



FPGA Development System User Manual

## **DUEPROLOGIC**

### **FPGA DEVELOPMENT SYSTEM User Manual**

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

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

Circuit designs, software and documentation are copyright © 2026, Earth People Technology, Inc

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



## FPGA Development System User Manual

### Table of Contents

|        |                                                    |     |
|--------|----------------------------------------------------|-----|
| 1      | Introduction and General Description.....          | 4   |
| 1.1    | Test Driving the Active Host Test Application..... | 5   |
| 1.2    | Hardware Description .....                         | 25  |
| 1.2.1  | Host PC Connection.....                            | 28  |
| 1.2.2  | CONF DONE LED .....                                | 29  |
| 1.2.3  | Jumper Settings .....                              | 31  |
| 1.2.4  | Inputs and Outputs .....                           | 36  |
| 1.2.5  | FPGA Configuration .....                           | 37  |
| 1.2.6  | FT2232H Dual Channel USB to Serial Chip .....      | 38  |
| 1.2.7  | SD Card Interface.....                             | 39  |
| 1.2.8  | SD Card Protocol .....                             | 41  |
| 1.2.9  | LEDs .....                                         | 48  |
| 1.2.10 | Pushbuttons .....                                  | 50  |
| 1.2.11 | Power Options.....                                 | 53  |
| 1.2.12 | Clock Domains.....                                 | 53  |
| 1.2.13 | JTAG Header .....                                  | 55  |
| 1.2.14 | PMOD Connectors.....                               | 56  |
| 1.2.15 | Slide Switches .....                               | 59  |
| 1.3    | FPGA Active Host Development.....                  | 61  |
| 1.4    | Active Host EndTerms .....                         | 62  |
| 1.5    | Active Transfer EndTerms .....                     | 66  |
| 2      | EPT Drivers.....                                   | 66  |
| 2.1    | USB Driver .....                                   | 67  |
| 2.2    | JTAG DLL Insert to Quartus Prime Lite .....        | 70  |
| 2.2.1  | Installing Quartus .....                           | 70  |
| 2.2.2  | Downloading Quartus .....                          | 70  |
| 2.2.3  | Quartus Installer .....                            | 76  |
| 2.2.4  | Adding the EPT_Blaster to Quartus Prime .....      | 84  |
| 2.3    | Active Host Application SDK.....                   | 91  |
| 3      | FPGA Active Transfer Library .....                 | 96  |
| 3.1    | EPT Active Transfer System Overview .....          | 97  |
| 3.2    | Active Transfer Library.....                       | 98  |
| 3.2.1  | Active Trigger EndTerm .....                       | 101 |
| 3.2.2  | Active Transfer EndTerm .....                      | 105 |
| 3.2.3  | Active Block EndTerm .....                         | 107 |
| 3.3    | Timing Diagram for Active Transfer EndTerms .....  | 110 |
| 3.3.1  | Active Trigger EndTerm Timing .....                | 110 |
| 3.3.2  | Active Transfer EndTerm Timing.....                | 110 |



## FPGA Development System User Manual

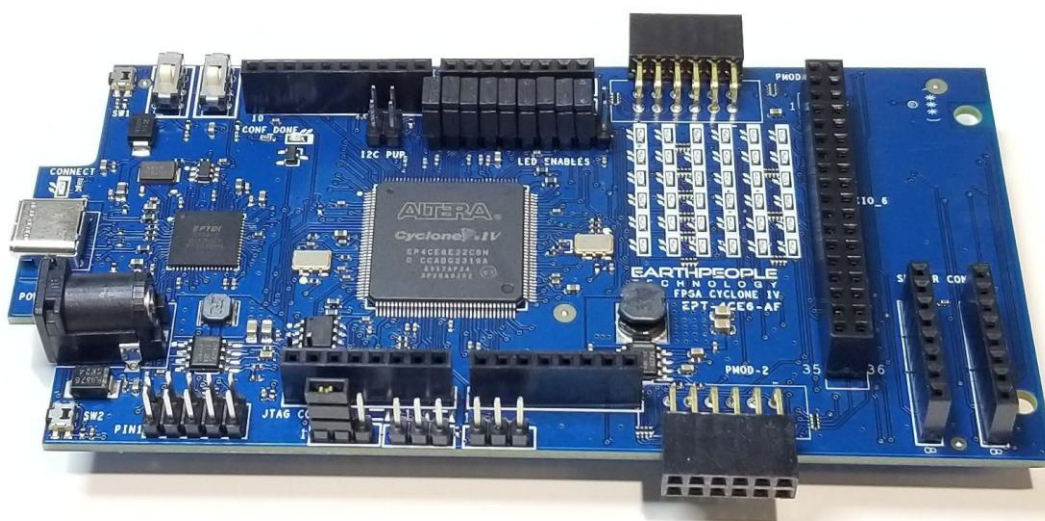
|       |                                                                    |     |
|-------|--------------------------------------------------------------------|-----|
| 3.3.3 | Active Block EndTerm Timing.....                                   | 111 |
| 4     | Compiling, Synthesizing, and Programming FPGA.....                 | 112 |
| 4.1   | Setting up the Project and Compiling .....                         | 112 |
| 4.1.1 | Selecting Pins and Synthesizing.....                               | 121 |
| 4.1.2 | Configuring the FPGA .....                                         | 130 |
| 5     | Active Host Application.....                                       | 142 |
| 5.1   | Trigger EndTerm.....                                               | 144 |
| 5.2   | Transfer(Byte) EndTerm.....                                        | 144 |
| 5.3   | Block EndTerm .....                                                | 144 |
| 5.4   | Active Host SDK .....                                              | 145 |
| 5.4.1 | Active Host Open Device.....                                       | 145 |
| 5.4.2 | Active Host Data Received Function.....                            | 148 |
| 5.4.3 | Active Host Triggers.....                                          | 151 |
| 5.4.4 | Active Host Byte Transfers.....                                    | 154 |
| 5.4.5 | Active Host Block Transfers.....                                   | 155 |
| 6     | Assembling, Building, and Executing a .NET Project on the PC ..... | 158 |
| 6.1   | Creating a Project.....                                            | 159 |
| 6.1.1 | Setting up the C# Express Environment x64 bit.....                 | 160 |
| 6.2   | Assembling Files into the Project .....                            | 167 |
| 6.2.1 | Changing Project Name .....                                        | 167 |
| 6.2.2 | Add Files to Project.....                                          | 171 |
| 6.2.3 | Adding Controls to the Project.....                                | 172 |
| 6.2.4 | Building the Project .....                                         | 176 |
| 6.2.5 | Testing the Project.....                                           | 177 |

## 1 Introduction and General Description

The DueProLogic gives learners the opportunity to have a hands-on approach when learning logic, exploring different iterations of schematic/code designs and the operation of those circuits.

