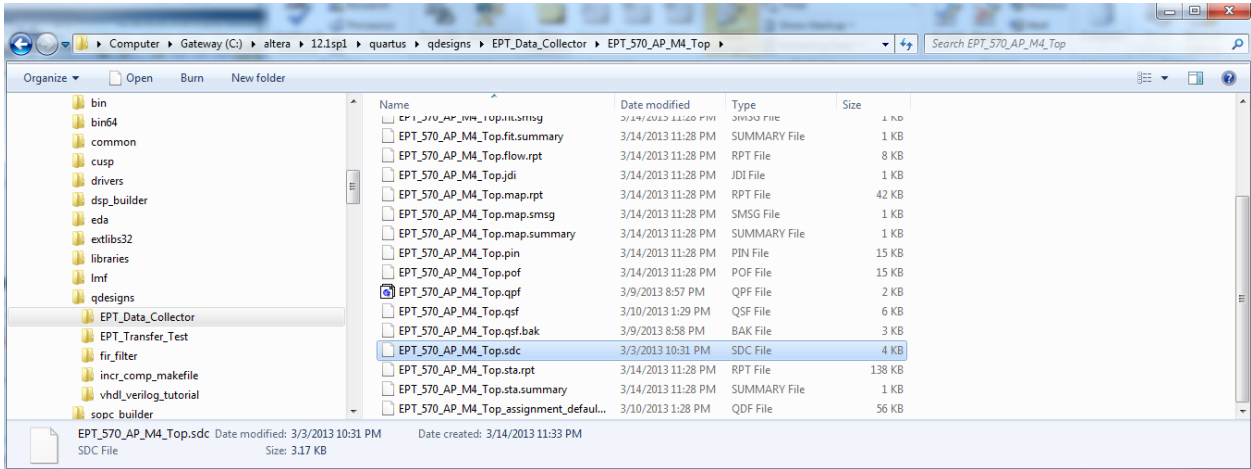
[http://www.altera.com/literature/hb/qts/qts_qii53018.pdf?GSA_pos=1&WT.oss_r=1&WT.oss=TimeQuest Timing Analyzer](http://www.altera.com/literature/hb/qts/qts_qii53018.pdf?GSA_pos=1&WT.oss_r=1&WT.oss=TimeQuest%20Timing%20Analyzer)

Browse to the
\\Projects_HDL\EPT_5M57_AP_M4_Data_Collector\EPT_5M57_AP_M4_Top folder
of the EPT USB-CPLD Development System DVD. Select the
“EPT_570_AP_M4_Top.sdc” file.



Copy the file and browse to c:\
intelFPGA_lite\xxx\quartus\qdesigns\EPT_Data_Collector directory. Paste the file.

USB CPLD Development System User Manual

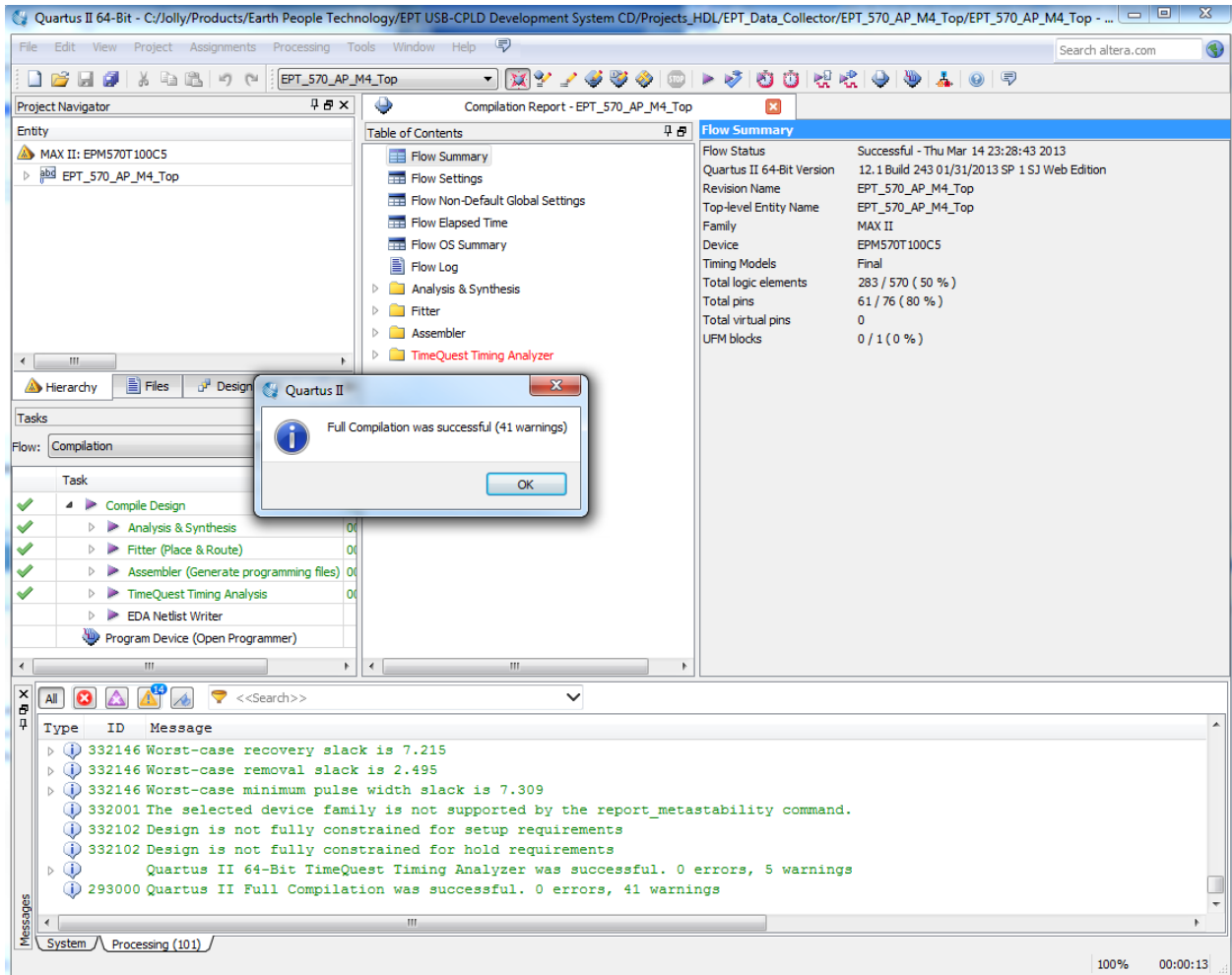


and select the Start Compilation button.



