USB CPLD Development System User Manual

EARTH PEOPLE TECHNOLOGY, Inc

MEGAMAX DEVELOPMENT SYSTEM
User Manual

The EPT MegaMax development system provides an innovative method of developing and debugging programmable logic code. It also provides a high speed data transfer mechanism between the board and a host PC. The MegaMax 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 cable. The Active Host SDK provides a highly configurable communications interface between the board 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.

http://www.earthpeopletechnology.com/

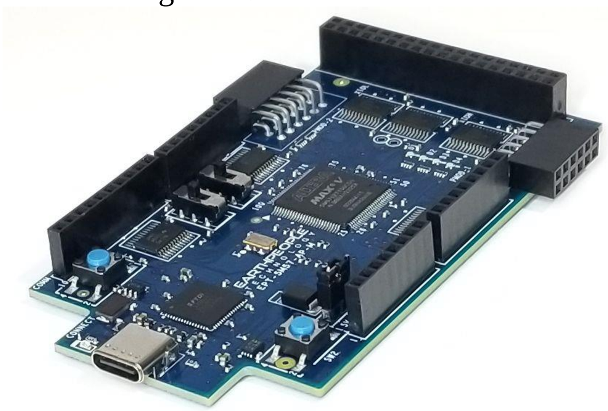
USB CPLD Development System User Manual

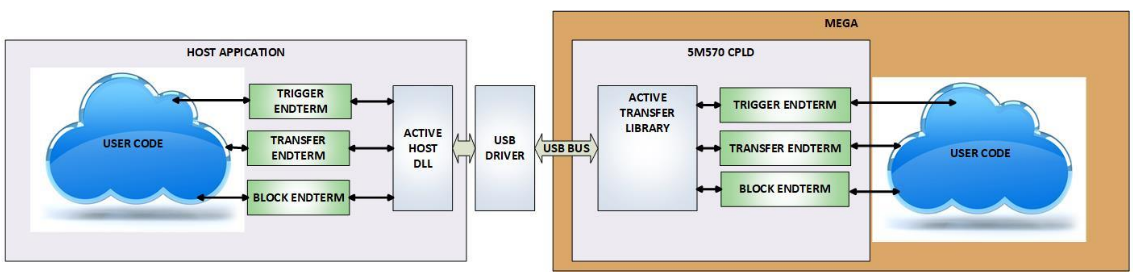
Table of Contents

USB CPLD Development System User Manual

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 High Speed transfer path. The software consists of the Active Host SDK for the PC. The firmware includes the Active Transfer Library which is used in the CPLD to provide advanced functions for control and data transfer to/from the MegaMax.



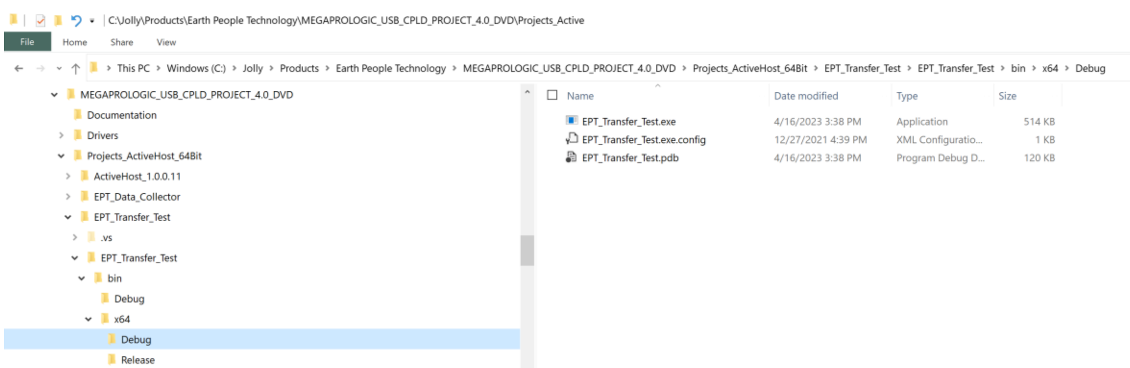


The EPT USB-CPLD Development System allows users to write HDL code (either Verilog or VHDL) that will implement any digital logic circuit. The user's HDL code is compiled and synthesized and packaged into a programming file. The programming file is programmed into the CPLD using the JTAG channel of the USB to Serial chip, the FT2232H.The Active Host SDK provides a COM Port serial path which maintains device connection, polling, writes and includes a unique receive mechanism that automatically transfers data from MegaMax when data is ready. It also alerts the user code when the software 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 SDK 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 .

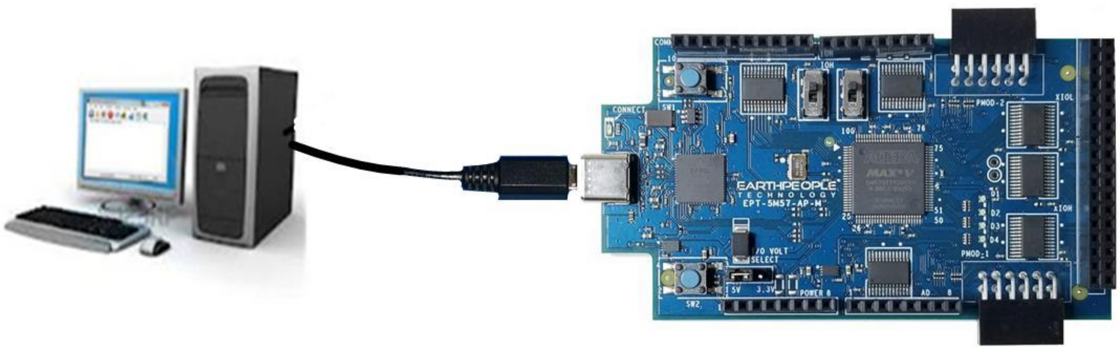
1.1 *Test Driving the Active Host Test Application*

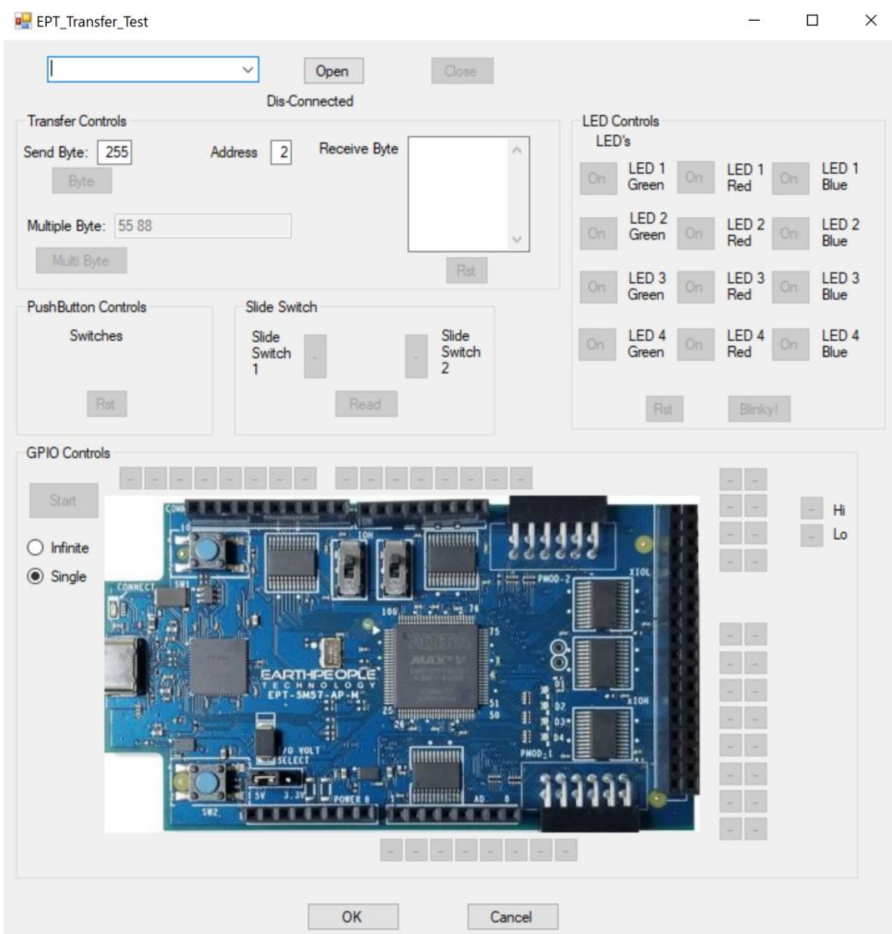
The MegaMax board comes pre-loaded with the EPT_Transfer_Test HDL project in the CPLD. This project allows the user to test out the functions of the Active Host API and the board hardware.



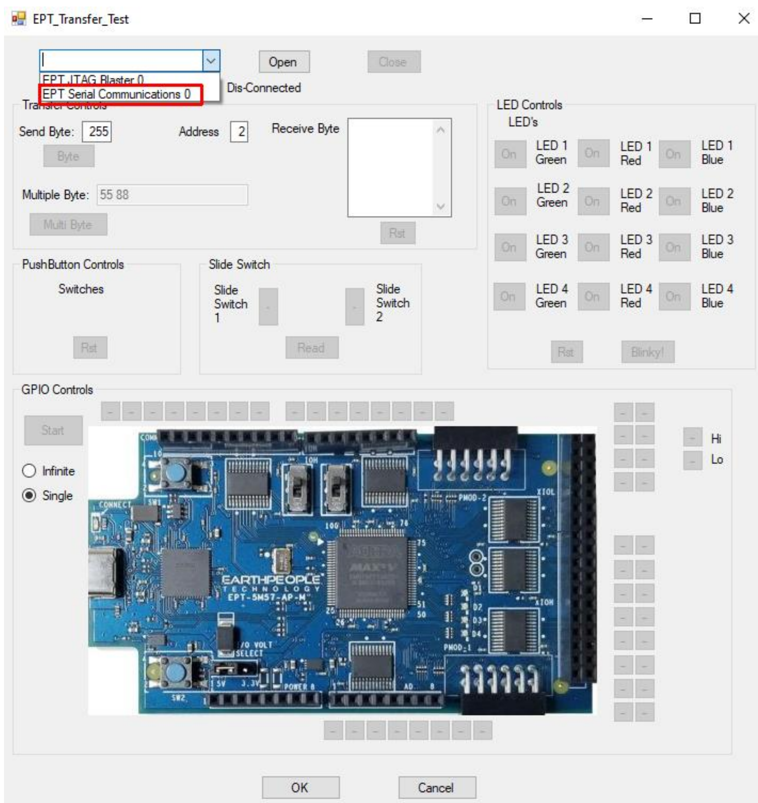
To test drive the application, connect the MegaMax to the Windows PC using Type A to USB-C cable. Load the driver for the board. See the section EPT Drivers for instructions on loading the MegaMax driver. If the USB driver fails to load, the Windows OS will indicate that no driver was loaded for the 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 MEGAMAX_CPLD_SYSTEM_PROJECT_4.x_DVD. The application should load with a Windows form.

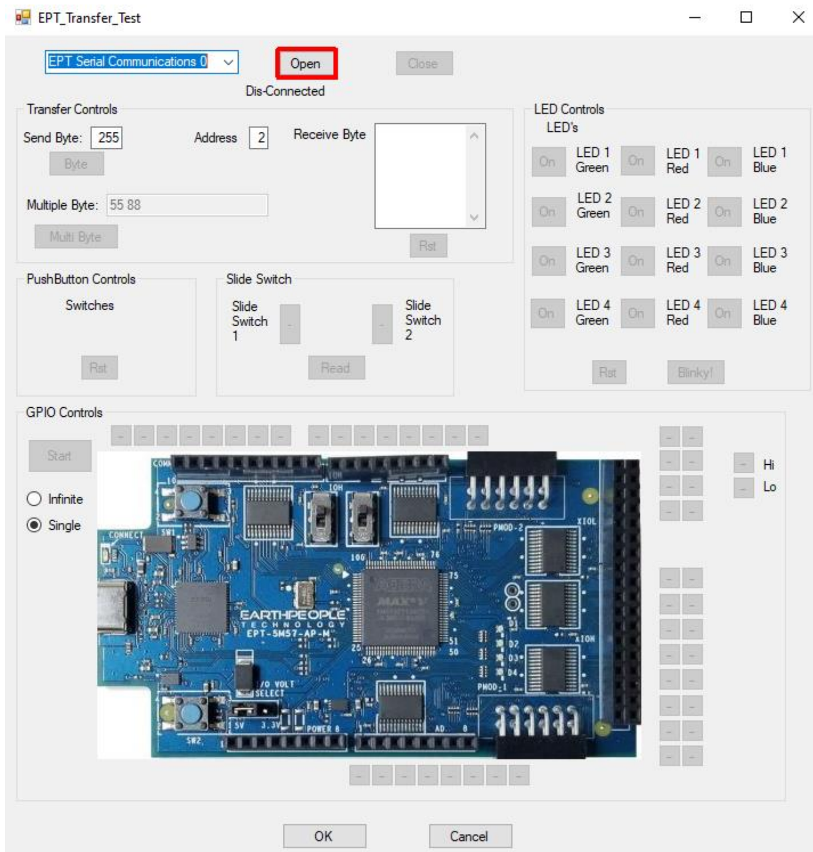




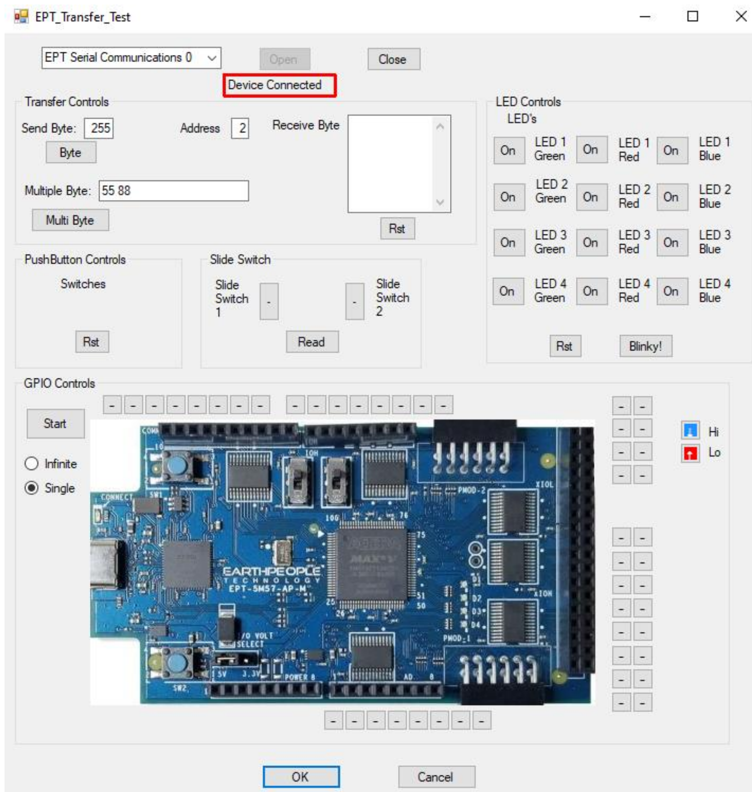
With the application loaded, select the EPT Serial Communications x board from the dropdown combo box.



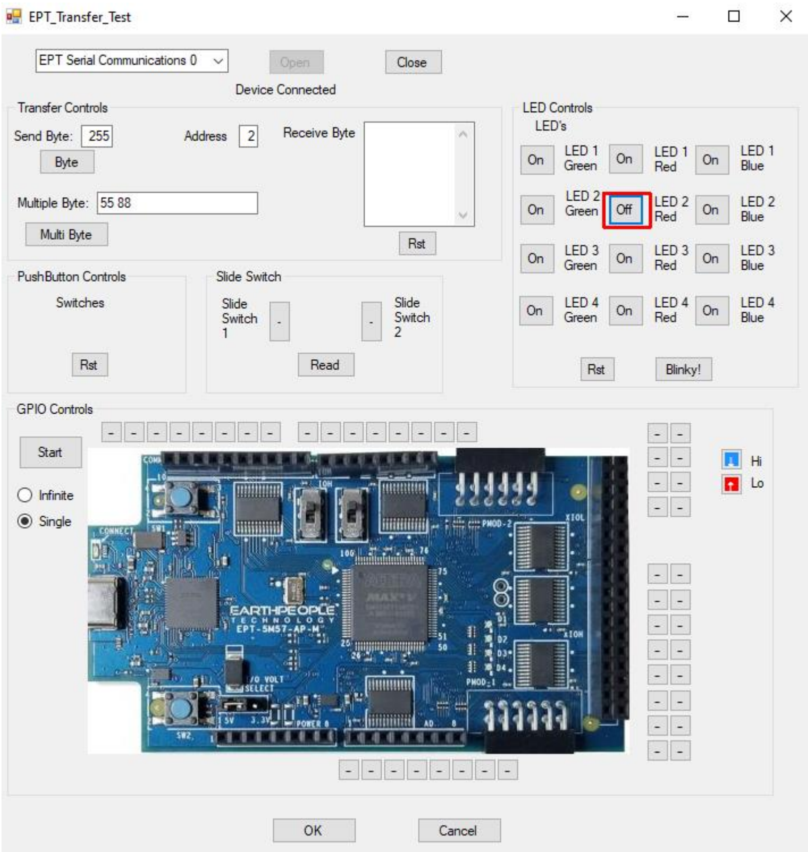
Next, click on the Open button.

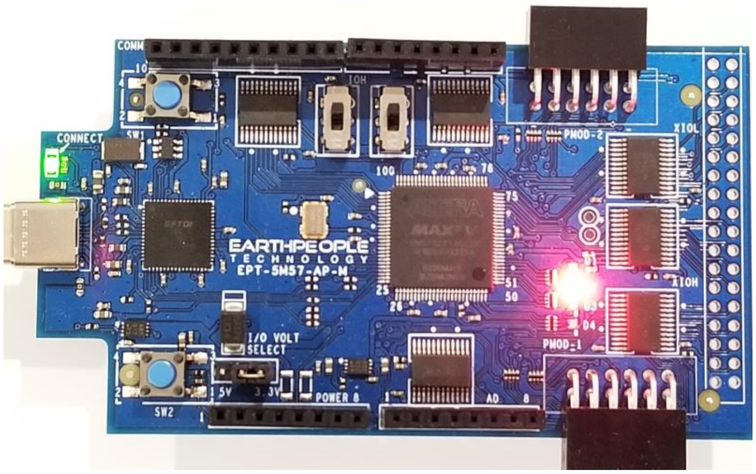


The “Device Connected” label should be seen to indicate the Windows App is now connected to the MegaMax.

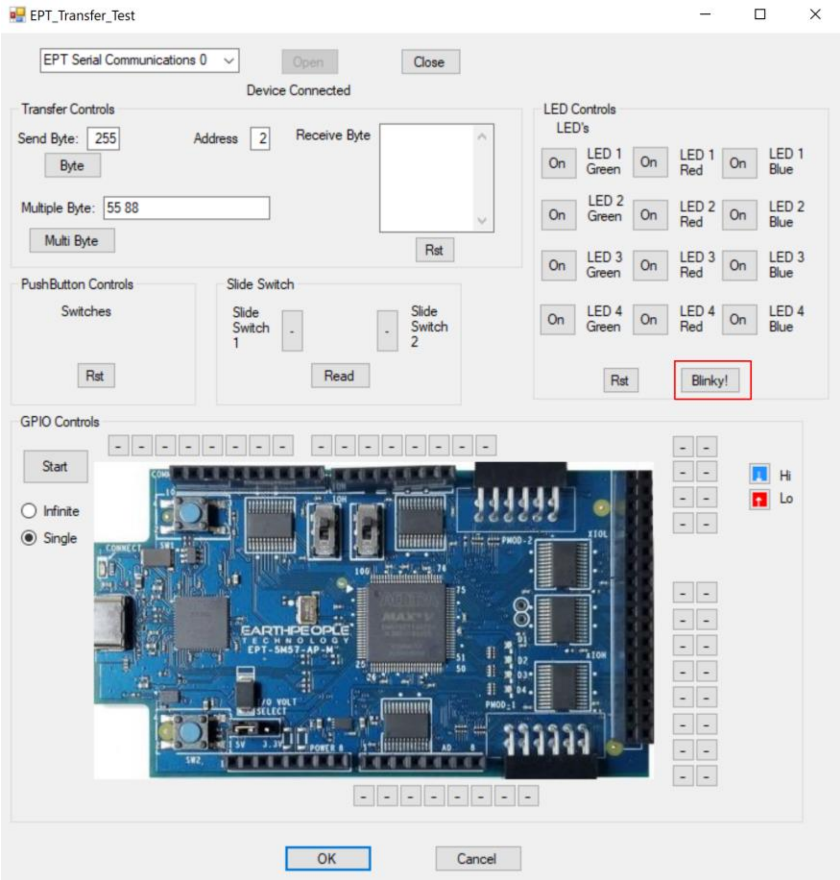


Click on one of the LED buttons in the middle of the window. The corresponding LED on the MegaMax board should light up.

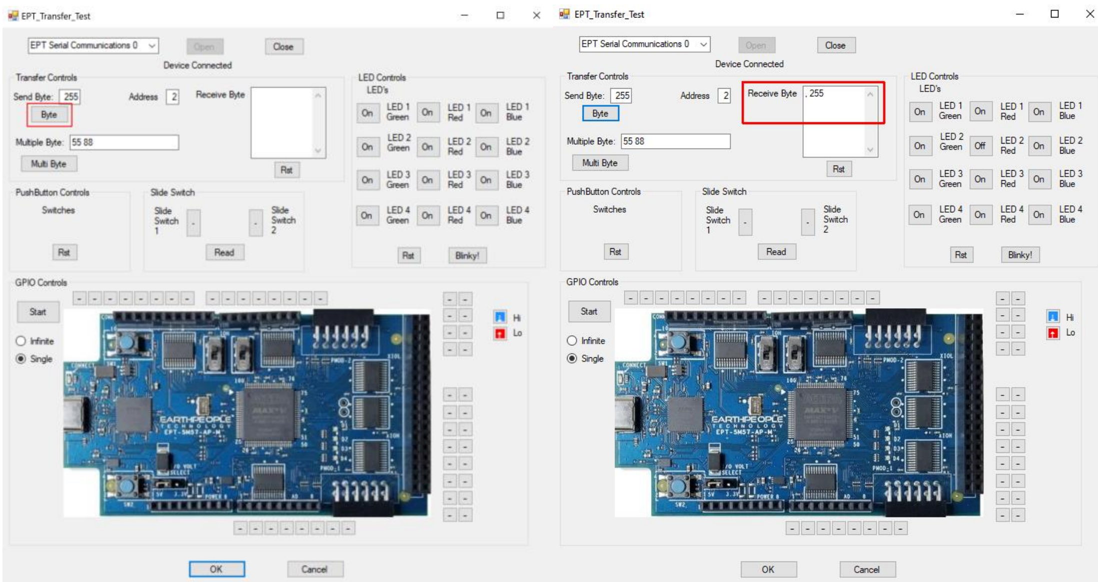




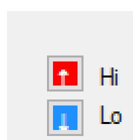
Clink on the Blinky button for a light show.

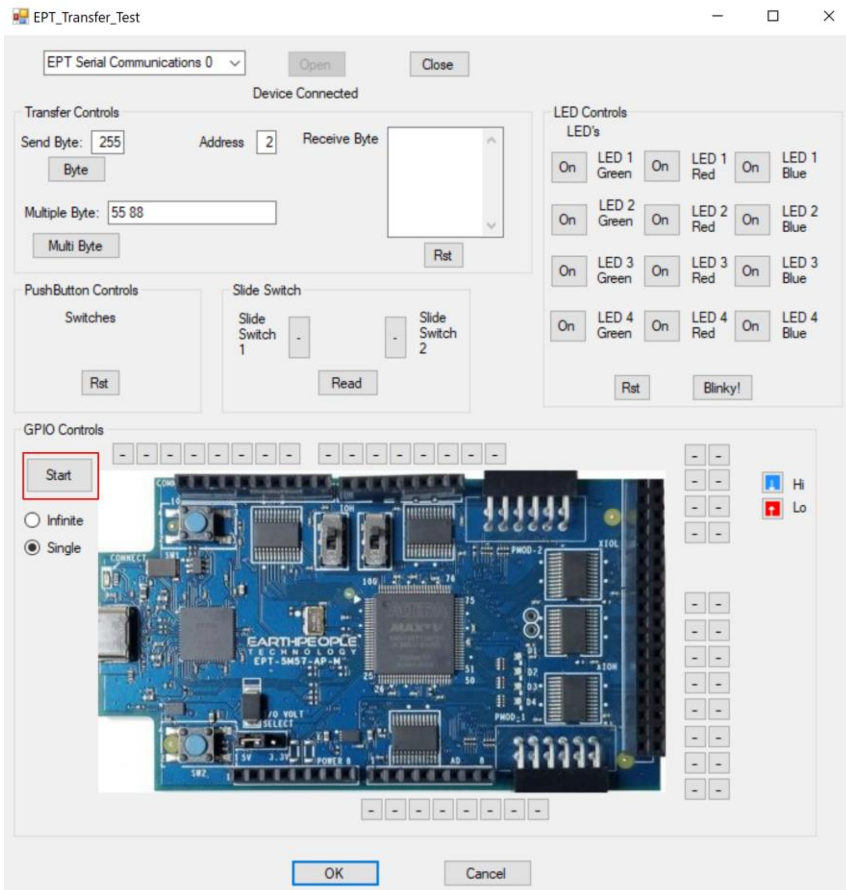


To exercise the Single Byte Transfer EndTerm, click the “Byte” button in the Transfer Controls group. Type in several numbers separated by a space and less 256 into the Multiple Byte textbox. Then hit the Multi Byte button. The numbers appear in the Receive Byte textbox.

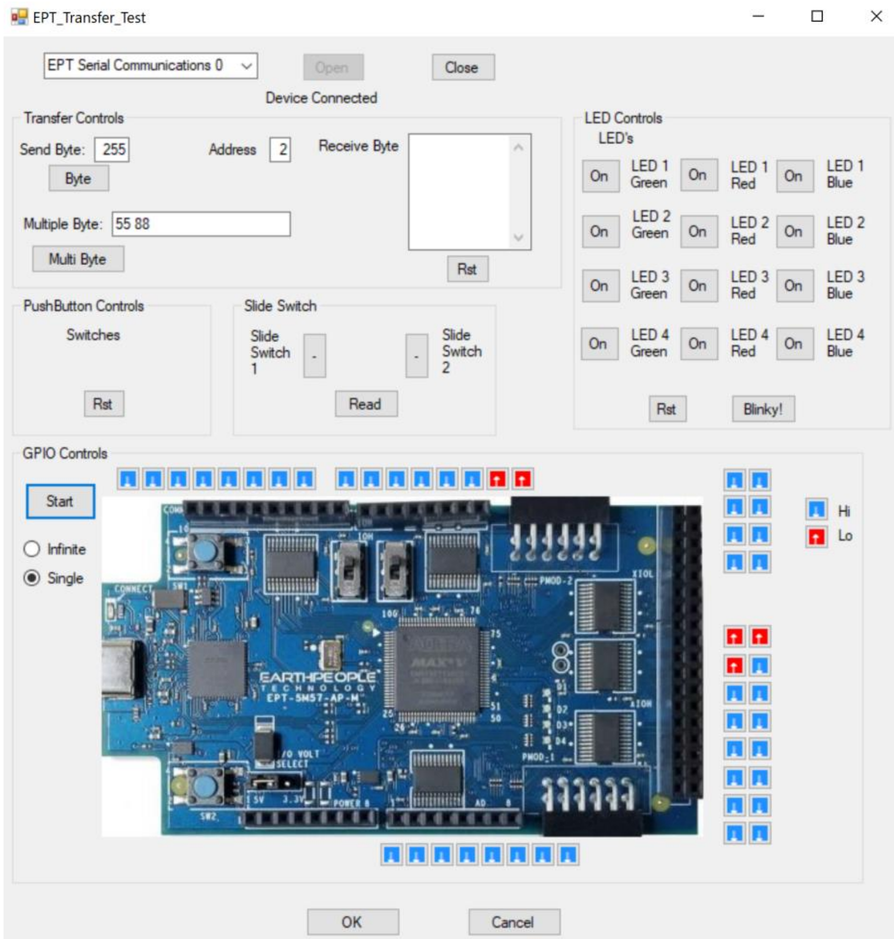


To exercise the Block Transfer EndTerm, click the “Start” button in the GPIO Controls group. The MegaMax will sample the state of each Input pin of the CPLD that is connected to a board edge connector. The Transfer Test Window will display the results of each pin, Hi or Lo

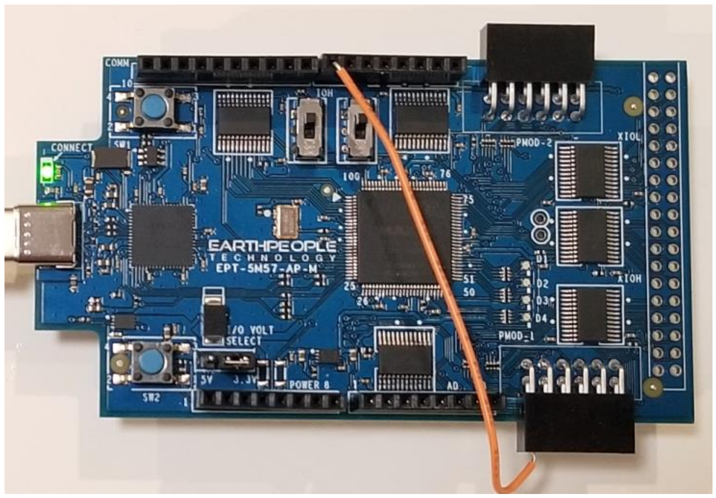


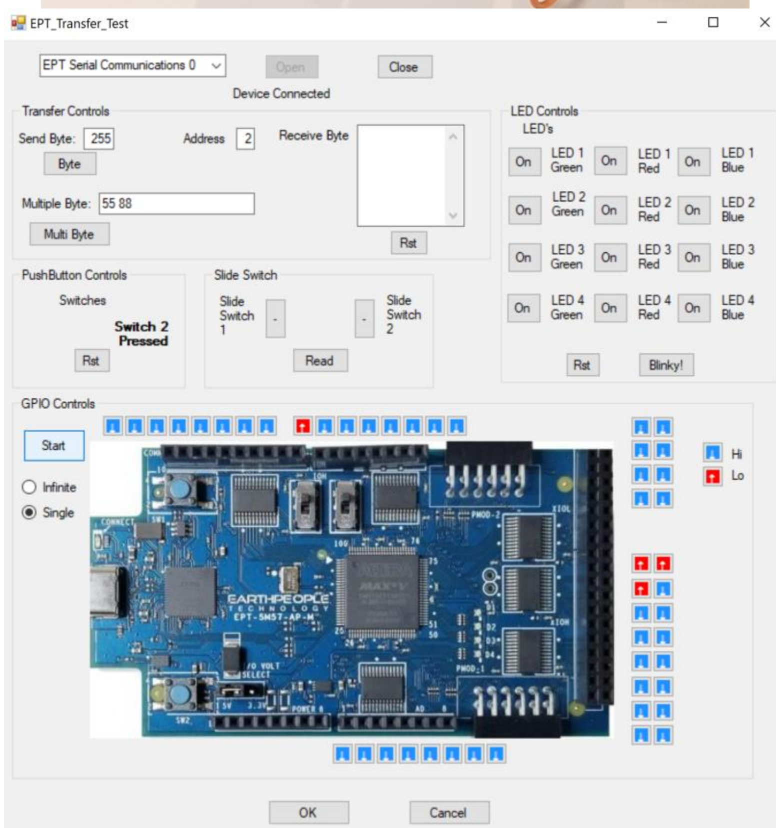


The results of each pin are displayed next to the image of the MegaMax in separate buttons.

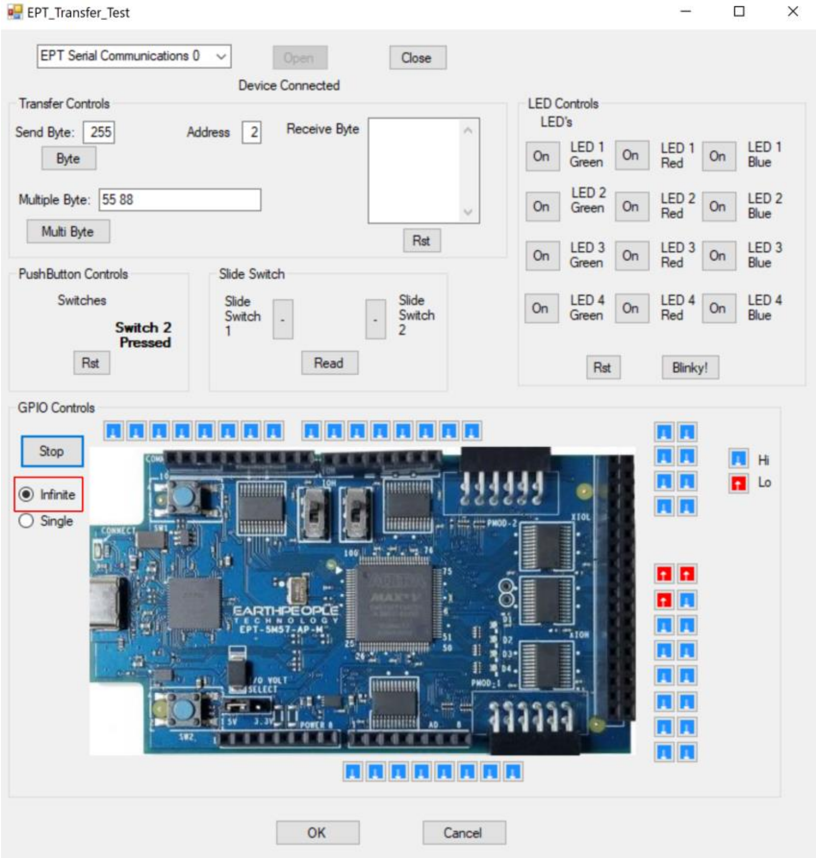


Use a jumper cable to connect between +3.3V and any of the Input pins to see the display detect a high on the selected pin.

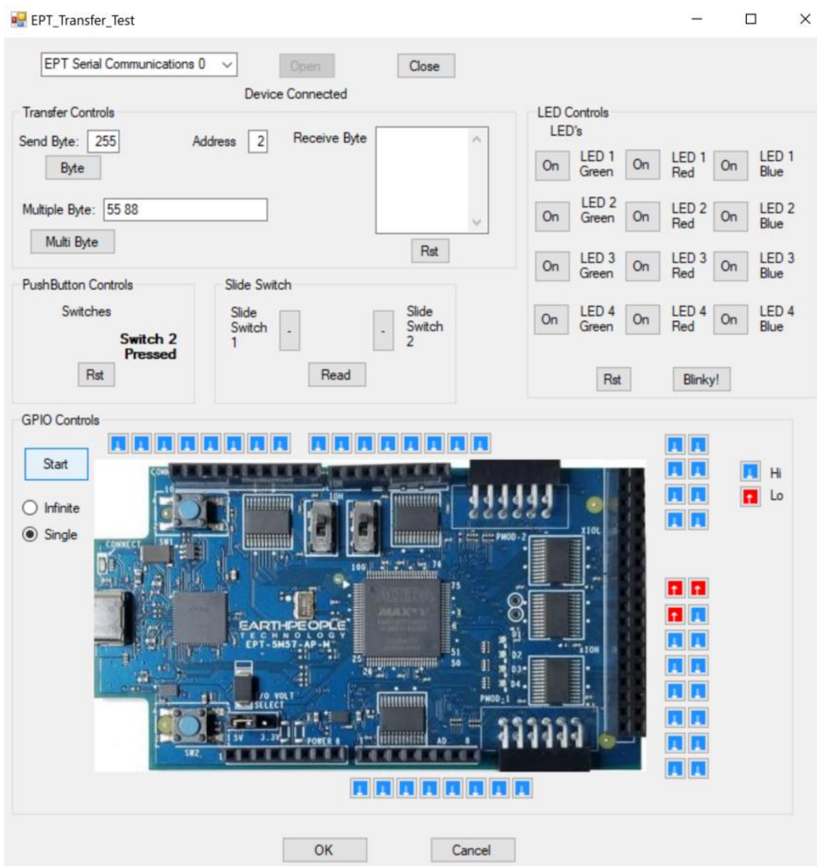

EARTHPEOPLE
TECHNOLOGY
EPT-5M57-AP-M
I/O VOLT
SELECT
CONNECT
SW1
SW2
PMOD-2
PMOD-1
XIOL
XIOH
POWER 8


EPT_Transfer_Test
EPT Serial Communications 0
Open
Close
Device Connected
Transfer Controls
Send Byte: 255
Address 2
Receive Byte
Byte
Multiple Byte: 55 88
Multi Byte
Rst
LED Controls
LED's
On LED 1 Green
On LED 1 Red
On LED 1 Blue
On LED 2 Green
On LED 2 Red
On LED 2 Blue
On LED 3 Green
On LED 3 Red
On LED 3 Blue
On LED 4 Green
On LED 4 Red
On LED 4 Blue
Rst
Blinky!
PushButton Controls
Switches
Switch 2 Pressed
Rst
Slide Switch
Slide Switch 1
Slide Switch 2
Read
GPIO Controls
Start
Infinite
Single
Hi
Lo
OK
Cancel

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

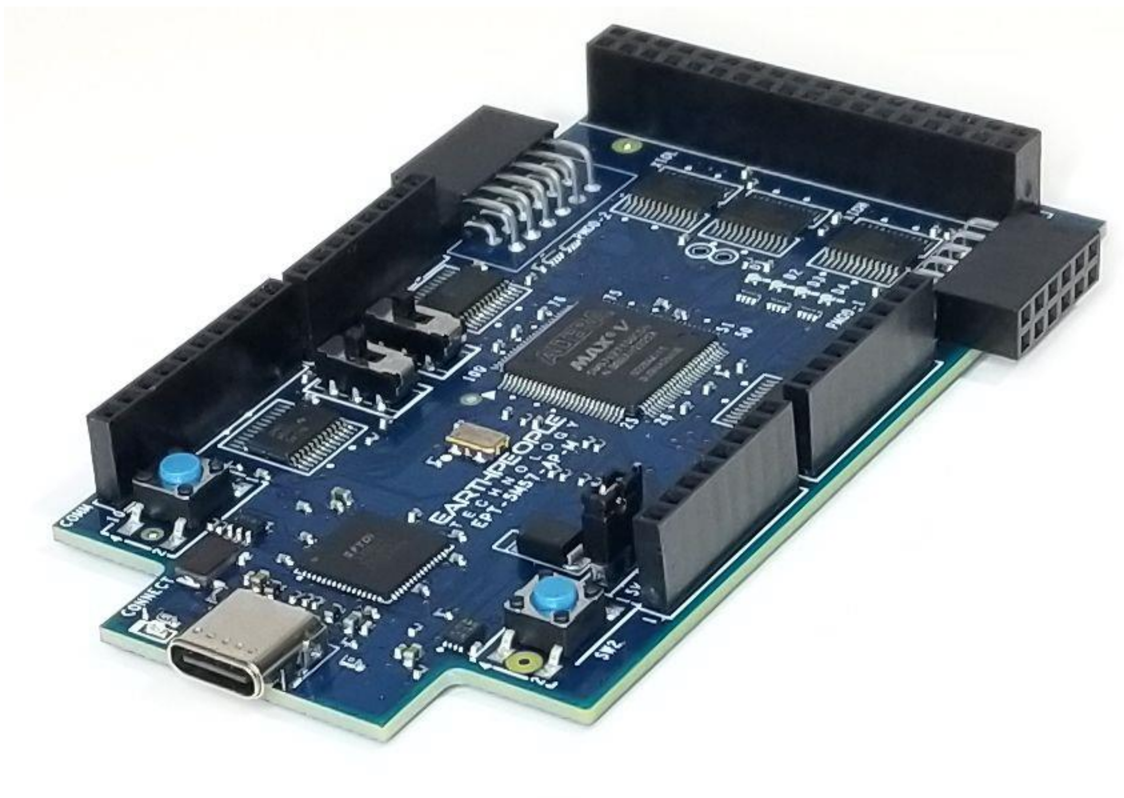


Press the PCB switches on the MegaMax to view the Switch Controls in action.