With the DPL's FPGA, projects can be easily attempted which rely on asynchronous, exceedingly fast, and even multiple separate concurrent logic structures operating in parallel. Logic circuits are implemented within the FPGA at few-nanosecond gate speeds and highly parallel in operation, effectively a few hundred MHz. Programmable logic is today's technology for logic learners and implementers. The DPL allows the learner to be more productive and better focus on the underlying logic and integration with the non-logic aspects of non-trivial projects.

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

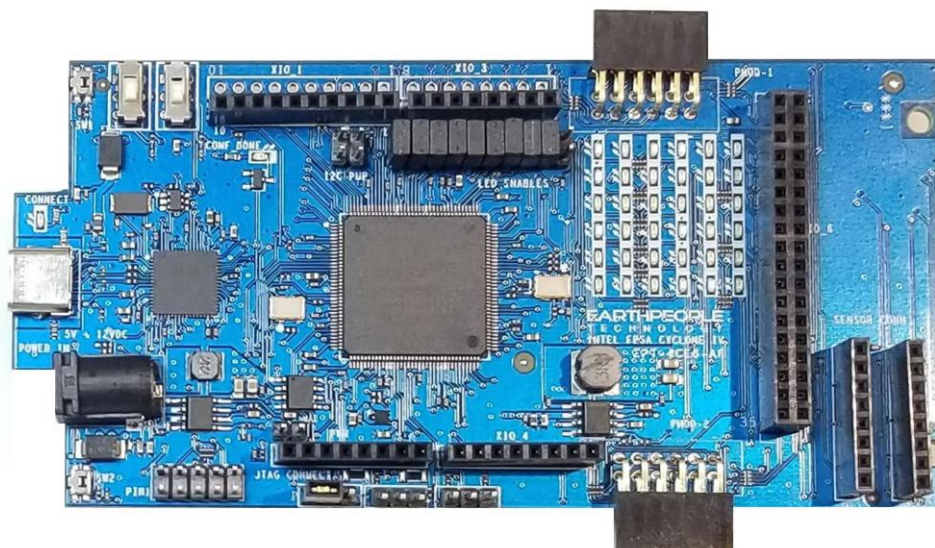




## FPGA Development System User Manual

The DueProLogic FPGA Development System allows users to write HDL code (either Verilog or VHDL) that will implement any digital logic circuit. The user's HDL code is compiled and synthesized and packaged into a programming file. The programming file is programmed into the Configuration Flash using one channel of the USB to Serial chip, the FT2232H. The Active Host SDK contains COM Port functions which maintains device connection, polling, writes and includes a unique receive mechanism that automatically transfers data from DPL when data is ready. It also alerts the user code when the COM Port 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 FPGA project to take advantage of the configurability of the Active Host SDK. All of the drivers, libraries, and project source code are available at [www.earthpeopletechnology.com](http://www.earthpeopletechnology.com).

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

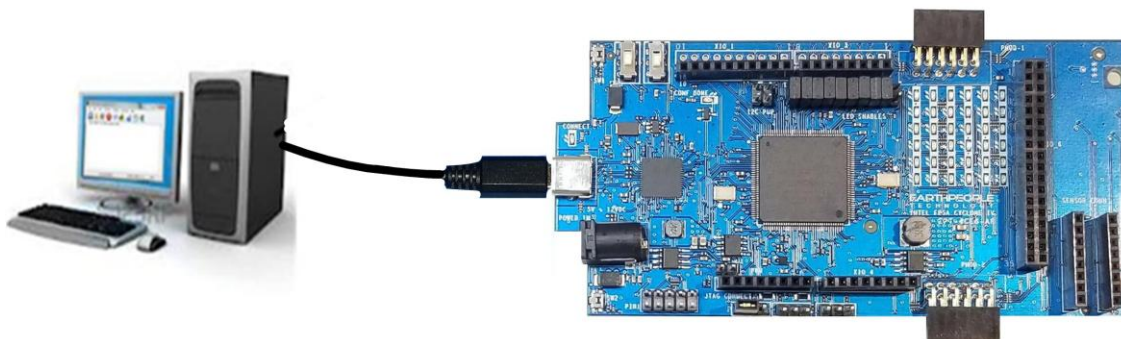




## FPGA Development System User Manual

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

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



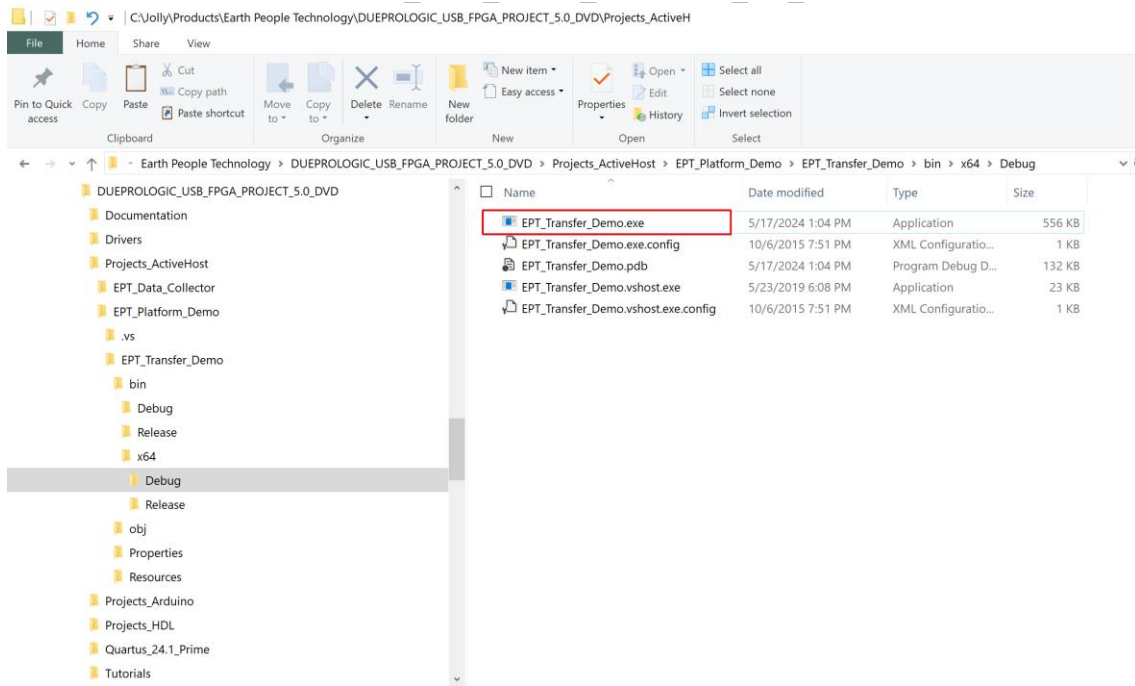
Next, open a Windows Explorer browser.