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

USB CPLD Development System User Manual



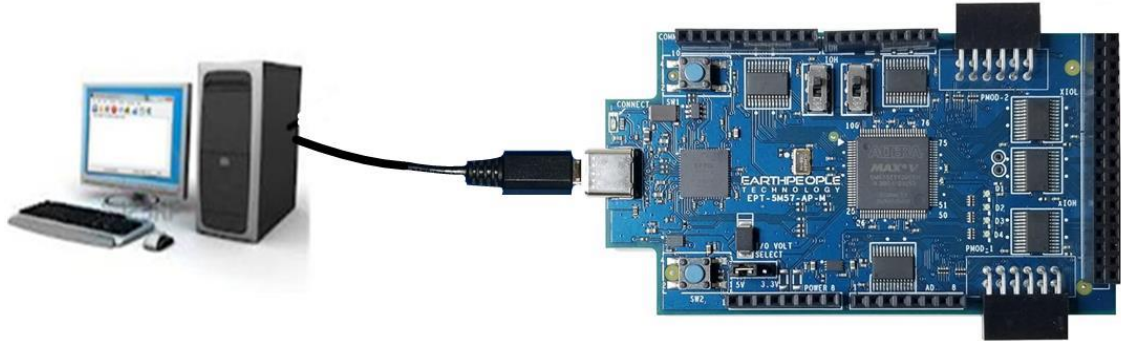
If the synthesis fails, you will see the failure message in the message window. Note that in addition to fatal errors, the compile process can produce “warnings” which do not necessarily prevent execution of the code but which should be corrected eventually.

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

7.1.10 **CPLD: Program the CPLD**

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

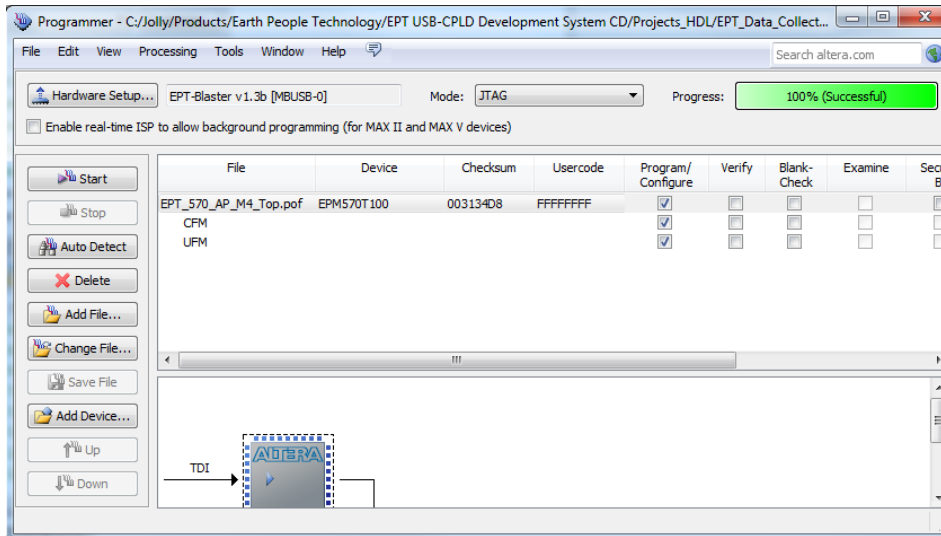
USB CPLD Development System User Manual



- Connect the MegaProLogic to the PC,
- Open up Quartus Prime,
- Open the programmer tool
- In the upper left corner of the Programmer Tool, there is a button labeled “Hardware Setup”. Verify that EPT-Blaster v1.0b” has been selected. If not, go to the section JTAG DLL Insert to Quartus Prime and follow the directions.
- Check the box under Program/Configure
- Click the Start button.

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

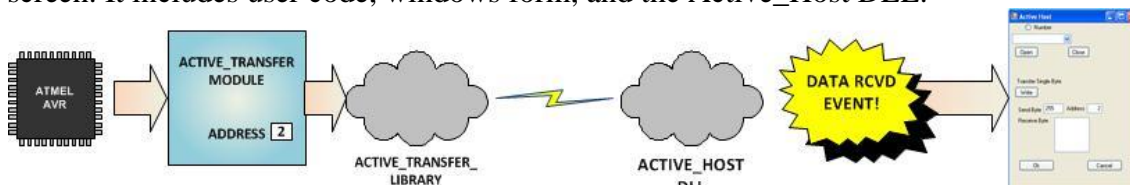
USB CPLD Development System User Manual



At this point, the MEGAPROLOGIC is programmed and ready for use.

7.1.11 PC: Design the Project

The final piece of the Data Collection Sampler is the PC application. This application will fetch the data from the CPLD of the MEGAPROLOGIC and display it on the screen. It includes user code, windows form, and the Active_Host DLL.



The Active_Host DLL is designed to transfer data from the 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, from the user code on the PC, makes the data transfer transparent. The data just appears in memory and the user code will direct the data to a textbox on the Windows Form.

The Data Collector project will perform the following functions.

- Find MEGAPROLOGIC Device.
- Open MEGAPROLOGIC Device.

USB CPLD Development System User Manual

- Start the Arduino data collection process.
- Wait for data from MEGAPROLOGIC.
- Display data from MEGAPROLOGIC in textbox.