EPT_Transfer_Test
EPT Serial Communications 0
Open
Close
Device Connected
Transfer Controls
Send Byte: 255
Address 2
Receive Byte
Byte
Multiple Byte: 55 88
Multi Byte
Rst
LED Controls
LED's
On
LED 1 Green
LED 1 Red
LED 1 Blue
LED 2 Green
LED 2 Red
LED 2 Blue
LED 3 Green
LED 3 Red
LED 3 Blue
LED 4 Green
LED 4 Red
LED 4 Blue
Blinky!
PushButton Controls
Switches
Switch 2 Pressed
Slide Switch
Slide Switch 1
Slide Switch 2
Read
GPIO Controls
Start
Infinite
Single
Hi
Lo
OK
Cancel

EARTHPEOPLE Technology

USB CPLD Development System User Manual

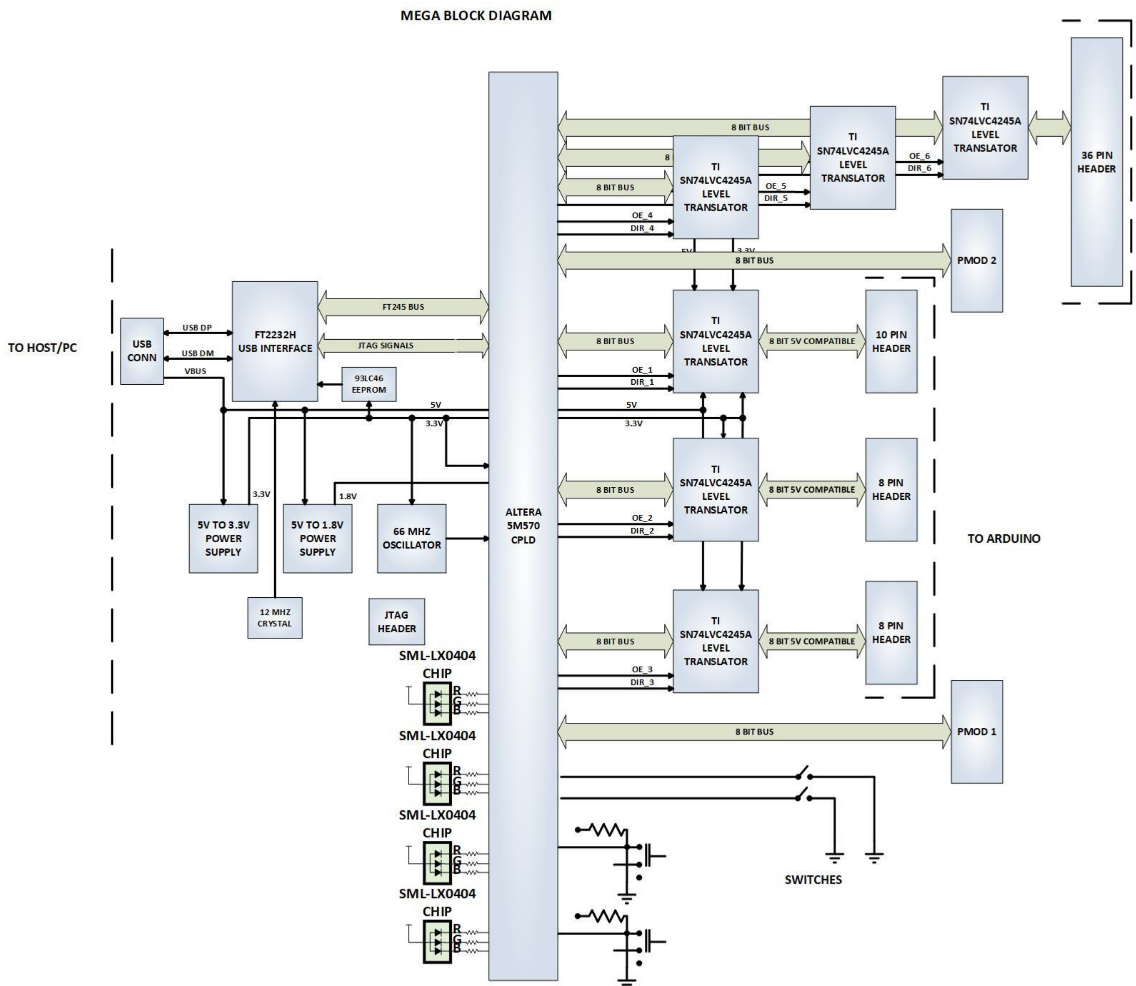


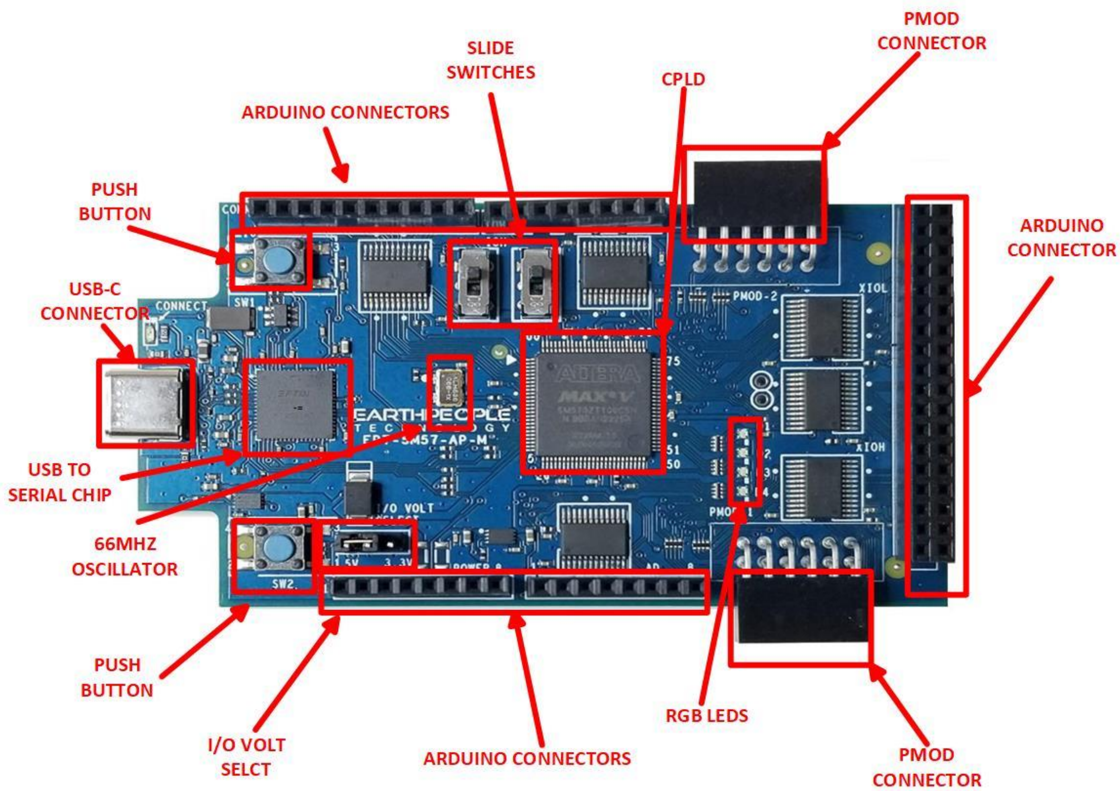
1.2 MegaMax

The MegaMax board is equipped with an Intel 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 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
- Four RGB LED's accessible by the user
- Two PCB switches accessible by the user
- Two Slide Switches
- Two PMOD Connectors

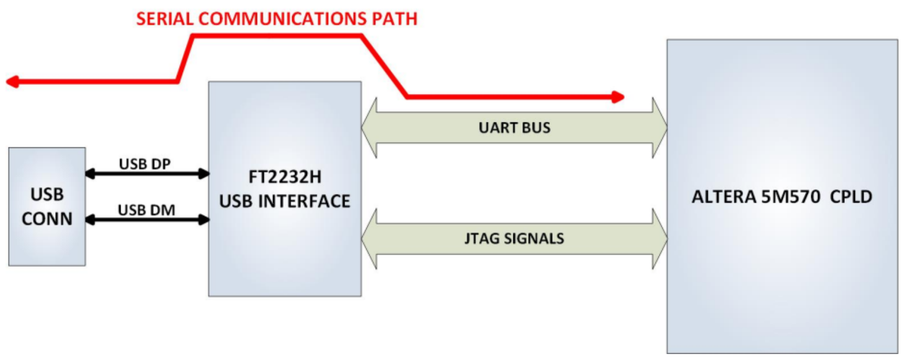




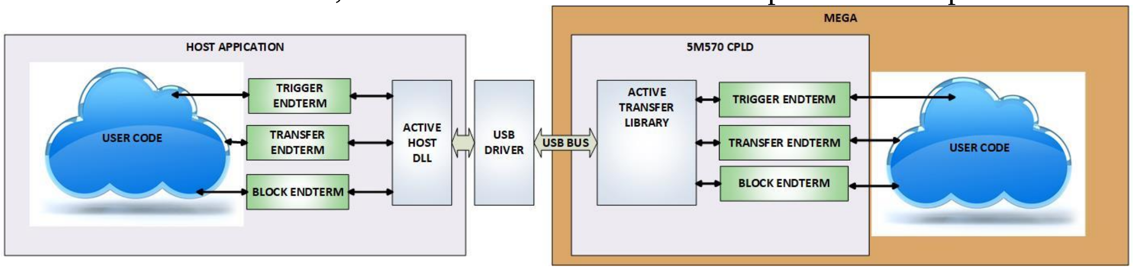
1.2.1 Serial USB Communications

The MegaMax 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 serial 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. The FT2232H chip provides its own 12 MHz clock and +3.3V and +1.8V power supplies. The +3.3V power supply output is used by the MegaMax for all of its +3.3V power budget.



The Serial Communications with the MegaMax 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:

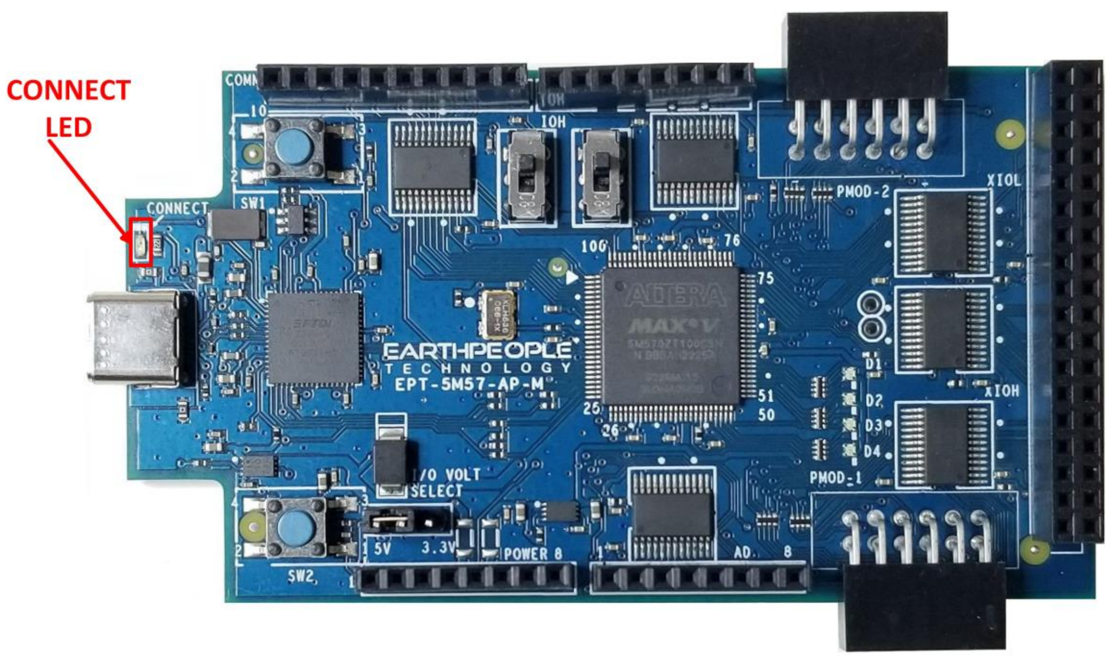
- Trigger Endterm
- Transfer Endterm
- Block Endterm

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

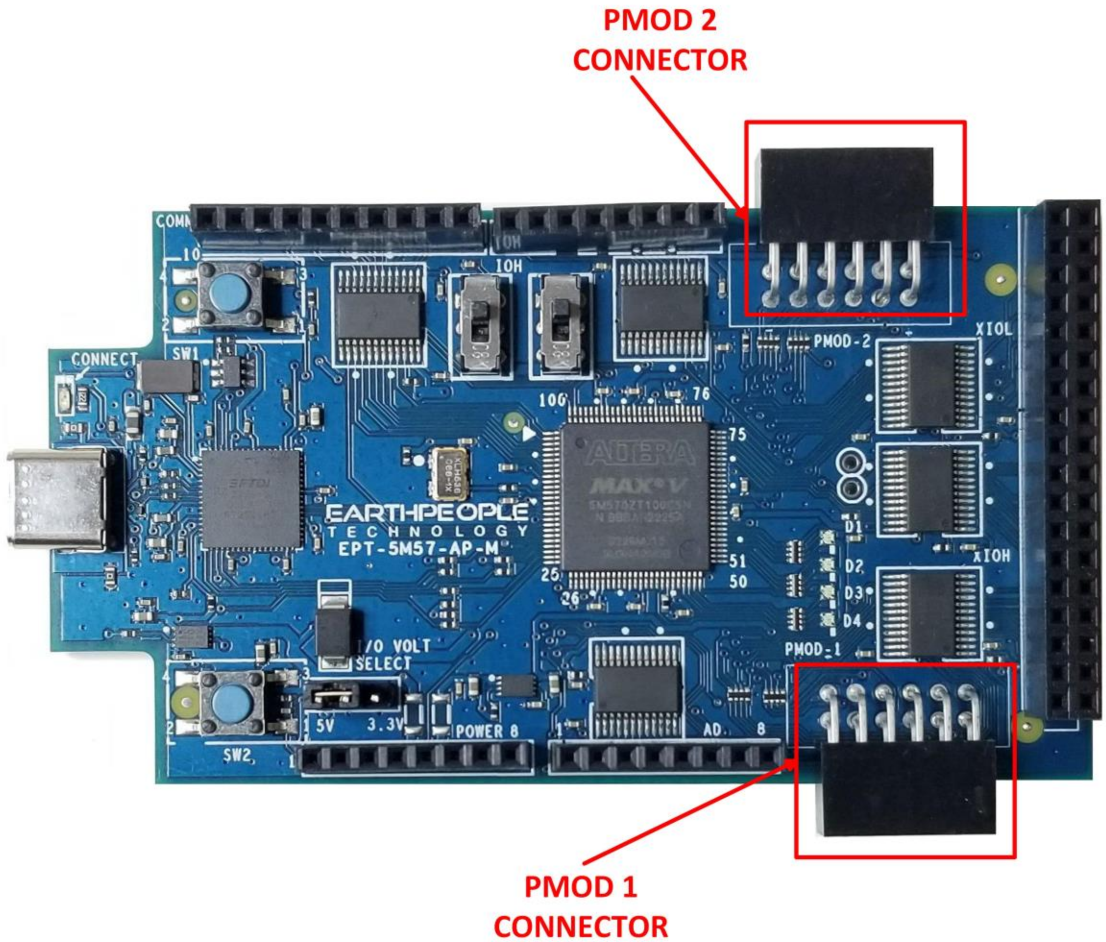
The MegaMax includes an LED that signifies the connection of the board with the Host PC. The connect LED has the word “CONNECT” in silkscreen next to the LED. This LED will only light up once the Host PC has correctly enumerated the USB device (FT2232HQ chip). When this LED is lit up it can tell the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

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

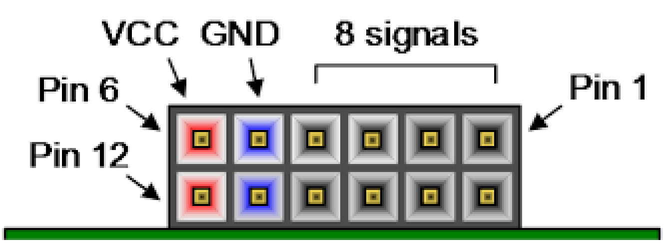


1.2.3 PMOD Connectors

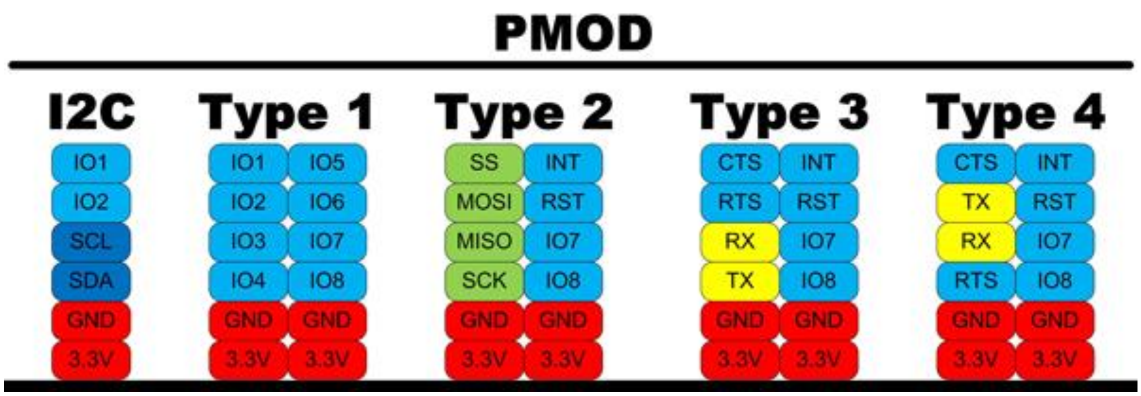
The MegaMax includes two PMOD Connectors. These two connectors are located towards the rear of the board.



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.

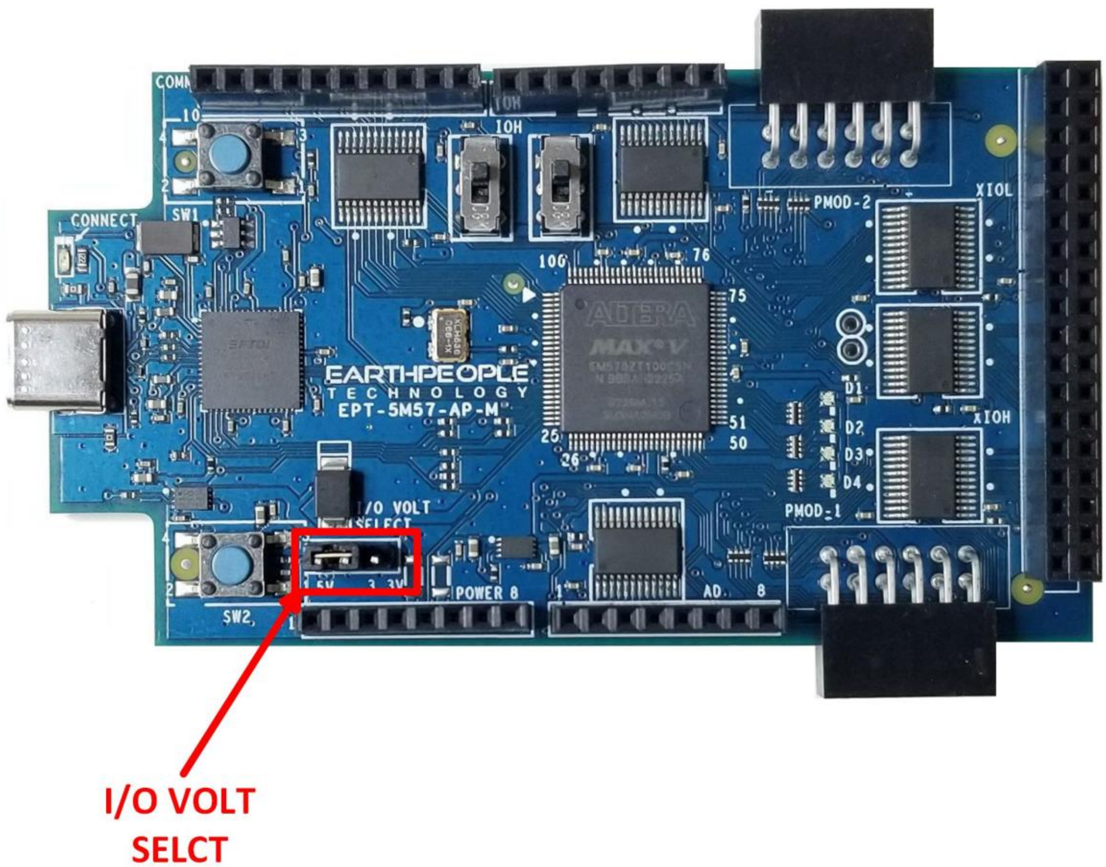


The PMOD have the following connections to the MAX V chip:

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

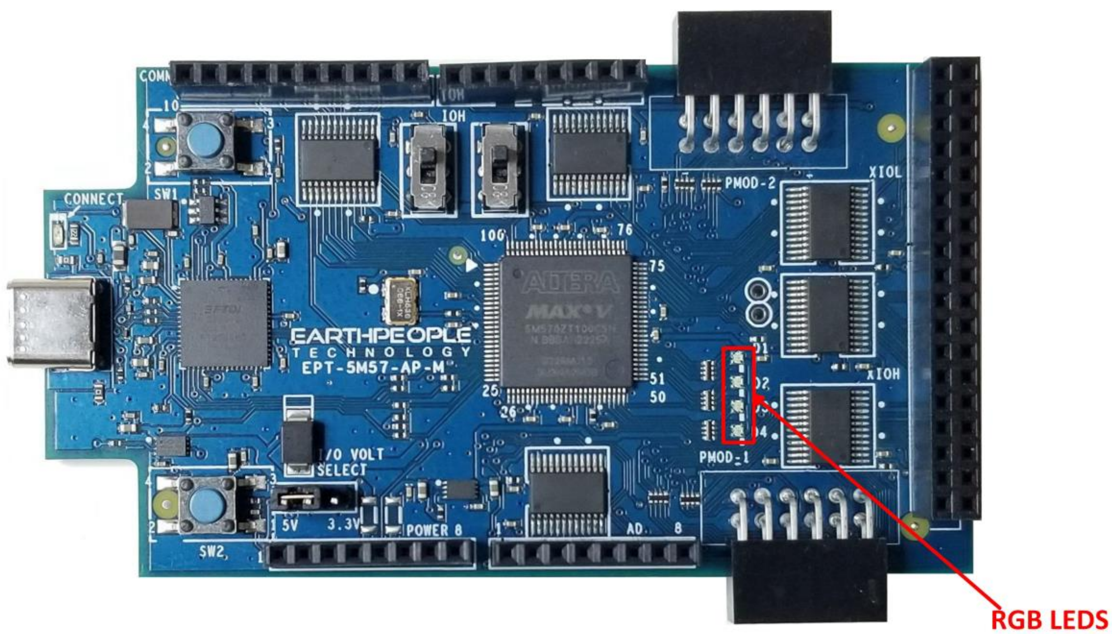
There are 32 Inputs/Outputs which are selectable between +3.3V and +5 Volt. JMP1 is used to select which voltage the 32 Inputs/Outputs are set to.



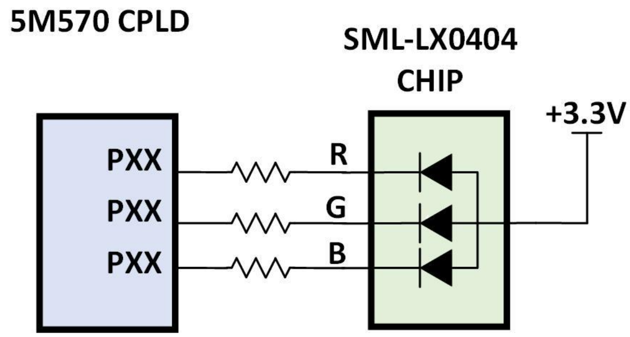
The I/O’s are organized as three 8 bit directional ports. Each port must be defined as input or output. This means that all 8 bits of a port will point in the same direction, depending on the direction bit of the transceiver. The direction bit can be changed at any time, so that a port can change from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



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.



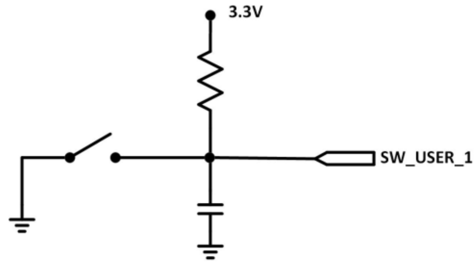
Each series resistor uses a 220 Ohm in a resistor array. In order to light up the each LED, the user code must assert a zero on the associated signal for the LED. To turn off the LED, assert High Z on the signal.

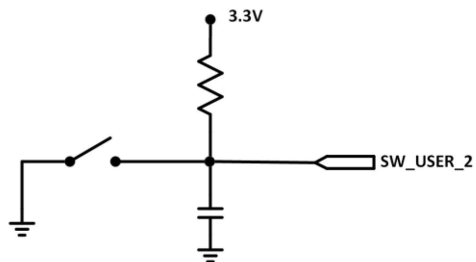
The LED RGB signals are organized on the following pins from the MAX V chip:

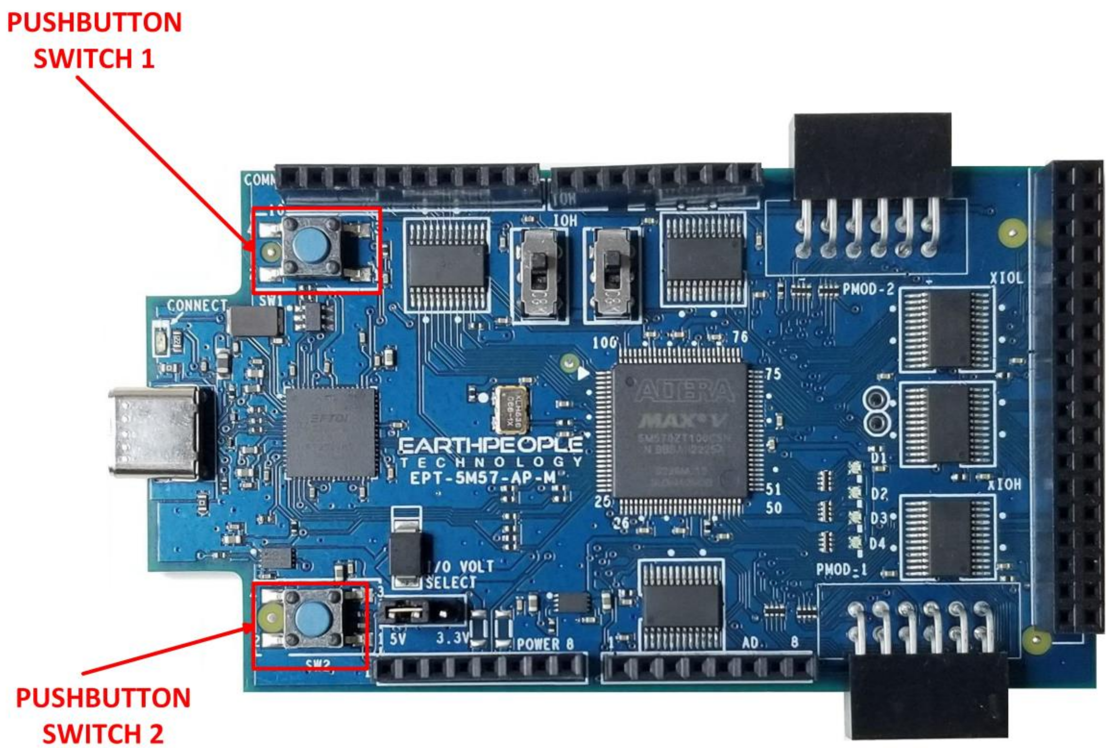
LED Number	Signal Name	MAX V Pin Number
D1	LED_GREEN_1_N	57
D1	LED _BLUE_1_N	56
D1	LED_RED_1_N	55
D2	LED_GREEN_2_N	54
D2	LED _BLUE_2_N	53
D2	LED_RED_2_N	52
D3	LED_GREEN_3_N	51
D3	LED _BLUE_3_N	50
D3	LED_RED_3_N	49
D4	LED_GREEN_4_N	48
D4	LED _BLUE_4_N	47
D4	LED_RED_4_N	43

1.2.6 User Push Buttons

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


3.3V
SW_USER_1

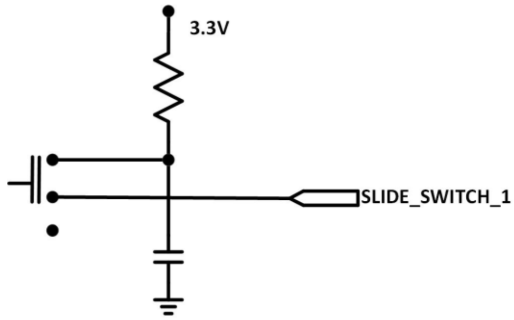

3.3V
SW_USER_2

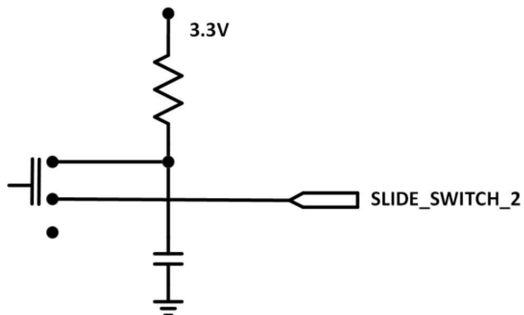


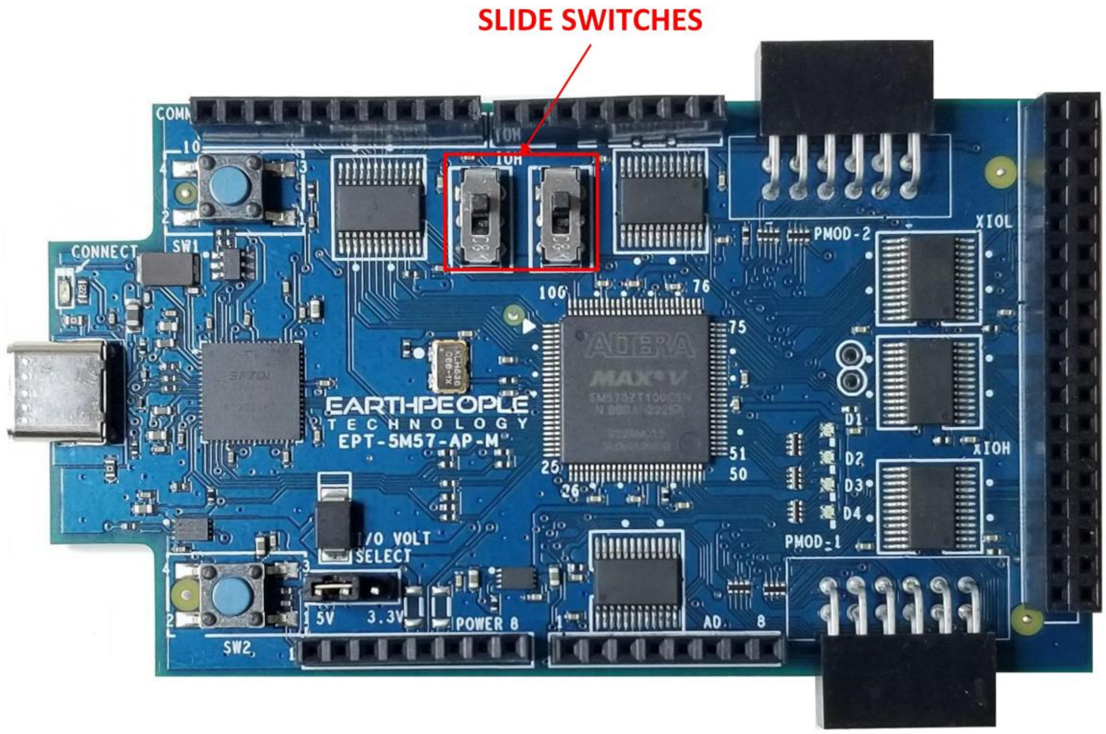
Component	Net Name	Pin on CPLD	Signal in EPT Project Pinout
SW1	SW_USER_1	20	SW_USER_1
SW2	SW_USER_2	21	SW_USER_2

1.2.7 Slide Switches

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


3.3V
SLIDE_SWITCH_1

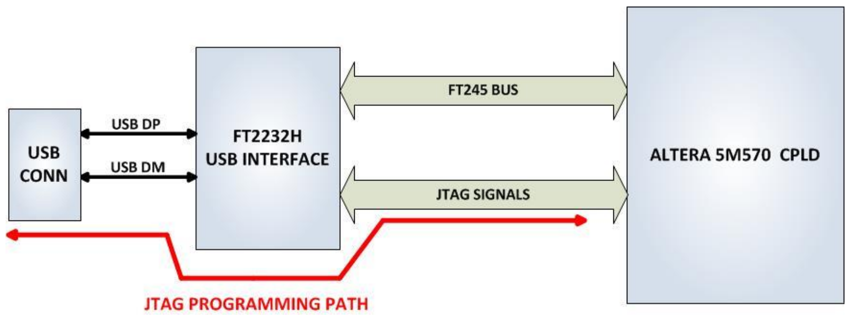

3.3V
SLIDE_SWITCH_2



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

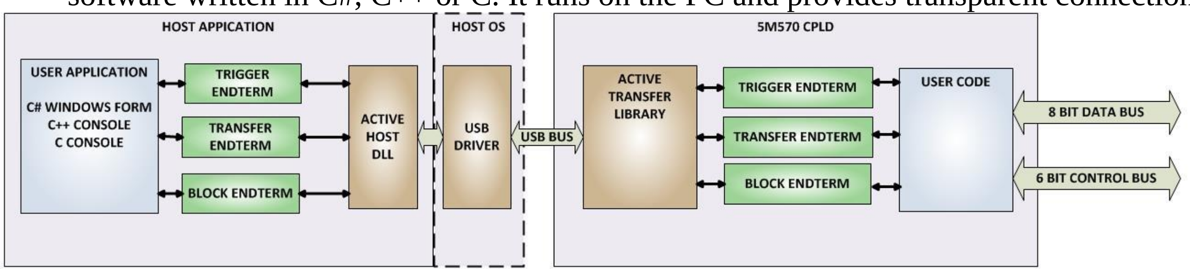
1.2.8 JTAG

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



1.3 *Active Host EndTerms*

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



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

- Trigger Endterm
- Transfer Endterm
- Block Endterm

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

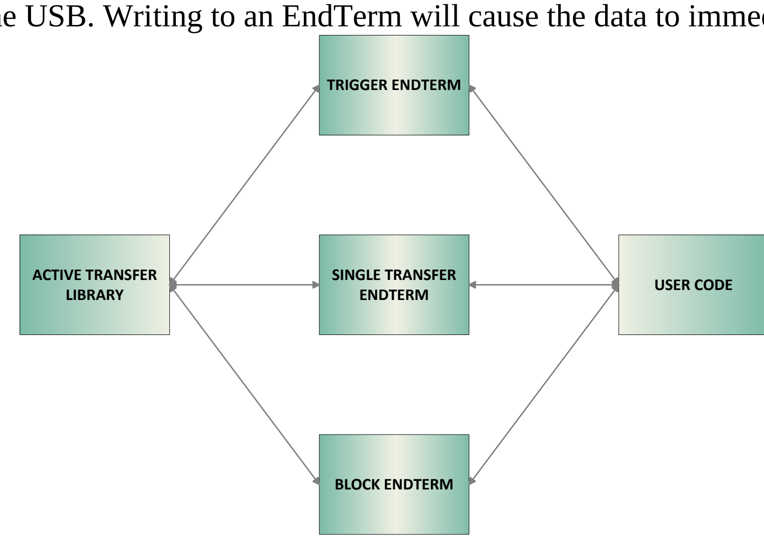
Receiving data from the CPLD is made simple by Active Host. Active Host transfers data from the CPLD as soon as it is available. It stores the transferred data into circular buffer. When the transfer is complete, Active Host invokes a callback function which is registered in the users application. This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

1.4 Active Transfer EndTerms

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

- Trigger Endterm
- Transfer Endterm
- 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.

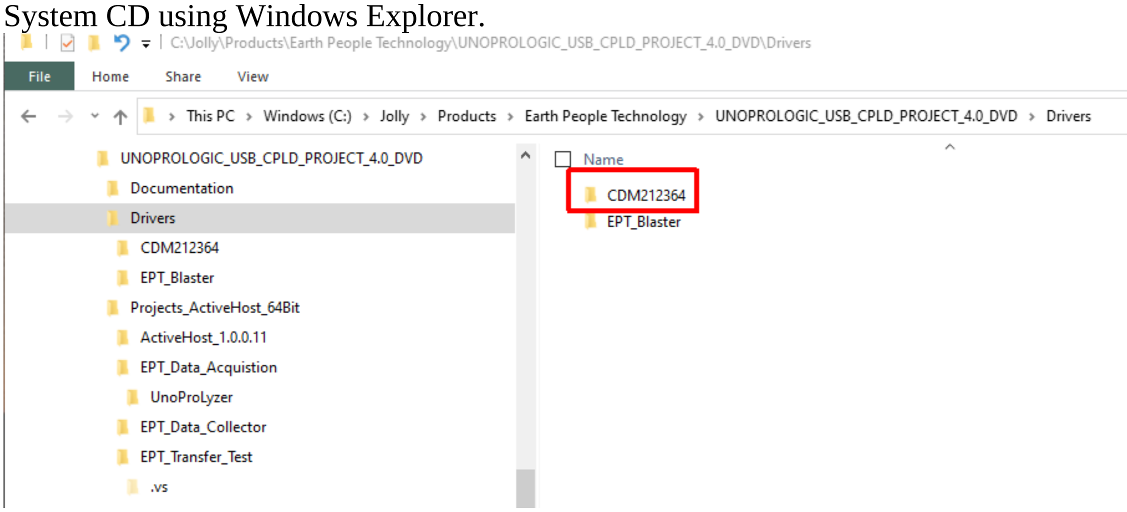
2 EPT Drivers

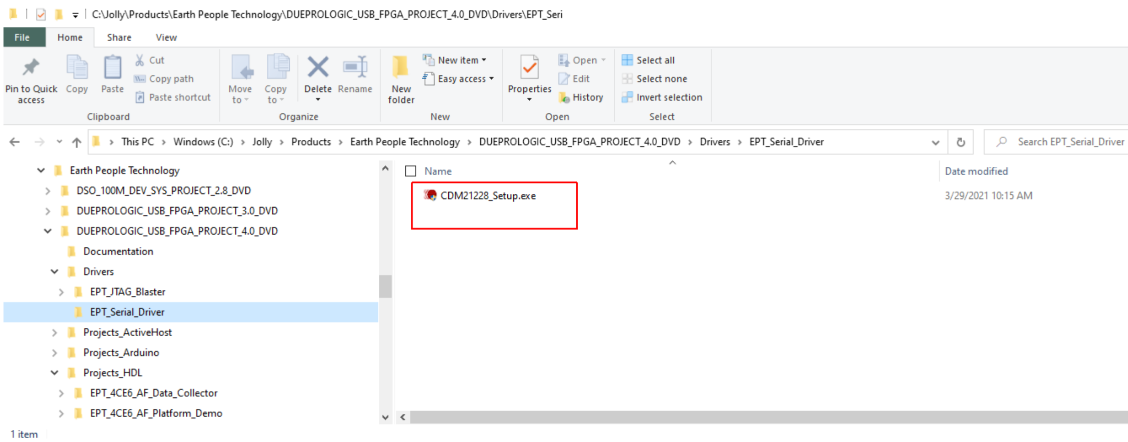
The MegaMax Development system requires drivers for any interaction between PC and the board. The communication between the two consists of programming the CPLD and data transfer. In both cases, the USB Driver is required. This will allow Windows to recognize the USB Chip and setup a pathway for Windows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

2.1 USB Driver

The MegaMax 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 MegaMax Development System CD using Windows Explorer.

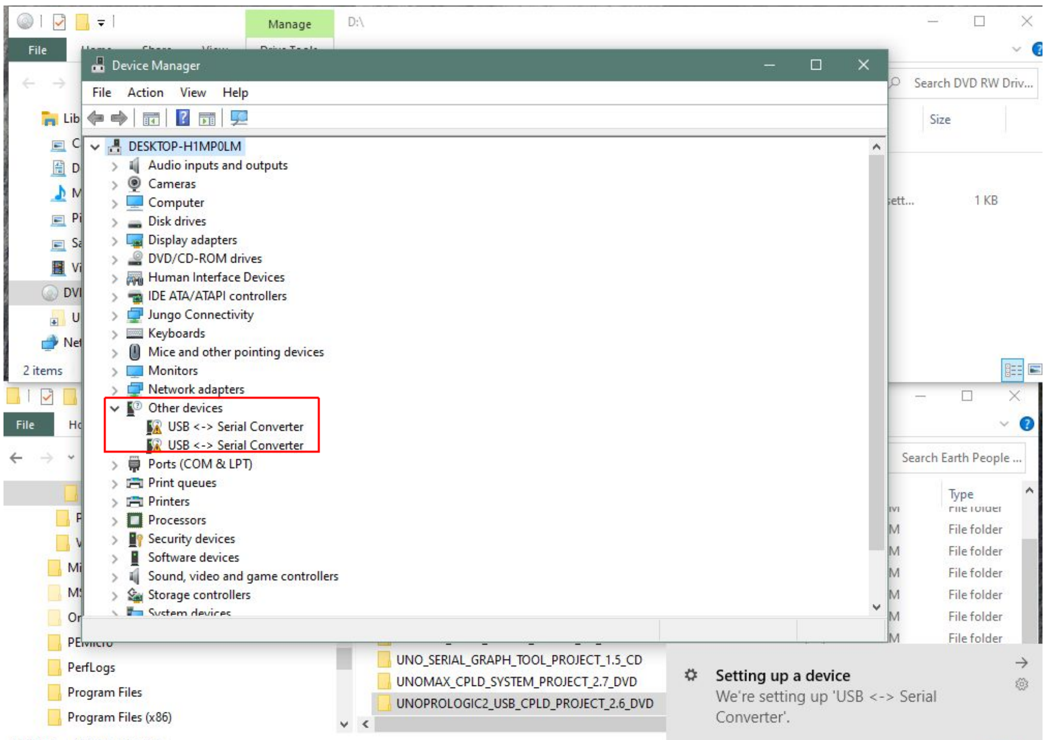




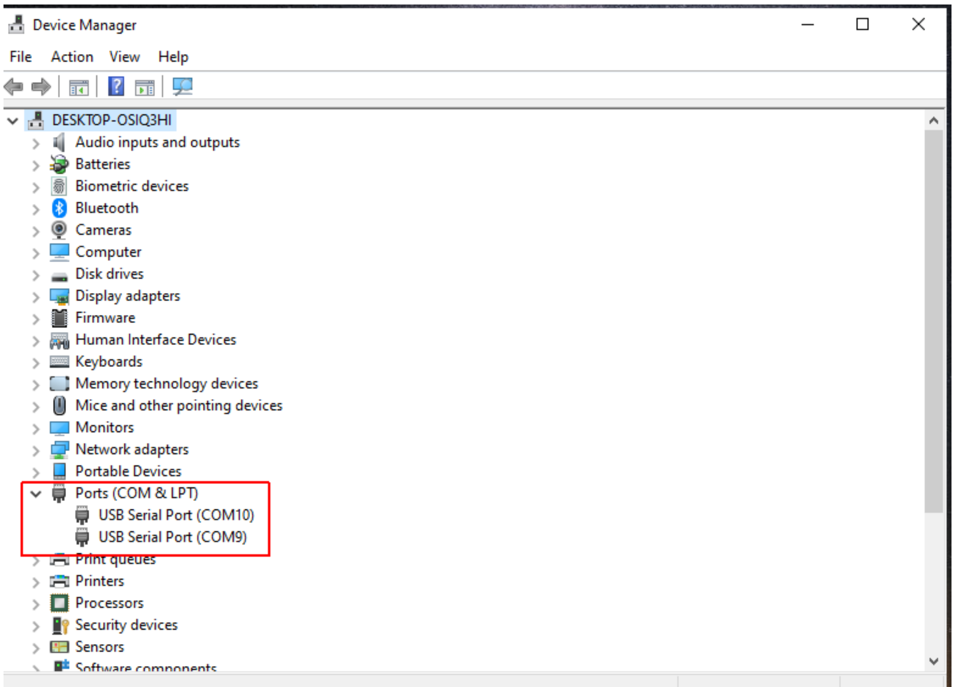
Double click on the *.exe file and select the default settings when the software tool queries the user.

Plug in the MegaMax 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 MegaMax, a notification window will inform the user that the driver load has failed for the device.



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 MegaMax. The MegaMax will show up as two COM Ports under the “Ports (COM &LPT)” under the Device Manager.



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

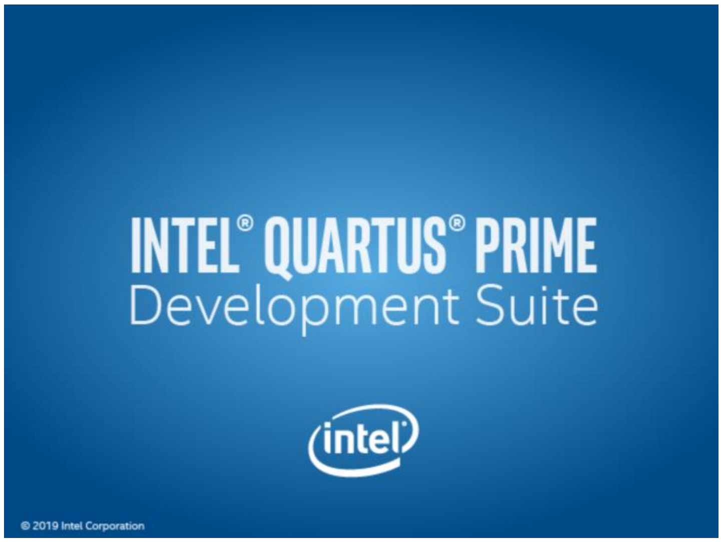
2.2 JTAG DLL Insert to Quartus Prime Lite

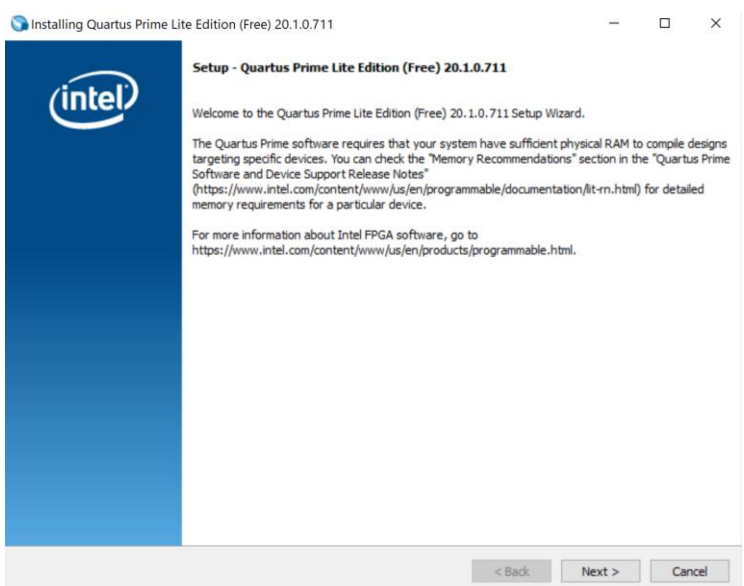
The JTAG DLL Insert to Quartus Prime Lite allows the Programmer Tool under Quartus to recognize the MegaMax. The MegaMax can then be selected and perform programming of the CPLD. The file, jtag_hw_mbftdi_blaster.dll must be placed into the folder that hosts the jtag_server for Quartus.