Browse to the  
Projects\_ActiveHost\EPT\_Platform\_Demo\EPT\_Transfer\_Demo\bin\X64\Debug\



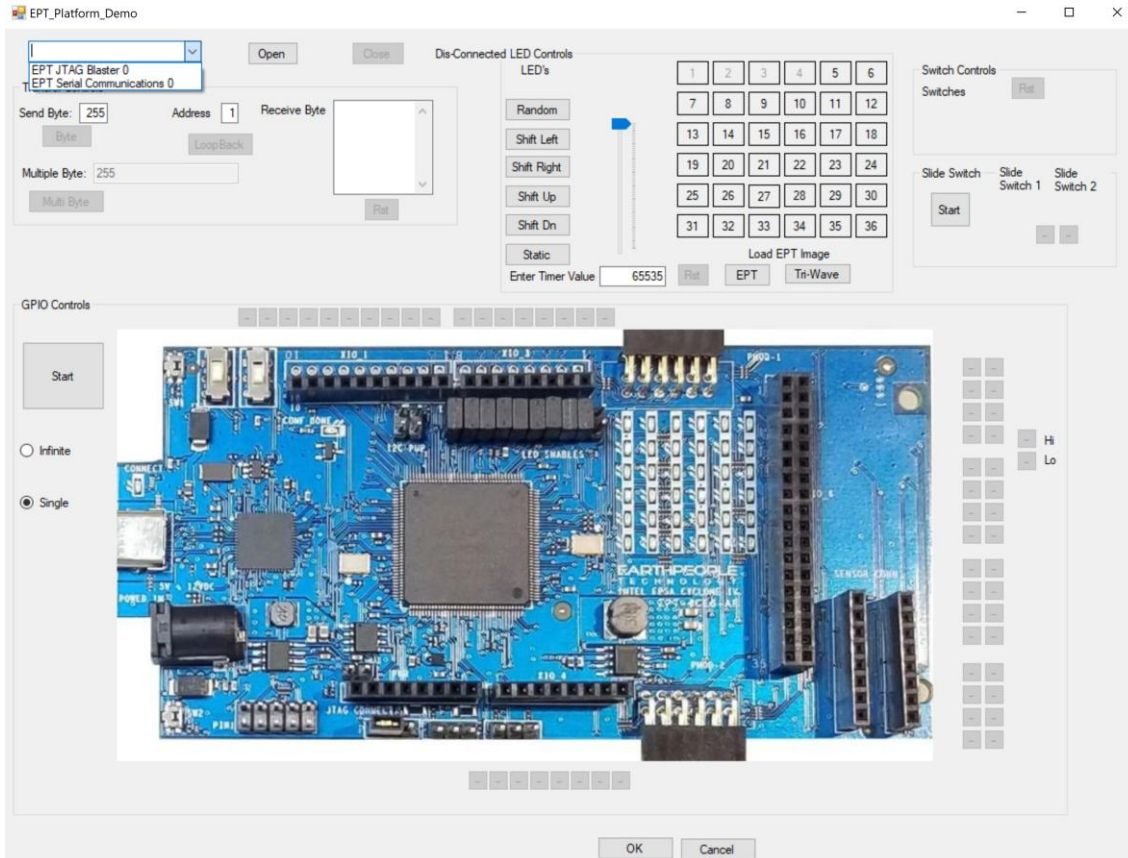
## FPGA Development System User Manual

folder on the DUEPROLOGIC\_USB\_FPGA\_PROJECT\_x.x\_DVD.