7.1.12 PC: Coding the Project

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

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

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

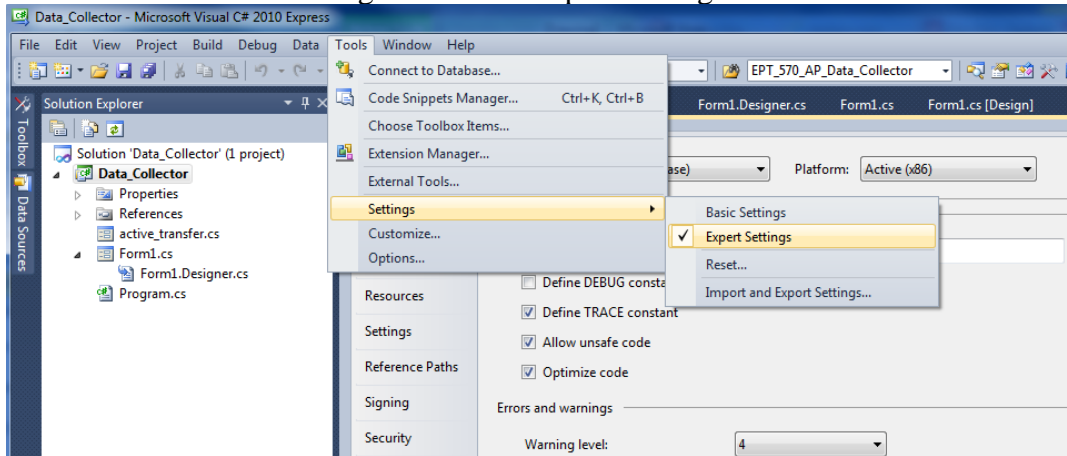
namespace Data_Collector
{
    public partial class Data_Collector : System.Windows.Forms.Form
    {
        public Data_Collector()...
```

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

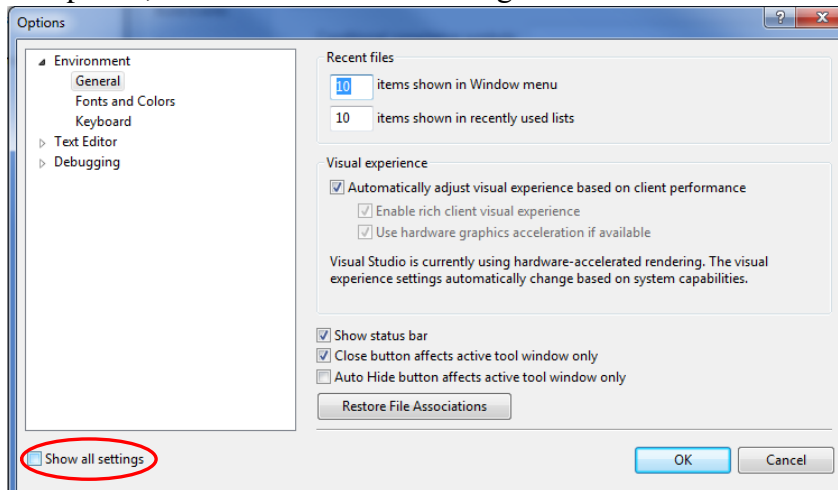
The project environment must be set up correctly in order to produce an application that runs correctly on the target platform. Visual C# Express defaults new projects to 32 bits. If you OS is a 64 bit platform, use the following directions to set up a 64 bit project.

USB CPLD Development System User Manual

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

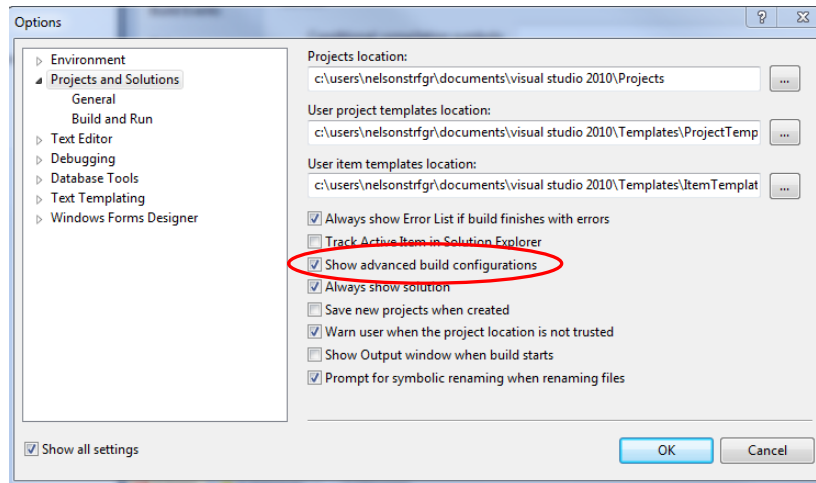


Go to Tools->Options, locate the “Show all settings” check box. Check the box.

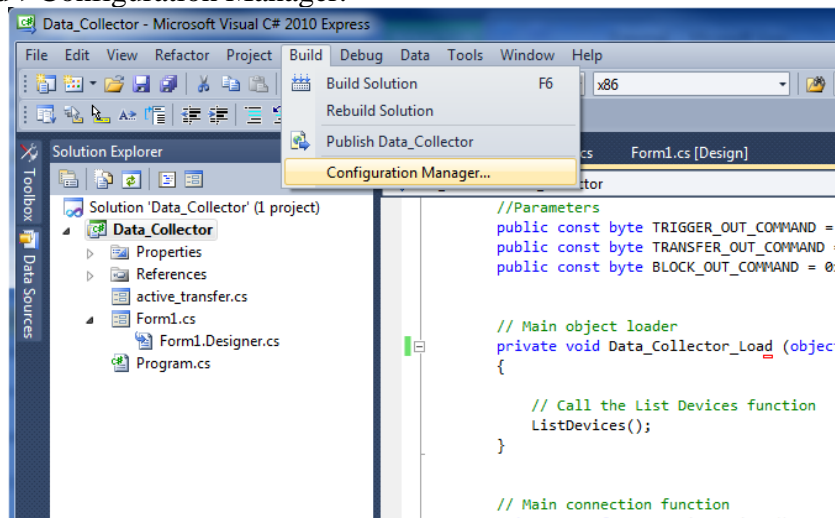


In the window on the left, go to “Projects and Solutions”. Locate the “Show advanced build configurations” check box. Check the box.

USB CPLD Development System User Manual

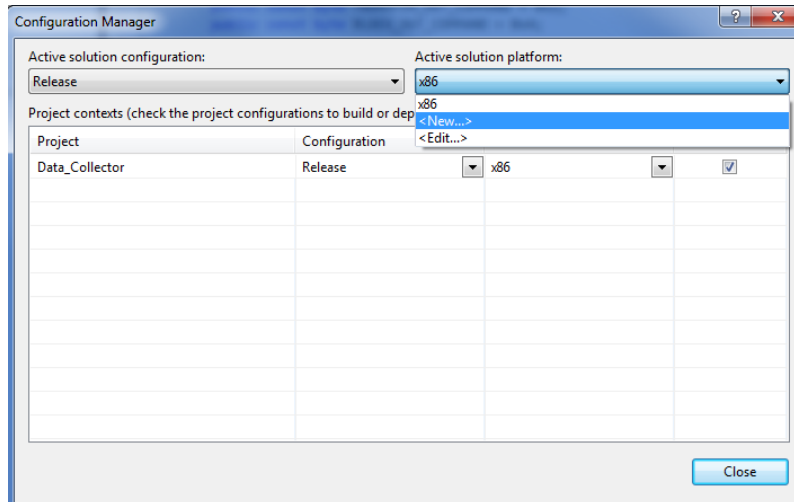


Go to Build->Configuration Manager.

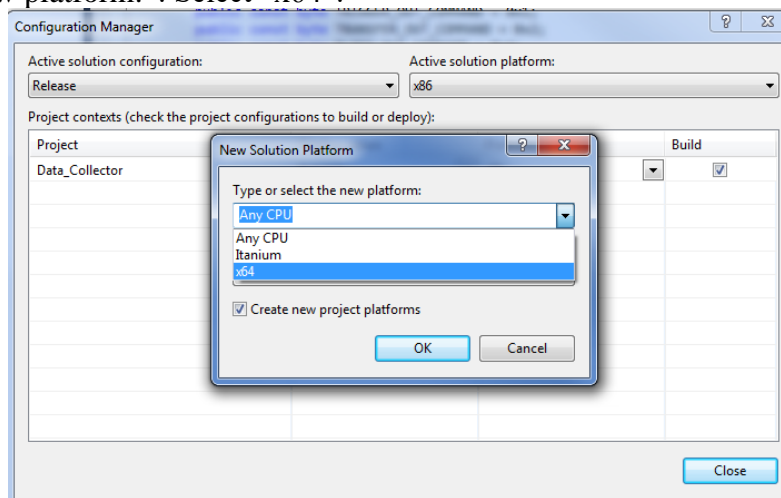


In the Configuration Manager window, locate the “Active solution platform:” label, select “New” from the drop down box.