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.





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

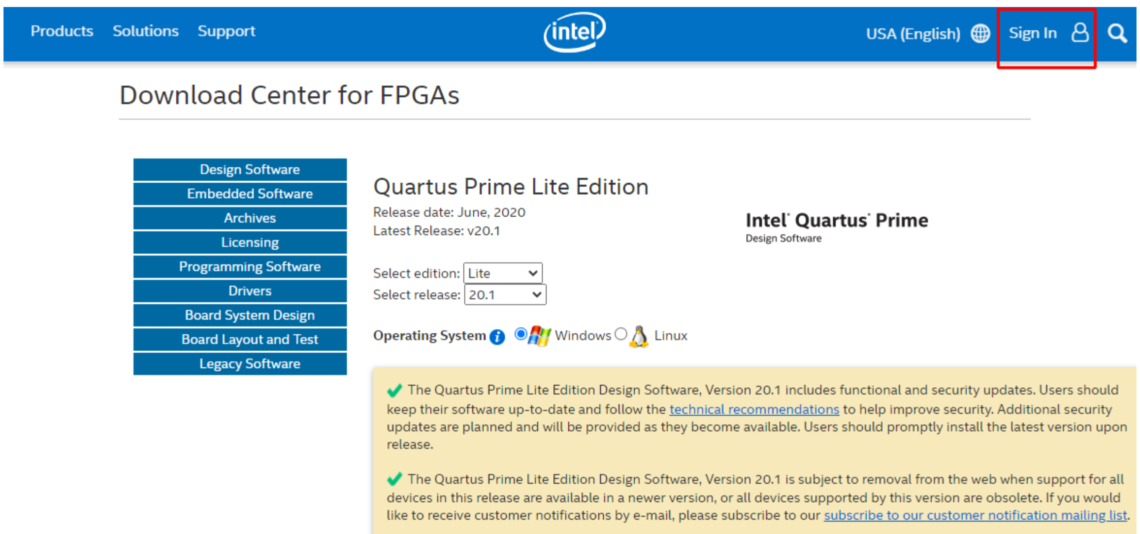
Next, skip the “Download Quartus” section. Go down to the “Quartus Installer” section to complete the Quartus installation.

2.2.2 Downloading Quartus

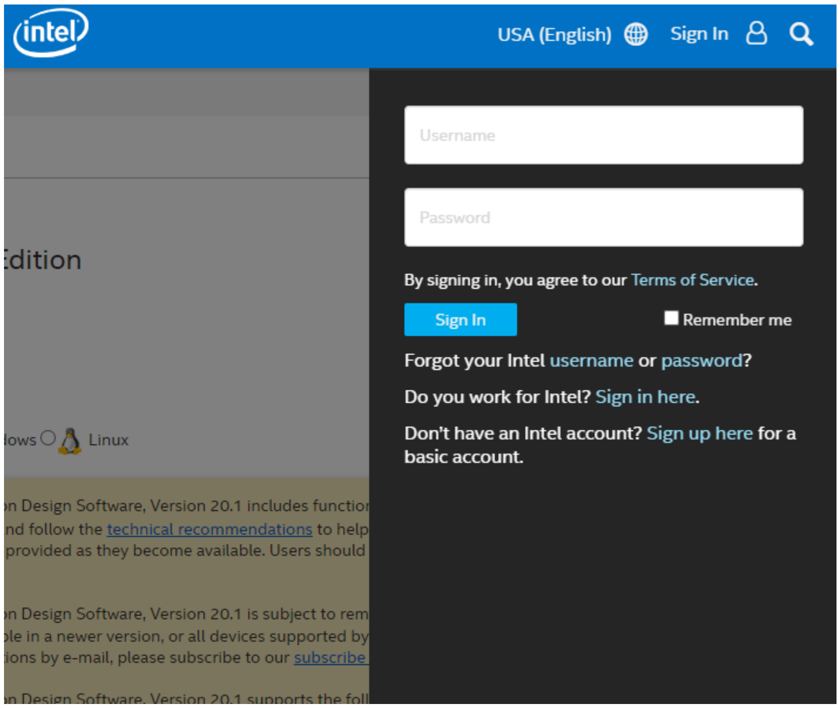
The first thing to do in order build a project in Quartus is to download and install the application. You can find the latest version of Quartus at:

Intel FPGA Quartus Prime Lite

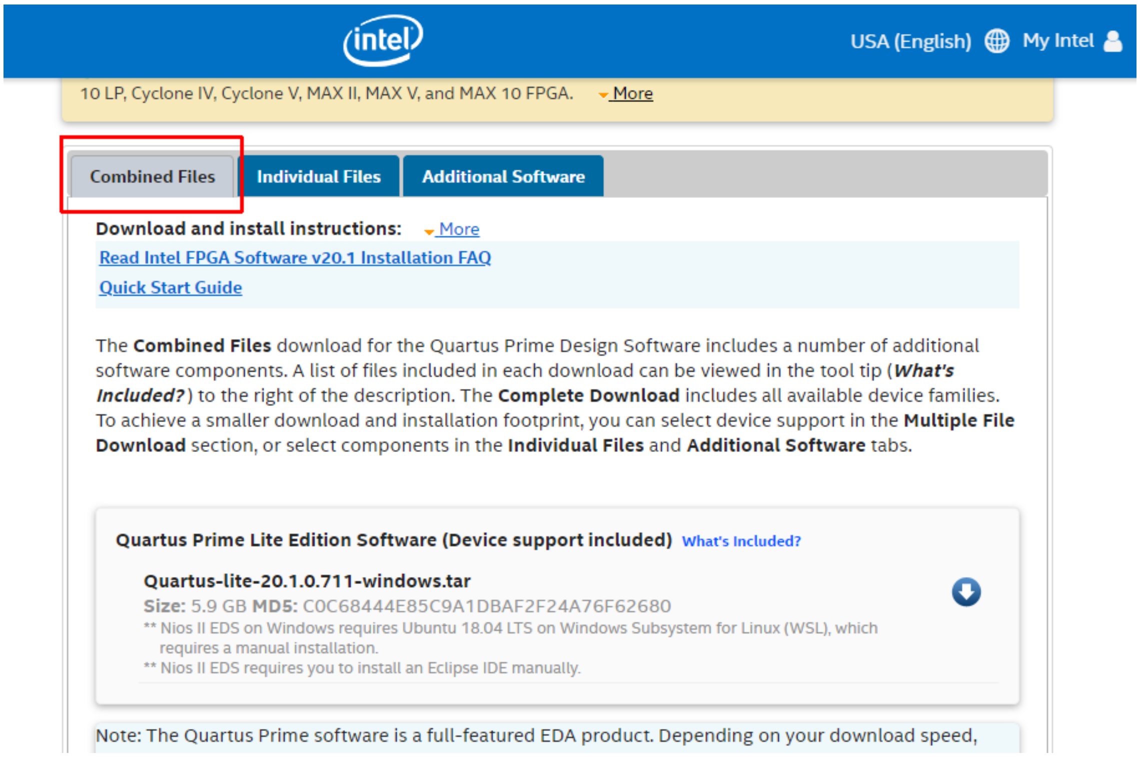
You will first need to apply for an account with Intel. Then use your login and password to access the download site. Click on the Download Windows Version.



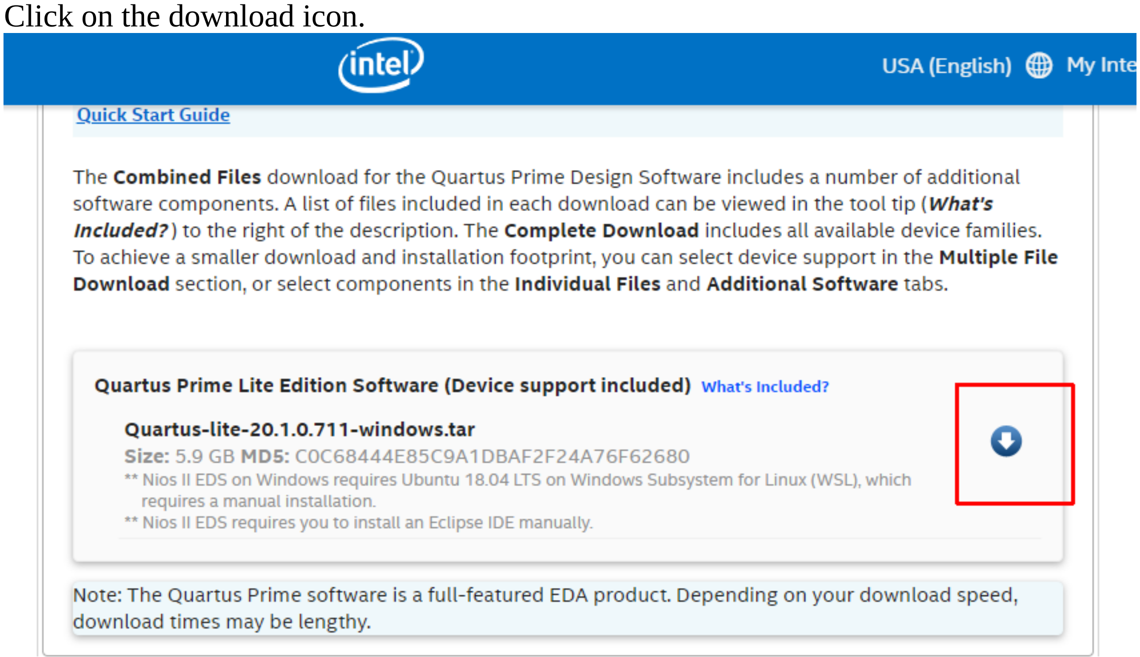
The next page will require you to sign into your "myAltera" account. If you do not have one, follow the directions under the box, "Don't have an account?"



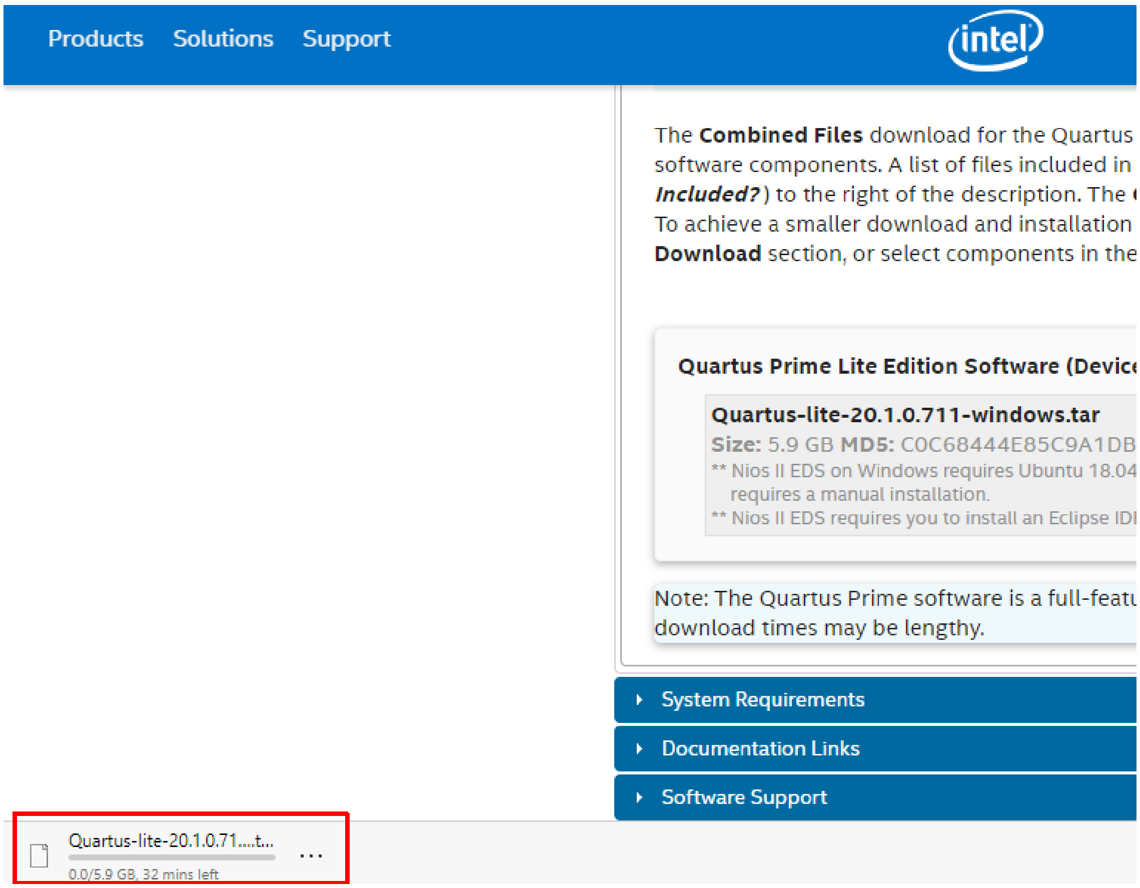
Once you have created your myAltera account, enter the User Name and Password. The next window will ask you to allow pop ups so that the file download can proceed.



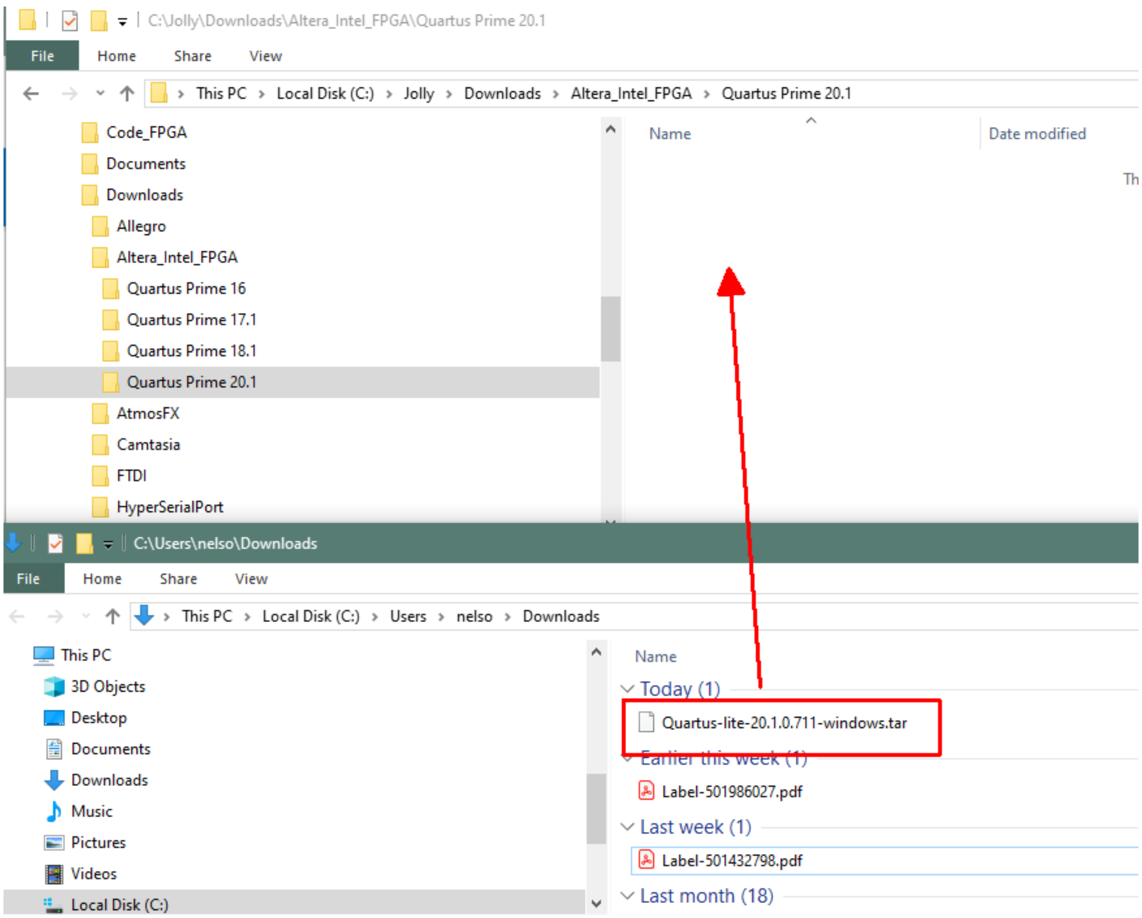
Click on the download icon.



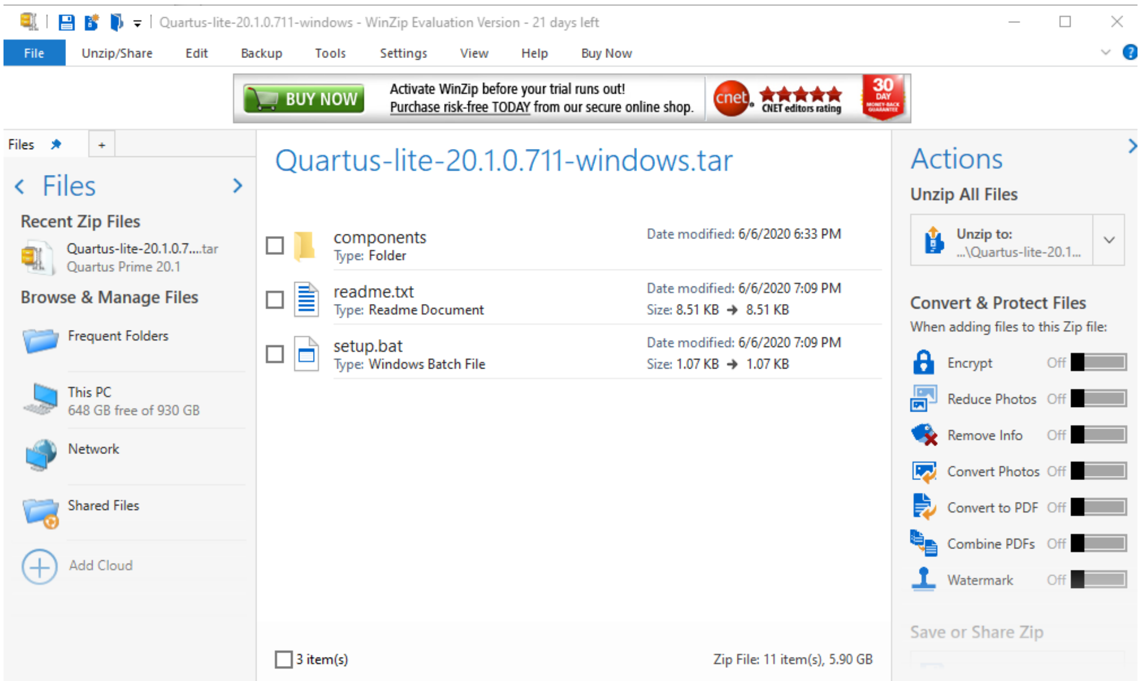
This will start the download.



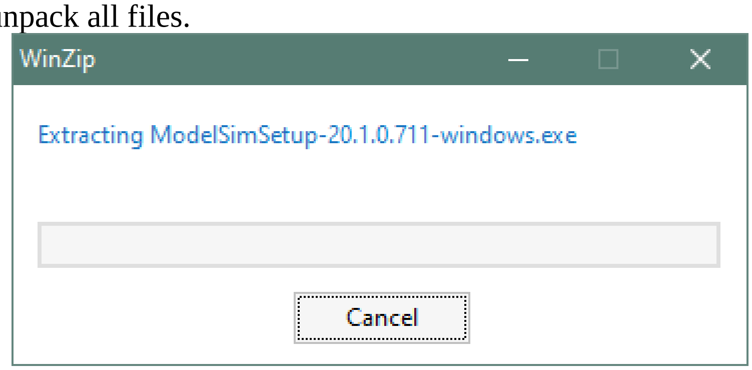
The file is 5.9 GB, so this could take a couple of hours depending on your internet connection. When download is complete, store the *.tar file in a directory on your PC.



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

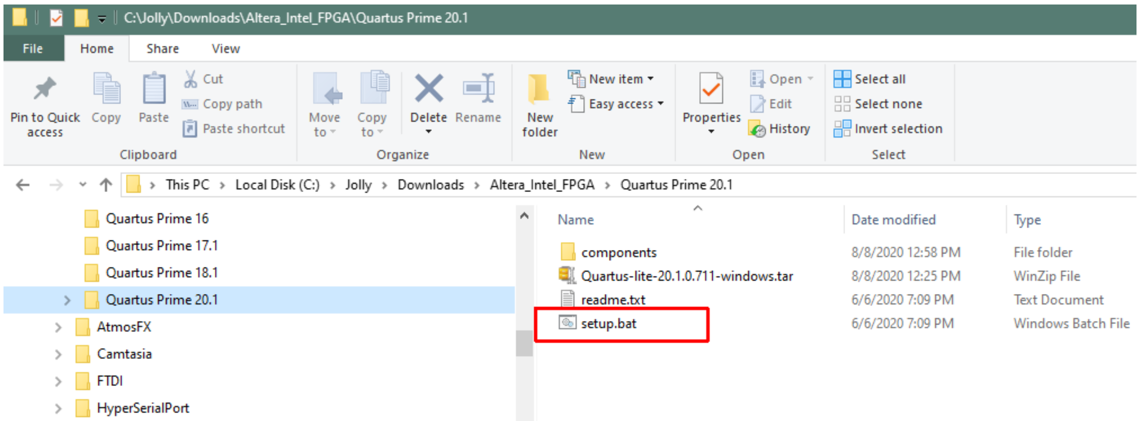


The tool will unpack all files.

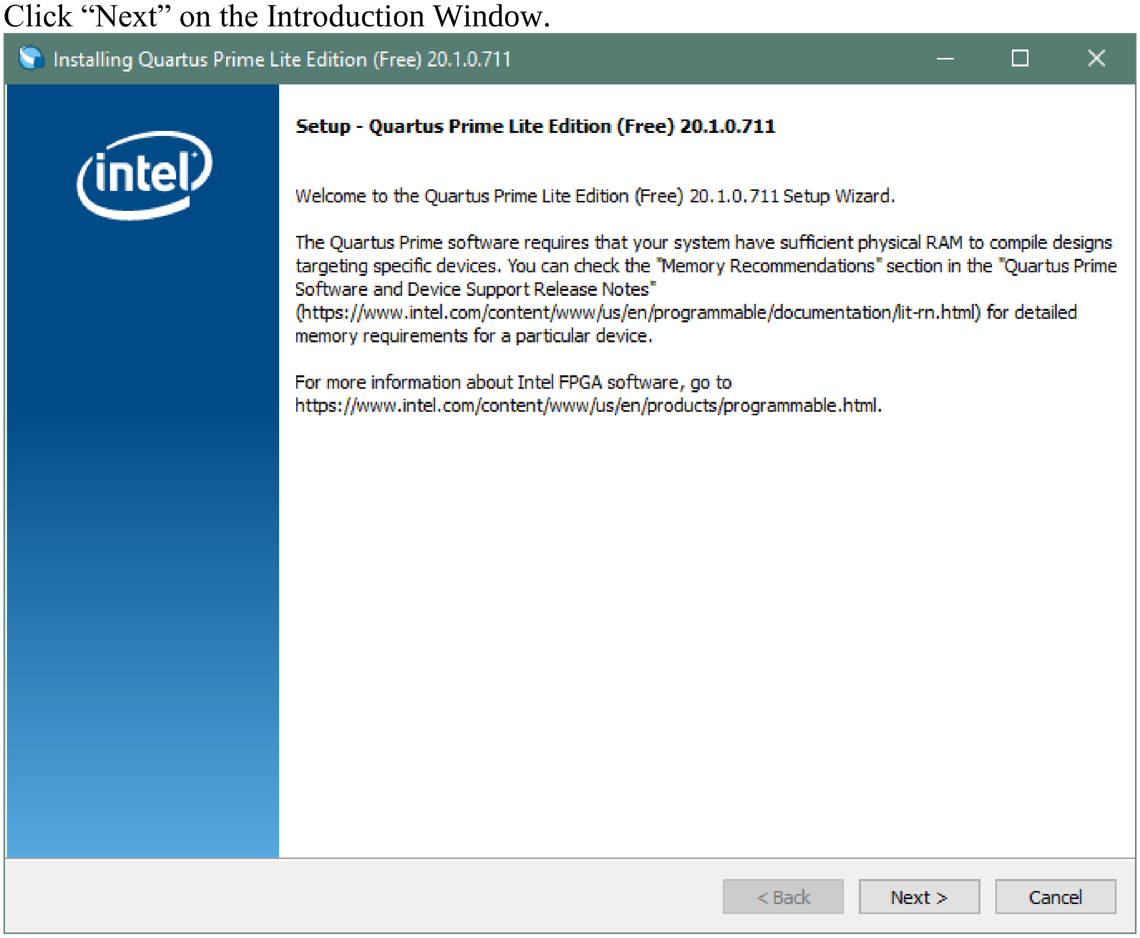


2.2.3 Quartus Installer

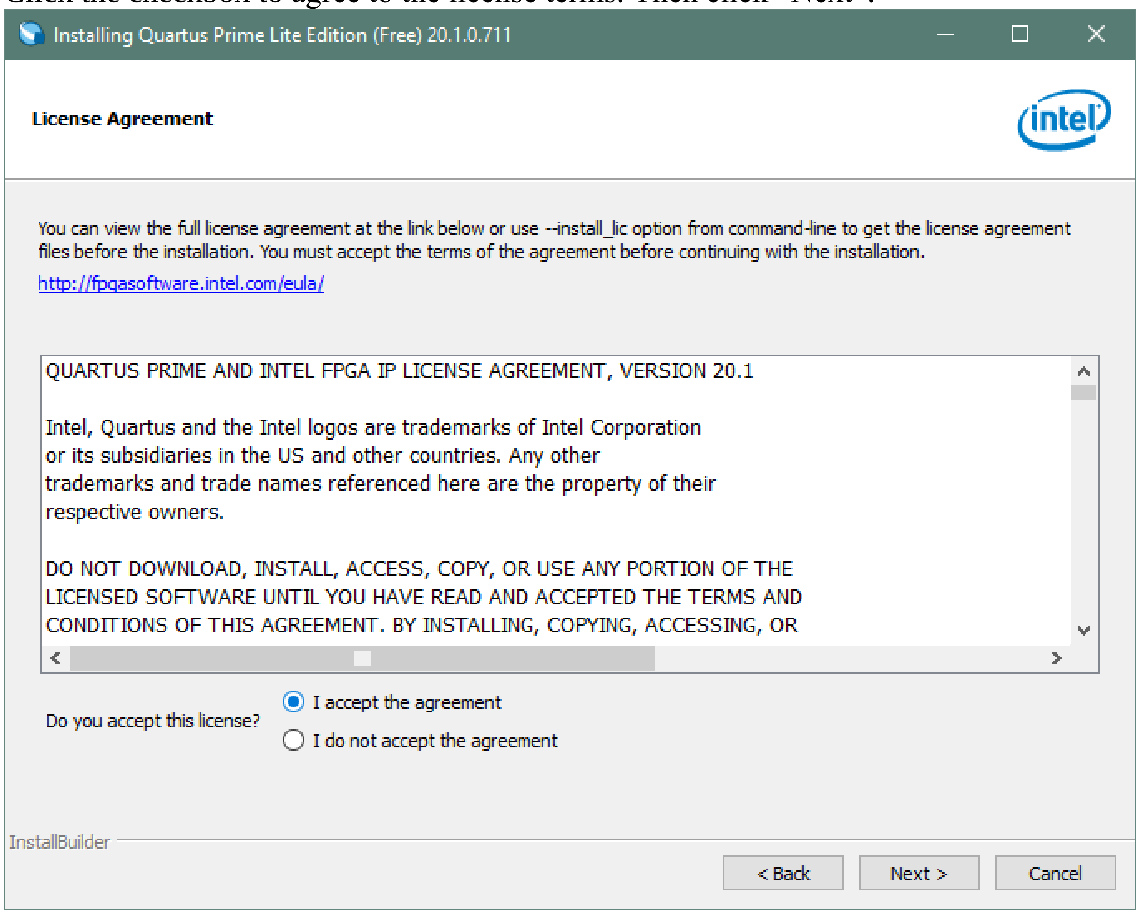
When the unpacking finishes from the previous section, double click the setup.bat file in the download folder.



Click “Next” on the Introduction Window.

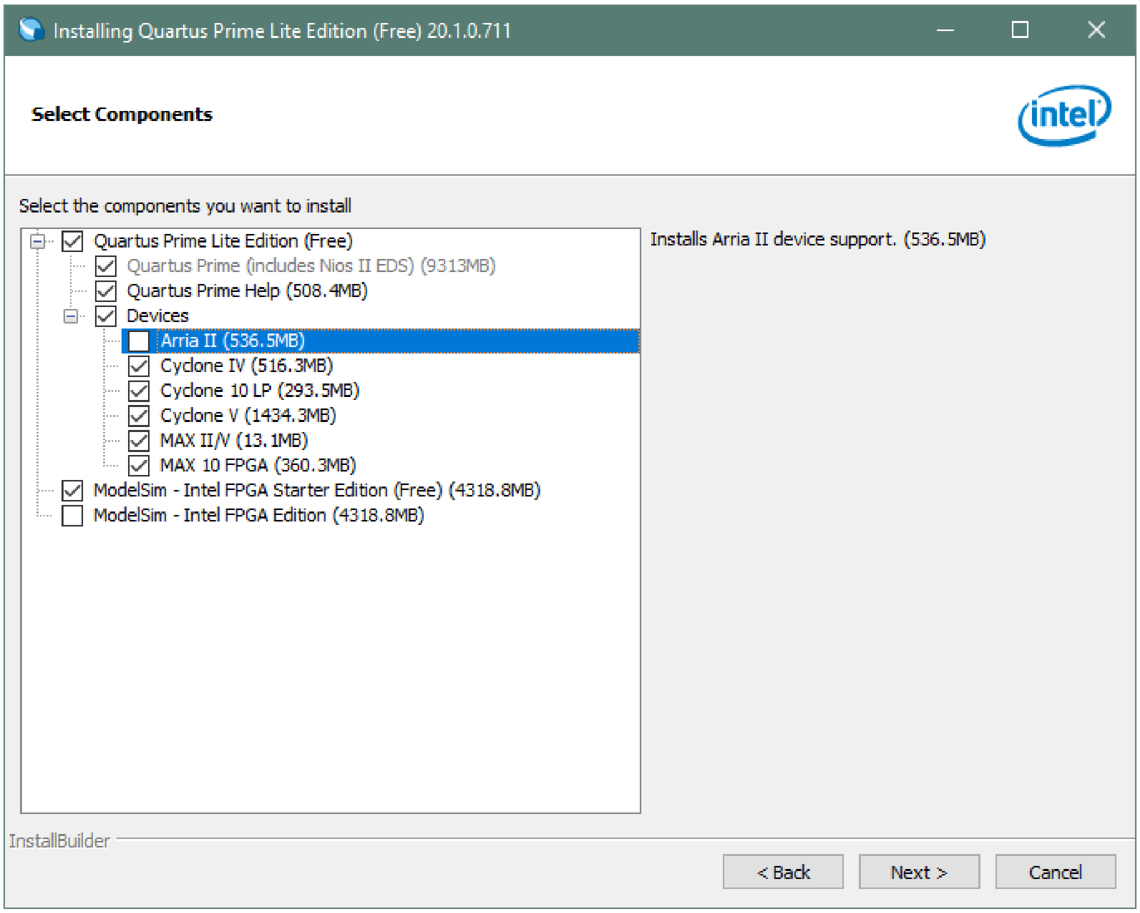


Click the checkbox to agree to the license terms. Then click "Next".

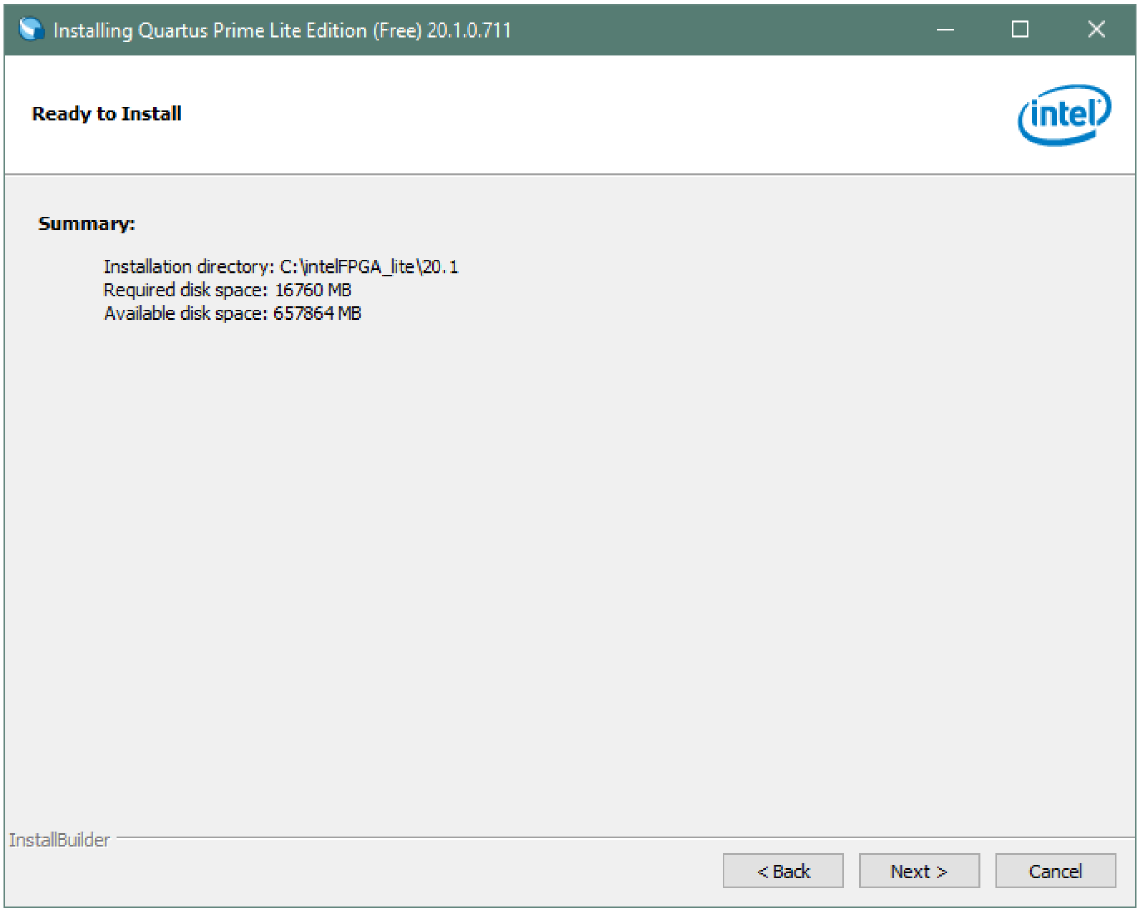


Click "Next" and accept the defaults.

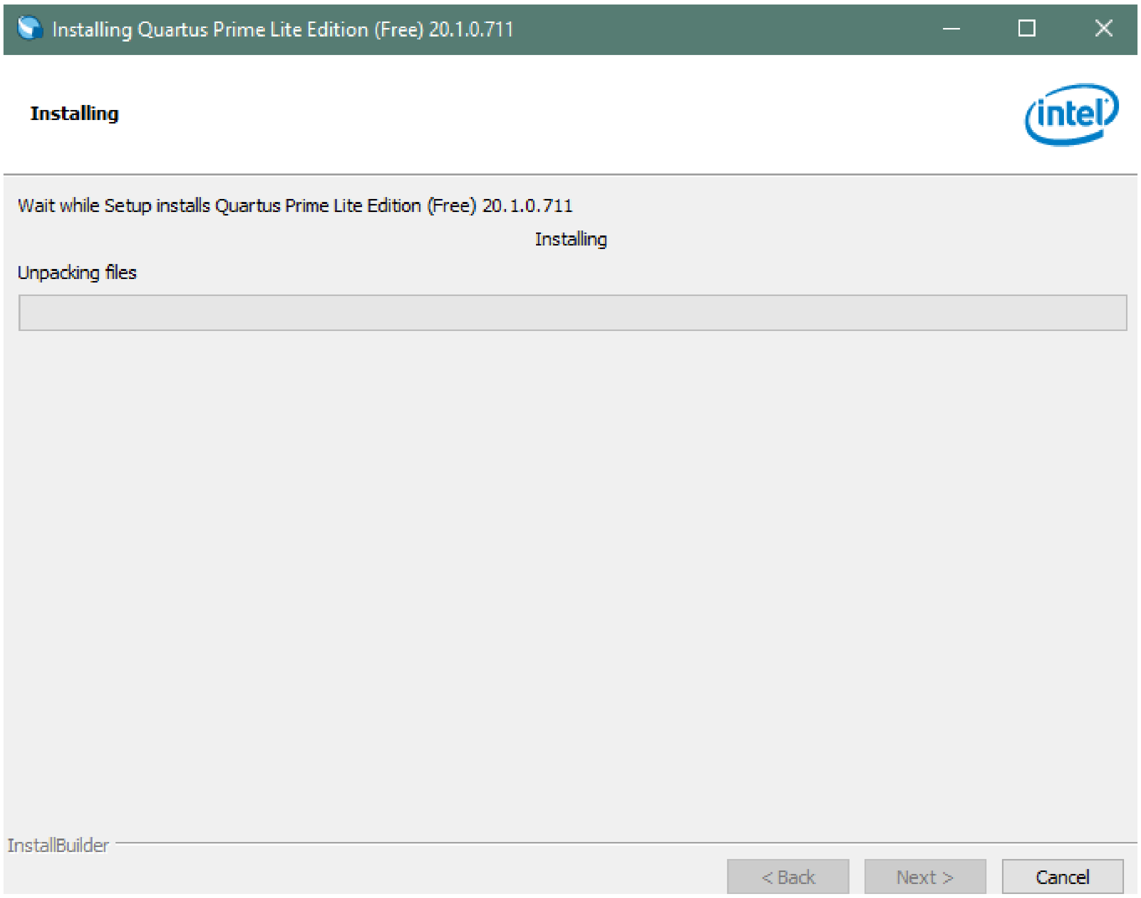
At the Select Products Window, de-select the Quartus Prime Supbscription Edition by clicking on its check box so that the box is not checked. Then click on the check box by the Quartus Prime Lite Edition (Free).



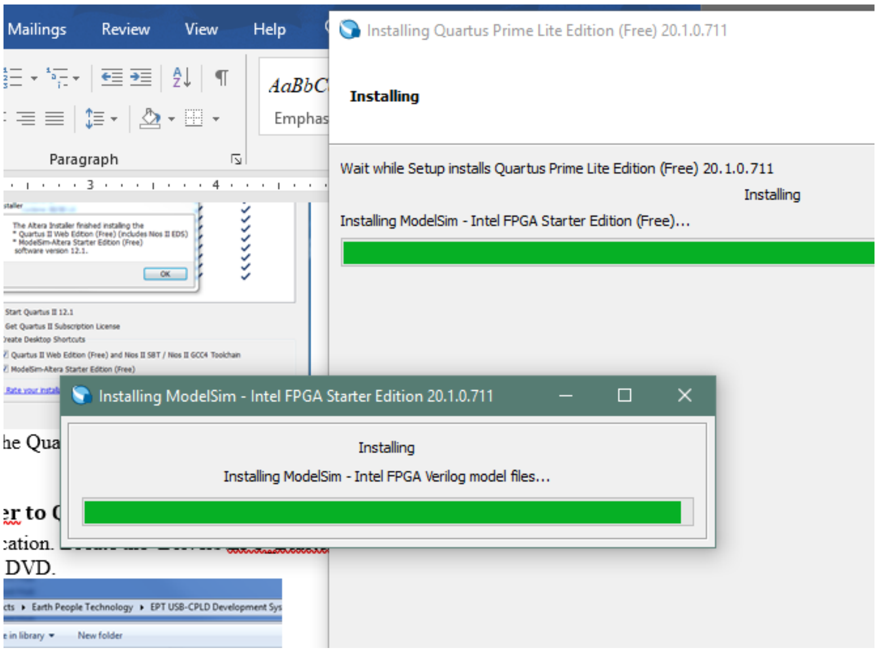
Click “Next” to accept the defaults

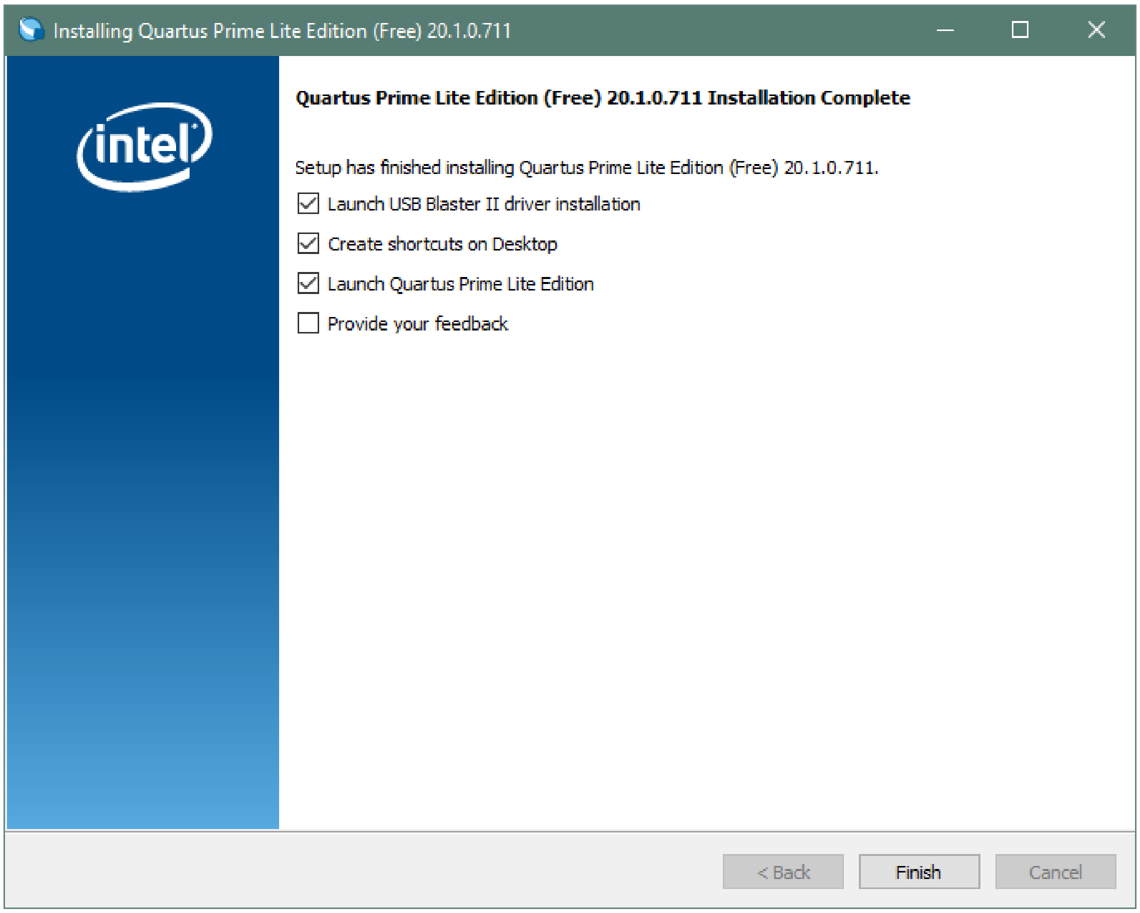


Click "Next" to accept the defaults

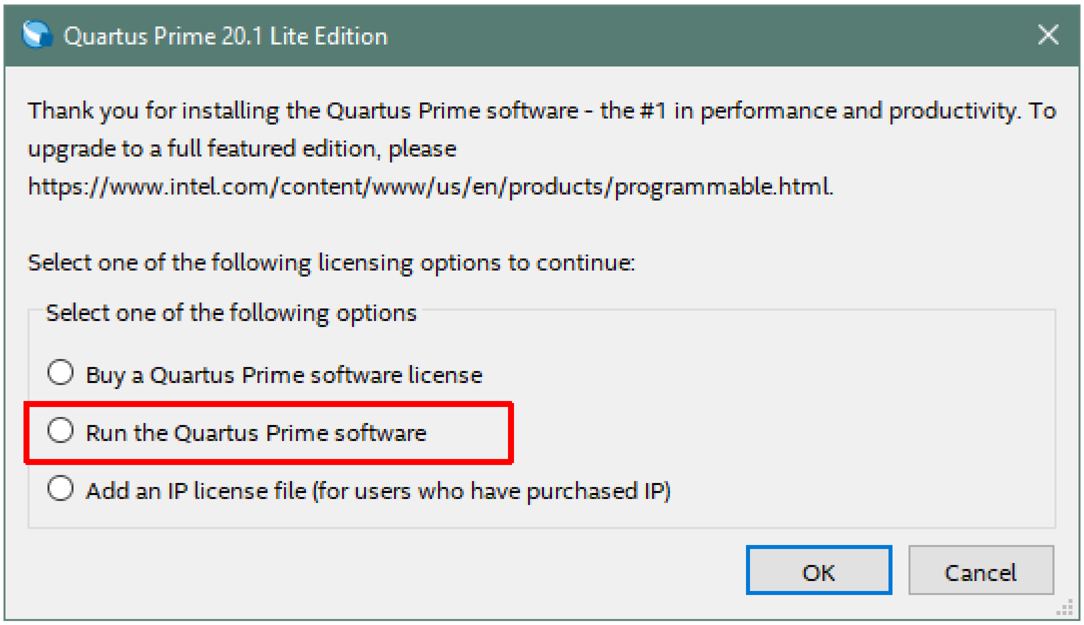


Wait for the installation to complete.