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

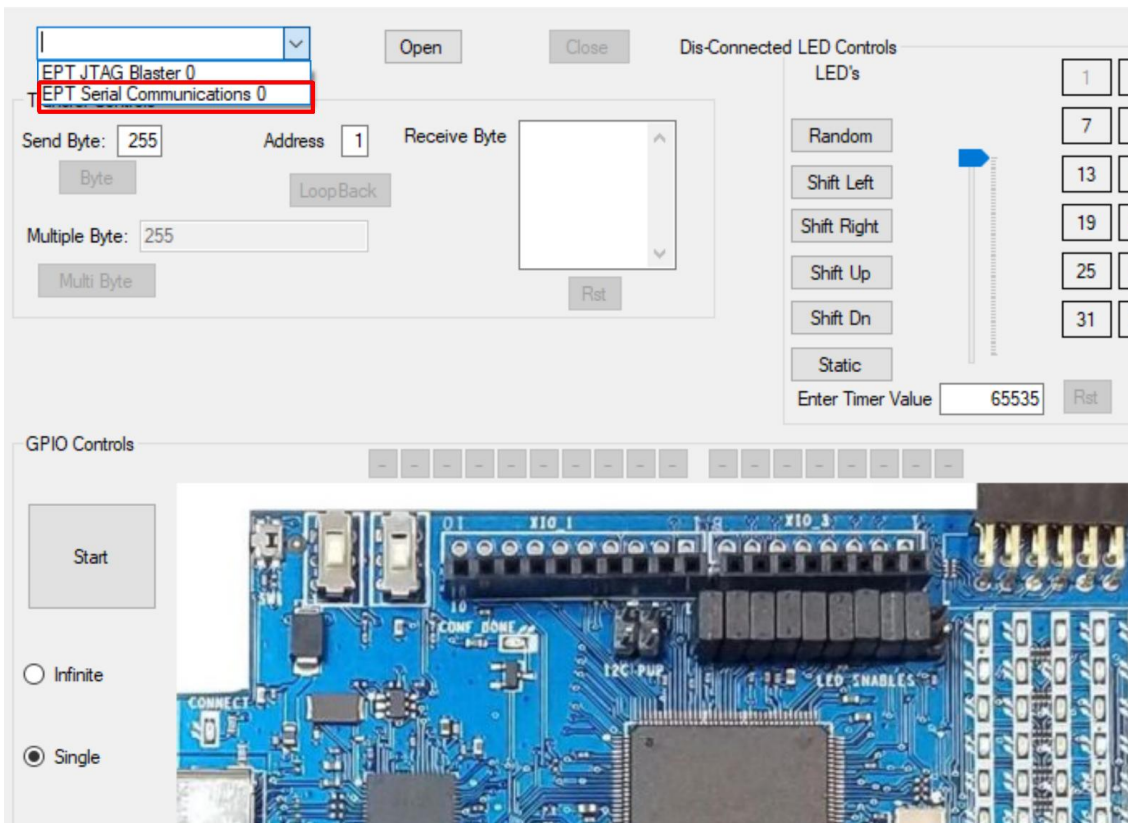
## FPGA Development System User Manual



With the application loaded, select the FPGA board (EPT Serial Communications x) from the dropdown combo box and click on the “Open” button.

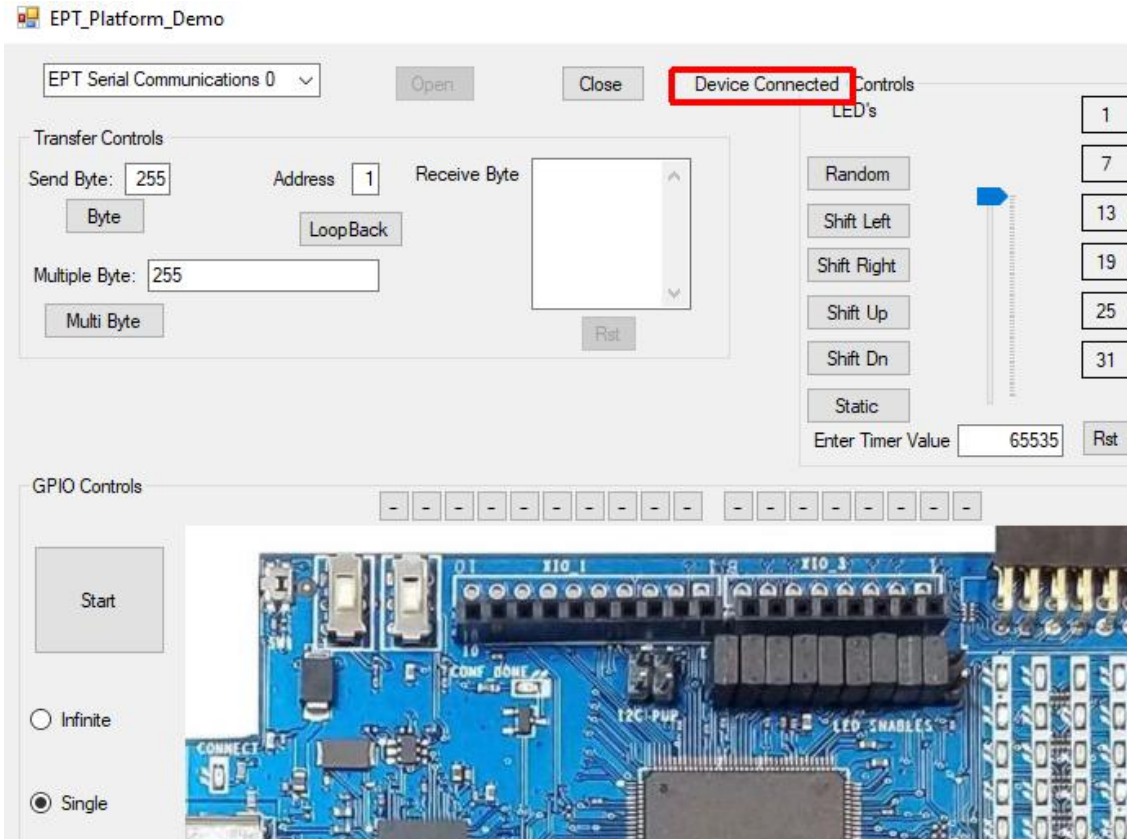
## FPGA Development System User Manual

EPT\_Platform\_Demo



When the EPT Platform Demo software has properly accessed the board, the “Device Connected” label will appear.