USB CPLD Development System User Manual

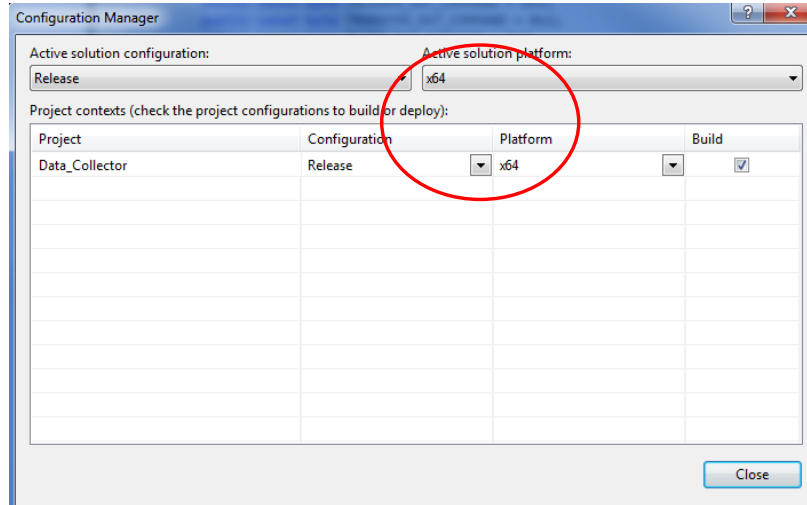


In the New Solution Platform window, click on the drop down box under “Type or select the new platform:”. Select “x64”.



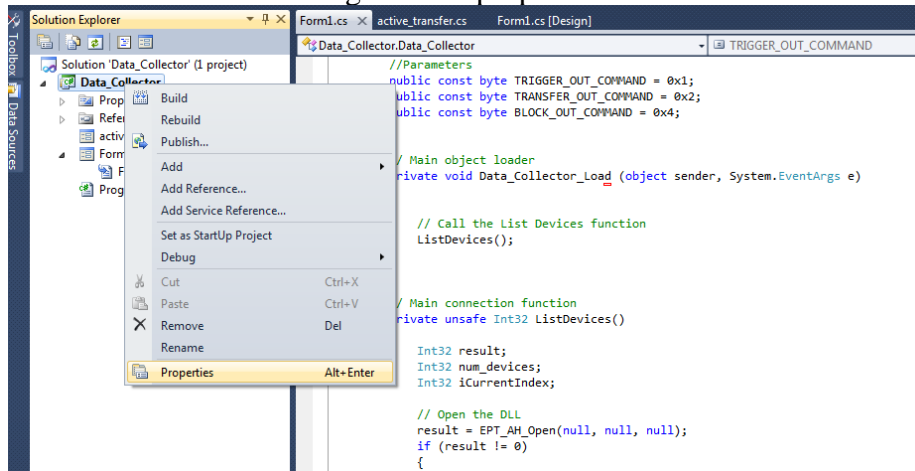
Click the Ok button. Verify that the “Active Solution Platform” and the “Platform” tab are both showing “x64”.

USB CPLD Development System User Manual



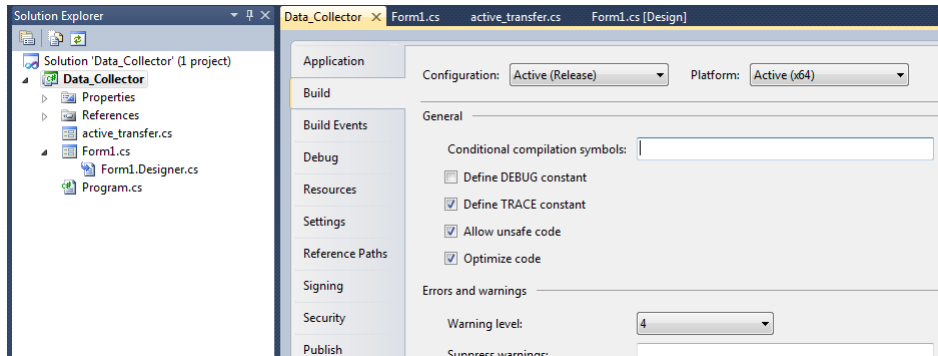
Click Close.

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.

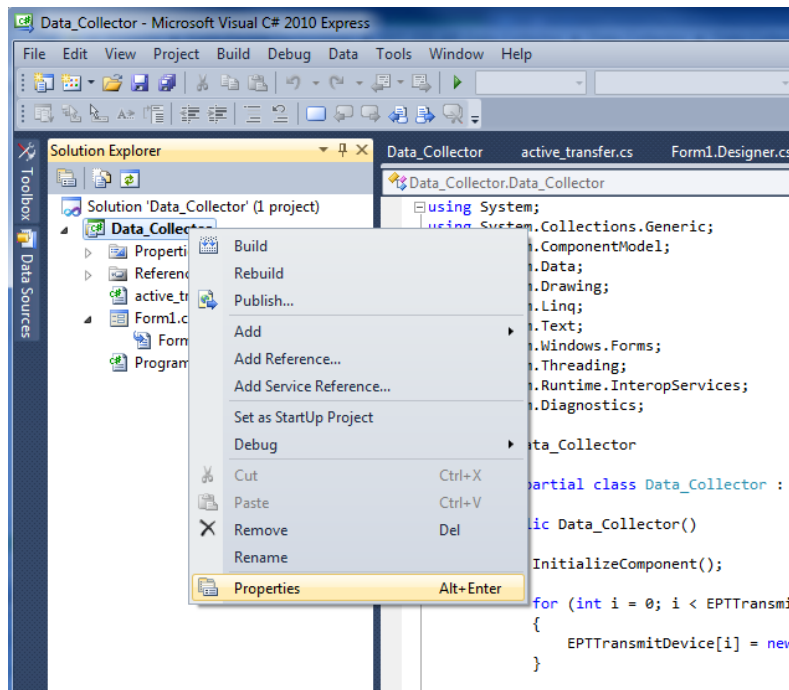


Verify that the “Platform:” label has “Active (x64)” selected from the drop down box.

USB CPLD Development System User Manual



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



Now we are ready to start coding.

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



USB CPLD Development System User Manual

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

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

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

The first function called when the Windows Form loads up is the `Data_Collector_Load()`. This function is called automatically upon the completion of the Windows Form, so there is no need to do anything to call it. Once this function is called, it in turn calls the `ListDevices()`. Use the function `List Devices()` to detect all EPT devices connected to the PC.