Installing Quartus Prime Lite Edition (Free) 20.1.0.711
Installing
Wait while Setup installs Quartus Prime Lite Edition (Free) 20.1.0.711
Installing
Installing ModelSim - Intel FPGA Starter Edition (Free)...
Installing ModelSim - Intel FPGA Starter Edition 20.1.0.711
Installing
Installing ModelSim - Intel FPGA Verilog model files...
Mailings
Review
View
Help
Paragraph

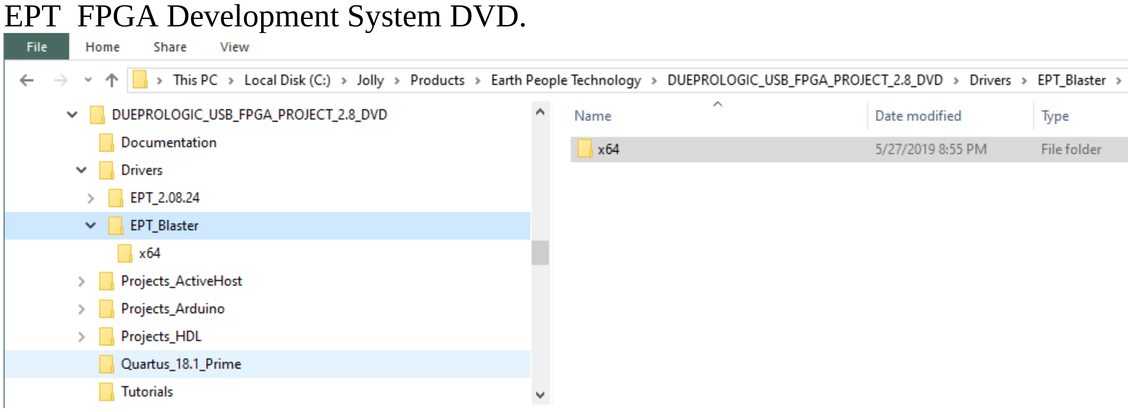


Click “Ok”, then click “Finish”. The Quartus Prime is now installed and ready to be used.



2.2.4 Adding the EPT_Blaster to Quartus Prime

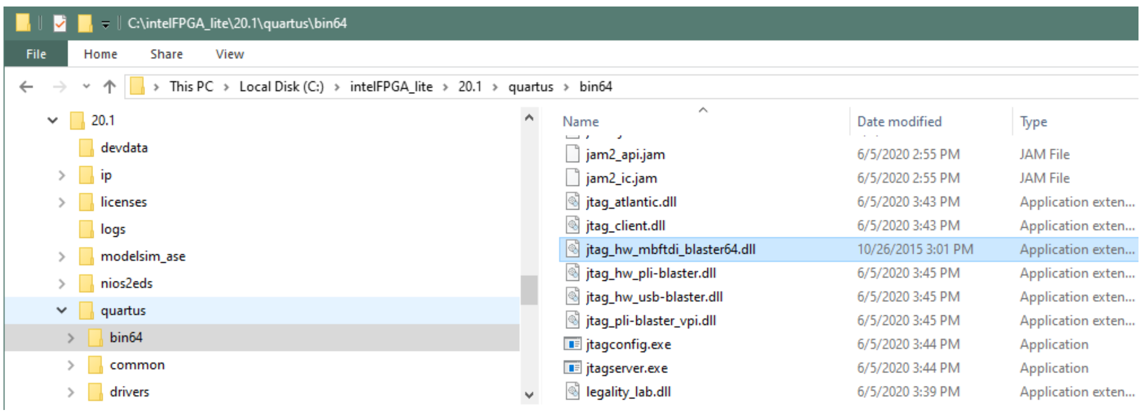
Close out the Quartus Prime application. Locate the \Drivers\EPT_Blaster folder on the EPT FPGA Development System DVD.



Follow these directions:

1. Open the C:\..\MegaMax_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.

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



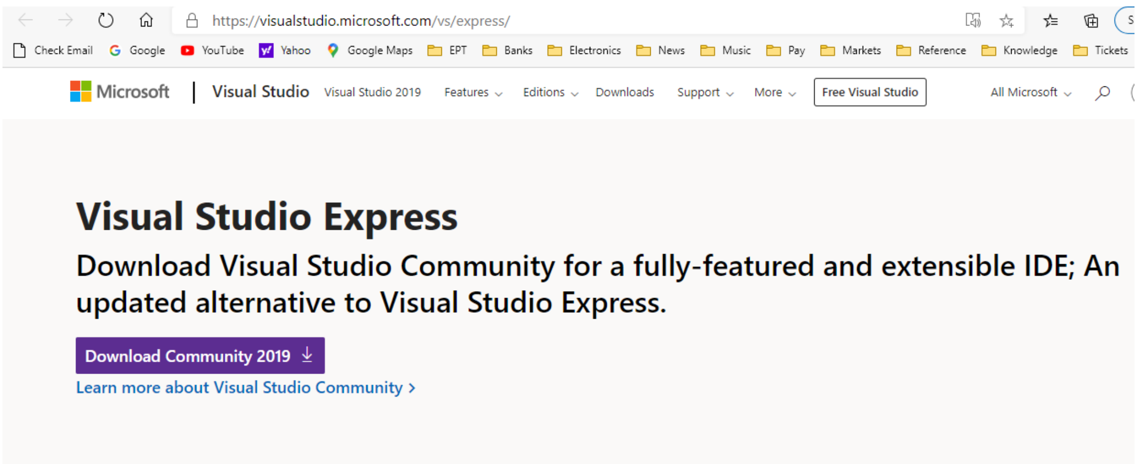
The DLL is installed and the JTAG server should recognize it. Go to the section "Programming the FPGA" of this manual for testing of the programming. If the driver is not found in the Programmer Tool->Hardware Setup box, see the JTAG DLL Insert to Quartus Prime Troubleshooting Guide.

2.3 Active Host Application SDK

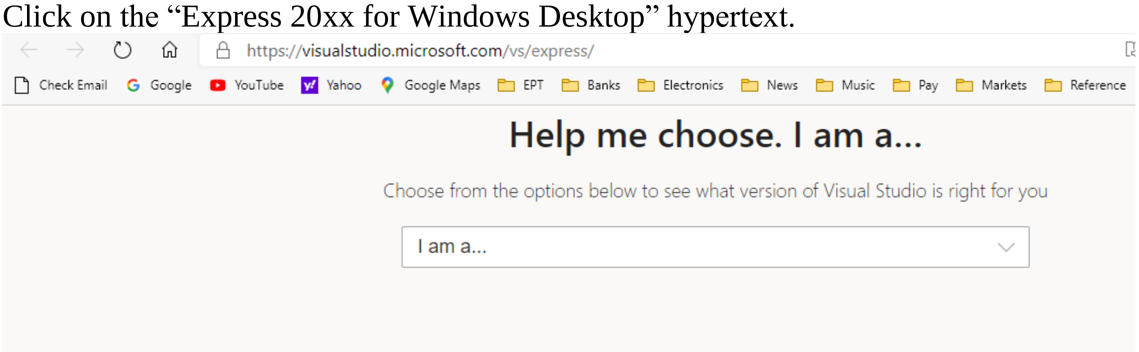
Download the latest version of Microsoft Visual C# Express environment from Microsoft. It's a free download.

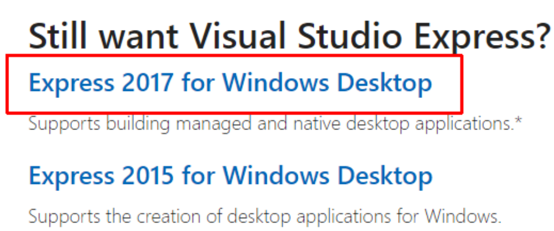
https://visualstudio.microsoft.com/vs/express/

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

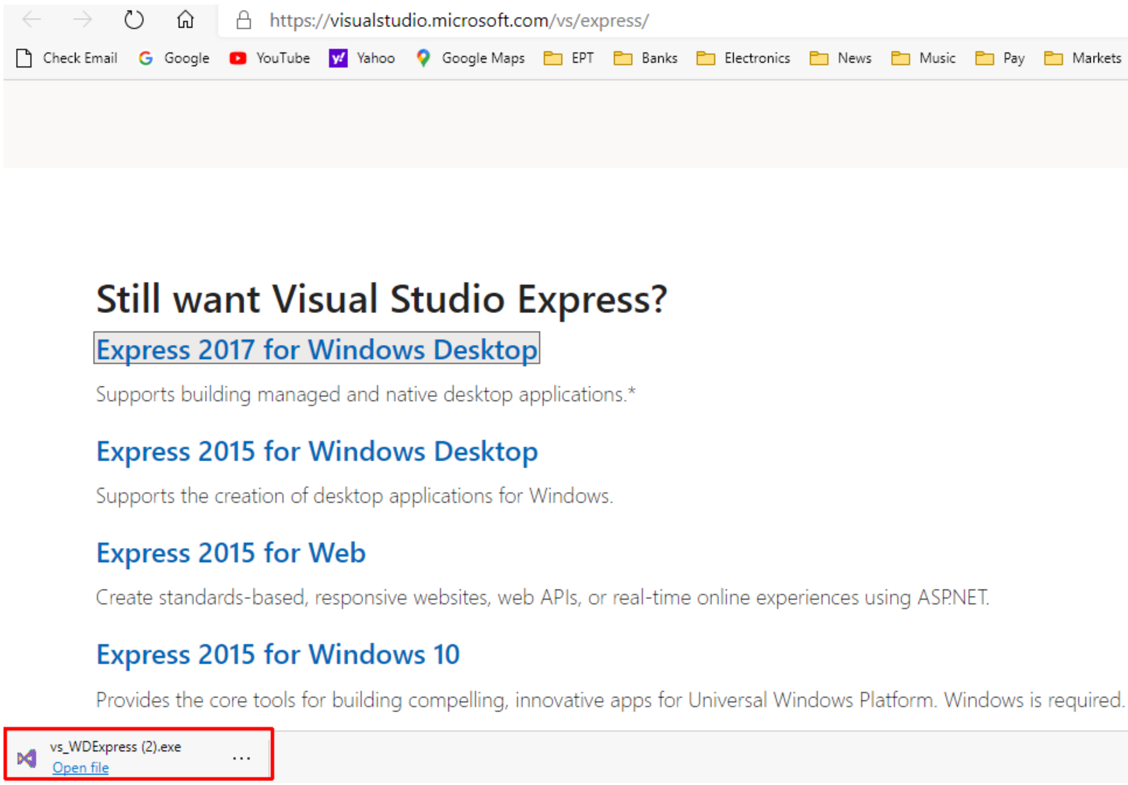


Click on the “Express 20xx for Windows Desktop” hypertext.

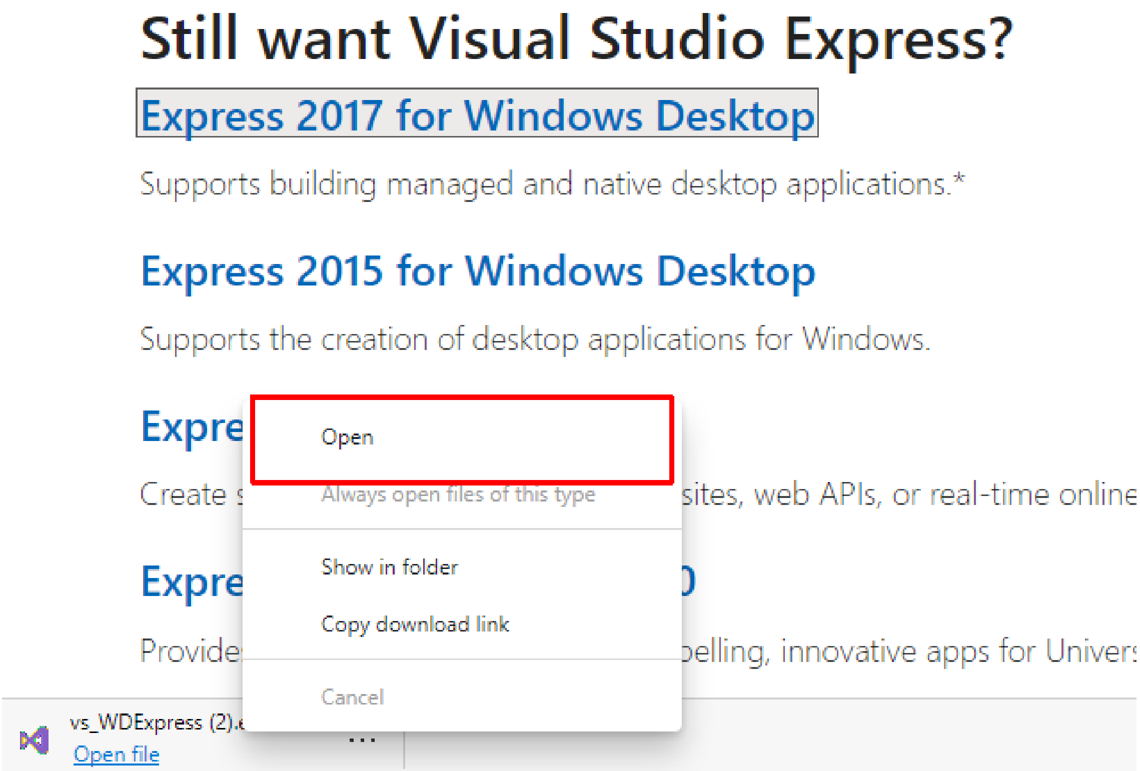




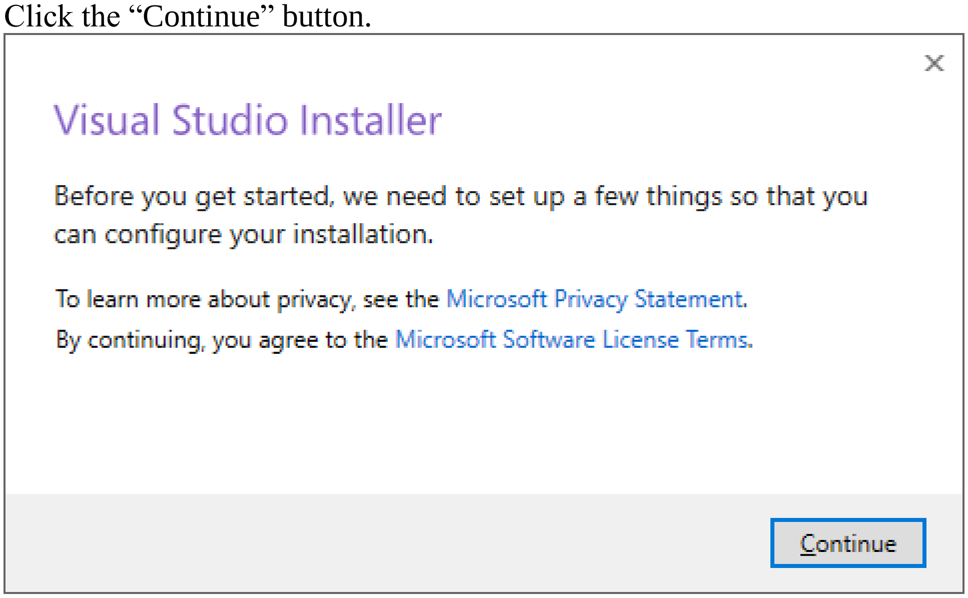
The download manager file will download the “WDExpress.exe” file.



Right click on the WDExpress.exe.

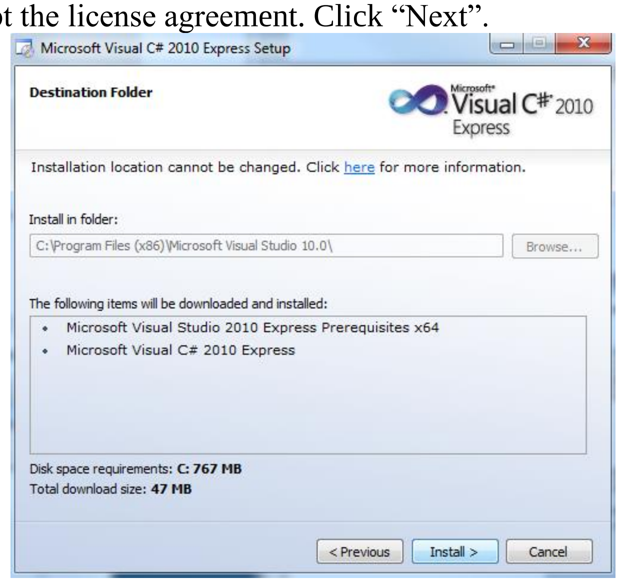


Click the “Continue” button.

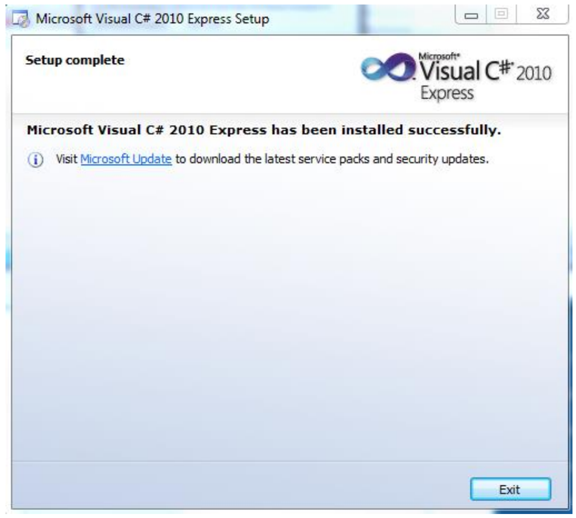


Next, follow the on screen windows and accept the default answers.

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



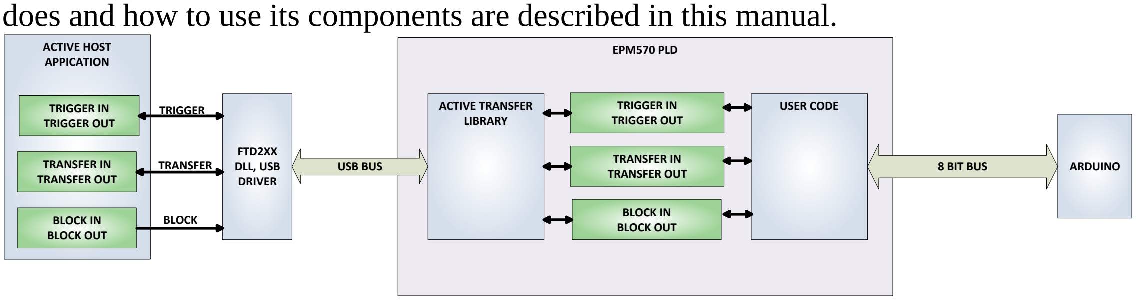
Visual C# 2010 Express will install. This may take up to twenty minutes depending on your internet connection.



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

3 Active Transfer Library

The Active Transfer Library is an HDL library designed to transfer data to and from the MegaMax 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.



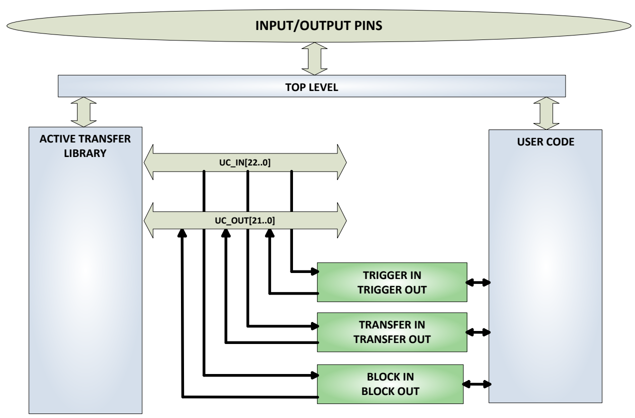
3.1 *EPT Active Transfer System Overview*

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

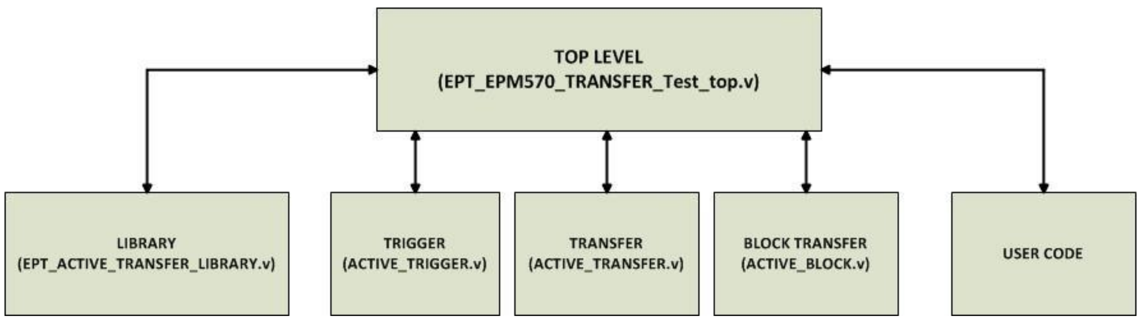
The Active_Serial_Library provides the communication to the USB hardware. While separate Input and Output buses provide bi-directional communications with the plug in modules.



The figure above shows how the modules of the EPT Active Transfer Library attach to the overall user project. The EPT Active_Transfer_Library.v, 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 tools allows users to focus on creating programmable logic applications and not have to become distracted by USB Host drivers and timing issues. The Active Transfer Library is pre-compiled file that the user will include in the project files.



```
//##########################################################################
//#
//# Copyright   Earth People Technology Inc. 2023
//#
//#
//# File Name:  EPT_5M57_AP_M4_Top.v
//# Author:     R. Jolly
//# Date:       January 14, 2023
//# Revision:   A
//#
//# Development: USB Test Tool Interface board
//# Application: Altera MAX V CPLD
//# Description: This file contains verilog code which will allow access
//#              to Active Transfer Library. The MAX V receives its commands via
//#              USB.
//#
//#
//#
//#**************************************************************************
//#
//# Revision History:
//#         DATE        VERSION     DETAILS
//#         1/14/23     A           Created         RJJ
//#                                 RJJ
//#
//#
//#
//##########################################################################

`timescale 1ns/1ps


```

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

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,


   //-----------------------------------------------
   // 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)

    );
```

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

The control 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.

```
wire [22*3-1:0]  uc_out_m;
eptWireOR # (.N(3)) wireOR (UC_OUT, uc_out_m);
    active_trigger             ACTIVE_TRIGGER_INST
    (
     .uc_clk                   (CLK_66),
     .uc_reset                 (RST),
     .uc_in                    (UC_IN),
     .uc_out                   (uc_out_m[ 0*22 +: 22 ]),

     .trigger_to_host          (trigger_to_host),
     .trigger_to_device        (trigger_in_byte)

    );
```

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.

```
//-----------------------------------------------
// Detect Trigger Out to Host
//-----------------------------------------------
always @(TRIGGER_OUT or trigger_in_reset or reset)
begin
   if(!reset)
       trigger_to_host = 8'h0;
   else if (trigger_in_reset)
       trigger_to_host = 8'h0;
   else if (TRIGGER_OUT > 8'h0)
       trigger_to_host = TRIGGER_OUT;
end

//-----------------------------------------------
// Reset Trigger Out to Host
//-----------------------------------------------
always @(posedge CLK_66 or negedge reset)
begin
   if(!reset)
   begin
       trigger_in_reset <= 0;
   end
   else
   begin
       if (trigger_to_host > 0)
           trigger_in_reset <= 1'b1;
       else
           trigger_in_reset <= 0;
   end
 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.

```
  //-----------------------------------------------
  // Detect Trigger In
  //-----------------------------------------------
always @(trigger_in_byte or trigger_in_reset or reset)
begin
   if(!reset)
   begin
       trigger_in_detect = 1'b0;
   end
   else if (trigger_in_reset)
   begin
       trigger_in_detect = 1'b0;
   end
   else if (trigger_in_byte > 8'h0)
   begin
       trigger_in_detect = 1'b1;
   end
end

  //-----------------------------------------------
  // Store the value of Trigger In
  //-----------------------------------------------
always @(posedge CLK_66 or negedge reset)
begin
   if(!reset)
   begin
       trigger_in_store <= 8'h0f;
       trigger_in_reg <= 1'b0;
       trigger_in_reset <= 1'b0;
   end
   else if (trigger_in_detect & !trigger_in_reg)
   begin
       if(trigger_in_byte != 0)
       trigger_in_store[7:0] <= trigger_in_byte[7:0];
       trigger_in_reg <= 1'b1;
   end
   else if (trigger_in_reg)
   begin
           trigger_in_reg <= 1'b0;
           trigger_in_reset <= 1'b1;
   end
   else if (!trigger_in_detect)
   begin
       trigger_in_reg <= 1'b0;
       trigger_in_reset <= 1'b0;
   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.

```
active_transfer             ACTIVE_TRANSFER_INST
(
 .uc_clk                    (CLK_66),
 .uc_reset                  (reset),
 .uc_in                     (UC_IN),
 .uc_out                    (uc_out_m[ 1*22 +: 22 ]),

 .start_transfer            (transfer_out_reg),
 .transfer_received         (transfer_in_received),

 .uc_addr                   (3'h2),

 .transfer_to_host          (transfer_out_byte),
 .transfer_to_device        (transfer_in_byte)
);
```

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.

```
//---------------------------------------------------
 // Transfer byte to Device
 //---------------------------------------------------
 always @(TRANSFER_OUT_EN or reset)
 begin
     if(!reset)
     begin
         transfer_out_detect = 1'b0;
     end
     else
     begin
         if(transfer_to_device_reset)
             transfer_out_detect = 1'b0;
         else if(TRANSFER_OUT_EN)
         begin
             transfer_out_byte = TRANSFER_OUT_BYTE;
             transfer_out_detect = 1'b1;
         end
     end
 end

 //---------------------------------------------------
 // Reset transfer_to_device_reset
 //---------------------------------------------------
  always @(posedge CLK_66 or negedge reset)
  begin
      if (!reset)
      begin
          transfer_to_device_reset <= 1'b0;
      end
      else
      begin
          if(transfer_out_detect)
             transfer_to_device_reset <= 1'b1;
          else
             transfer_to_device_reset <= 1'b0;
      end
  end
```

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

```
//------------------------------------------------
// Transfer to Host
//------------------------------------------------
 always @(posedge CLK_66 or negedge reset)
 begin
      if (!reset)
      begin
          transfer_out <= 1'b0;
          transfer_out_reg <= 1'b0;
          transfer_out_byte <= 8'h0;
      end
      else
      begin
          if(start_transfer_byte & !transfer_out)
          begin
              transfer_out_byte <= TRANSFER_HOST_BYTE;
              transfer_out_reg <= 1'b1;
              transfer_out <= 1'b1;
          end
          else if(start_transfer_byte & transfer_out)
          begin
              transfer_out_reg <= 1'b0;
              transfer_out <= 1'b1;
          end
          else if(!start_transfer_byte & transfer_out)
          begin
              transfer_out_reg <= 1'b0;
              transfer_out <= 1'b0;
          end
      end
  end
```

3.2.3 Active Block EndTerm

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

```
active_block                    BLOCK_TRANSFER_INST
(
 .uc_clk                        (CLK_66),
 .uc_reset                      (RST),
 .uc_in                         (UC_IN),
 .uc_out                        (uc_out_m[ 2*22 +: 22 ]),

 .start_transfer                (block_out_reg),
 .transfer_received             (block_in_rcv),

 .transfer_ready                (block_byte_ready),

 .uc_addr                       (3'h4),
 .uc_length                     (BLOCK_COUNT_8),

 .transfer_to_host              (block_out_byte),
 .transfer_to_device            (block_in_data),

.STATE_OUT                       (block_state_out),
.TEST_BUS                        (block_out_test_bus)

);
```

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

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

```
 //------------------------------------------------
 // Registers to start Block Transfer Out
 //------------------------------------------------
always @(posedge CLK_66 or negedge RST)
begin
  if(!RST)
  begin
        block_out_reg <= 1'b0;
       start_block_transfer_reg <= 1'b0;
  end
  else
  begin
     if(start_block_transfer & !start_block_transfer_reg)
       start_block_transfer_reg <= 1'b1;
     else if(start_block_transfer_reg & !block_out_reg)
     begin
       block_out_reg <= 1'b1;
     end
     else if(block_out_counter >= BLOCK_COUNT_8)
     begin
         block_out_reg <= 1'b0;
         start_block_transfer_reg <= 1'b0;
     end
  end
end

 //------------------------------------------------
 // Data for Block Transfer Out
 //------------------------------------------------
always @(posedge CLK_66 or negedge RST)
begin
  if(!RST)
  begin
     block_out_counter <= 0;
  end
  else
  begin
       if(block_byte_ready)
       begin
         block_out_counter <= block_out_counter + 1'd1;
       end
       else if(block_out_counter >= BLOCK_COUNT_8 )
       begin
         block_out_counter <= 0;
       end
  end
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

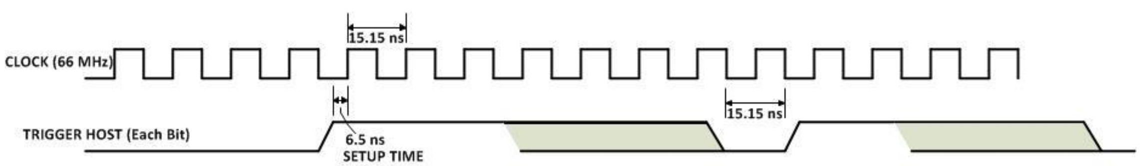


Figure xx Active Trigger to Host Timing

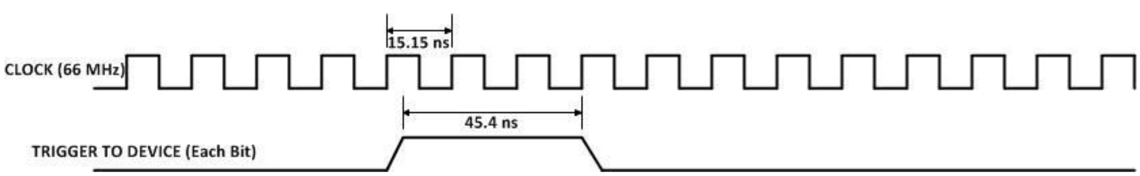


Figure xx Active Trigger to Device Timing

3.3.2 Active Transfer EndTerm Timing

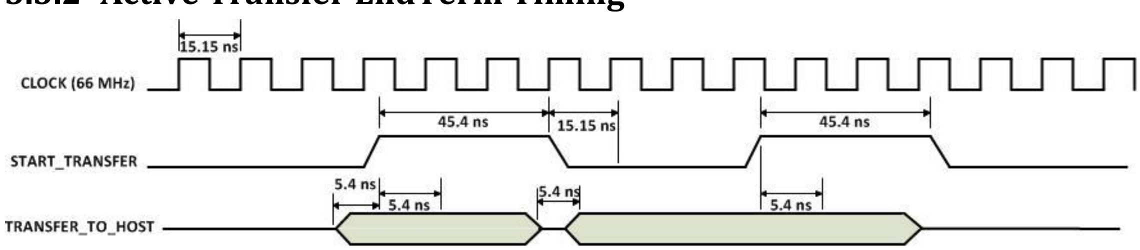


Figure xx Active Transfer To Host Timing

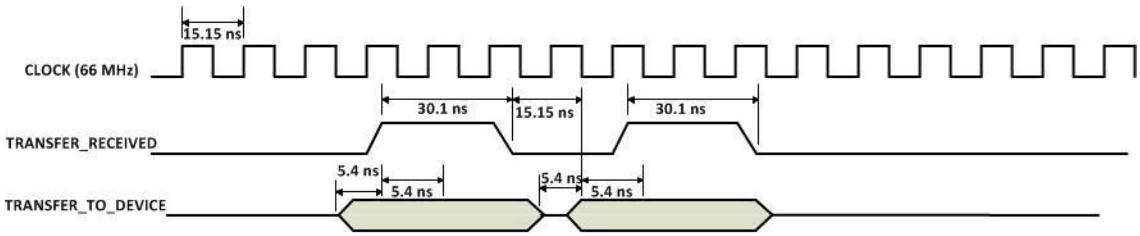


Figure xx Active Transfer To Device Timing

3.3.3 Active Block EndTerm Timing

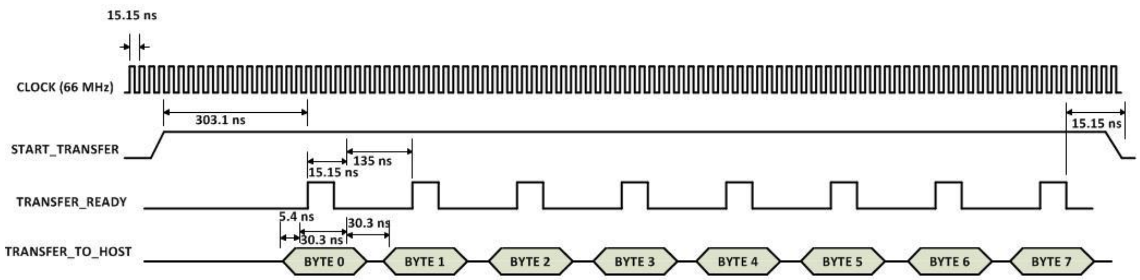


Figure xx Active Block To Host Timing

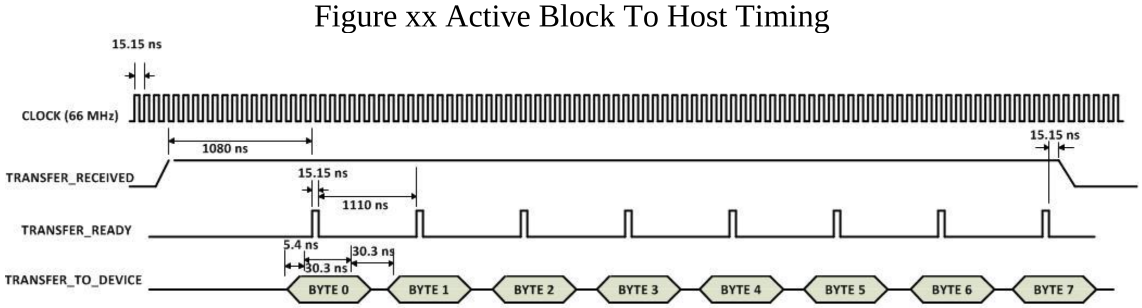
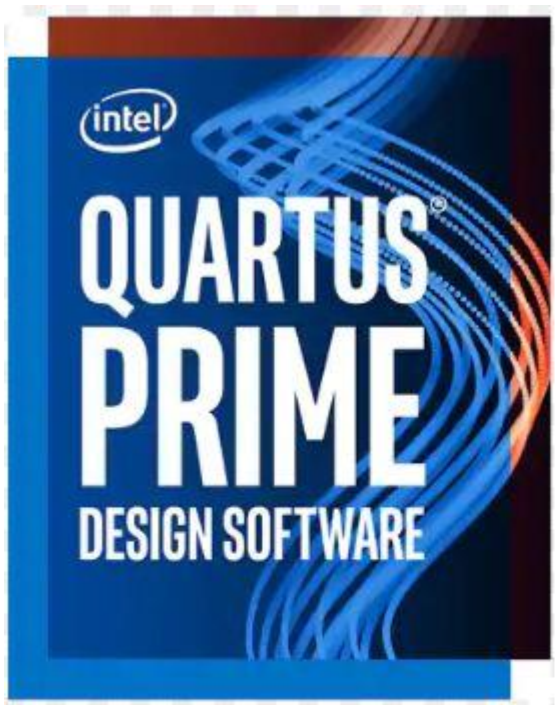


Figure xx Active Block To Device Timing

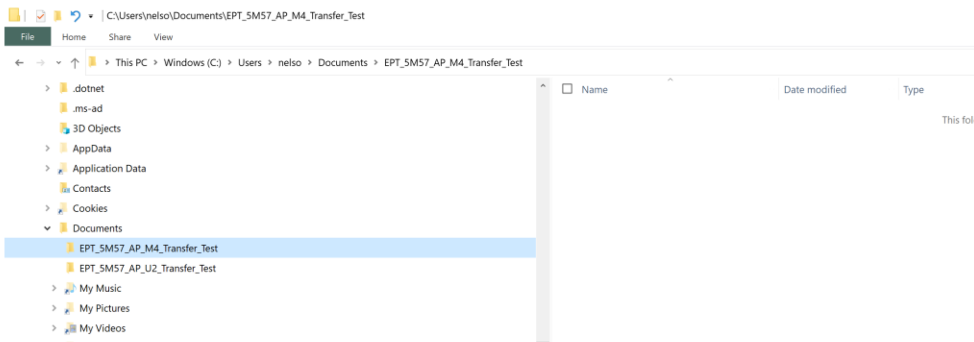
4 Compiling, Synthesizing, and Programming CPLD



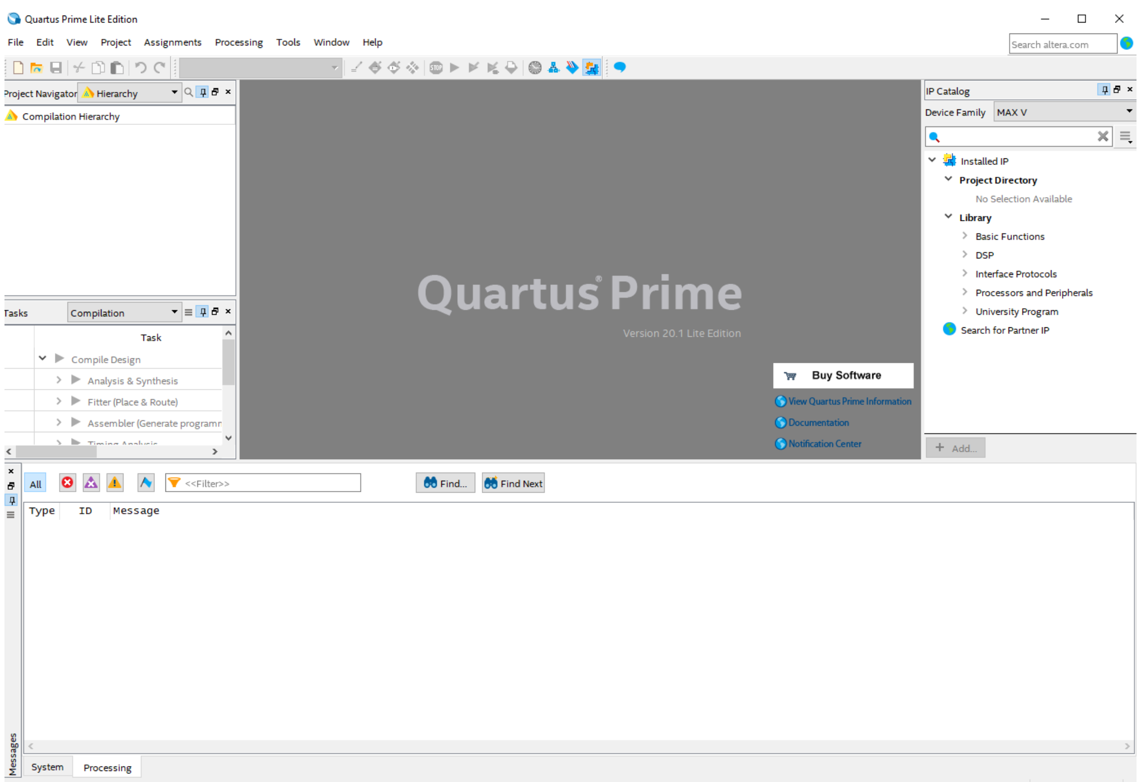
The CPLD on the MegaMax can be programmed with the Active Transfer Library and custom HDL code created by the user. Programming the CPLD requires the use of the Quartus Prime software and a standard USB cable. There are no extra parts to buy, just plug in the USB cable. Once the user HDL code is written 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*

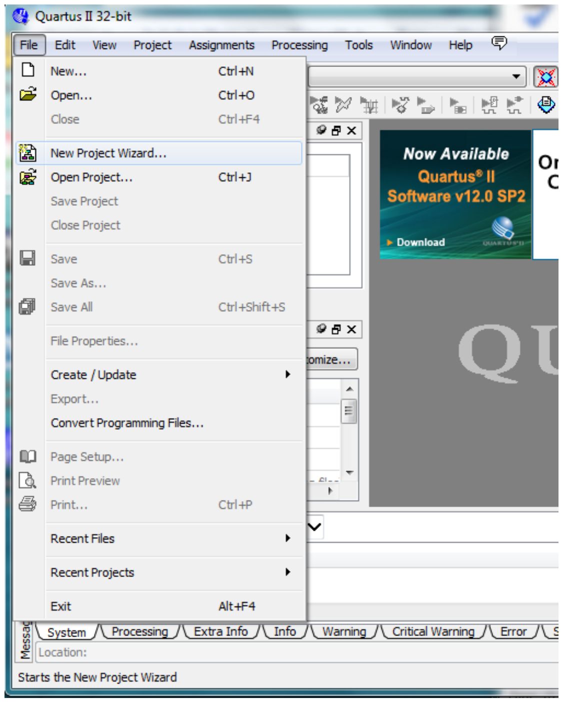
Once the HDL code (Verilog or VHDL) is written and verified using a simulator, a project can be created using Quartus Prime. Writing the HDL code and simulating it will be covered in later sections. Bring up Quartus Prime, then use Windows Explorer to browse to C:\Users\nelso\Documents\EPT_5M57_AP_M4_Transfer_Test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



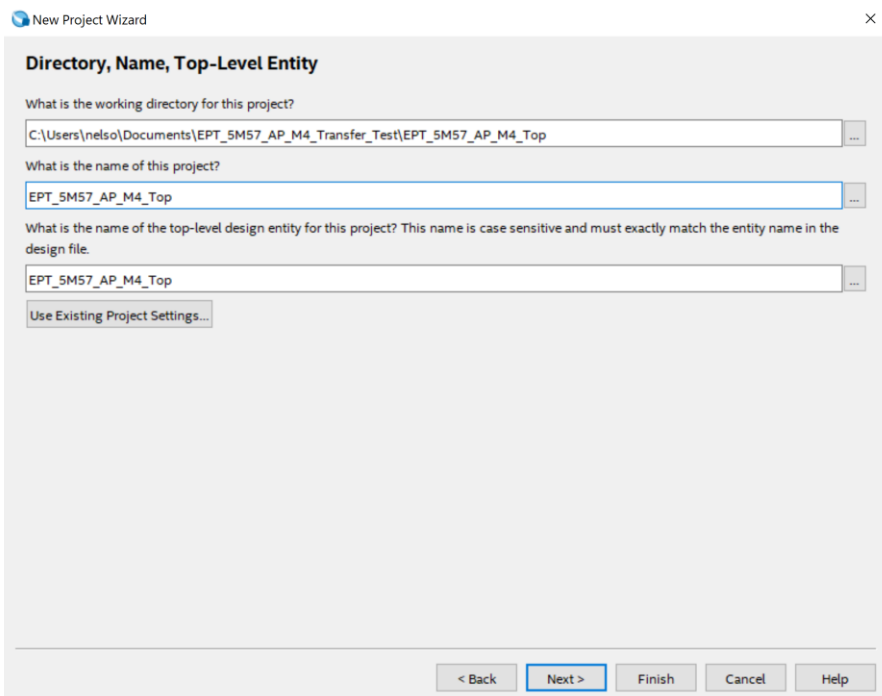
Open Quartus Prime by clicking on the icon .