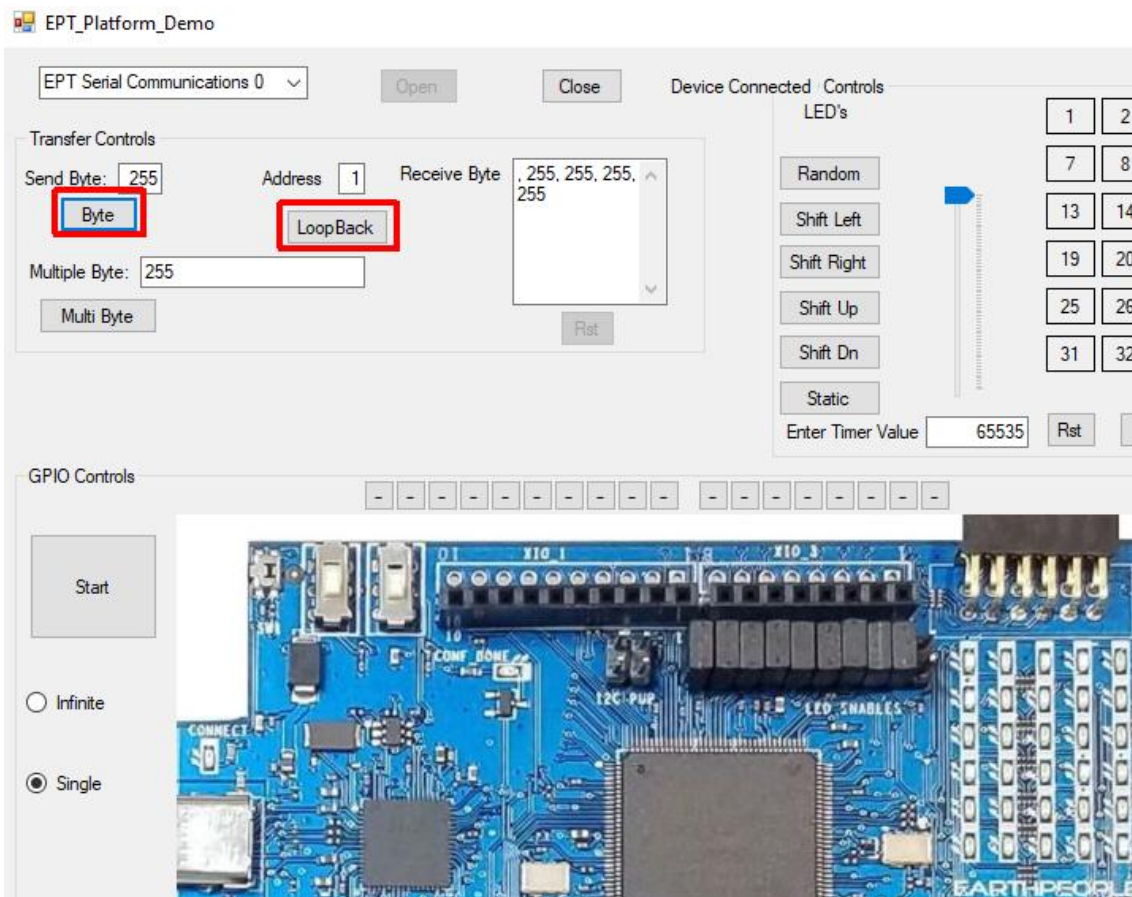
## FPGA Development System User Manual



In the “Transfer Controls” Group, leave the Address set at 1 for the Transfer Controls Group.

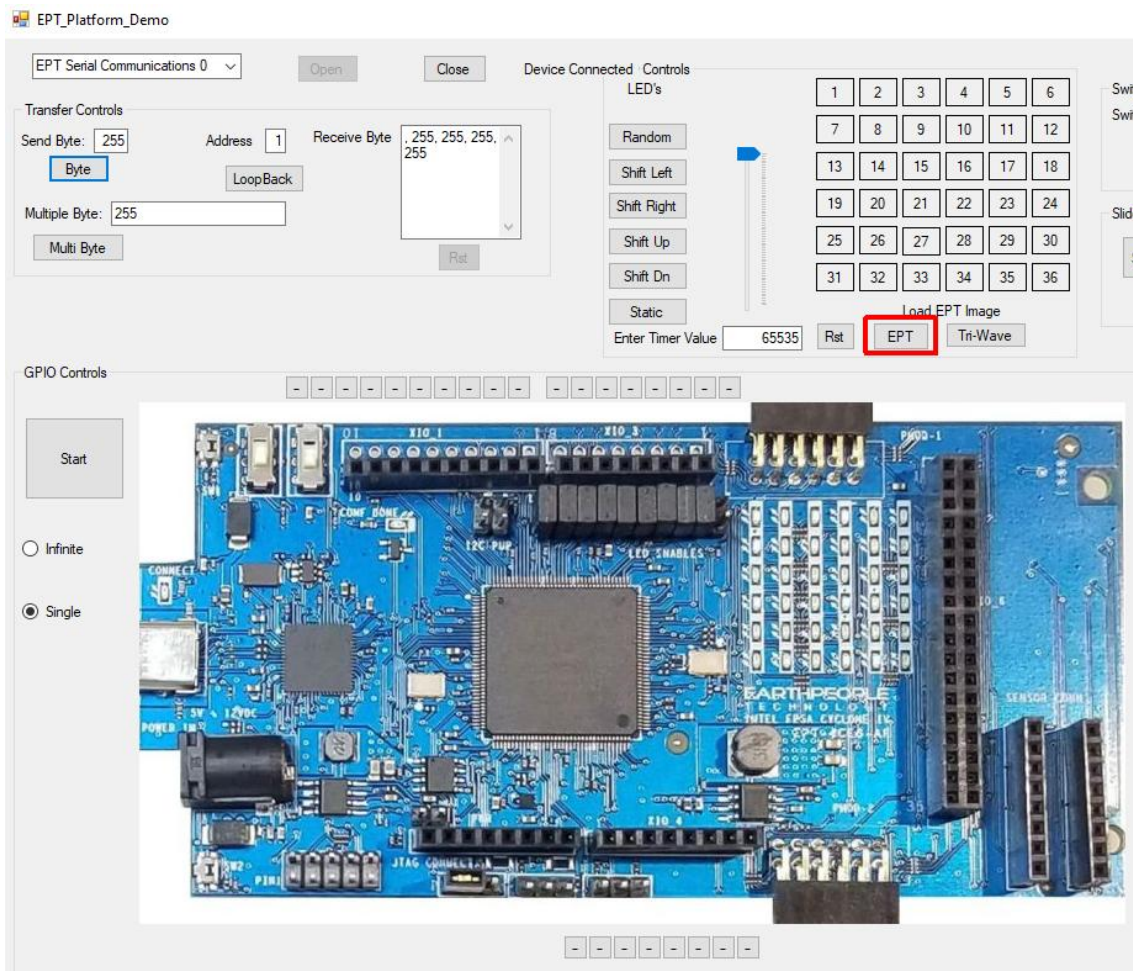
To exercise the Single Byte Transfer EndTerm, click the “LoopBack” button in the Transfer Controls group. Type in several numbers separated by a space and less 256 into the Multiple Byte textbox. Then hit the Multi Byte button. The numbers appear in the Receive Byte textbox.