```
private void EPT_Transfer_Test_Load(object sender, System.EventArgs e)
{
    //String buffer
    String PortText = "";

    //Index registers
    int Index = 0, EPTgroupNumber = 0;

    // Call the List Devices function
    List<string> names = ComPortNames("0403", "6010");

    // Get a list of serial port names.
    string[] ports;
    ports = SerialPort.GetPortNames();
}
```

USB CPLD Development System User Manual

```
if (names.Count > 0)
{
    foreach (String port in ports)
    {
        //Compare port name with the found VID/PID
        //combinations. Add them to Matching port list
        //and comboDevList
        if (names.Contains(port))
        {
            MatchingComPortList[Index] = port;

            if (Index == 0)
            {
                PortText = "EPT JTAG Blaster " + EPTgroupNumber;
                Index++;
            }
            else
            {
                PortText = "EPT Serial Communications " + EPTgroupNumber++;
                Index++;
            }
            cmbDevList.Items.Add(PortText);
        }
    }
}
else
    MessageBox.Show("No EPT Devices found!");
```

The ListDevices() function calls the

```
ports = SerialPort.GetPortNames();
```

to determine the Serial devices attached to the PC. Next,

```
if (names.Contains(port))
```

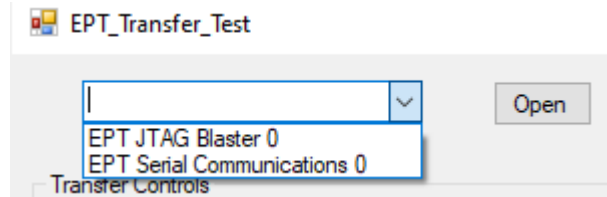
is called inside a for loop to return the ASCII name of each Serial device attached to the PC. It will automatically populate the combo box, cmbDevList with all the EPT devices it finds.

```
cmbDevList.Items.Add(PortText);
```

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

USB CPLD Development System User Manual

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

OpenSerialPort1()

function. The device_index is passed into the function. If the function is successful, the device name is displayed in the label, labelDeviceCnt. Next, the Open button is grayed out and the Close button is made active.