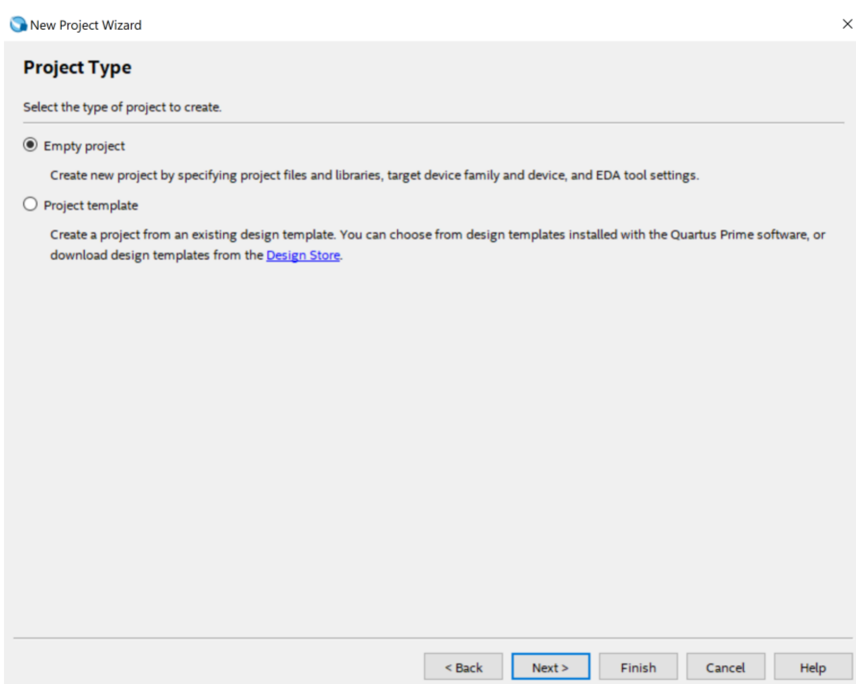


Under Quartus, Select File->New Project Wizard. The Wizard will walk you through setting up files and directories for your project.

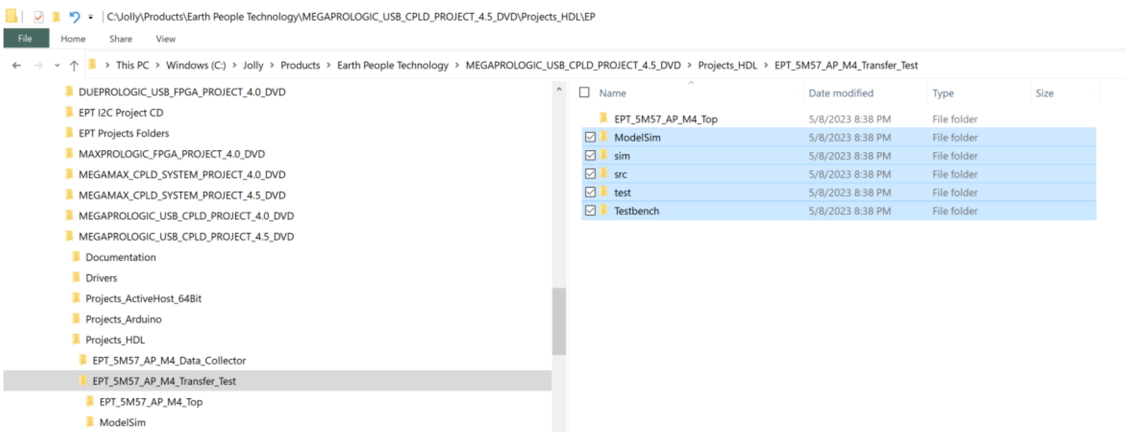


At the Top-Level Entity page, browse to the C:\Users\<your name>\Documents\EPT_5M57_AP_M4_Transfer_Test\EPT_5M57_AP_M4_Top directory to store your project. Type in a name for your project "EPT_570_AP_M4_Top".

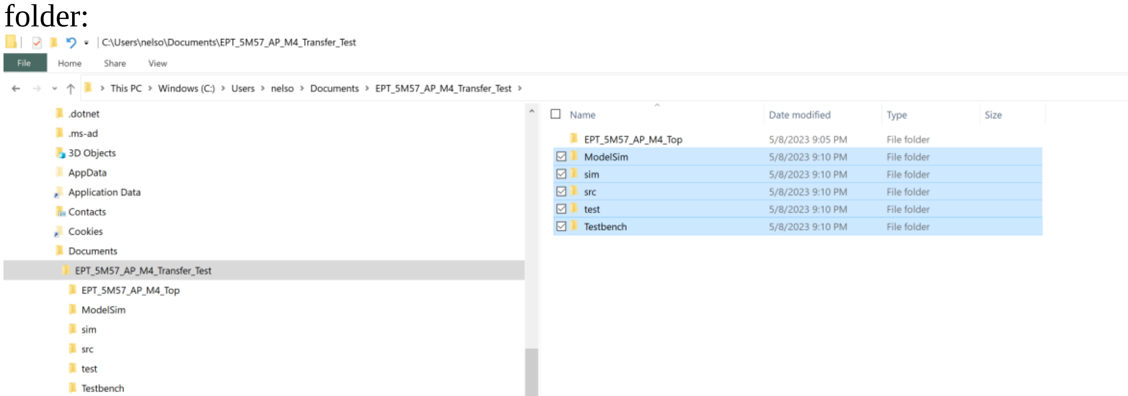

New Project Wizard
Directory, Name, Top-Level Entity
What is the working directory for this project?
C:\Users\nelso\Documents\EPT_5M57_AP_M4_Transfer_Test\EPT_5M57_AP_M4_Top
What is the name of this project?
EPT_5M57_AP_M4_Top
What is the name of the top-level design entity for this project? This name is case sensitive and must exactly match the entity name in the design file.
EPT_5M57_AP_M4_Top
Use Existing Project Settings...
< 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

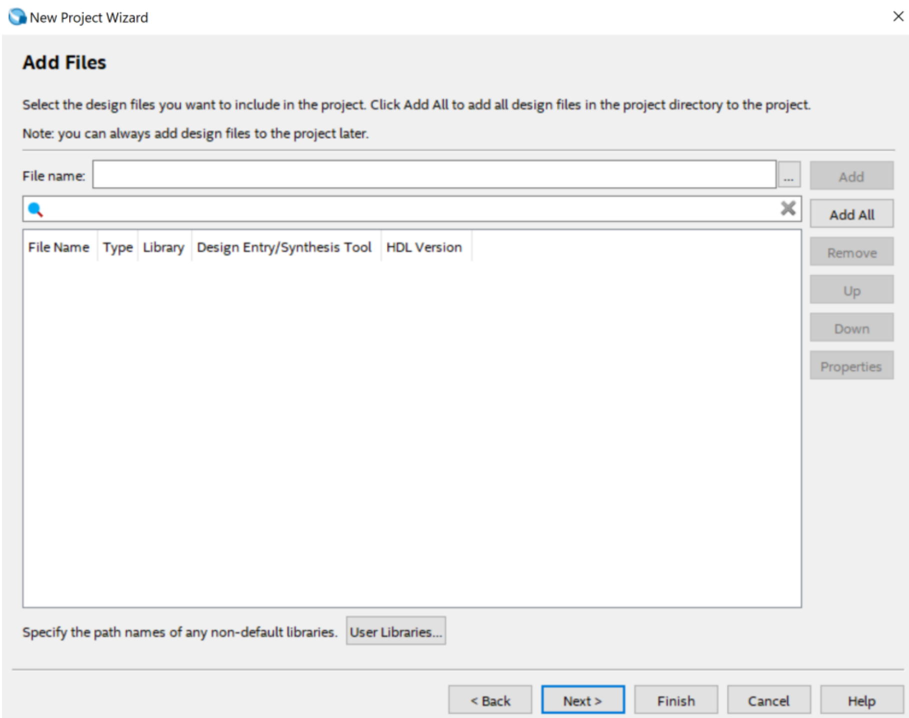
Select Next. At the Add Files window: Browse to the \Projects_HDL\EPT_Transfer_Test \src folder of the MegaMax Development System DVD. Copy the files from the following directory.



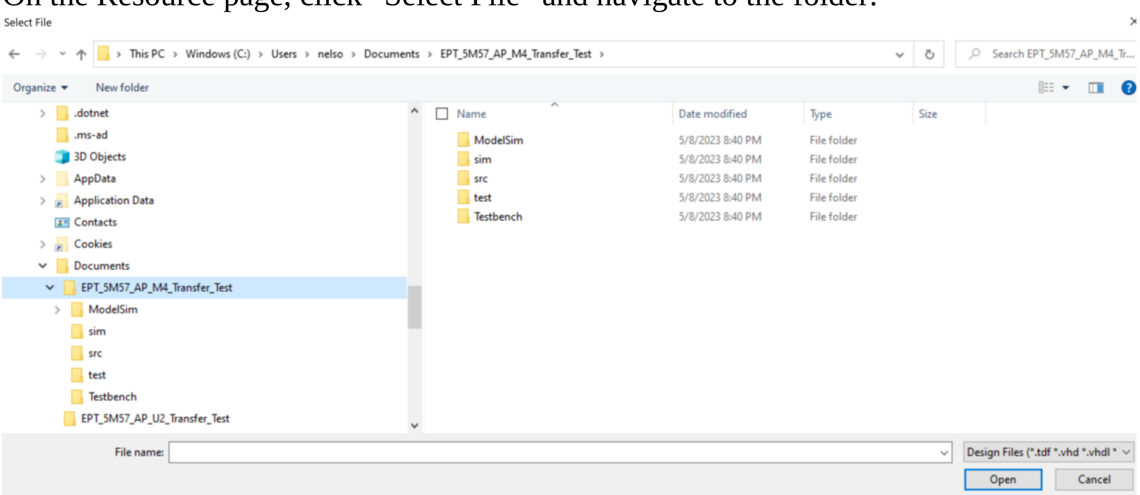
Then paste them in the C:/users/<your name>/Documents/EPT_5M57_AP_M4_Tranfer_Test\EPT_5M57_AP_M4_Top folder:



Then go to the “Add Files” page:

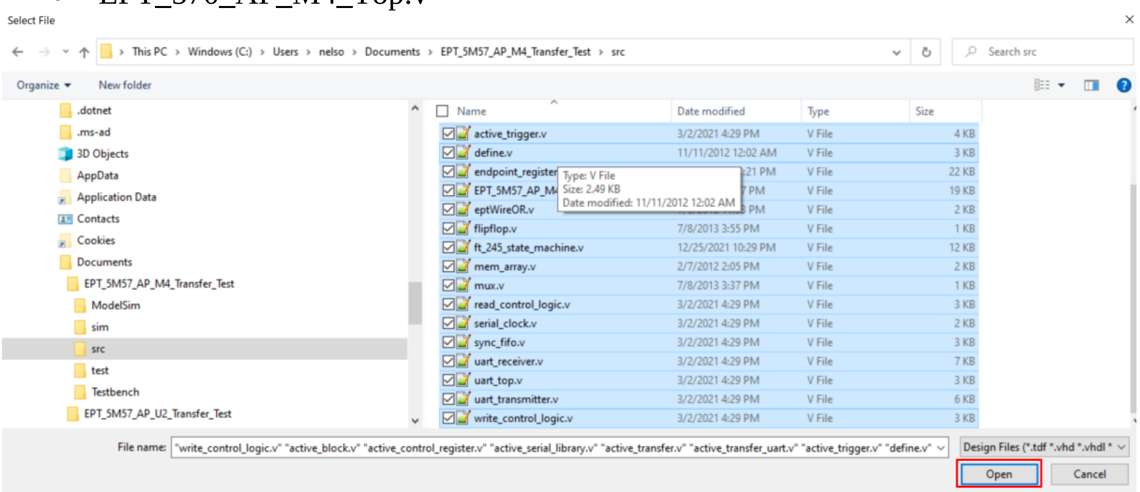


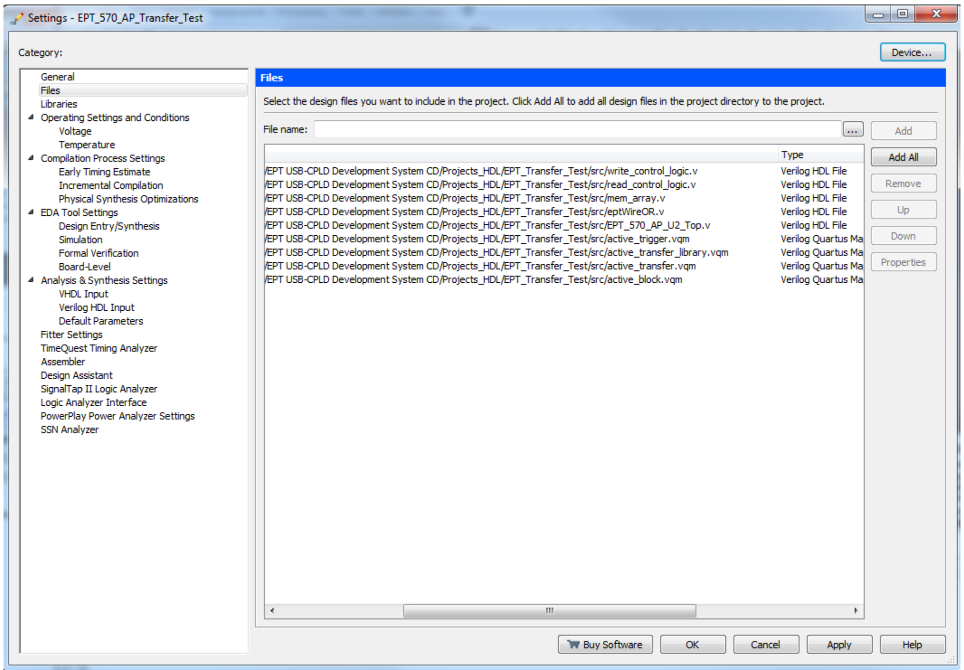
On the Resource page, click “Select File” and navigate to the folder:



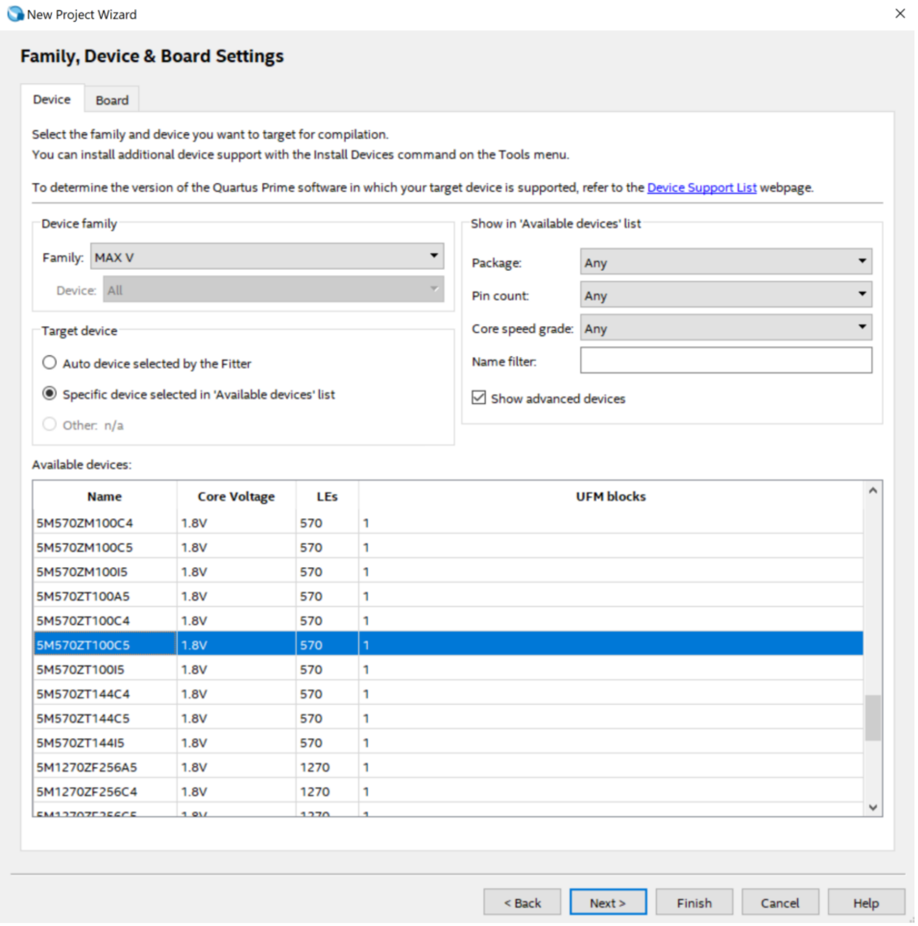
- Active_block.v
- Active_transfer.v
- Active_trigger.v
- Active_Serial_library.v
- Endpoint_registers.v

- 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

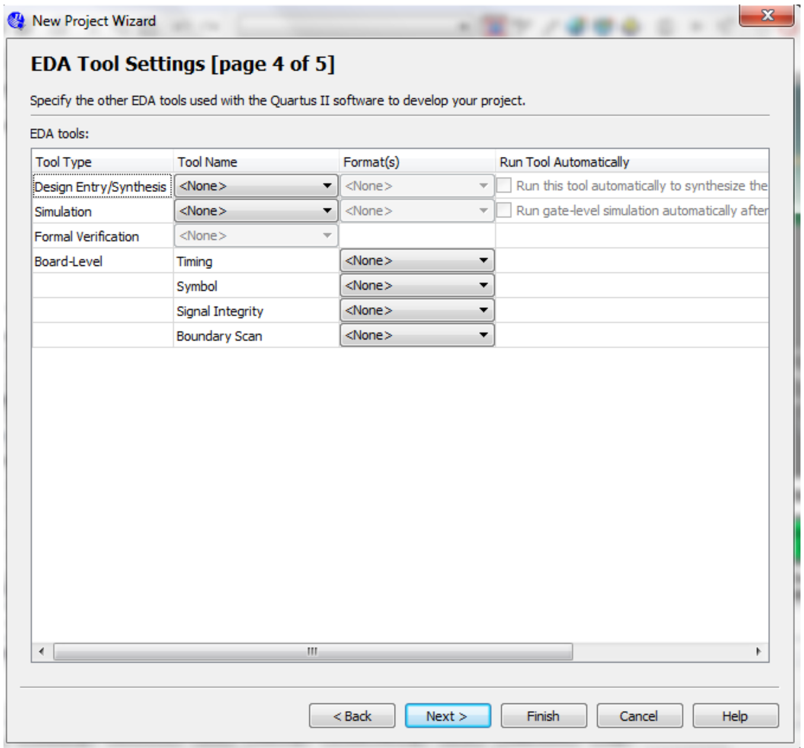




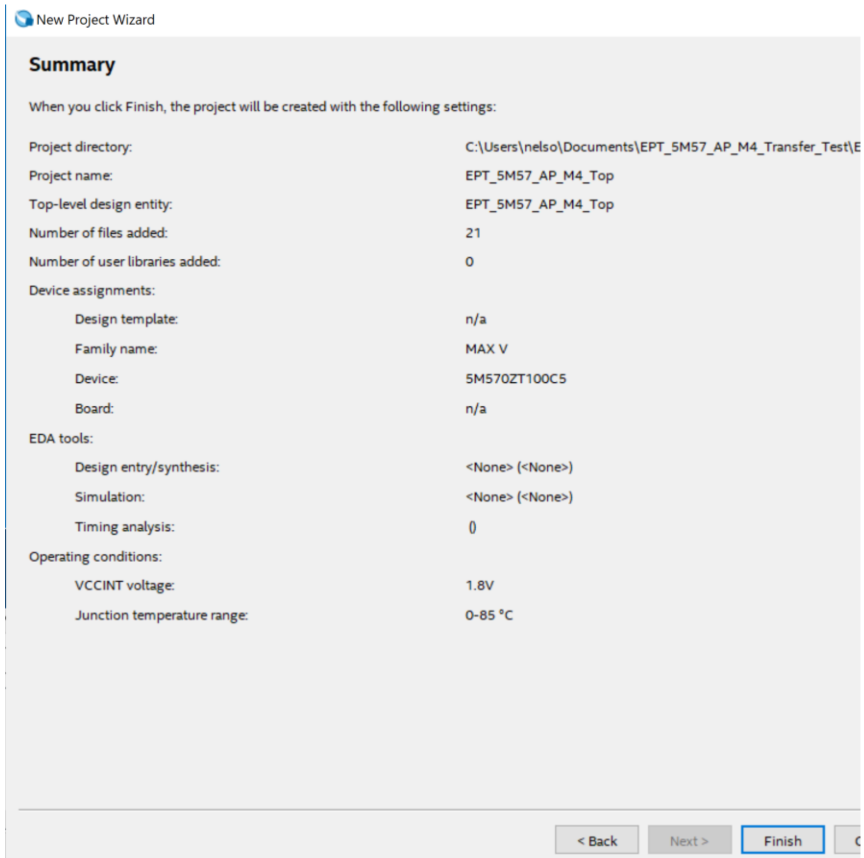
Select Next, at the Device Family group, select MAX V for Family. In the Available Devices group, browse down to 5M570ZT100C5 for Name.



Select Next, leave defaults for the EDA Tool Settings.



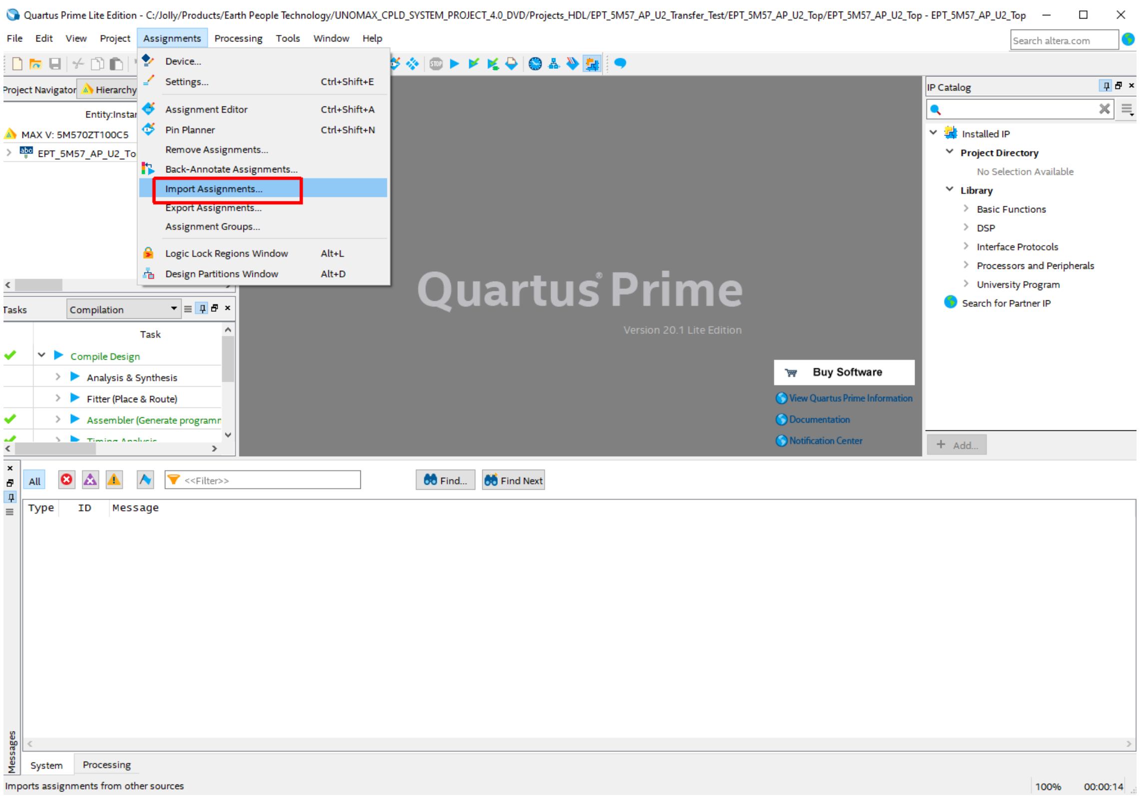
Select Next, then select Finish. You are done with the project level selections.



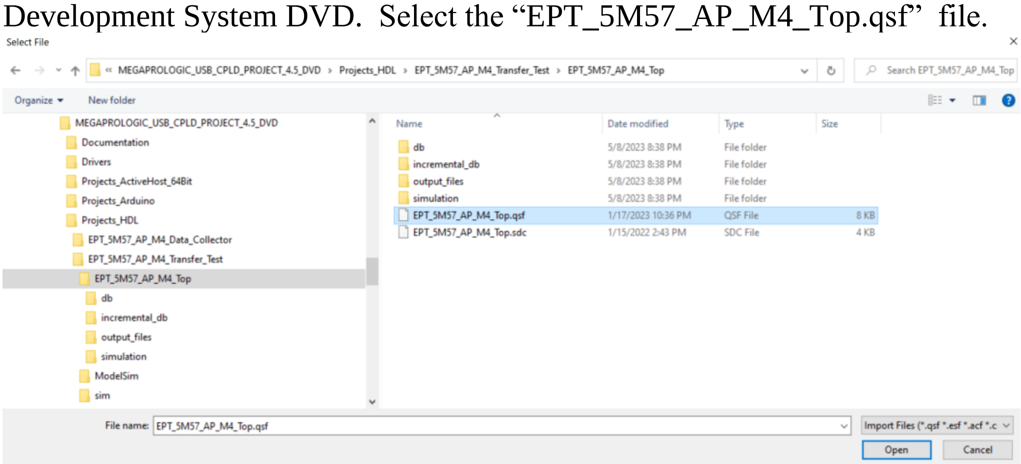
Next, we will select the pins and synthesize the project.

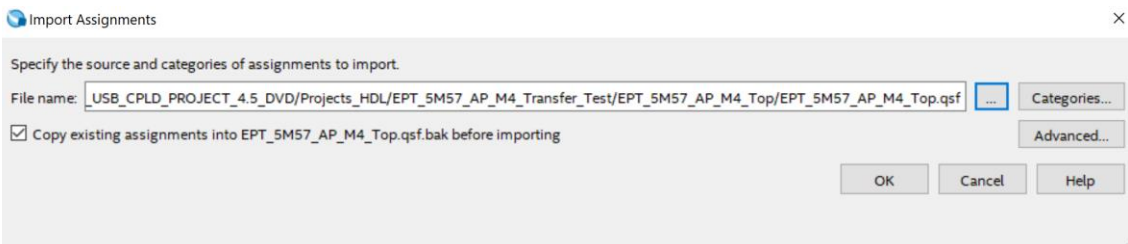
4.1.1 Selecting Pins and Synthesizing

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.

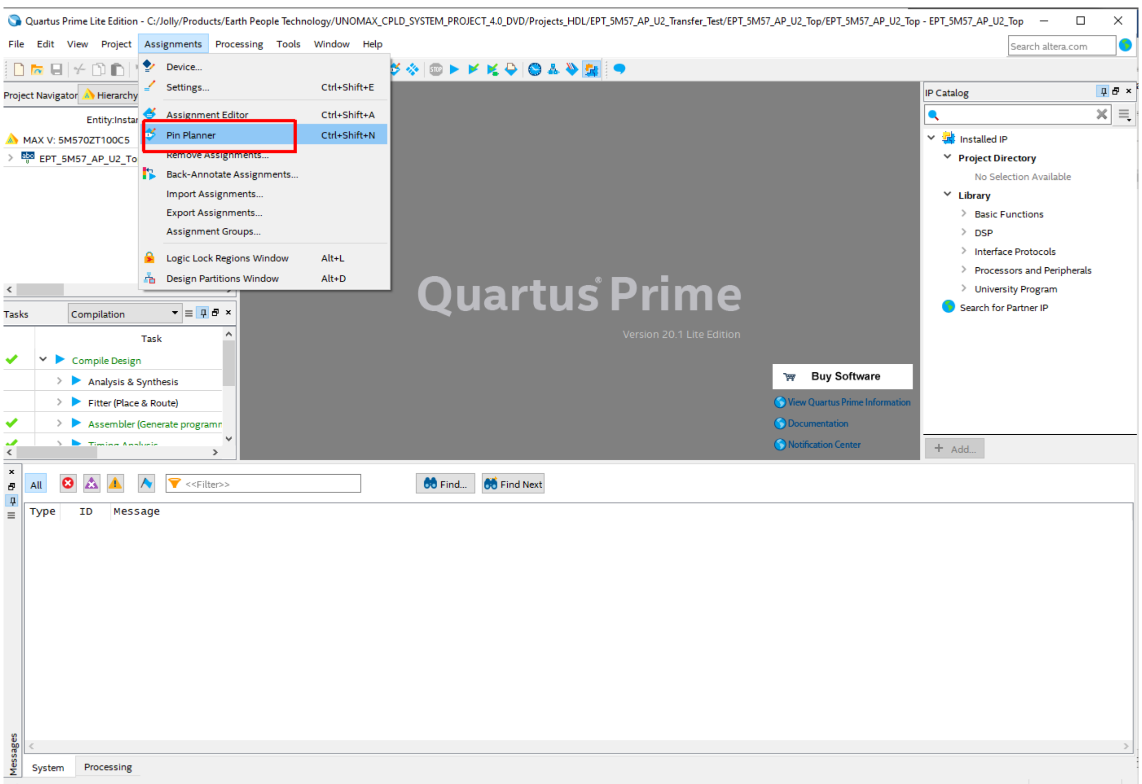


At the Import Assignment dialog box, Browse to the \Projects_HDL\EPT_Transfer_Test \ EPT_MegaMax_TOP folder of the MegaMax Development System DVD. Select the "EPT_5M57_AP_M4_Top.qsf" file.

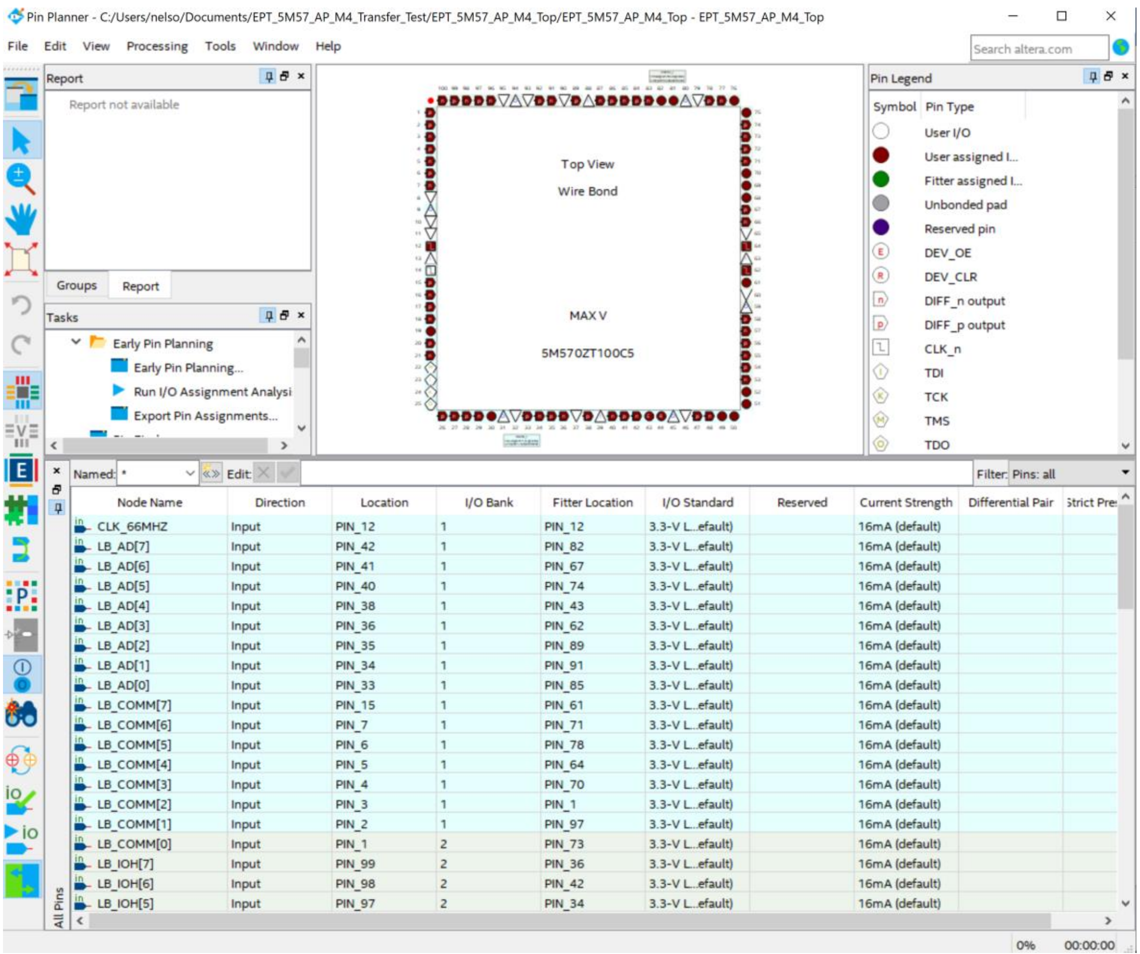




Click Ok. Under Assignments, Select Pin Planner. Verify the pins have been imported correctly.



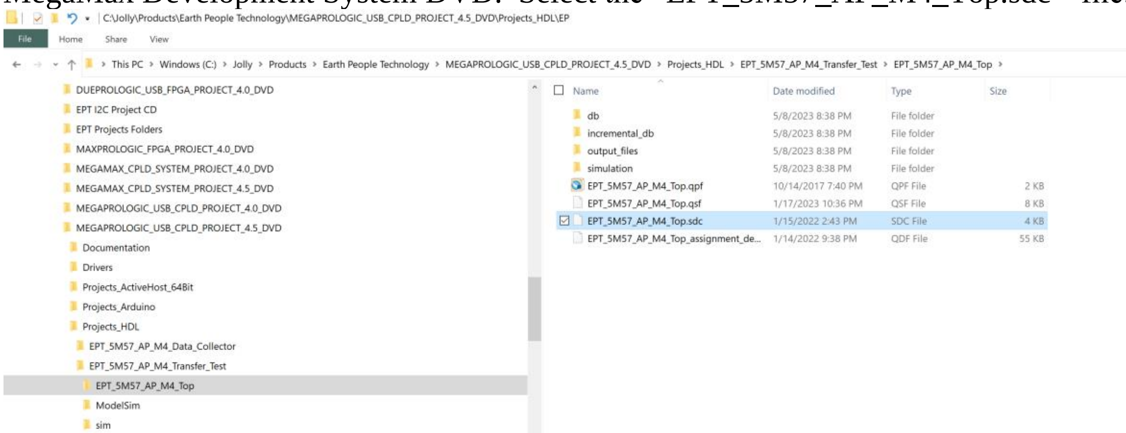
The pin locations should not need to be changed for EPT USB CPLD Development System. However, if you need to change any pin location, just click on the “location” column for the particular node you wish to change. Then, select the new pin location from the drop down box.



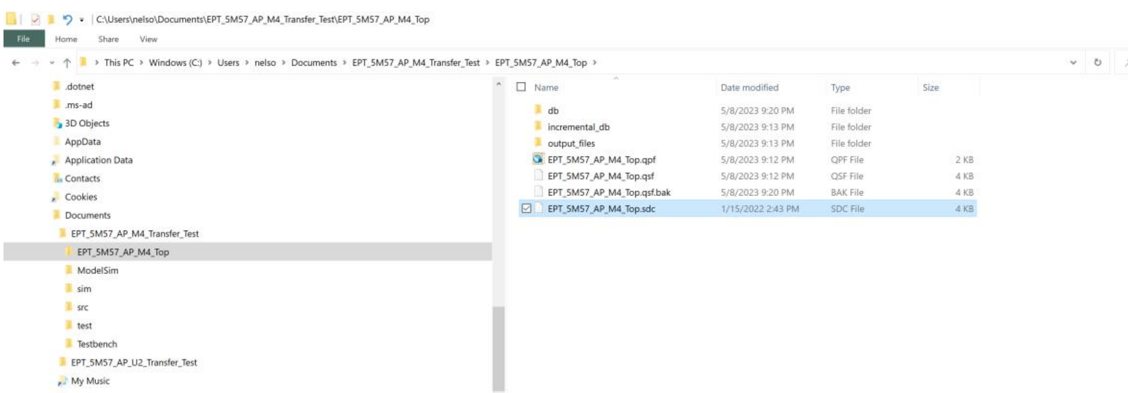
Exit the Pin Planner. Next, we need to add the Synopsys Design Constraint file. This file contains timing constraints which forces the built in tool called TimeQuest Timing Analyzer to analyze the path of the synthesized HDL code with setup and hold times of the internal registers. It takes note of any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

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

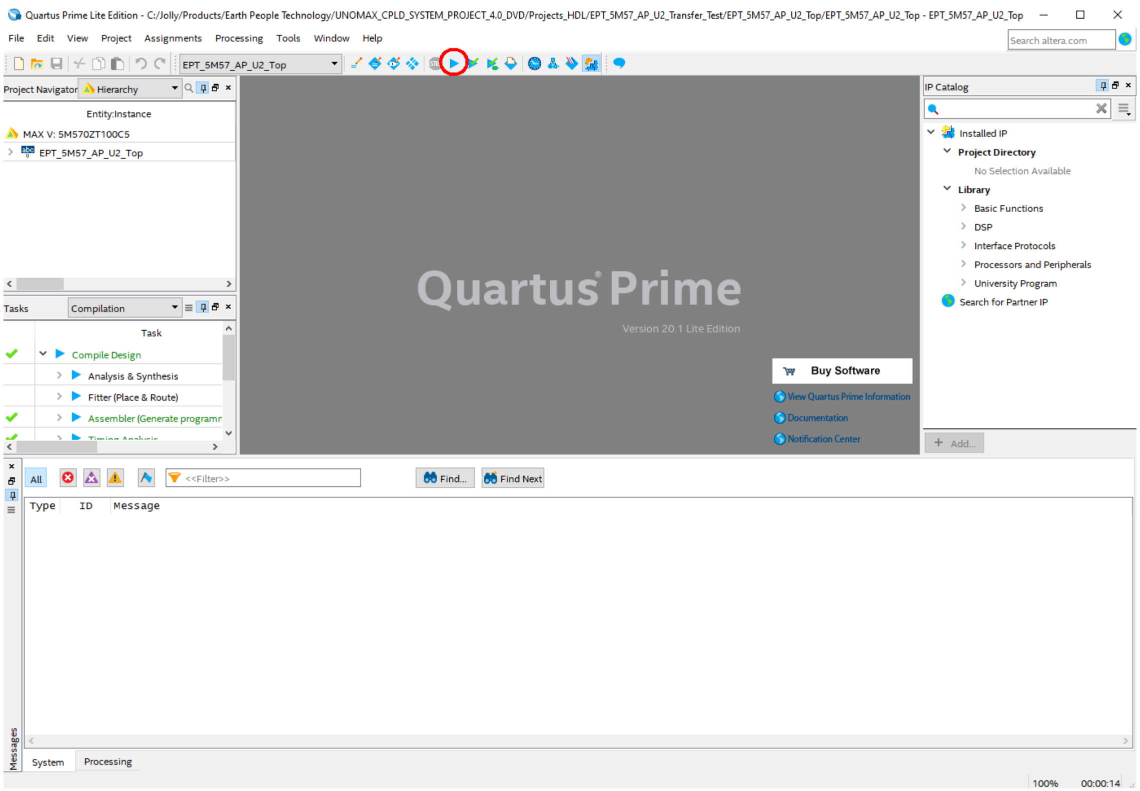
Browse to the \Projects_HDL\EPT_Transfer_Test \ EPT_MegaMax_TOP folder of the MegaMax Development System DVD. Select the “EPT_5M57_AP_M4_Top.sdc” file.