## FPGA Development System User Manual



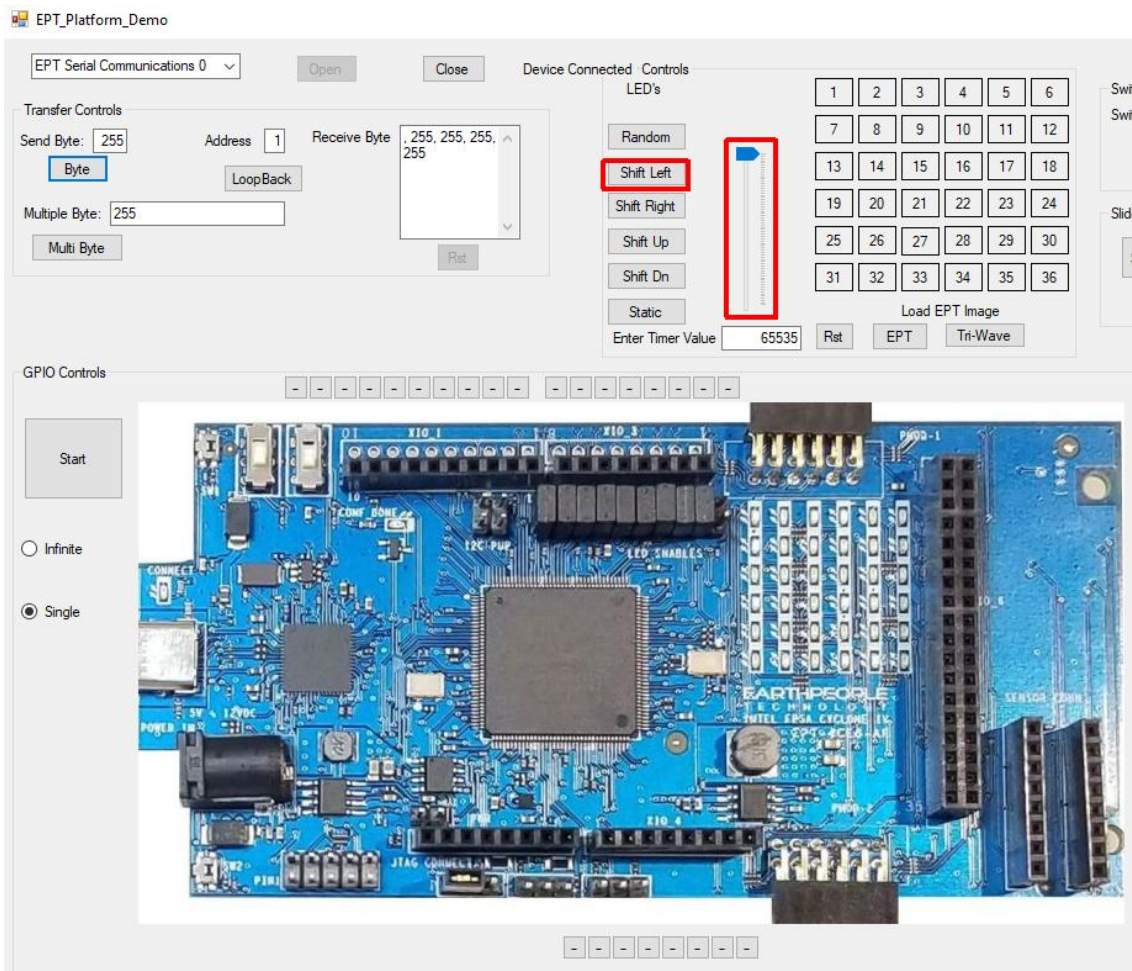
Under LED Controls, press the “EPT” button under the “Load EPT Image”.