```

1 reference
public bool OpenSerialPort1()
{
    try
    {
        //Set the serial port parameters
        serialPort_AH.PortName = PortName;
        serialPort_AH.BaudRate = Convert.ToInt32(BaudRate);
        serialPort_AH.Parity = (Parity)Enum.Parse(typeof(Parity), vParity);
        serialPort_AH.DataBits = Convert.ToInt16(DataBits);
        serialPort_AH.StopBits = (StopBits)Enum.Parse(typeof(StopBits), StopBits);
        serialPort_AH.Handshake = (Handshake)Enum.Parse(typeof(Handshake), pHandshake);

        if (!serialPort_AH.IsOpen)
        {
            serialPort_AH.Open();
            btnOpenDevice.Enabled = false;
            btnCloseDevice.Enabled = true;
            //textBox1.ReadOnly = false;
            return true;
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    return false;
}

```


USB CPLD Development System User Manual

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

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

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

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

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

- `btnWriteByte`
- `btnTransferReset`
- `btnOk`
- `btnClose`
- `btnResetBlock`

Refer to section 6.1.4 Adding Controls to the Project for details about using the ToolBox to place controls on a design. The `btnWriteByte` click event calls the `EPT_AH_SendTransferControlByte()`. This function is used to turn on/off bits in the Control Register in the CPLD code. The `btnWriteByte` will set the `start_stop_cntrl` signal in the CPLD to one. This signal starts the Arduino Data Collector sending its random word to the CPLD.



USB CPLD Development System User Manual

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

The btnTransferReset sets the start_stop_cntrl bit in the Control Register to zero. This action will cause the Arduino Data Collector to stop sending the random word to the CPLD.

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

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

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

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



USB CPLD Development System User Manual

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

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

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

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

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

7.1.13 PC: Compiling the Active Host Application

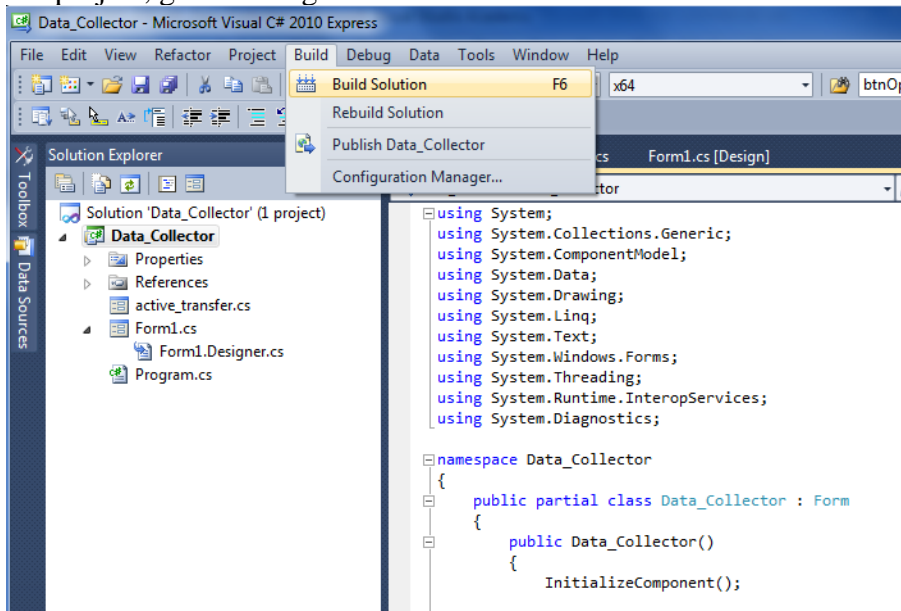
Building the Data_Collector project will compile the code in the project and produce an executable file. It will link all of the functions declared in the opening of the Data_Collector Class with the Active Host dll. The project will also automatically link the FTD2XX.dll to the object code. Follow section: Assembling, Building, and Executing a .NET Project on the PC. Browse to the \Projects_ActiveHost_xxBit\EPT_Data_Collector\Data_Collector\ folder of the EPT USB-CPLD Development System CD. Copy the following files into the project.

- Active_transfer_xxx.cs
- Data_Collector.csproj
- Data_Collector.csproj.user
- Form1.cs

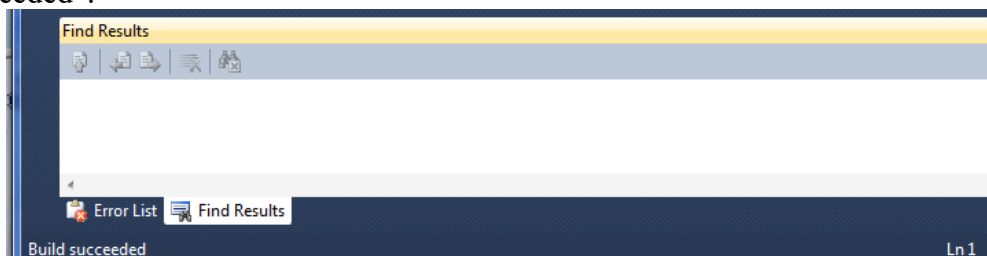
USB CPLD Development System User Manual

- Form1.Designer.cs
- Program.cs

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



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



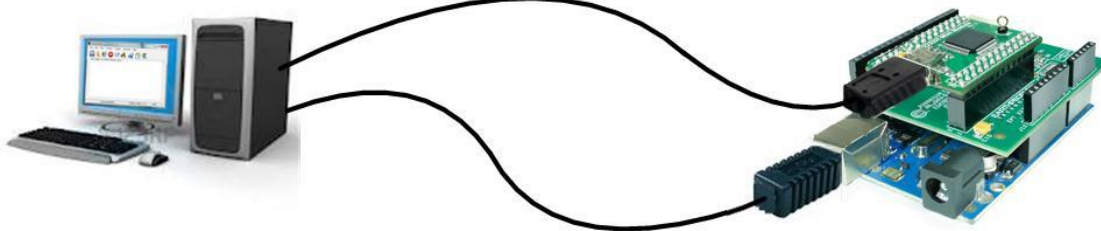
If the build fails, you will have to examine each error in the “Error List” and fix it accordingly. If you cannot fix the error using troubleshooting methods, post a topic in the Earth People Technology Forum. All topics will be answered by a member of the technical staff as soon as possible.

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

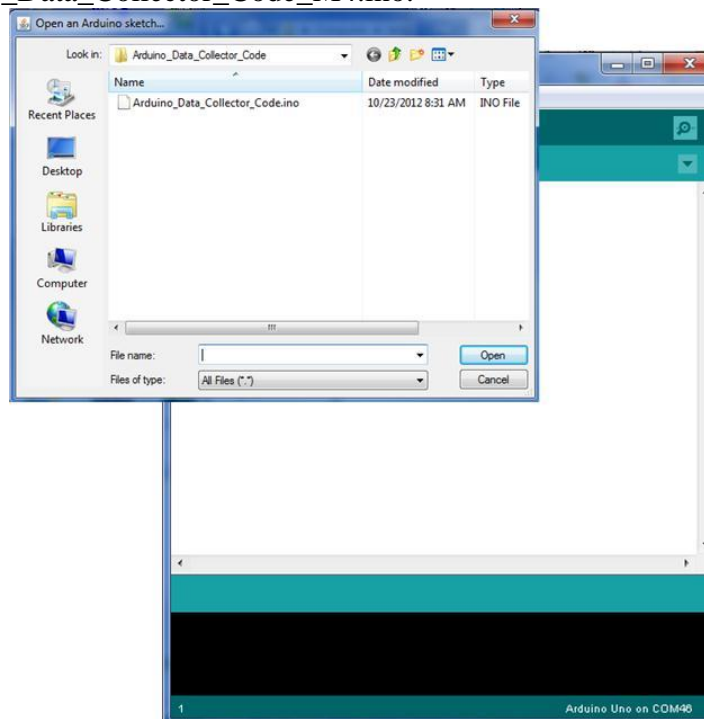
USB CPLD Development System User Manual

7.1.14 Connecting the Project Together

Now we will connect the Arduino, EPT 570-AP-M4, and the PC to make a Data Collector. First, connect a USB cable from a USB port on the PC to the Arduino. Second, connect a USB cable from an open USB port on the PC to the EPT 570-AP-M4.

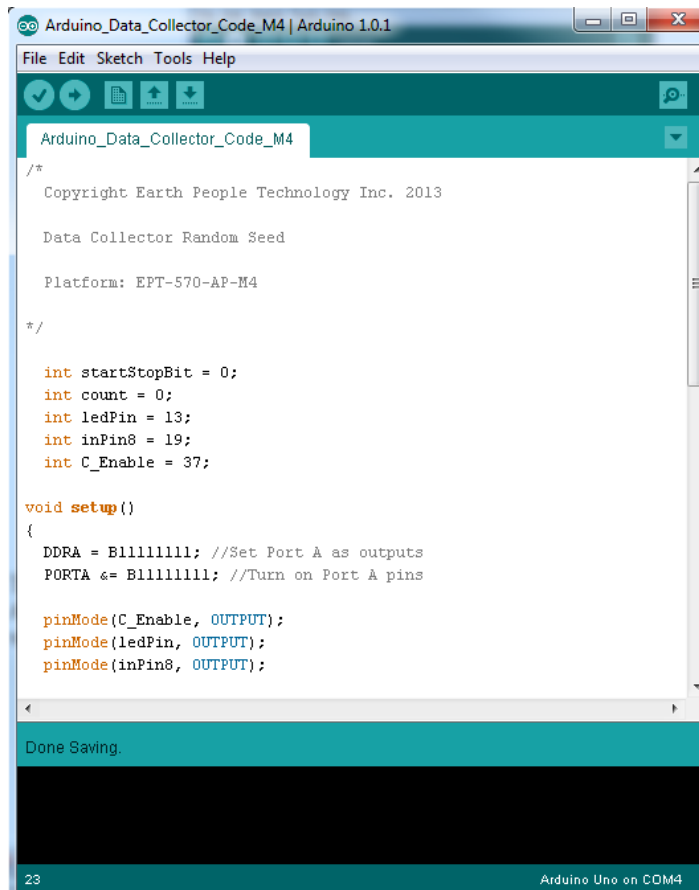


Next, open the Arduino IDE and select File->Open and select your sketch created earlier, `Arduino_Data_Collector_Code_M4.ino`.



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

USB CPLD Development System User Manual