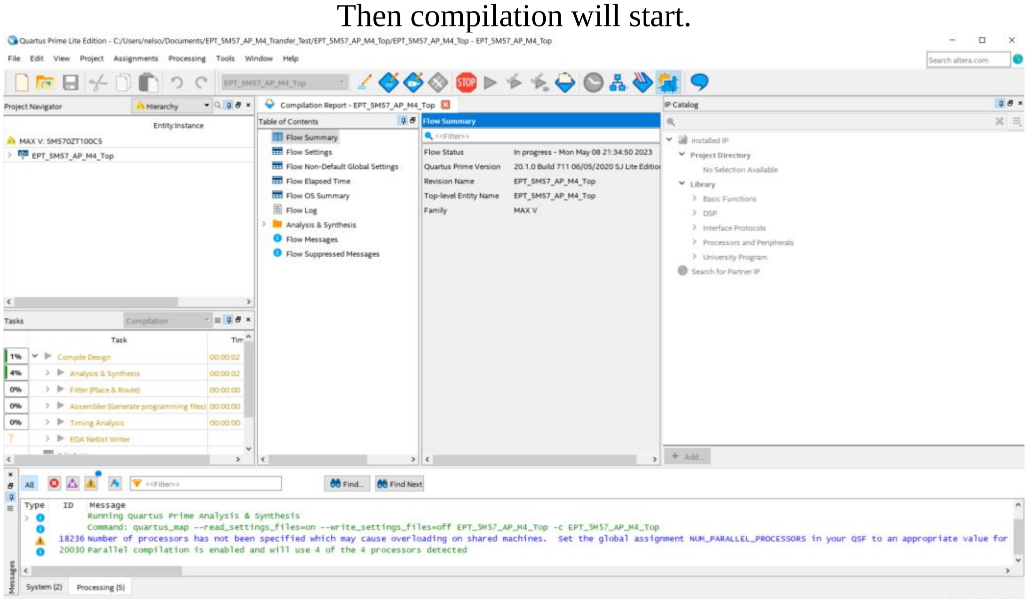
Copy the file and browse to C:\Users\nelso\Documents\EPT_5M57_AP_M4_Transfer_Test\EPT_5M57_AP_M4_Top directory. Paste the file.



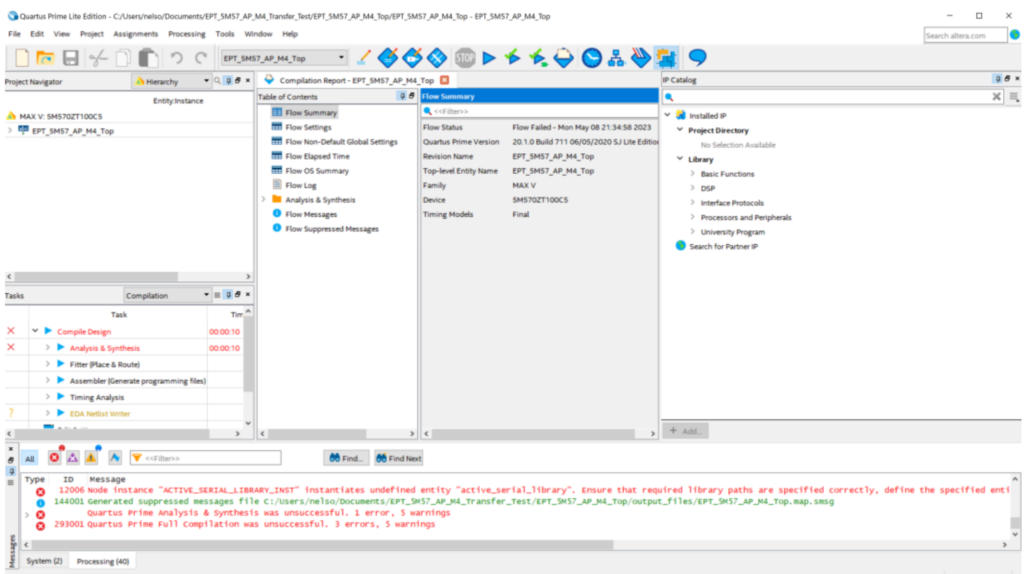
Select the Start Compilation button.



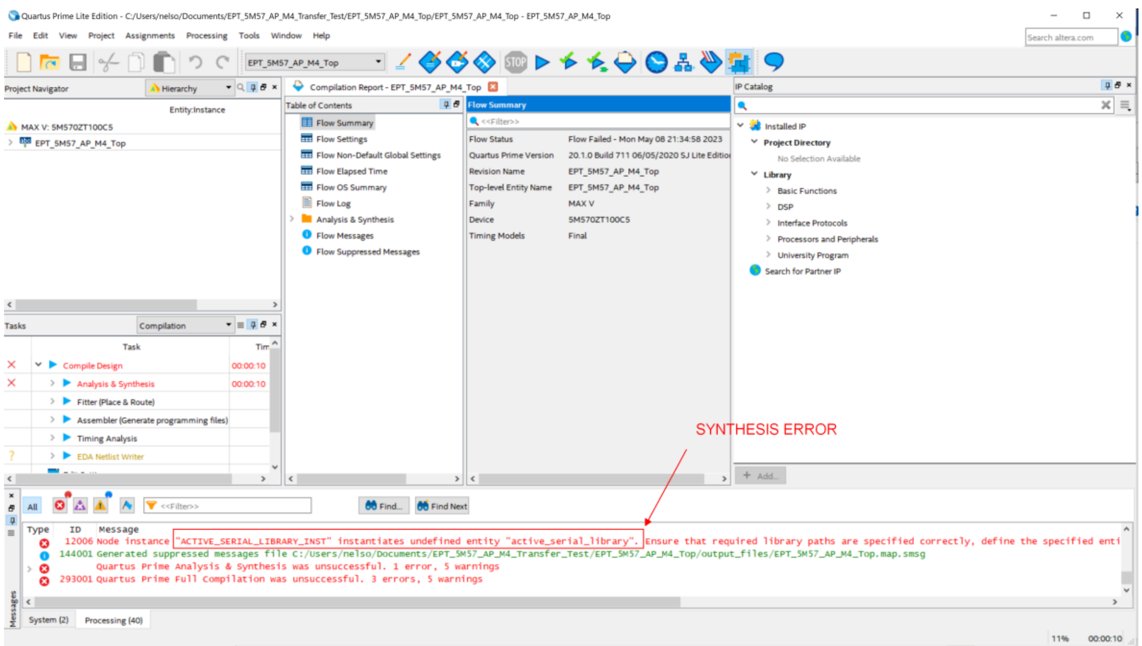
Then compilation will start.



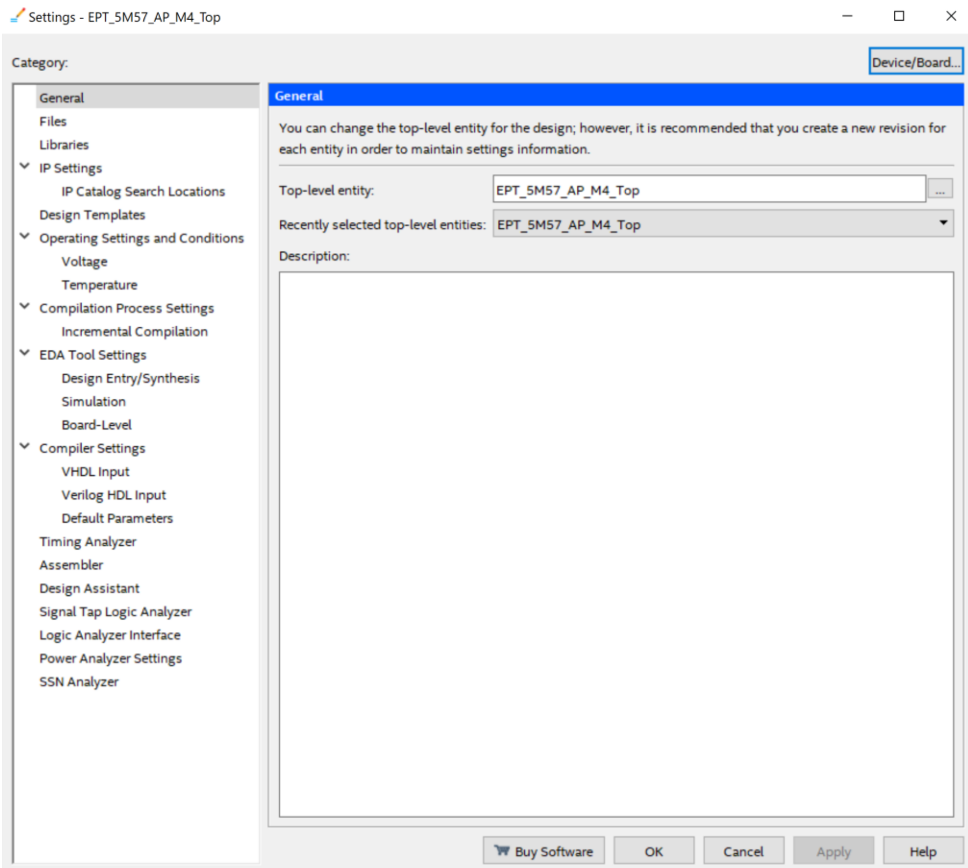
If you forget to include a file or some other error you should expect to see a screen similar to this:



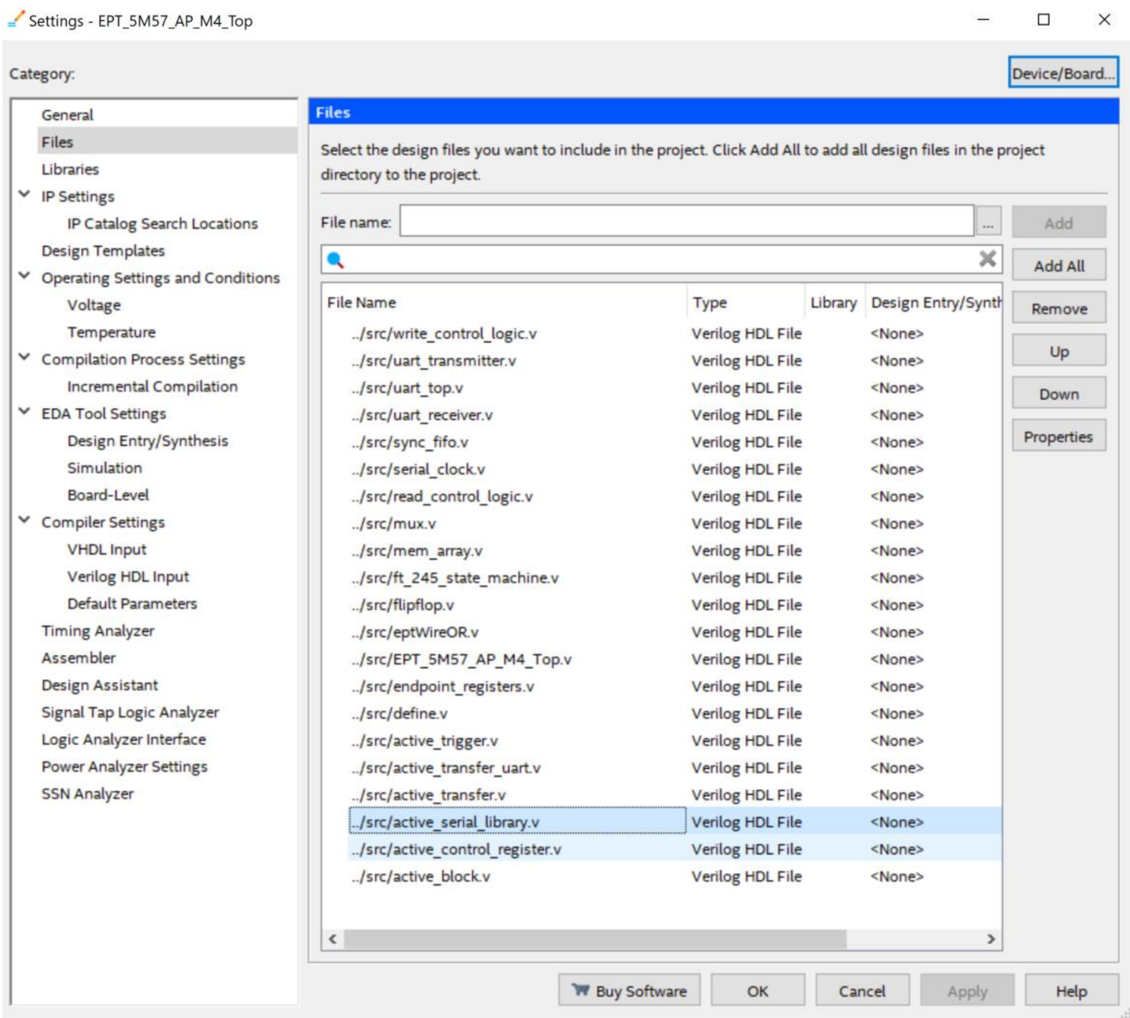
Click on the “Processing” tab at bottom to see the error.



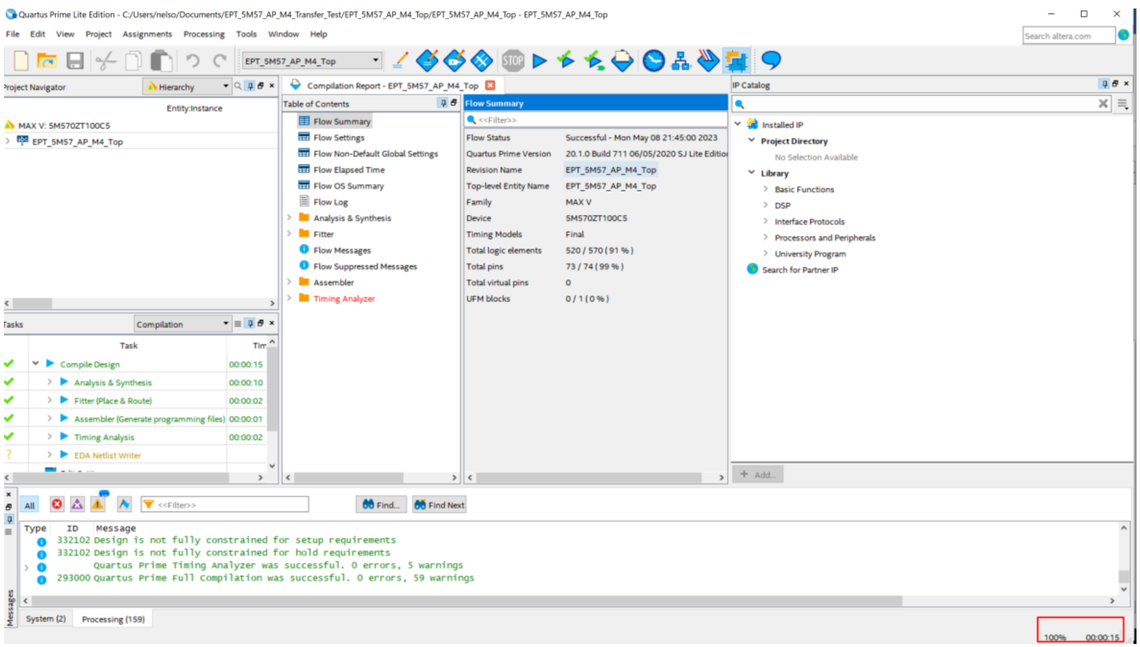
The error in this case is the missing file “active_serial_library.v”. Click on the Assignment menu,



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



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



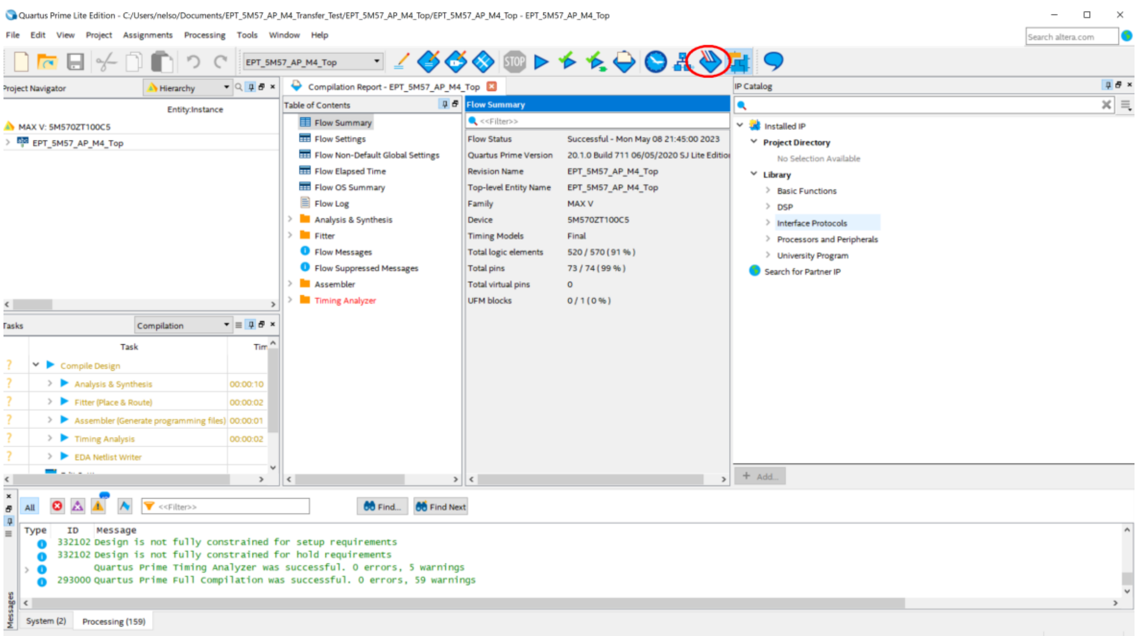
At this point the project has been successfully compiled, synthesized and a programming file has been produce. See the next section on how to program the CPLD.

4.1.2 Programming the CPLD

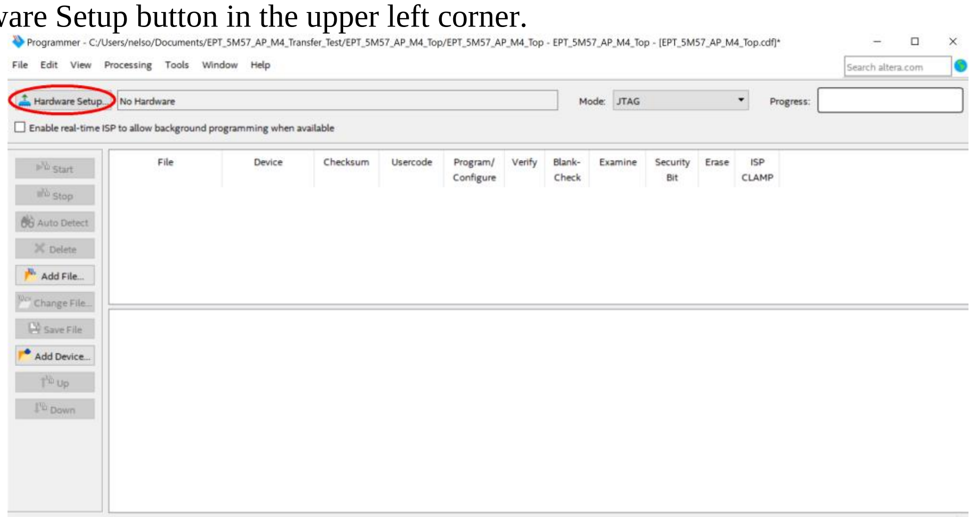
Programming the CPLD is quick and easy. All that is required is a standard USB-C cable. Connect the MegaMax to the PC, open up Quartus Prime, open the programmer tool, and click the Start button. To program the CPLD, follow the steps to install the USB Driver and the JTAG Driver Insert for Quartus Prime.

USB CPLD Development System User Manual

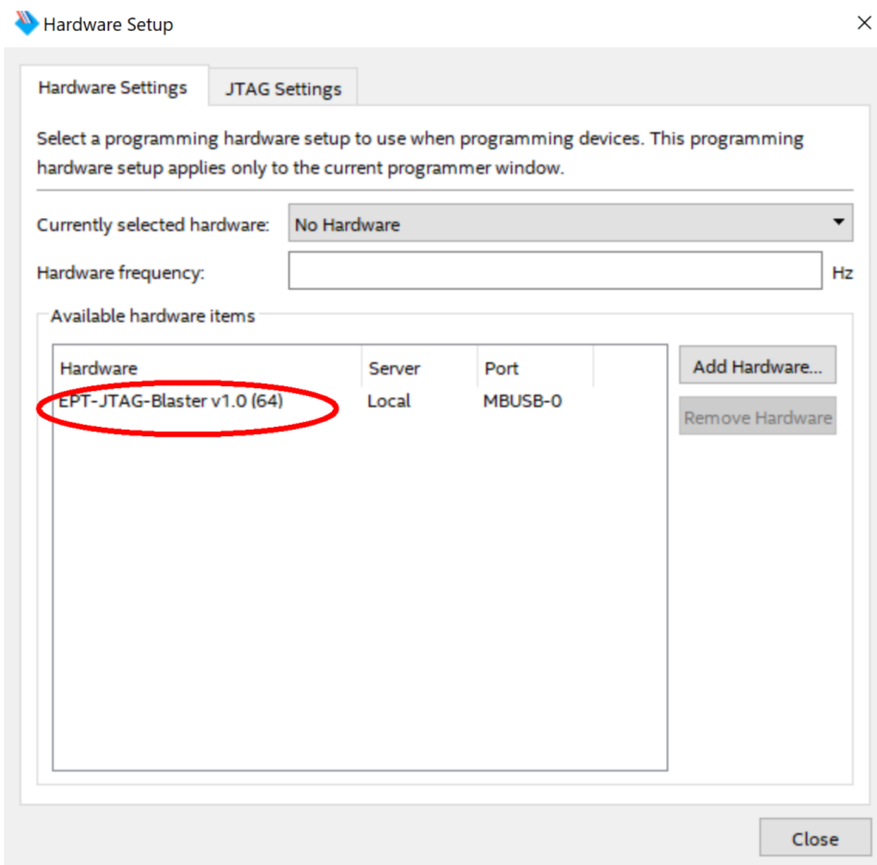
If the project created in the previous sections is not open, open it. Click on the Programmer button.



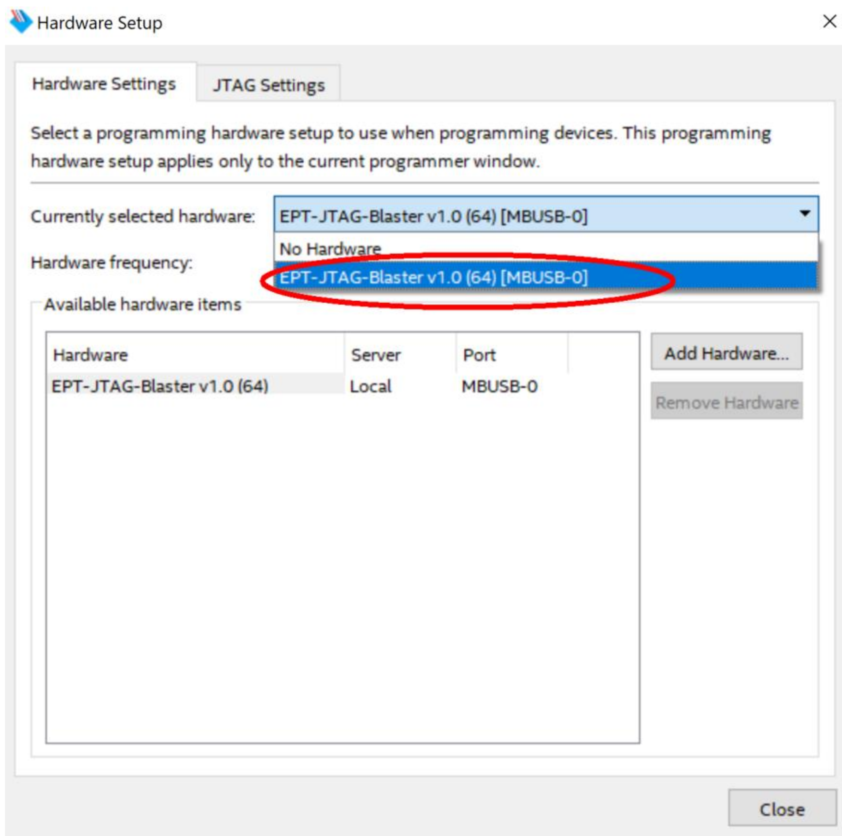
The Programmer Window will open up with the programming file selected. Click on the Hardware Setup button in the upper left corner.



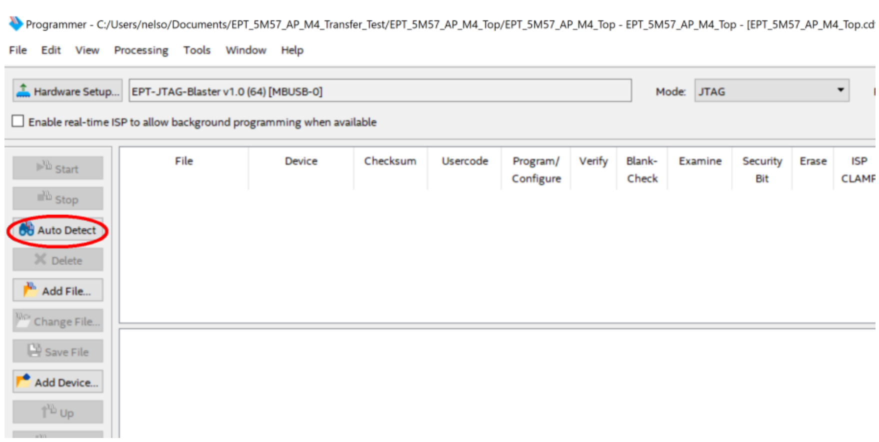
The Hardware Setup Window will open. In the “Available hardware items”, double click on “EPT-Blaster v1.0”.



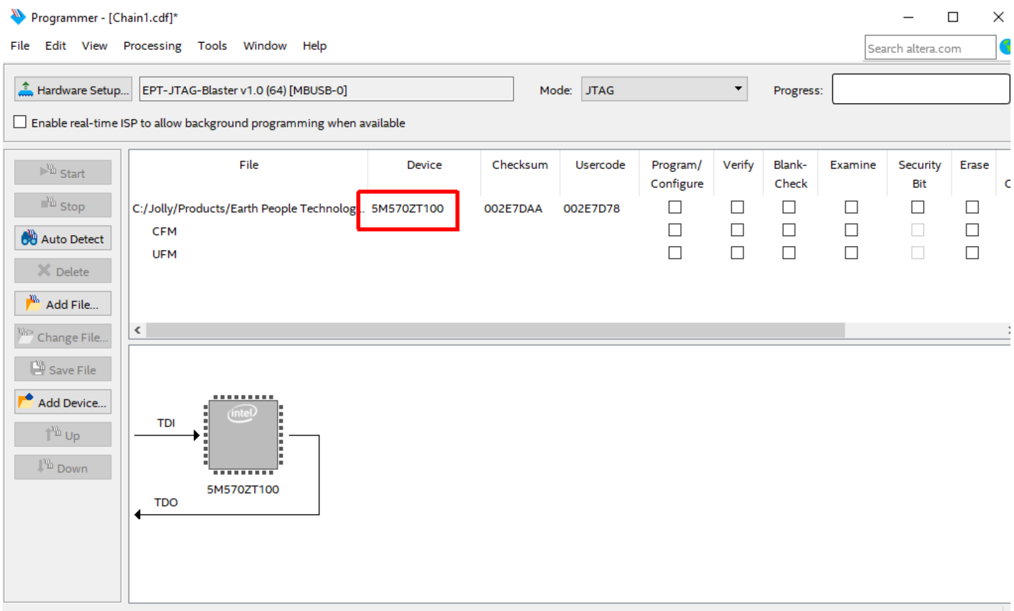
If you successfully double clicked, the “Currently selected hardware:” dropdown box will show the “EPT-Blaster v1.0 (64)”.



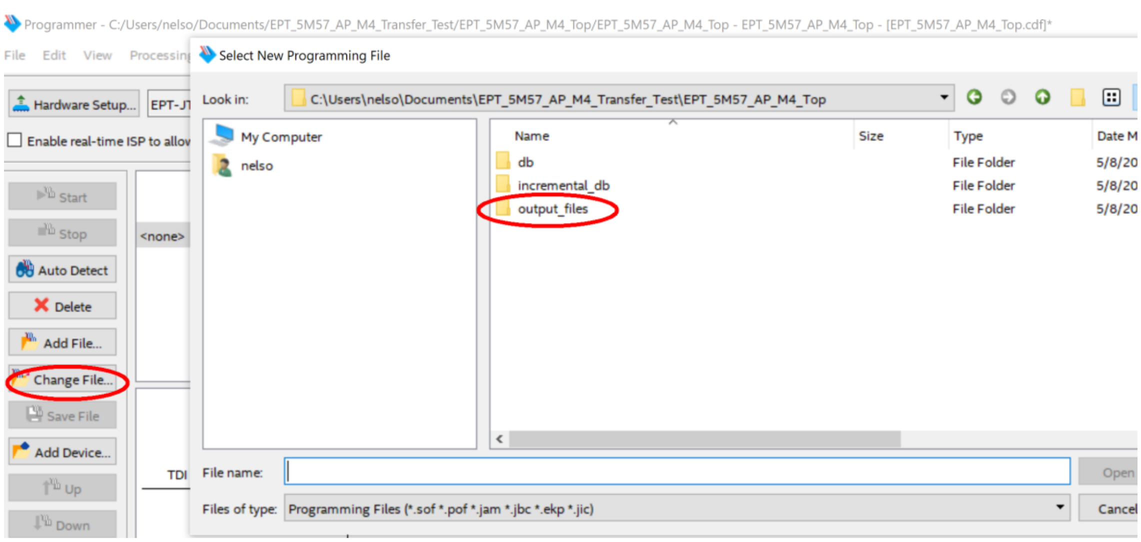
Click on the Auto-Detect button. This will verify that the EPT-Blaster driver can connect with the MegaMax device.



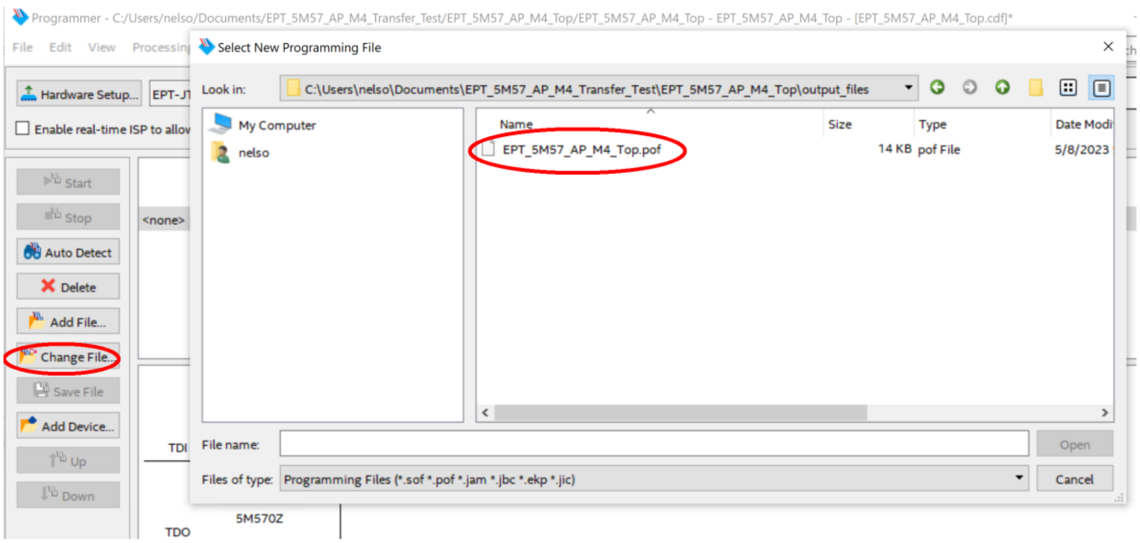
Select the 5M570 under “Device”.



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

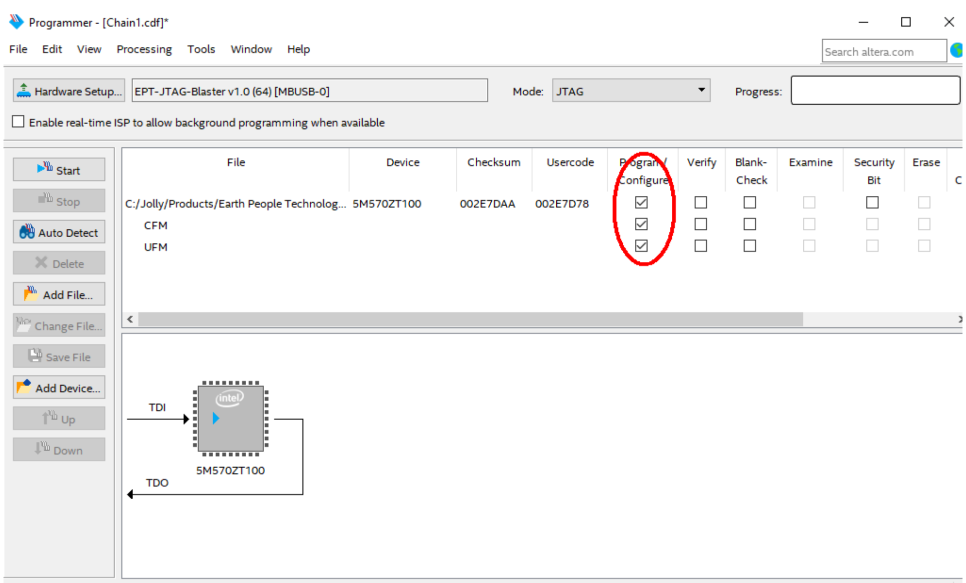


Click on the EPT_5M57_AP_M4_Top.pof file to select it.

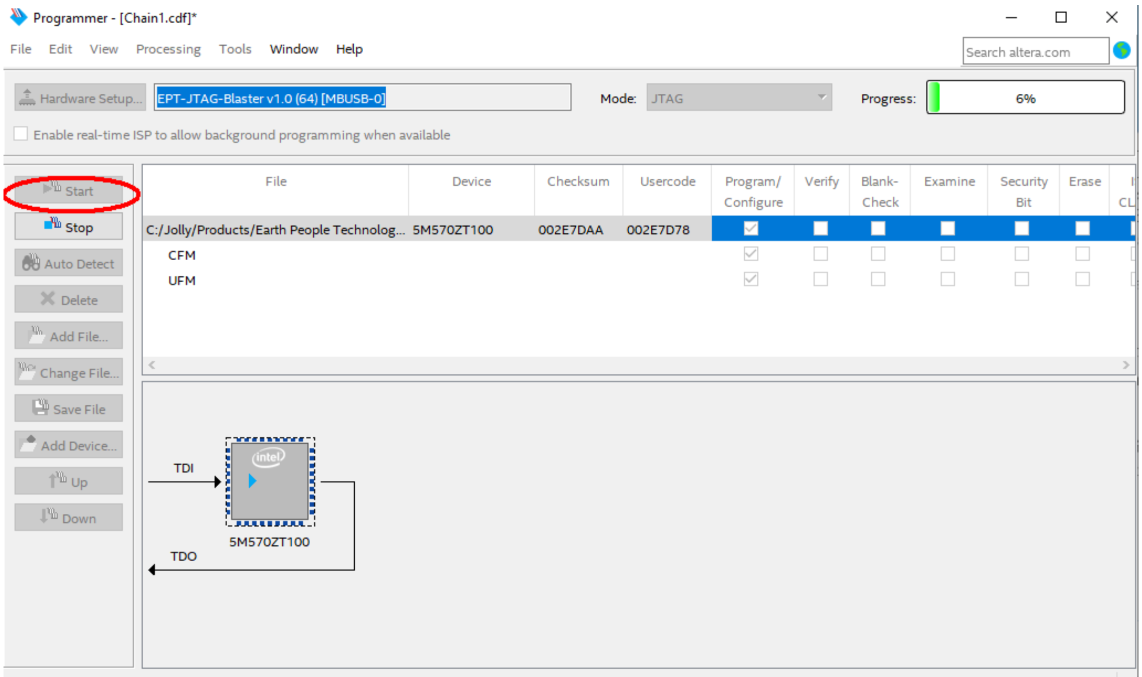


Click the Open button in the lower right corner.

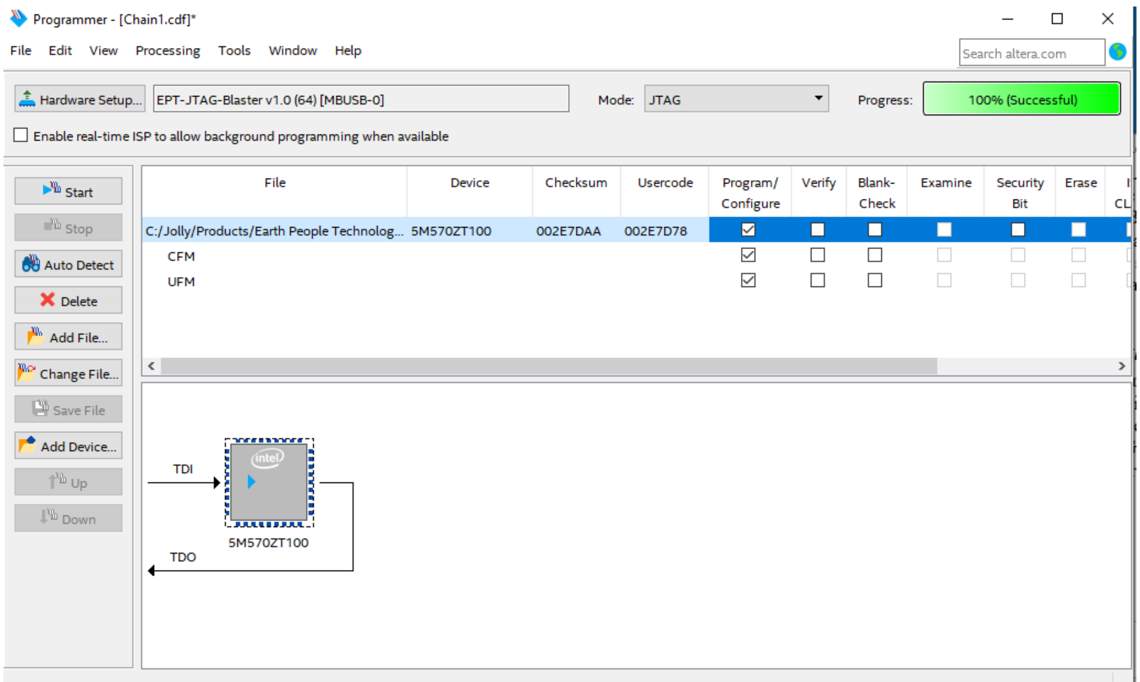
Next, selet the checkbox under the “Program/Configure” of the Programmer Tool. The checkboxes for the CFM and UFM will be selected automatically.



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



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



At this point, the MegaMax 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.

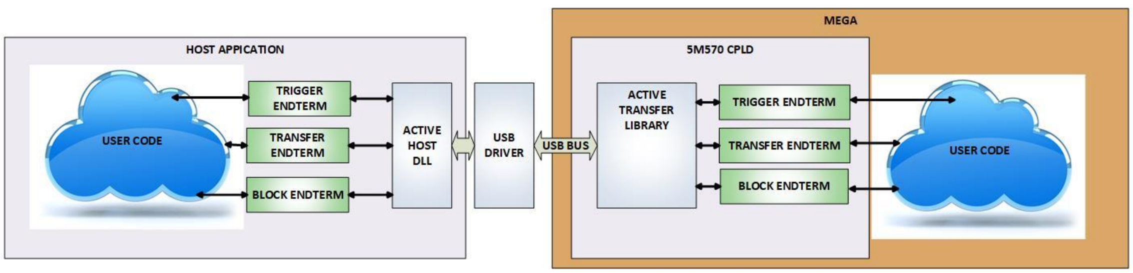
5 Active Host Application

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

- Trigger Endterm
- Transfer Endterm
- Block Endterm

User code writes to the Endterms as function calls. Just include the address of the individual module (there are eight individually addressable modules of each Endterm).

Immediately after writing to the selected Endterm, the value is received at the HDL Endterm in the CPLD. The Trigger Endterms are used as “switches”. The user code can set a Trigger bit in the CPLD and cause an event to occur. The Transfer Endterm sends one byte to the CPLD. The Block Endterm sends a block of bytes. By using one of the Active Host Endterms, the user can create a dynamic, bi-directional, and configurable data transfer design.



5.1 *Trigger EndTerm*

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

5.2 *Transfer(Byte) EndTerm*

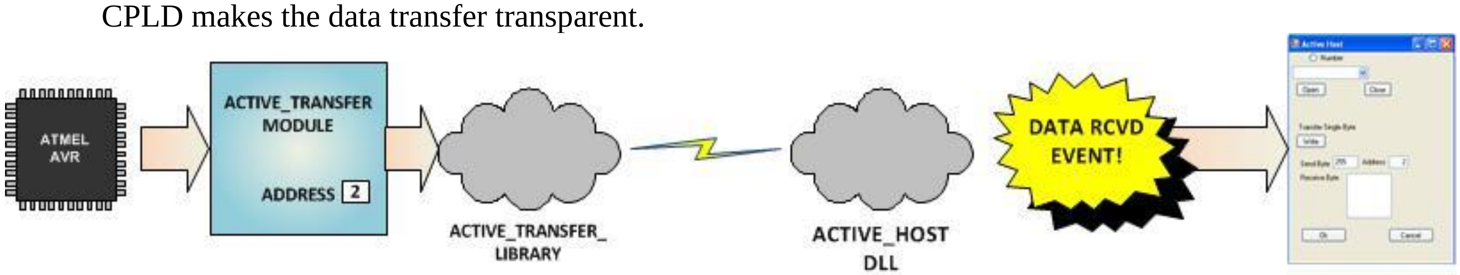
The Transfer EndTerm is a software component that provides a direct path from the users application to the commensurate Transfer EndTerm in the CPLD. It is used to transfer a byte to and from the CPLD. Eight separate Transfer EndTerm modules can be instantiated in the CPLD. Each module is addressed by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

5.3 *Block EndTerm*

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

5.4 *Active Host DLL*

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



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

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

- EPT_AH_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.

```
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();
```

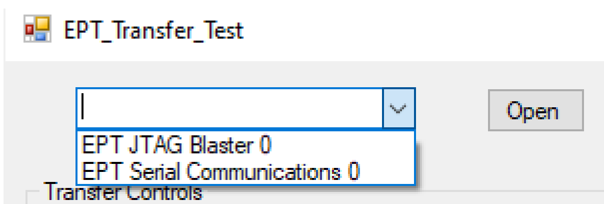
to determine the Serial devices attached to the PC. Next,

```
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.

```
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

- TRANSFER_IN
- BLOCK_IN

```
2 references
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;
```

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

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

```
public void Receive_Trigger_In(object sender, EventArgs e)
{
    switch (ept_data.Payload)
    {
        case 0x01:
            lLableSwitch1.Text = "Switch 1\n Pressed";
            break;
        case 0x02:
            lLableSwitch2.Text = "Switch 2\n Pressed";
            break;
        case 0x04:
            lLableSwitch1.Text = "";
            lLableSwitch2.Text = "";
            break;
    }
}
```

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

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 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 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 it’s transmit FIFO. If data exists, the Host will command the Device to send it’s data. The received data is then stored into local memory and register bits are set that will indicate data has been received from a particular address.

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

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

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

```
public void TransferOutReceive()
{
    string WriteRcvChar = "";
    WriteRcvChar = String.Format("{0}", (int)EPTReceiveData.Payload);
    tbDataBytes.AppendText(WriteRcvChar + ' ');
    tbAddress.Text = String.Format("{0:x2}", (uint)System.Convert.ToUInt32(EPTReceiveData.Address.ToString()
}
```

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

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.