## FPGA Development System User Manual



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

## FPGA Development System User Manual

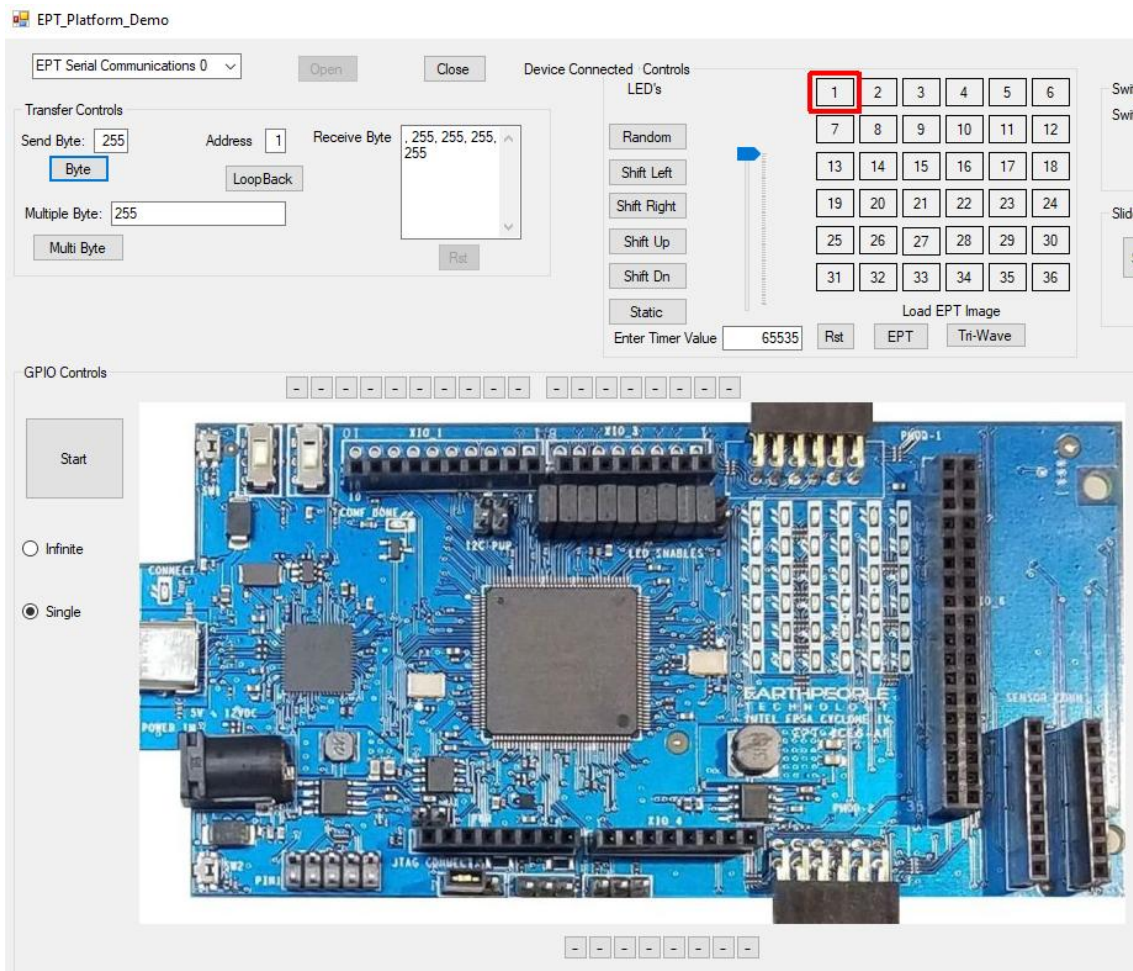


## FPGA Development System User Manual

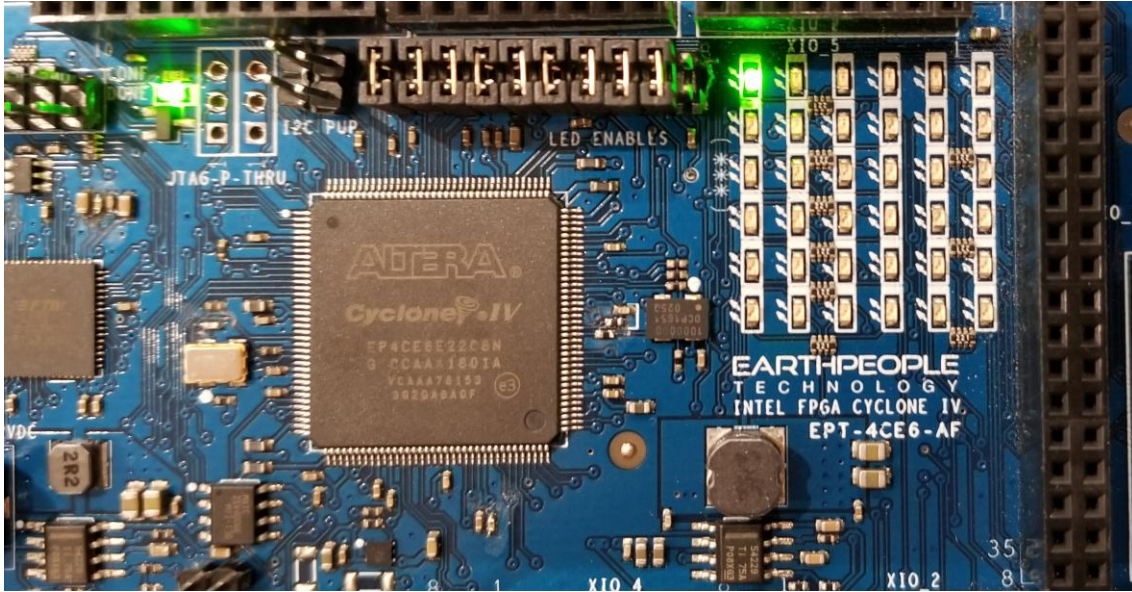


Next press any of the numbered buttons in the “LED Controls” group. This will toggle the corresponding LED in the 6x6 LED Array. It performs this operation as a Block Write. So, an entire LED frame will be transferred to the DueProLogic each time a button is pressed.