```
Arduino_Data_Collector_Code_M4 | Arduino 1.0.1
File Edit Sketch Tools Help
Arduino_Data_Collector_Code_M4
/*
  Copyright Earth People Technology Inc. 2013

  Data Collector Random Seed

  Platform: EPT-570-AP-M4
*/

int startStopBit = 0;
int count = 0;
int ledPin = 13;
int inPin8 = 19;
int C_Enable = 37;

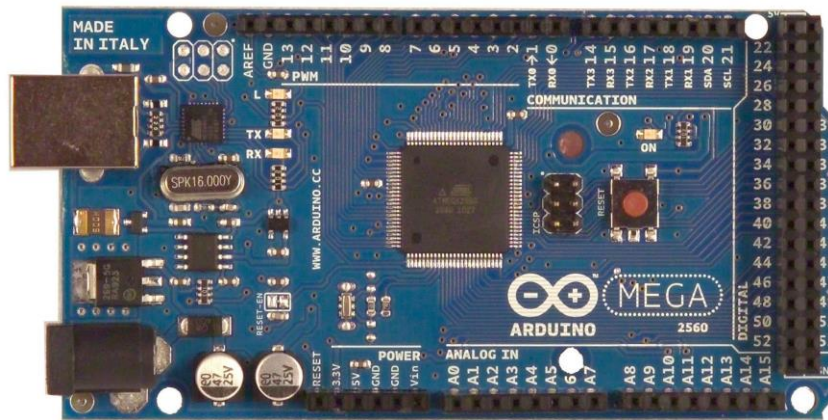
void setup()
{
  DDRA = B11111111; //Set Port A as outputs
  PORTA &= B11111111; //Turn on Port A pins

  pinMode(C_Enable, OUTPUT);
  pinMode(ledPin, OUTPUT);
  pinMode(inPin8, OUTPUT);
}