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 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.

```
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

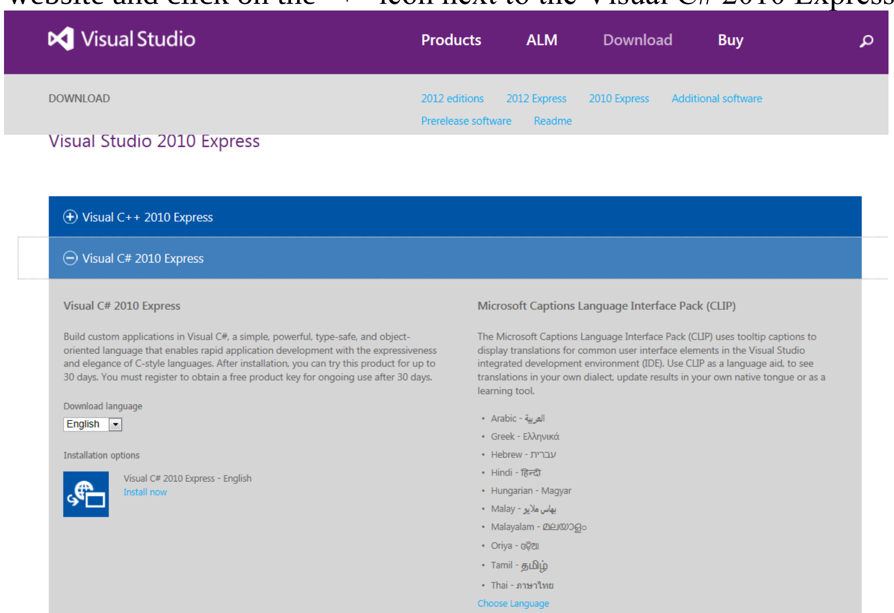
The Active Host Application DLL is used to build a custom standalone executable on the PC that can perform Triggers and Transfer data to/from the MegaMax. A standalone project can range from a simple program to display and send data from the user to/from the MegaMax. Or it can more complex to include receiving data, processing it, and start or end a process on the board. 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.

6.1 *Creating a Project*

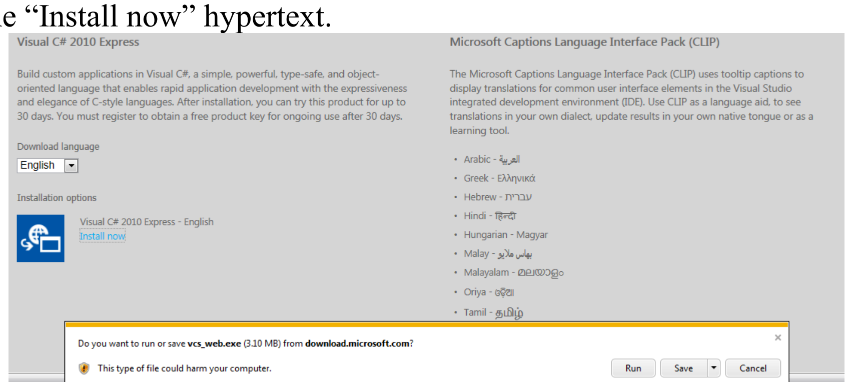
Download the latest version of Microsoft Visual C# 2010 Express environment from Microsoft. It's a free download.

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.

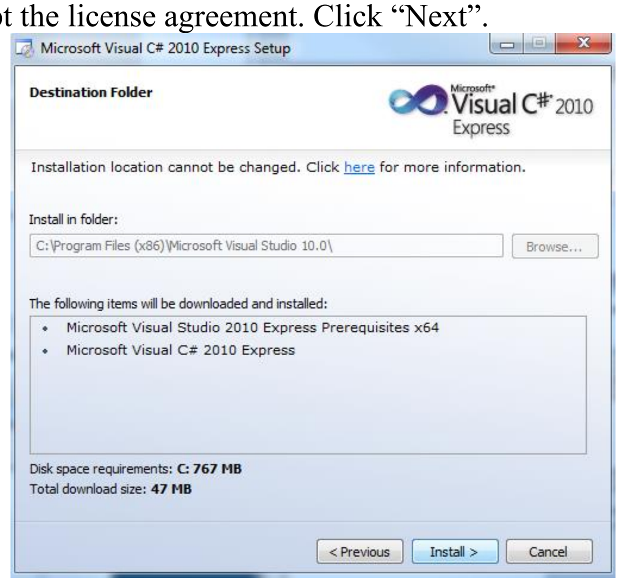


Click on the "Install now" hypertext.

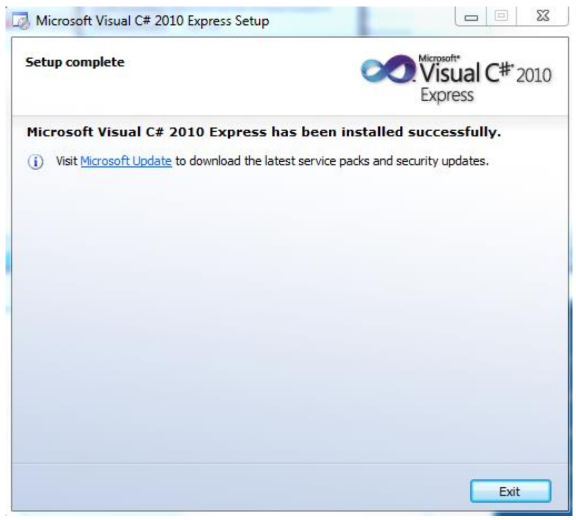


Click the "Run" button.

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

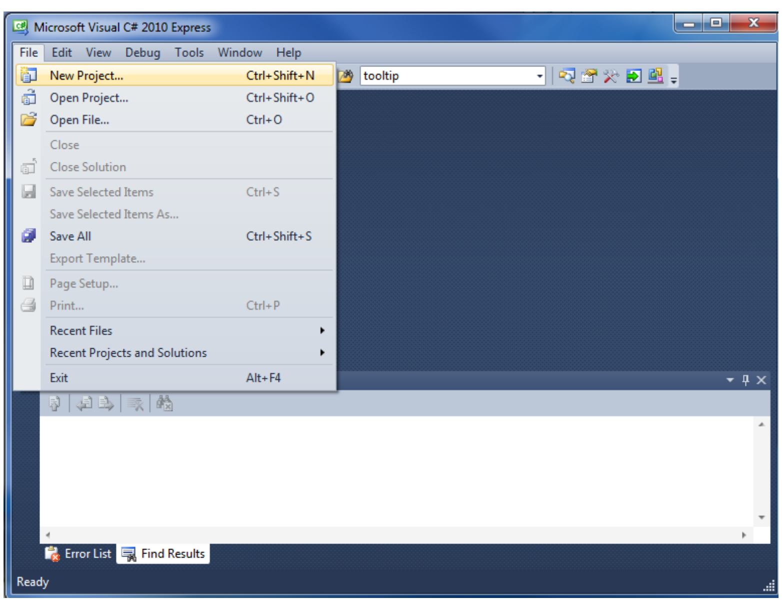


Visual C# 2010 Express will install. This may take up to twenty minutes depending on your internet connection.

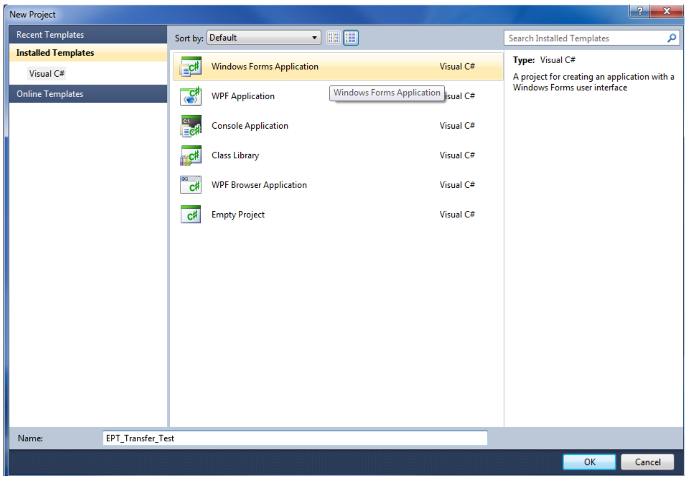


The installed successfully window will be displayed when Visual C# Express is ready to use.

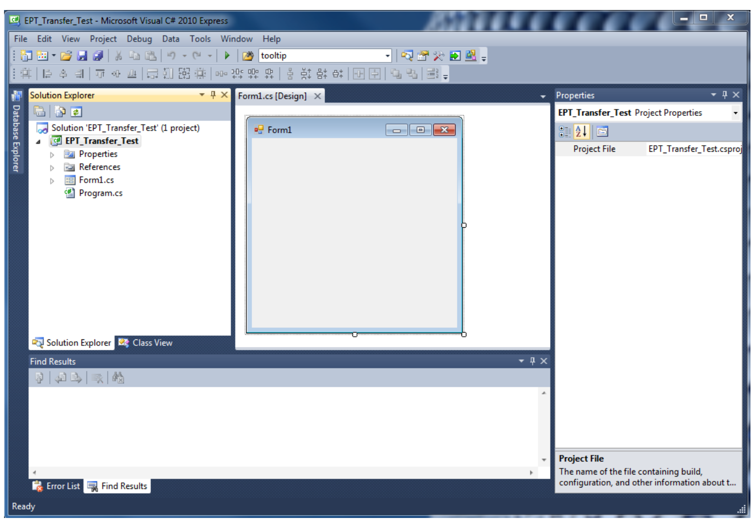
Once the application is installed, open it up. Click on File->New Project.



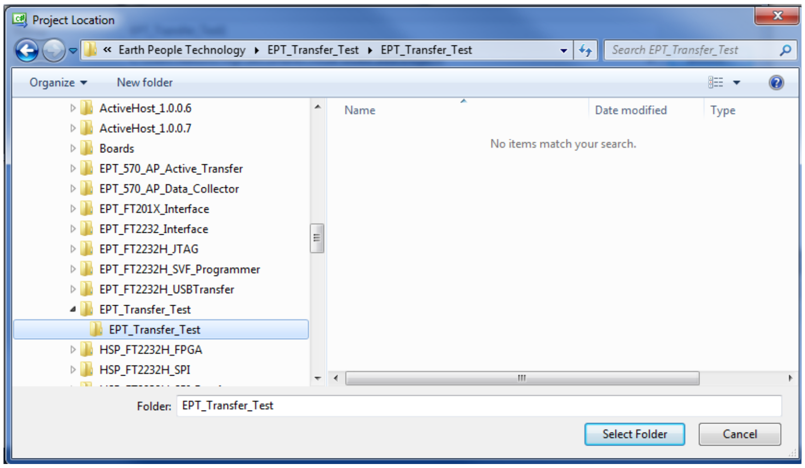
At the New Project window, select the Windows Forms Application. Then, at the Name: box, type in EPT_Transfer_Test



The project creation is complete.

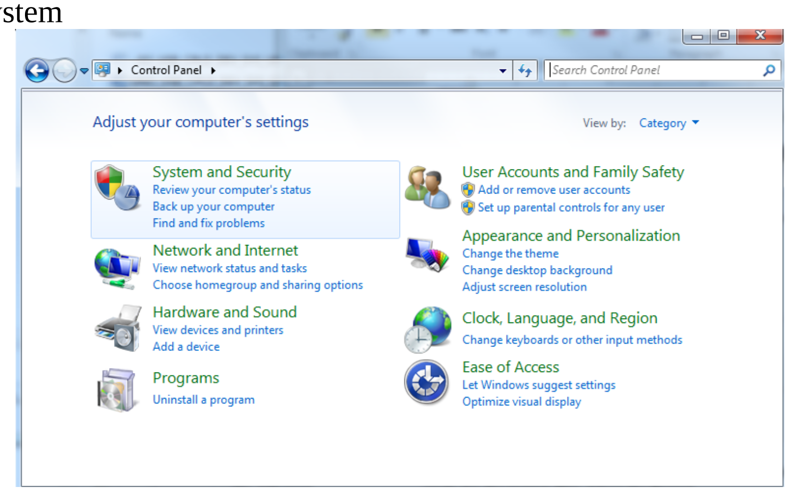


Save the project, go to File->Save as, browse to a folder to create EPT_Transfer_Test folder. The default location is c:\Users\<Users Name>\documents\visual studio 2010\Projects.

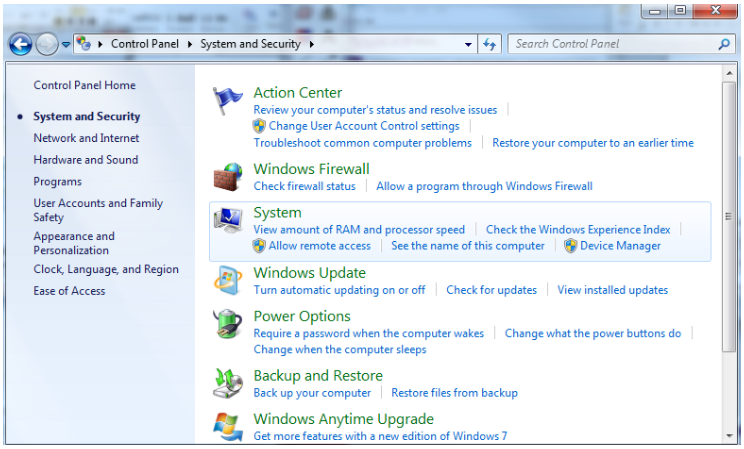


6.1.1 Setting up the C# Express Environment for x64 bit

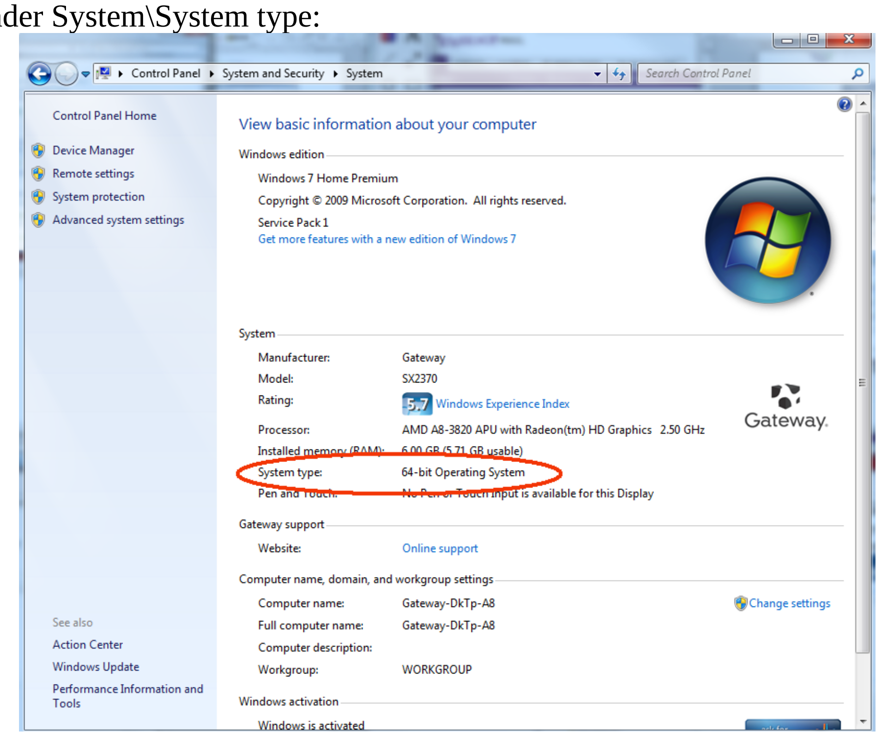
The project environment must be set up correctly in order to produce an application that runs correctly on the target platform. If your system supports 64 bit operation, perform the following steps. Otherwise if your system is 32 bit skip to the Section, Assembling Files into the Project. Visual C# Express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



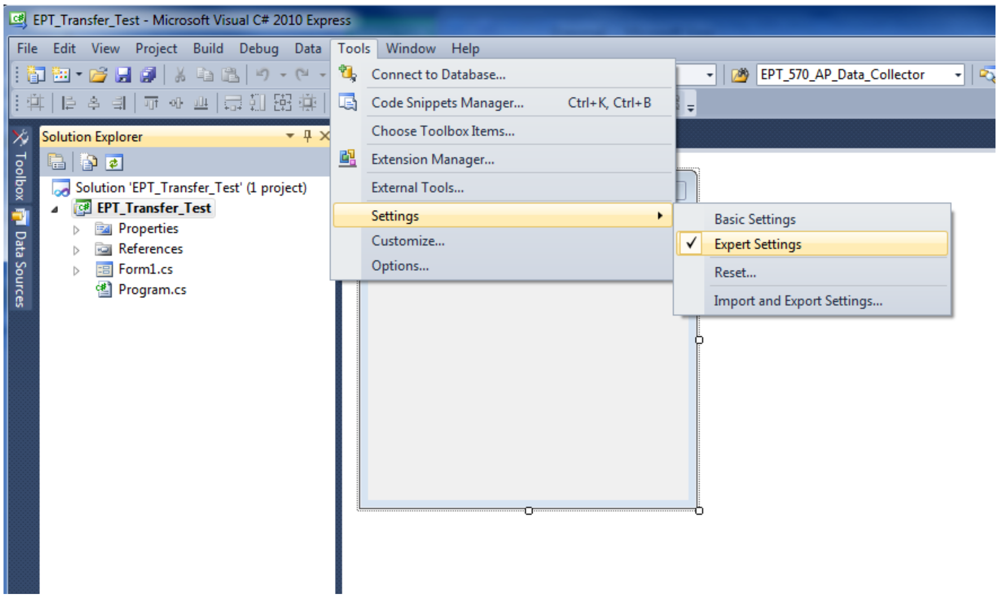
Click on System.



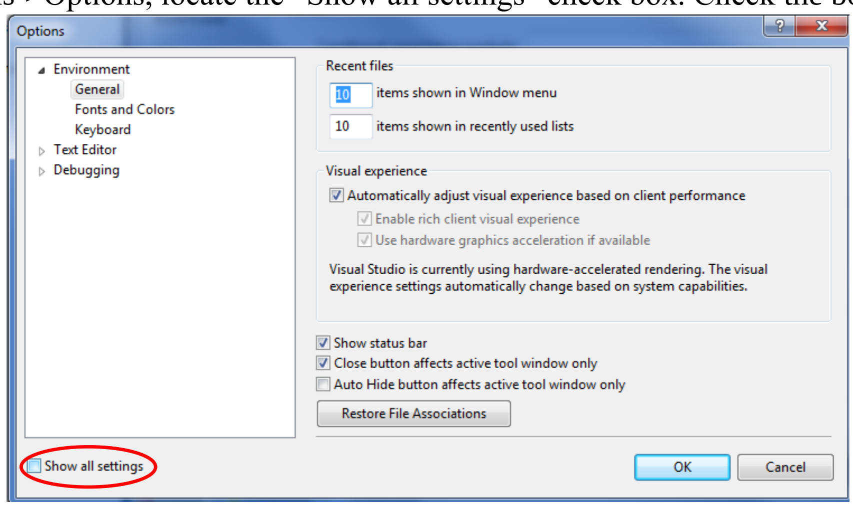
Check under System\System type:



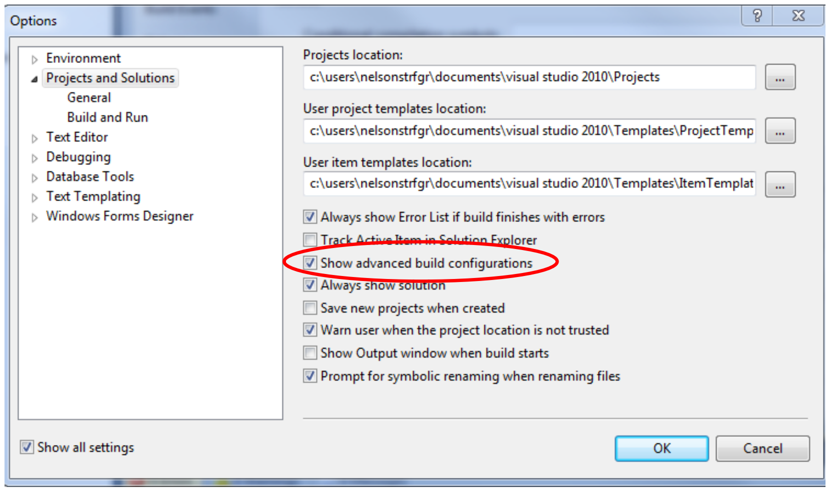
First, we need tell C# Express to produce 64 bit code if we are running on a x64 platform. Go to Tools->Settings and select Expert Settings



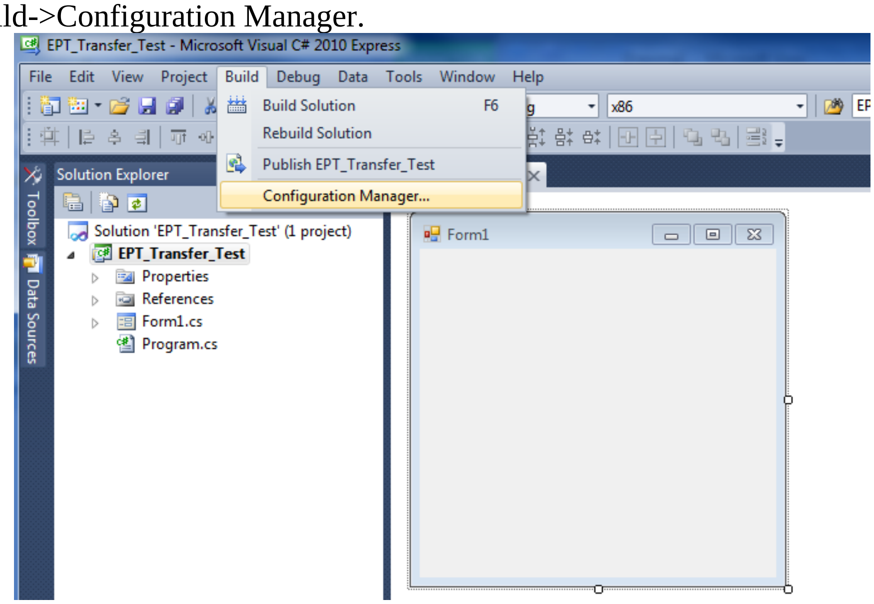
Go to Tools->Options, locate the “Show all settings” check box. Check the box.



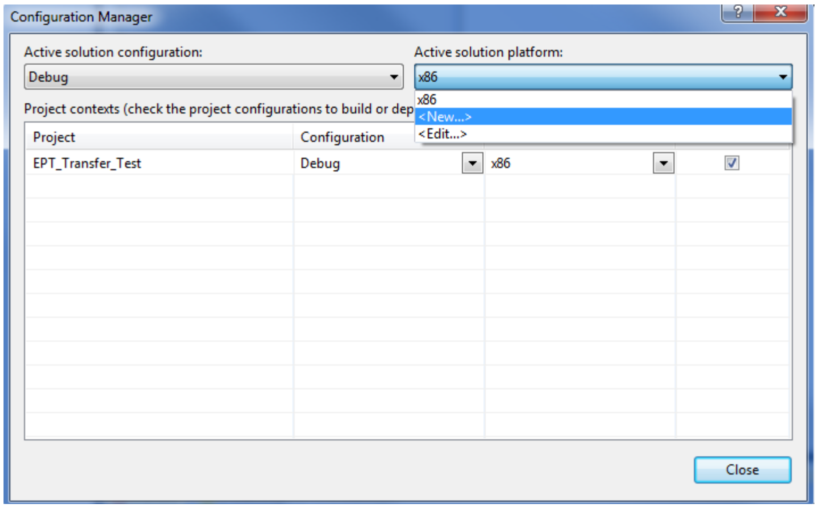
In the window on the left, go to “Projects and Solutions”. Locate the “Show advanced build configurations” check box. Check the box.



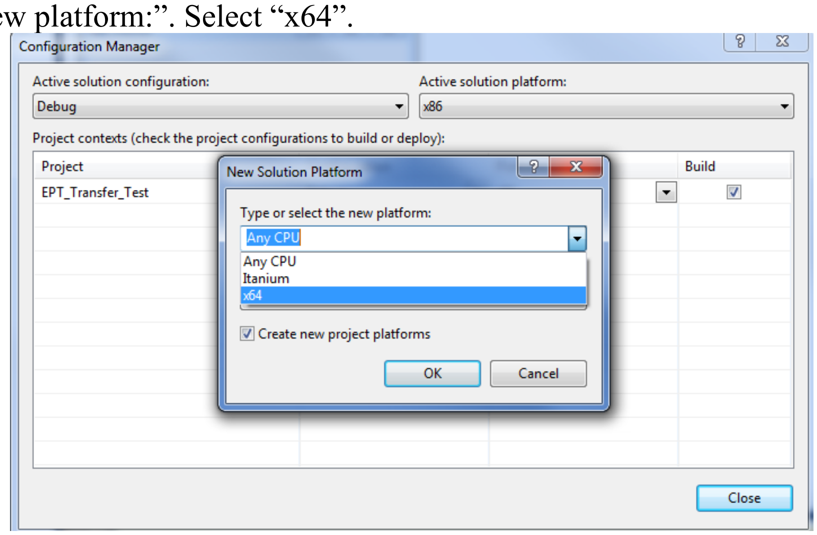
Go to Build->Configuration Manager.



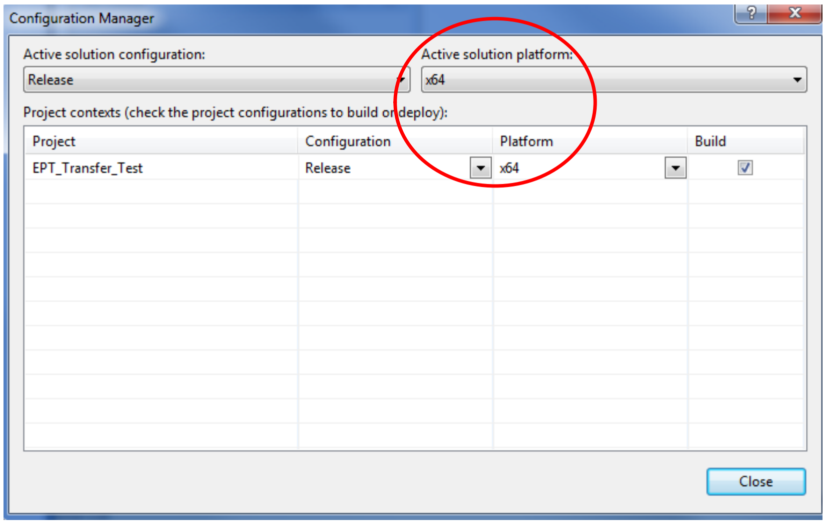
In the Configuration Manager window, locate the “Active solution platform:” label, select “New” from the drop down box.



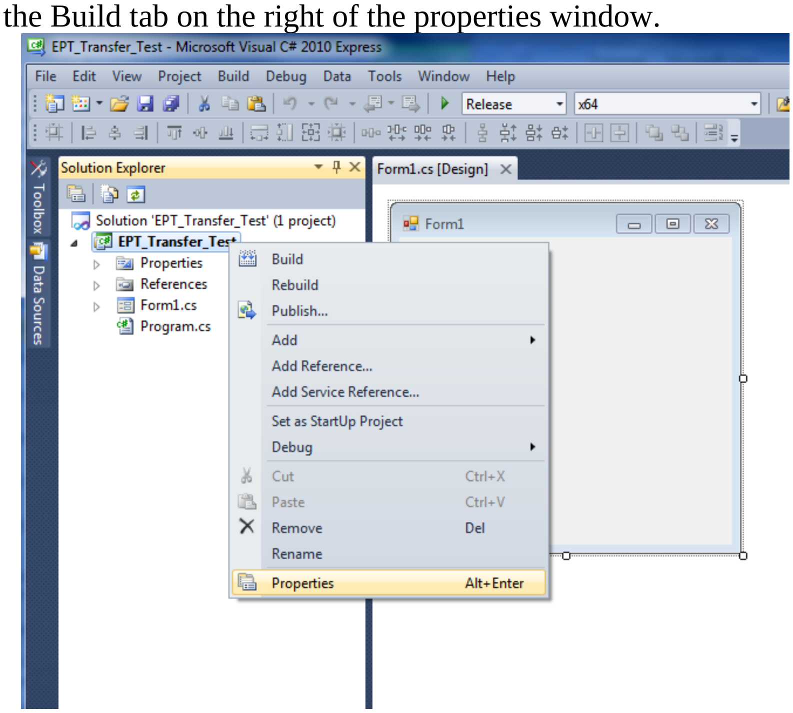
In the New Solution Platform window, click on the drop down box under "Type or select the new platform:". Select "x64".



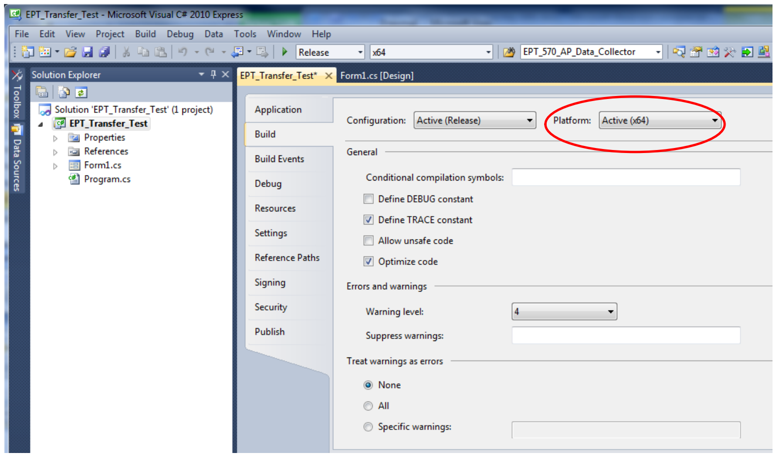
Click the Ok button. Verify that the "Active Solution Platform" and the "Platform" tab are both showing "x64".



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



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

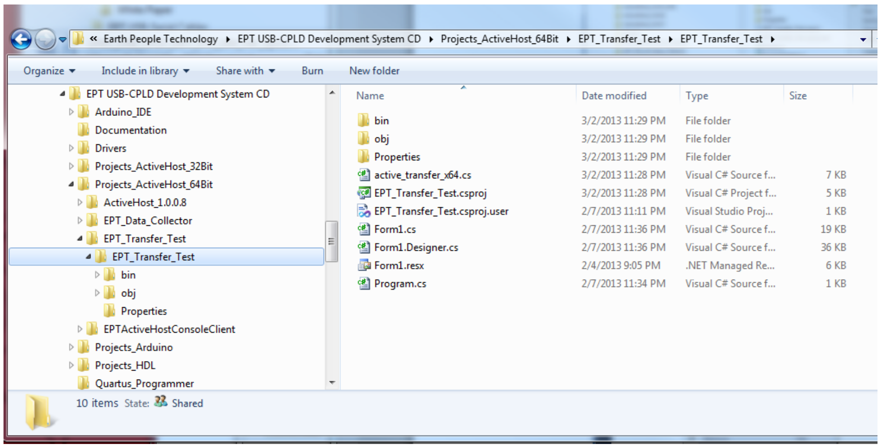


Click on the Save All button on the tool bar. The project environment is now setup and ready for the project files. Close the Project.

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

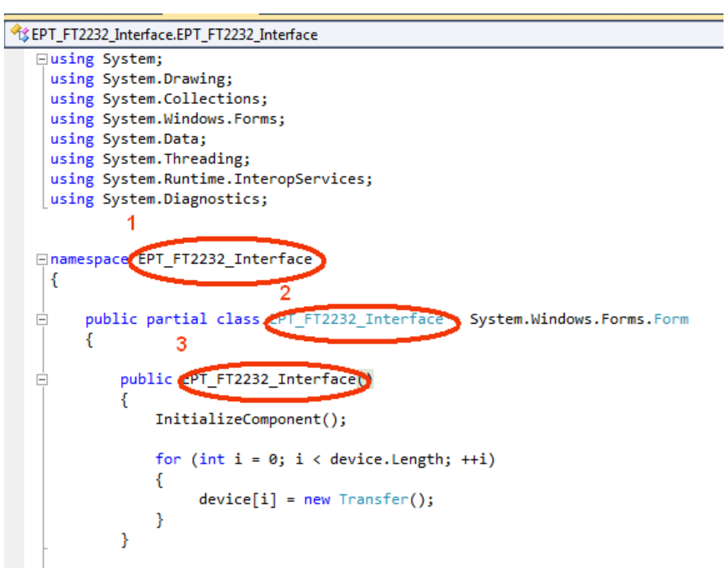


6.2.1 Changing Project Name

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 chages can be made by modifying the following:

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



4. Change EPT_Transfer_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 clase of Form1.Designer.cs to new project name.

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

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

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

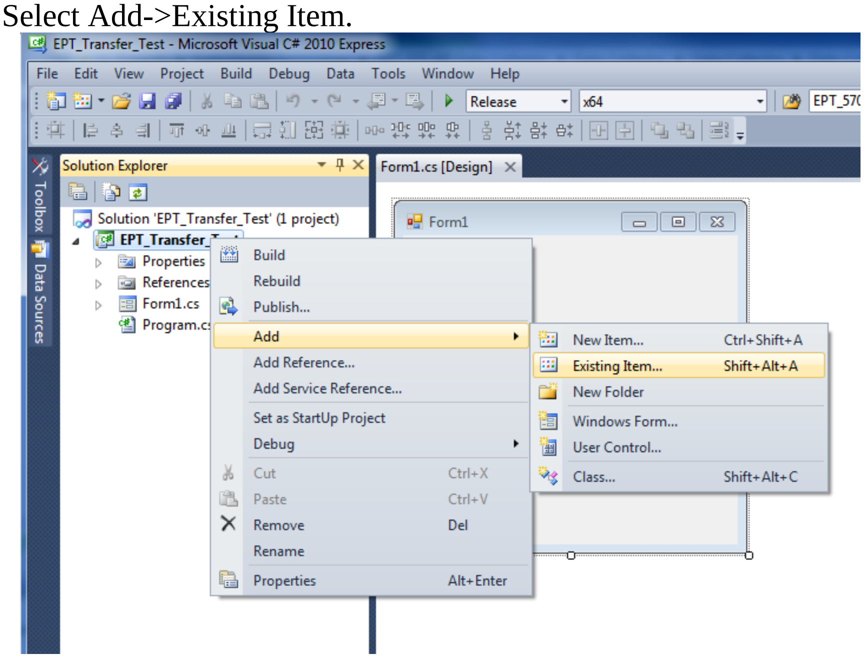
```
this.Controls.Add(this.btnTrigger3);
this.Controls.Add(this.btnTrigger2);
this.Controls.Add(this.btnTrigger1);
this.Controls.Add(this.btnCloseDevice);
this.Controls.Add(this.btnOpenDevice);
this.Controls.Add(this.cmbDevList);
this.Controls.Add(this.LEDBox);
this.Controls.Add(this.gbTriggerOut);
this.Controls.Add(this.gbTransferControl);
this.Controls.Add(this.groupBox1);
this.Name = "EPT_FT2232_Interface";
this.Text = "EPT_FT2232_Interface";
this.Load += new System.EventHandler(this.EPT_FT2232_Interface_Load);
this.LEDBox.ResumeLayout(false);
this.LEDBox.PerformLayout();
this.gbTriggerOut.ResumeLayout(false);
this.gbTransferControl.ResumeLayout(false);
this.gbTransferControl.PerformLayout();
this.groupBox1.ResumeLayout(false);
this.groupBox1.PerformLayout();
this.ResumeLayout(false);
this.PerformLayout();
```

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

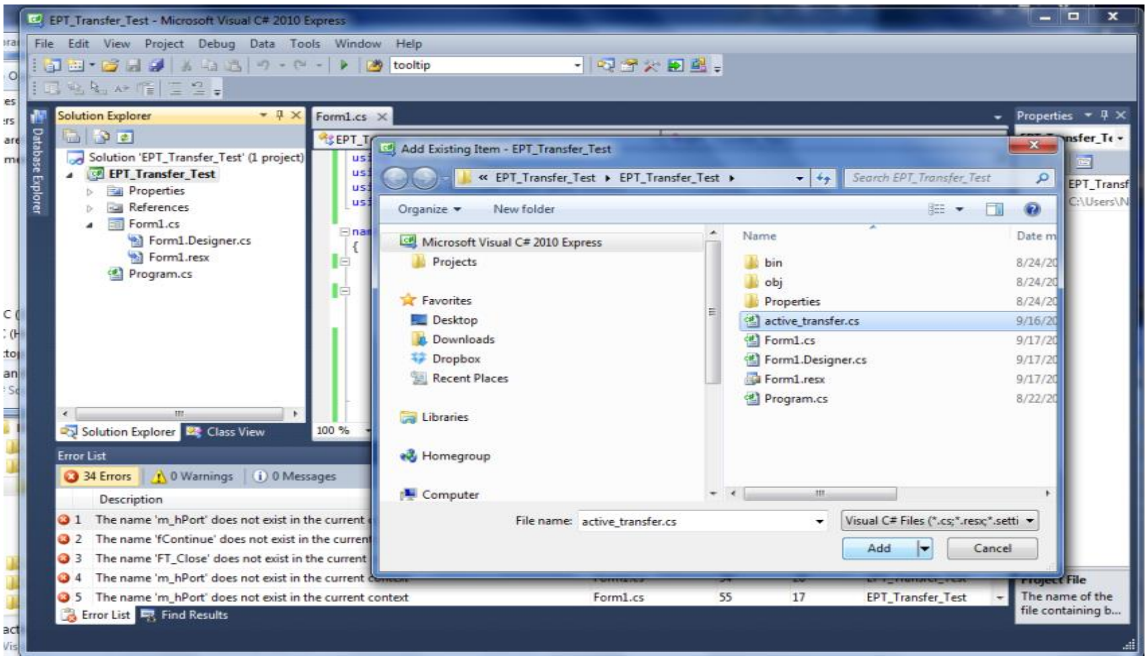
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;

namespace EPT_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

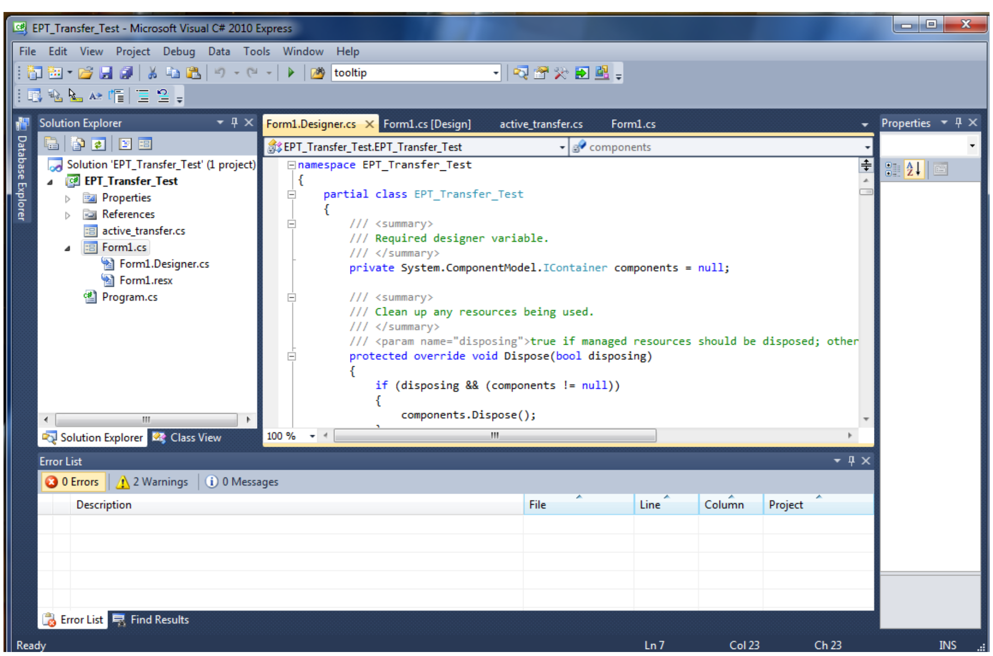
Open the EPT_Transfer_Test project. Right click on the project in the Solutions Explorer. Select Add->Existing Item.



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.



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