Done Saving.
23 Arduino Uno on COM4
```

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

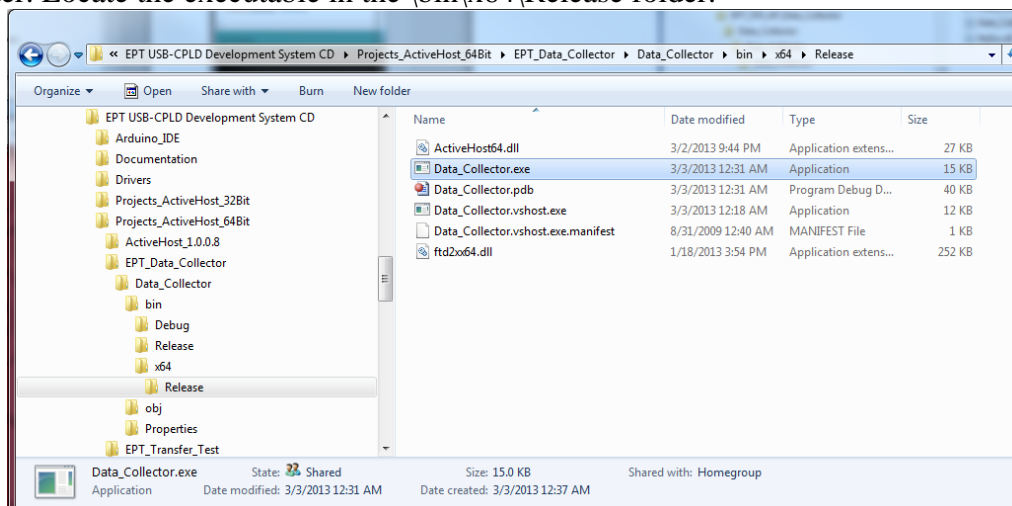
USB CPLD Development System User Manual



If this LED is blinking at the rate of once per second, the Arduino and the Data Collector project are ready for the EPT 570-AP-M4 code.

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

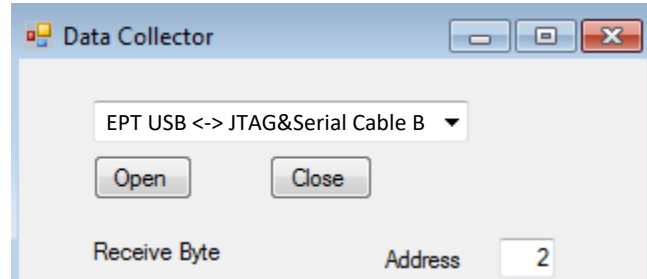
Open the EPT Data Collector on the PC by browsing to the Data Collector project folder. Locate the executable in the \bin\x64\Release folder.



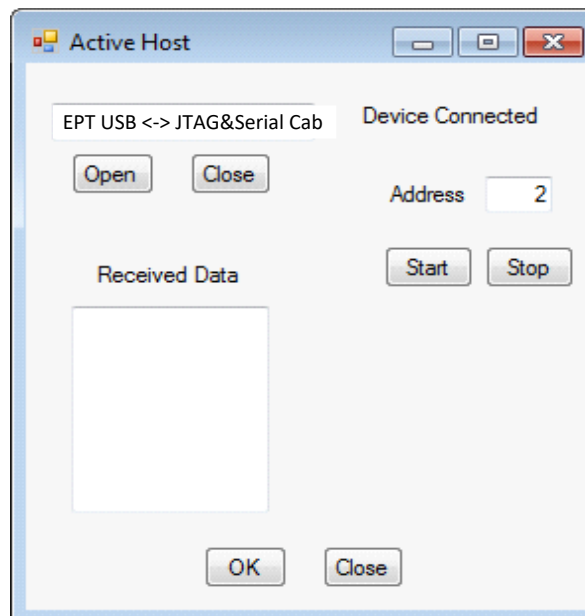
Initiate the application by double clicking the application icon in the \Release folder of the project. The application will open and automatically load the Active Host dll. The

USB CPLD Development System User Manual

application will locate the EPT_5M57-AP-M4 device. Next, the combo box at the top will be populated with the name of the device.



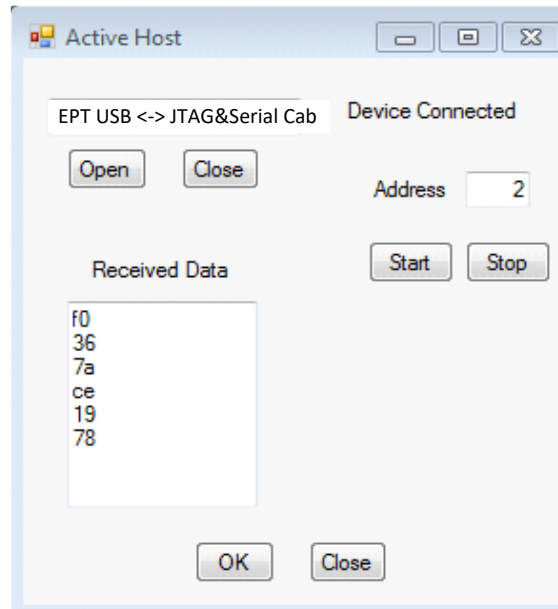
Select the EPT-5M57-AP device and click the Open button. If the Active Host application connects to the device, a label will indicate “Device Connected”. Next, select the address of the Active Transfer module in the CPLD. In our case it is “2”.



7.1.15 Testing the Project

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

USB CPLD Development System User Manual



And that's all there is to the Data Collector Project. It's up to the user to use this project as a base to create much larger projects. You can easily make a volt meter using this project by turning off the Random number generator in the Arduino and reading the Analog Pins. Also, reformat the textBox display that it shows in decimal instead of the Hexadecimal display.



USB CPLD Development System User Manual

APPENDIX I

List of Abbreviations and Acronyms

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