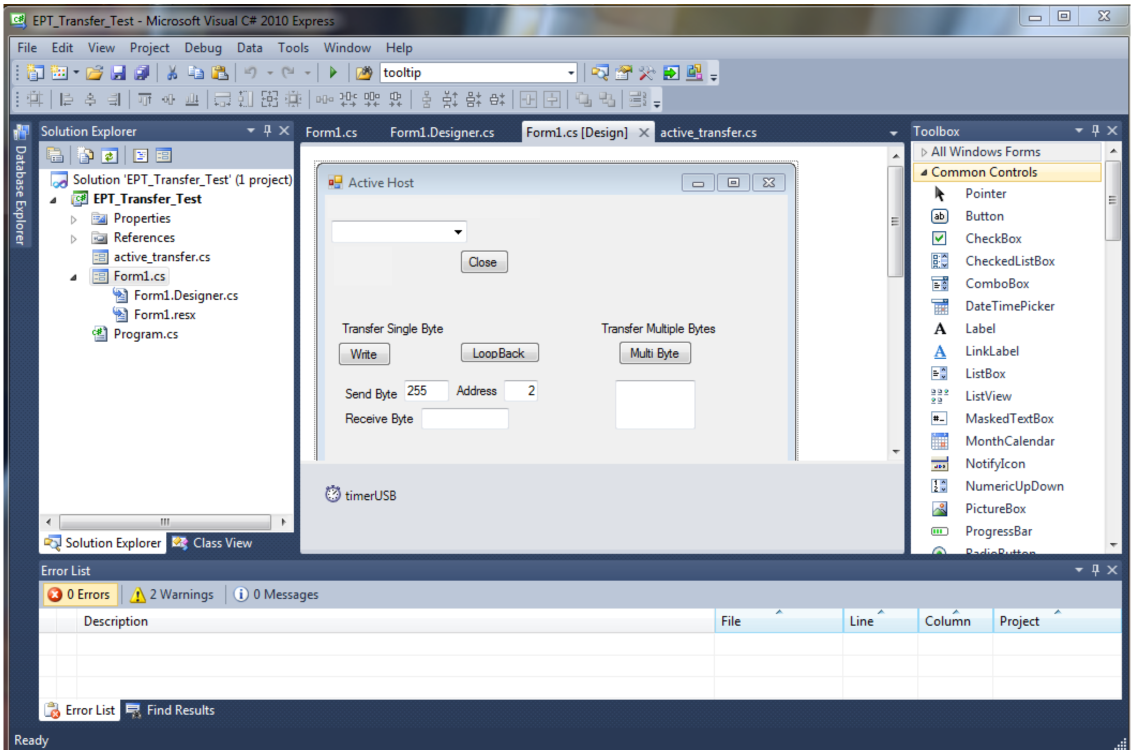


6.2.3 Adding Controls to the Project

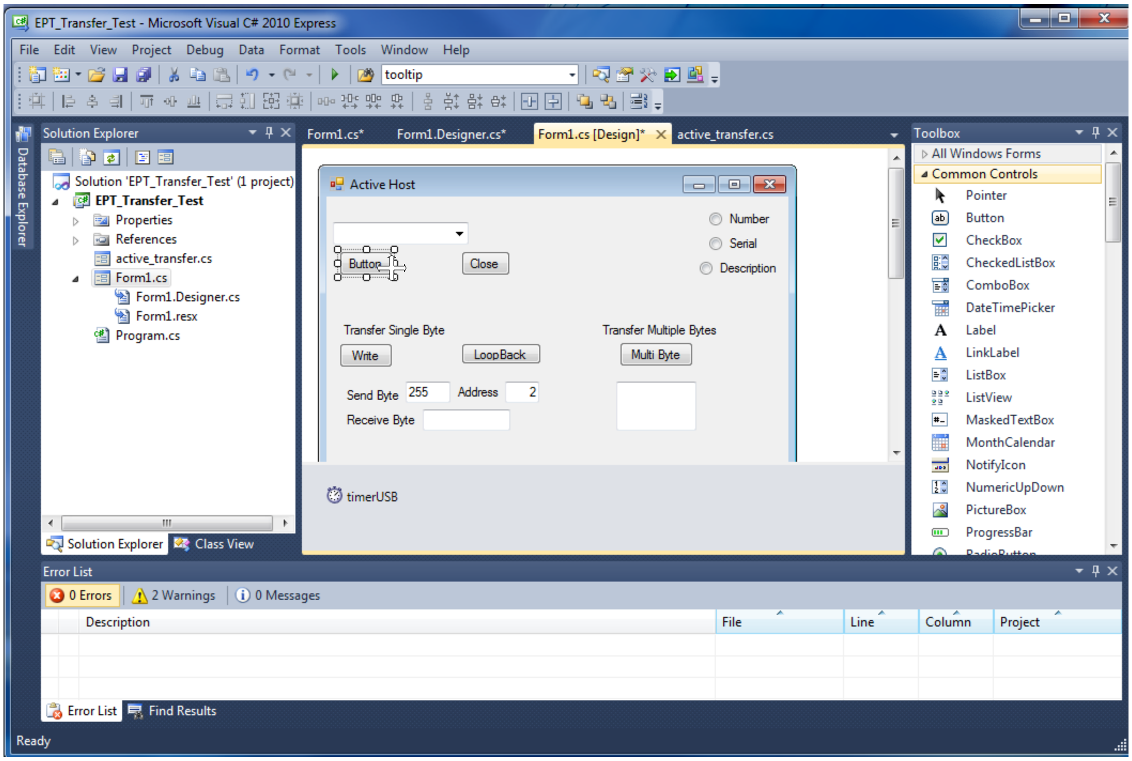
Although, the C# language is very similar to C Code, there are a few major differences. The first is C# .NET environment is event based. A second is C# utilizes classes. This guide will keep the details of these items hidden to keep things simple. However, a brief introduction to events and classes will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

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

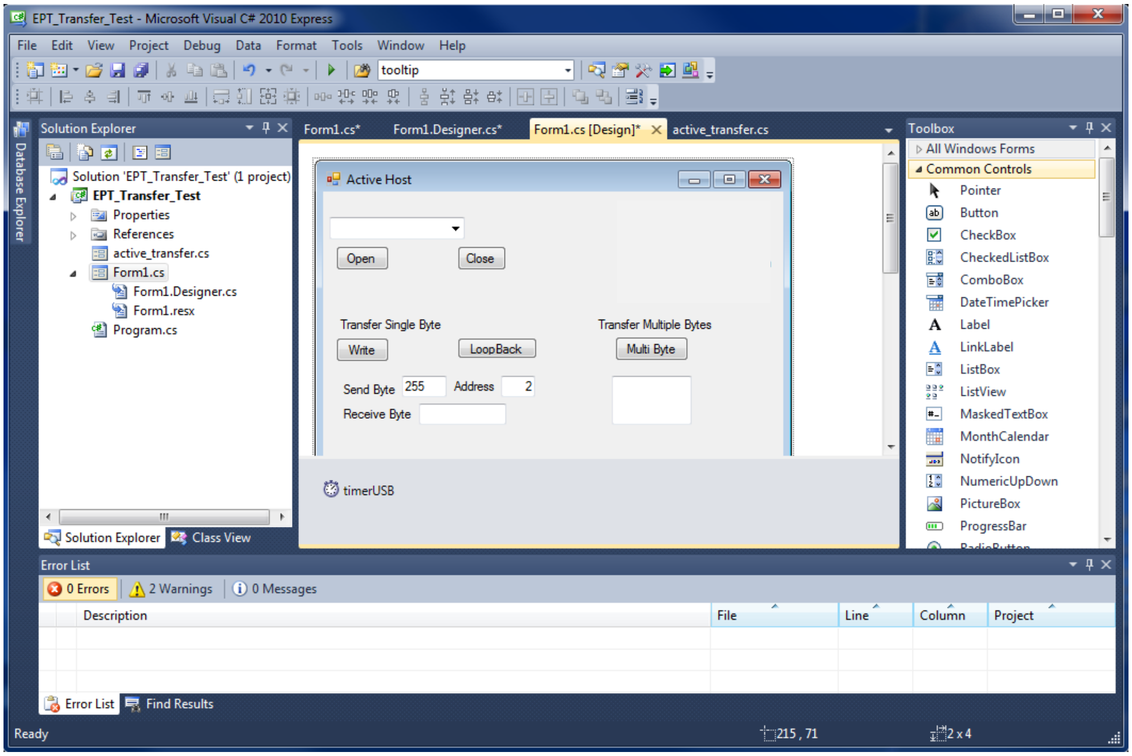
The easiest way to add a button to a form is to double click the Form1.cs in the Solution Explorer. Click on the  button to launch the Toolbox.



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



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



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

Private void btnOpenDevice_click(object sender, System.EventArgs e)

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

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

This code sets up the button, size, placement, and text. It also declares the "System.EventHandler()". This statement sets the click method (which is a member of the button class) of the btnOpenDevice button to call the EventHandler – btnOpenDevice_Click. This is where the magic of the button click event happens.

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

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

When btnOpenDevice_Click is called, it calls the function "OpenDevice()". This function is defined in the dll and will connect to the device selected in the combo box. This is a quick view of how to create, add files, and add controls to a C# project. The user is encouraged to spend some time reviewing the online tutorial at http://www.homeandlearn.co.uk/csharp/csharp.html to become intimately familiar with Visual C# .NET programming. In the meantime, follow the examples from the Earth

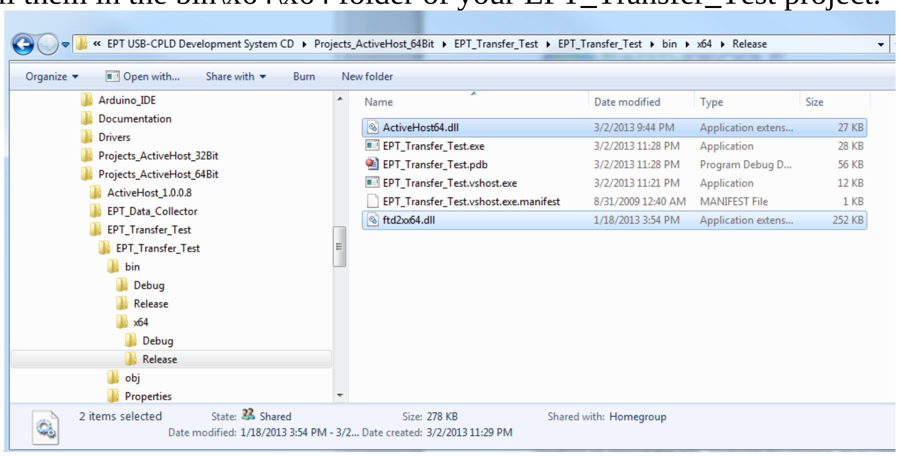
People Technology to perform some simple reads and writes to the EPT USB-CPLD Development System.

6.2.4 Adding the DLL's to the Project

Locate the EPT USB-CPLD Development System CD installed on your PC. Browse to the Projects_ActiveHost Open the Bin folder, copy the following files:

- ActiveHostXX.dll
- ftd2xxXX.dll

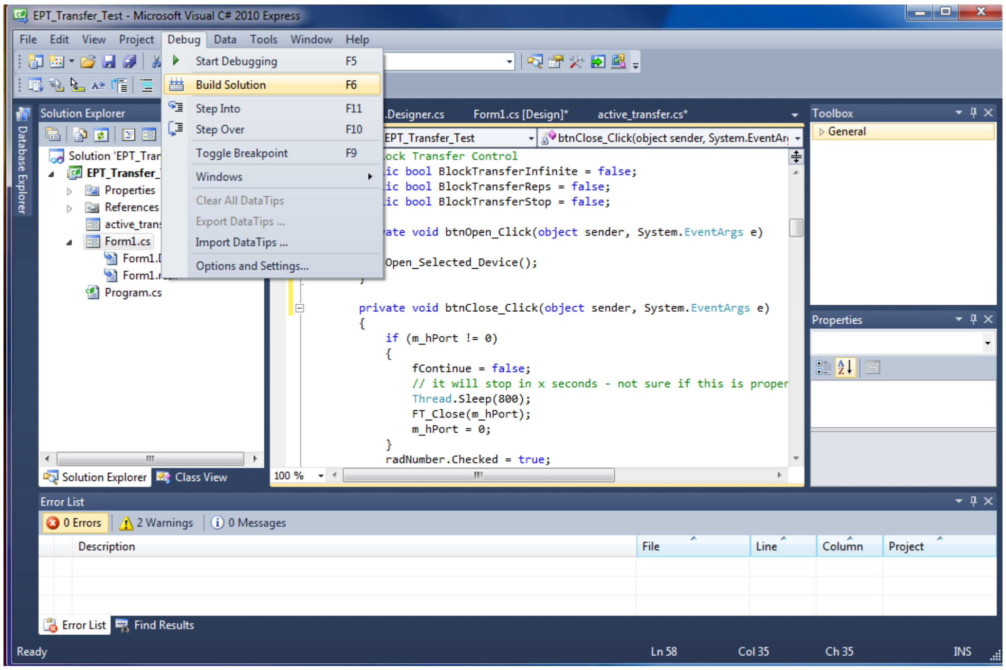
and install them in the bin\x64\x64 folder of your EPT_Transfer_Test project.



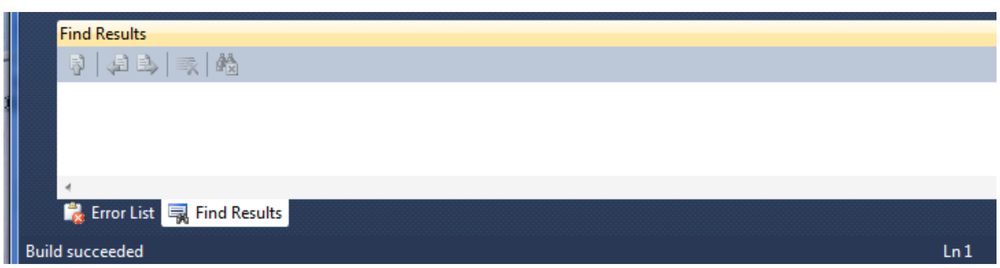
Save the project.

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.

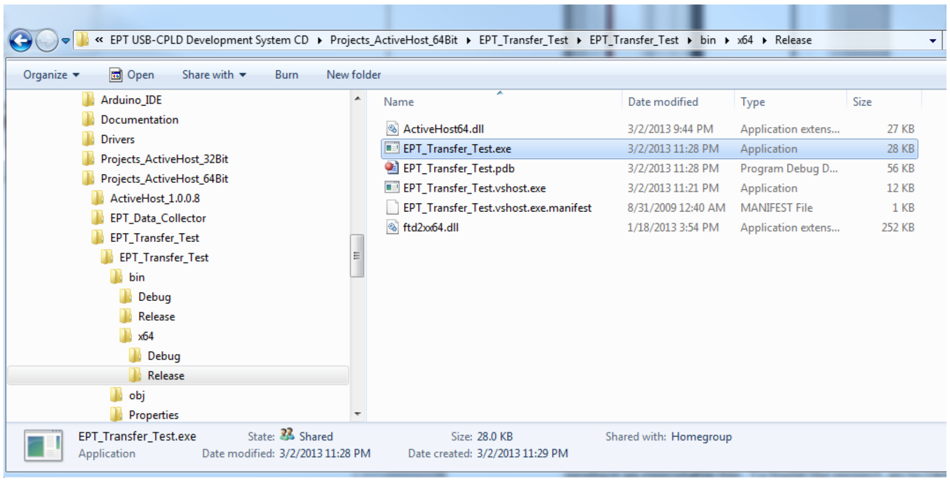


The C# Express compiler will start the building process. If there are no errors with code syntax, function usage, or linking, then the environment responds with “Build Succeeded”.

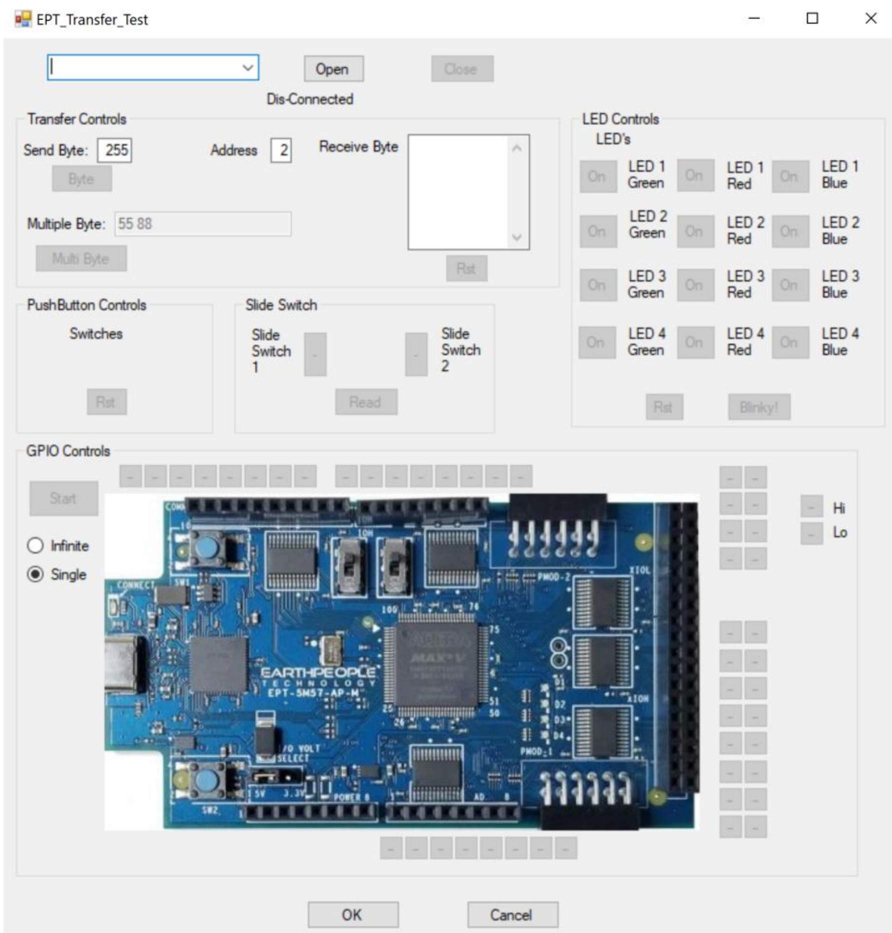


6.2.6 Testing the Project

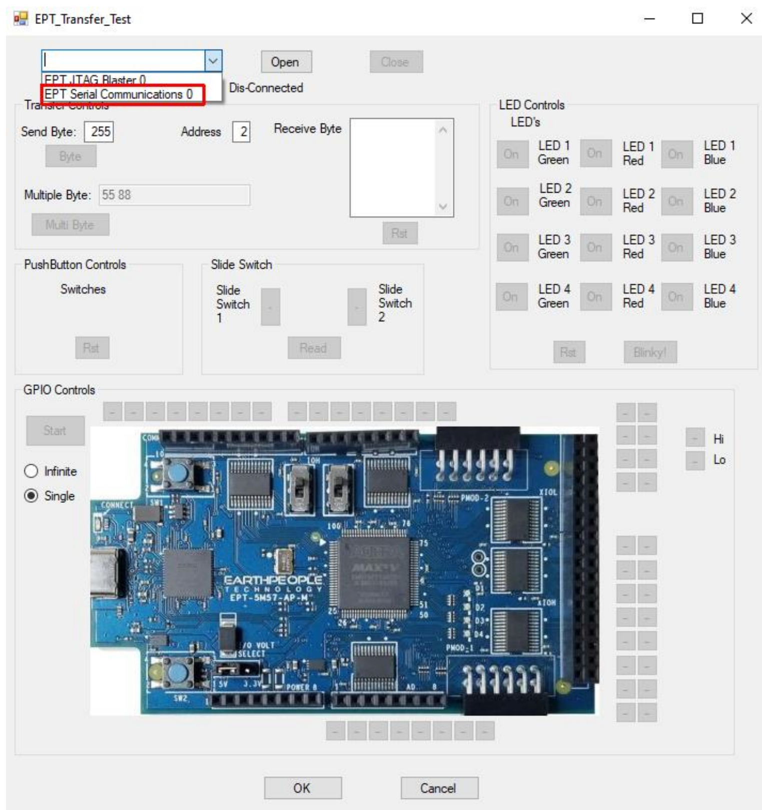
Once the project has been successfully built, it produces an *.exe file. The file will be saved in the Release or Debug folders.



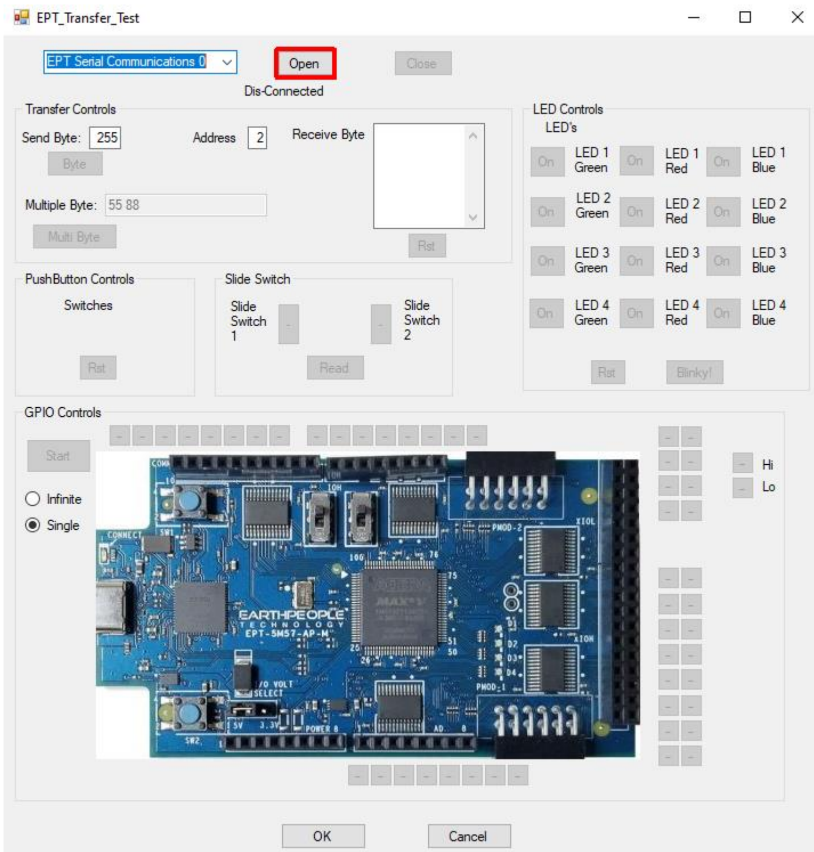
The EPT_Transfer_Text.exe file can now be tested using the MegaMax board. To test the file, connect the MegaMax to the Windows PC using Type A to USB-C cable. Make sure the driver for the board loads. If the USB driver fails to load, the Windows OS will indicate that no driver was loaded for the device.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



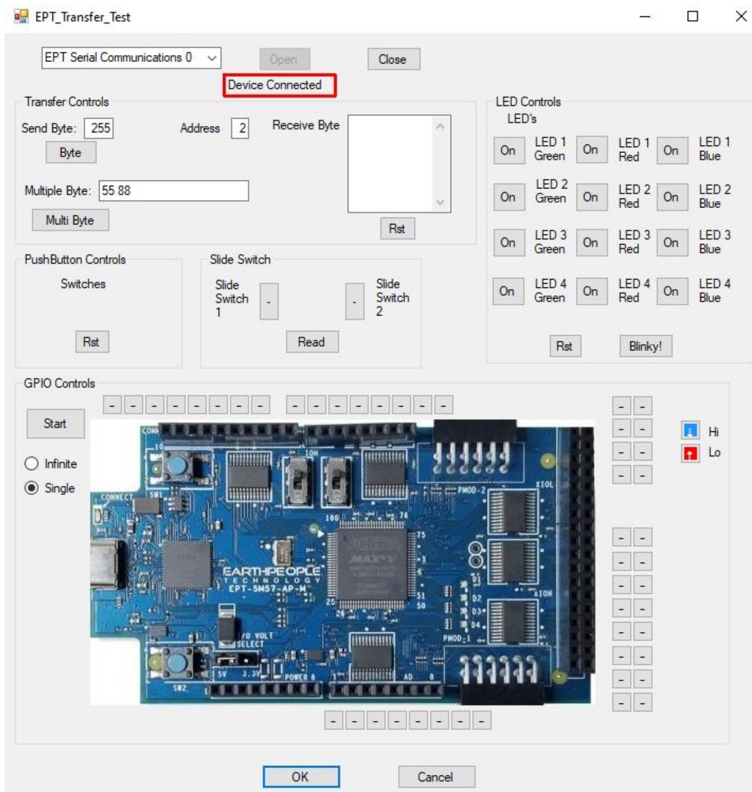
With the application loaded, select the EPT Serial Communications x board from the dropdown combo box.



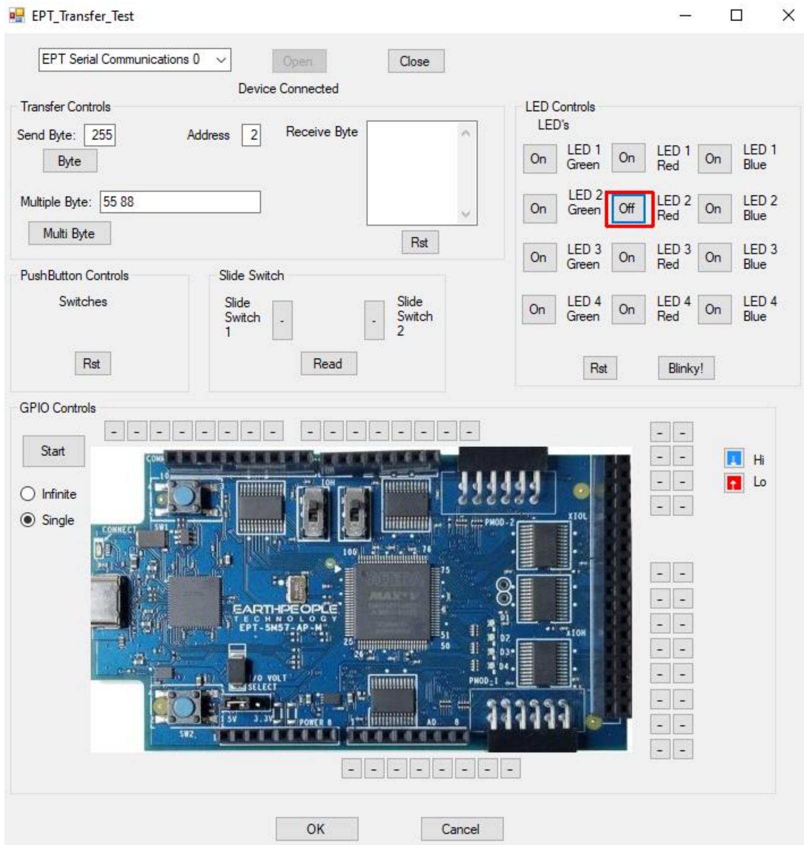
Next, click on the Open button.

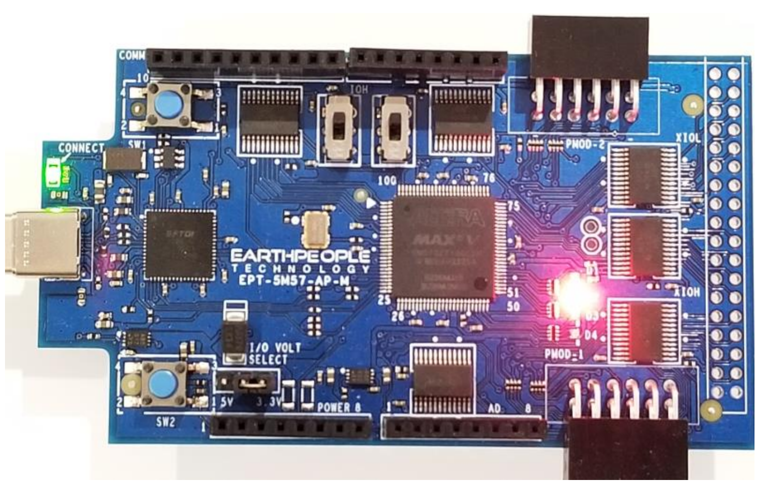


The “Device Connected” label should be seen to indicate the Windows App is now connected to the MegaMax.

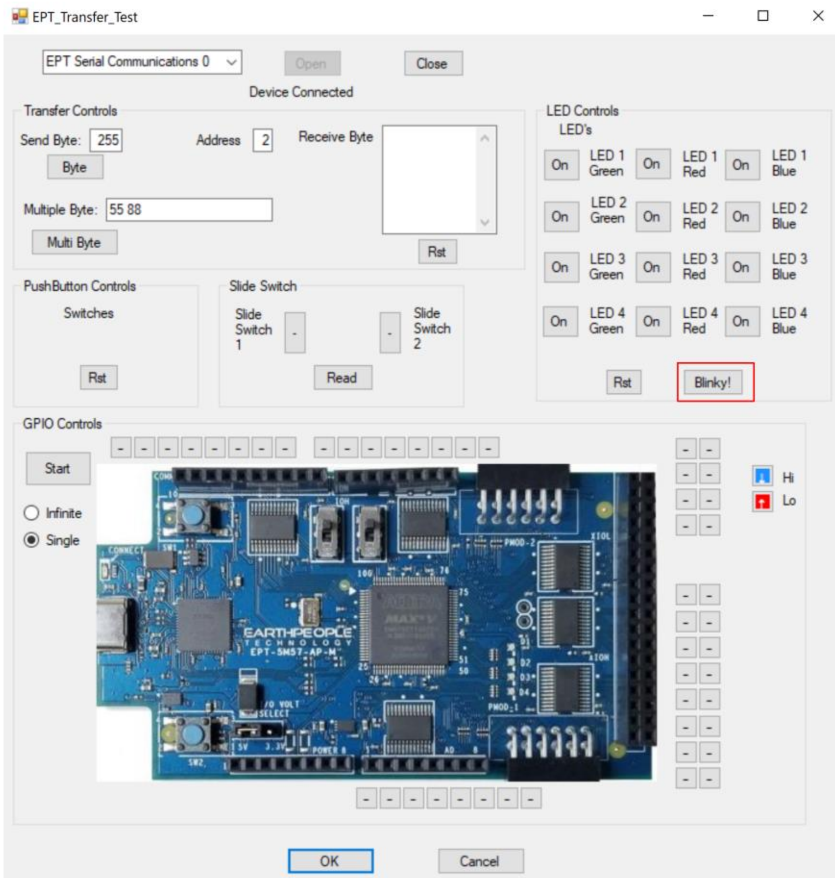


Click on one of the LED buttons in the middle of the window. The corresponding LED on the MegaMax board should light up.

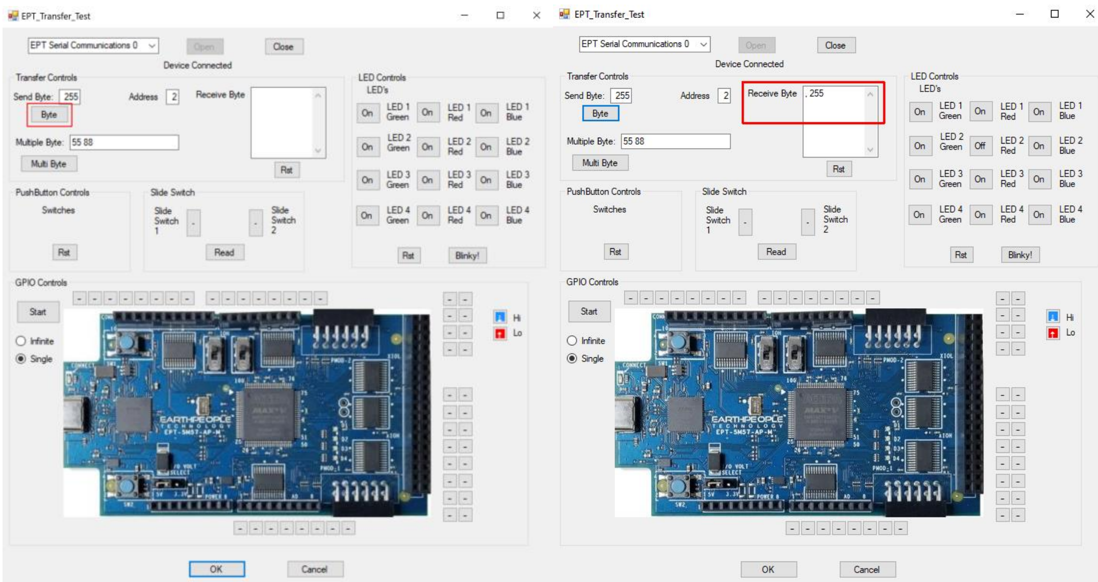




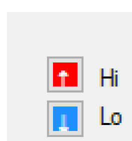
Clink on the Blinky button for a light show.

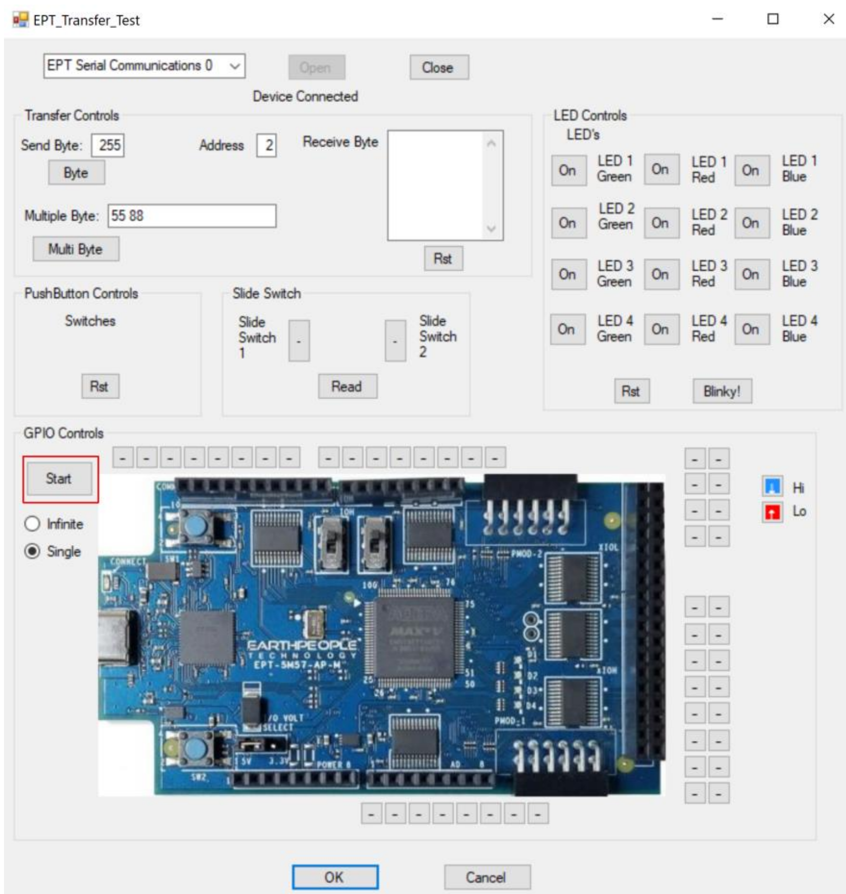


To exercise the Single Byte Transfer EndTerm, click the “Byte” button in the Transfer Controls group. Type in several numbers separated by a space and less 256 into the Multiple Byte textbox. Then hit the Multi Byte button. The numbers appear in the Receive Byte textbox.

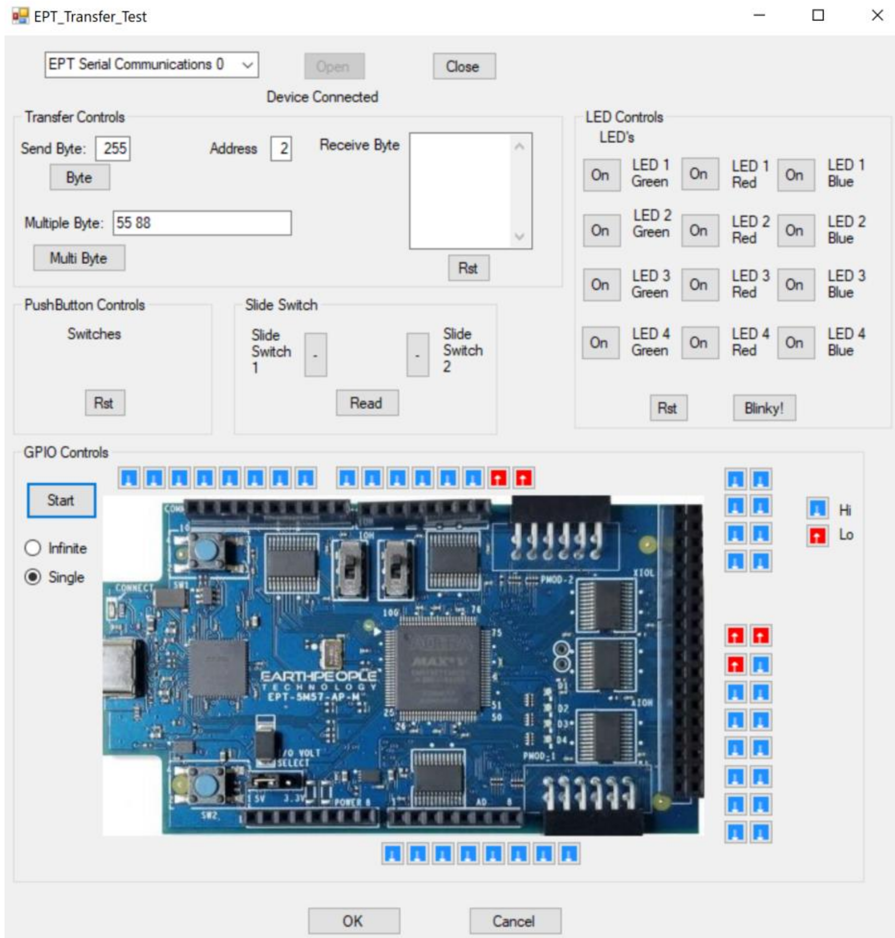


To exercise the Block Transfer EndTerm, click the “Start” button in the GPIO Controls group. The MegaMax will sample the state of each Input pin of the CPLD that is connected to a board edge connector. The Transfer Test Window will display the results of each pin, Hi or Lo



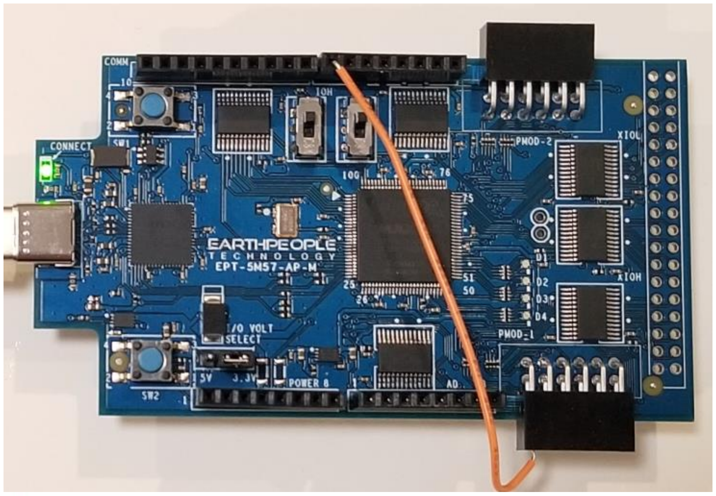


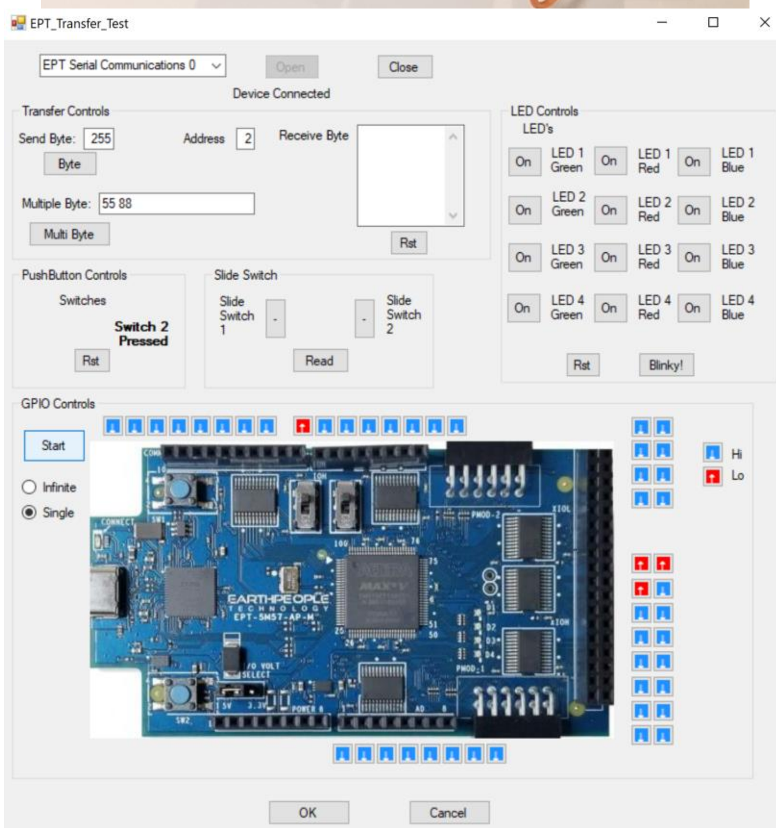
The results of each pin are displayed next to the image of the MegaMax in separate buttons.



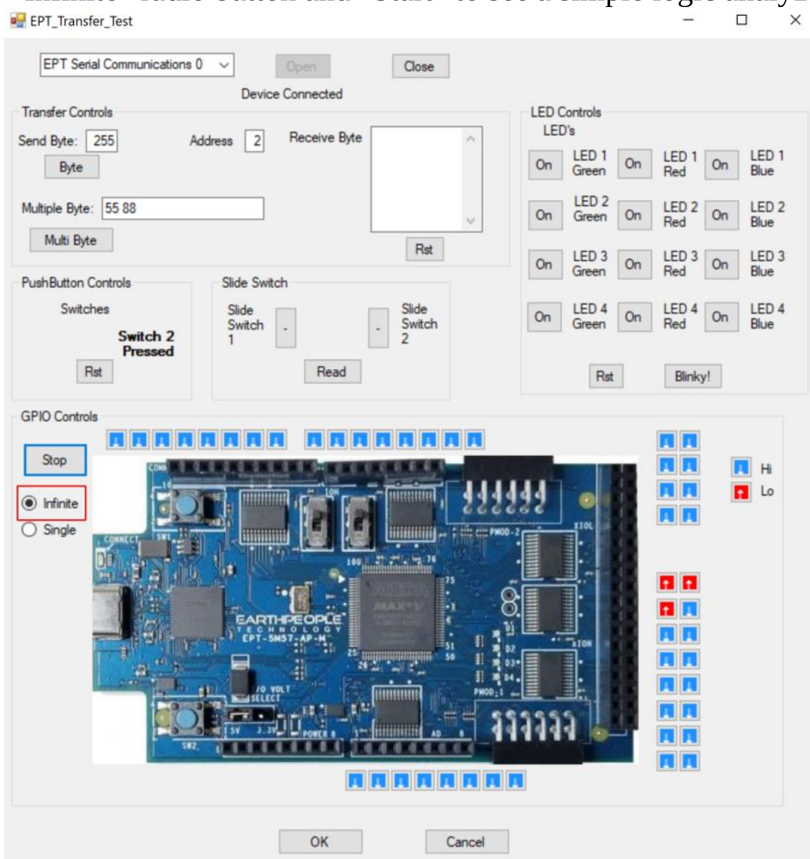
Use a jumper cable to connect between +3.3V and any of the Input pins to see the display detect a high on the selected pin.

USB CPLD Development System User Manual

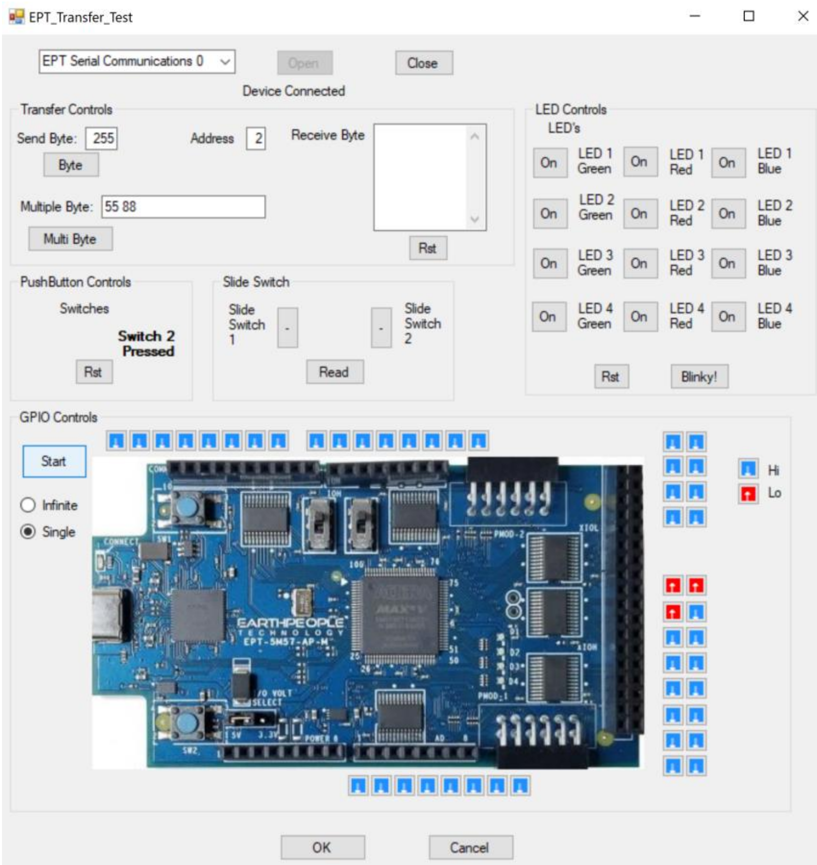




Click on the “Infinite” radio button and “Start” to see a simple logic analyzer.



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


EPT_Transfer_Test
EPT Serial Communications 0
Open
Close
Device Connected
Transfer Controls
Send Byte:
255
Byte
Address
2
Receive Byte
Multiple Byte:
55 88
Multi Byte
Rst
LED Controls
LED's
On
LED 1 Green
LED 1 Red
LED 1 Blue
LED 2 Green
LED 2 Red
LED 2 Blue
LED 3 Green
LED 3 Red
LED 3 Blue
LED 4 Green
LED 4 Red
LED 4 Blue
Blinky!
PushButton Controls
Switches
Switch 2 Pressed
Slide Switch
Slide Switch 1
Slide Switch 2
Read
GPIO Controls
Start
Infinite
Single
Hi
Lo
EARTHPEOPLE TECHNOLOGY
EPT-5M57-AP-M
OK
Cancel