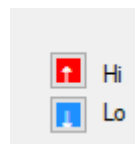
## FPGA Development System User Manual



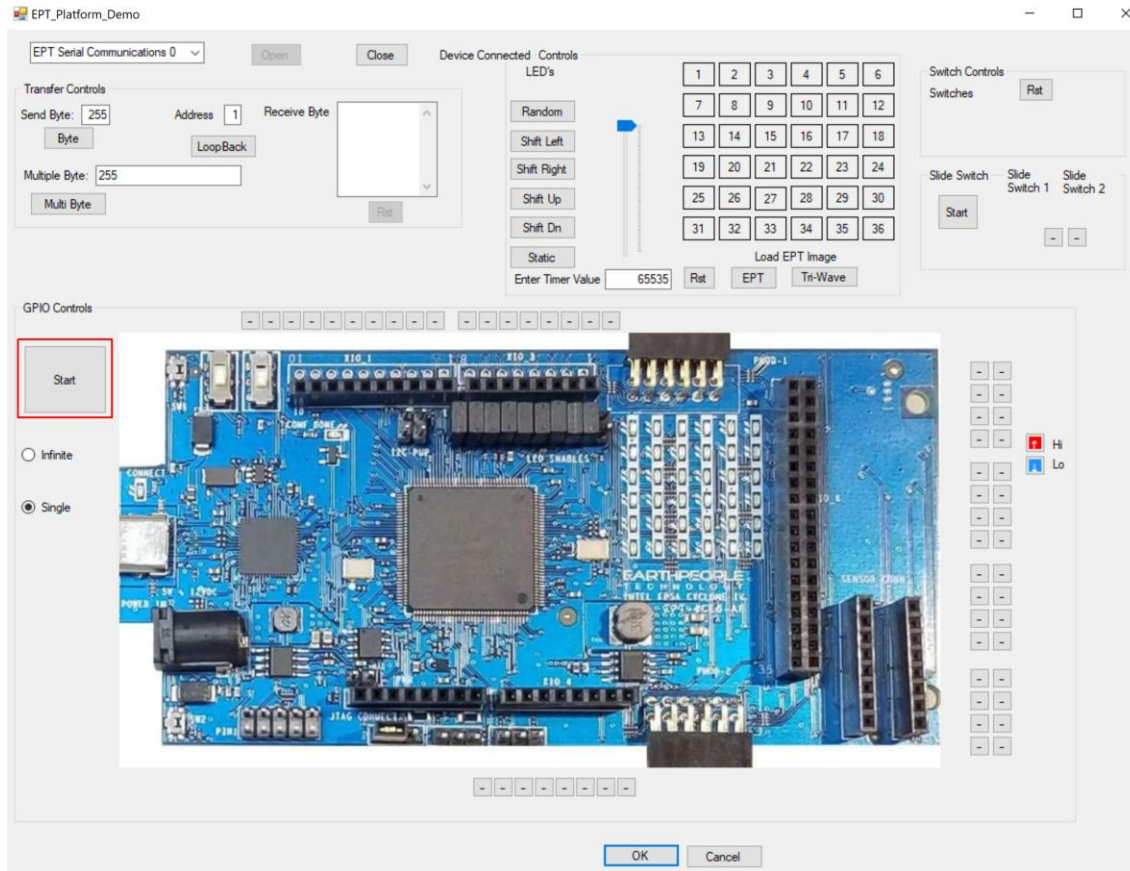
## FPGA Development System User Manual



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

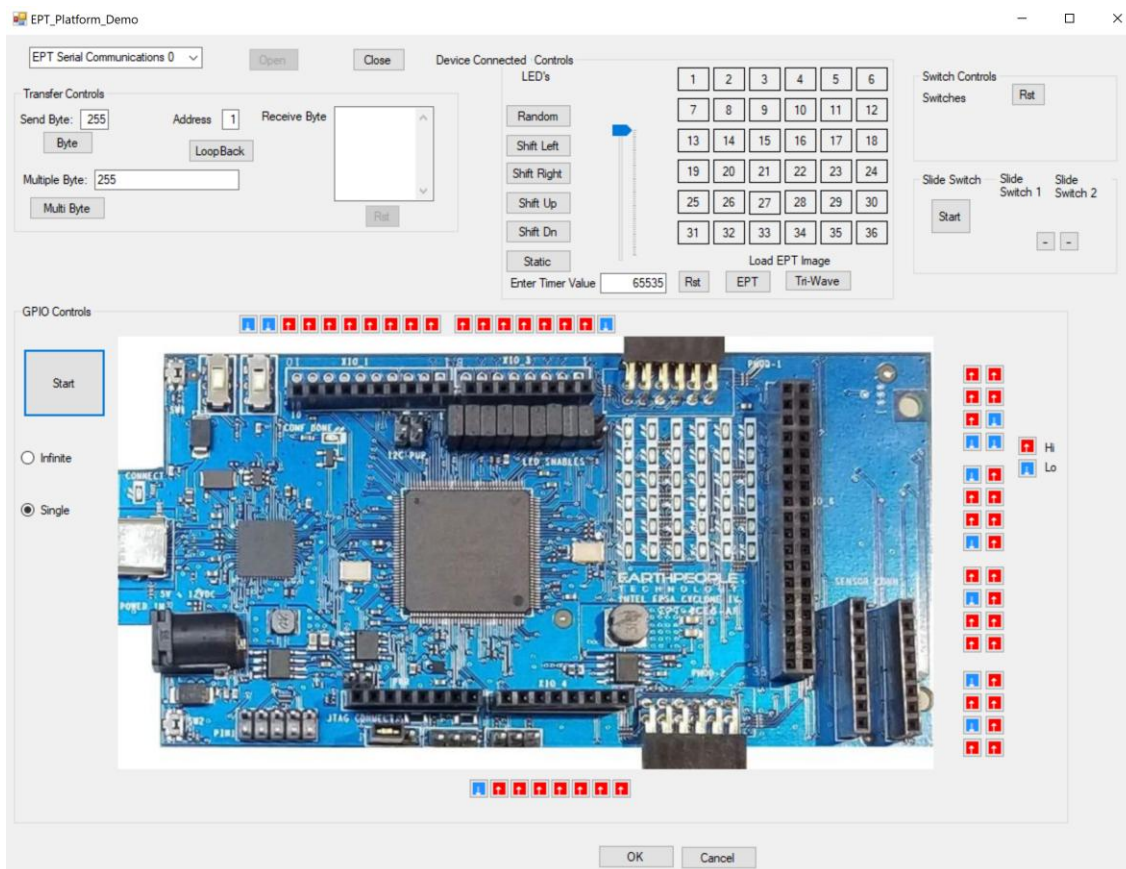


## FPGA Development System User Manual



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

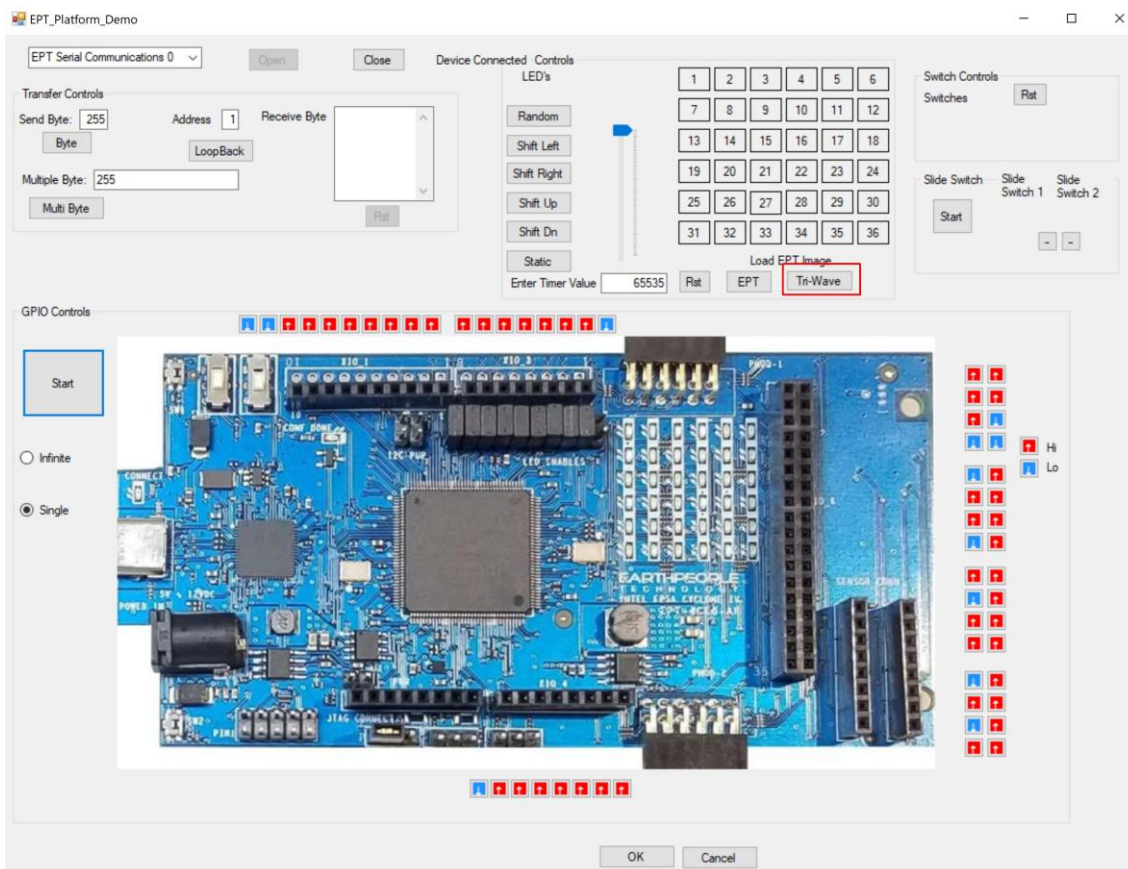
## FPGA Development System User Manual



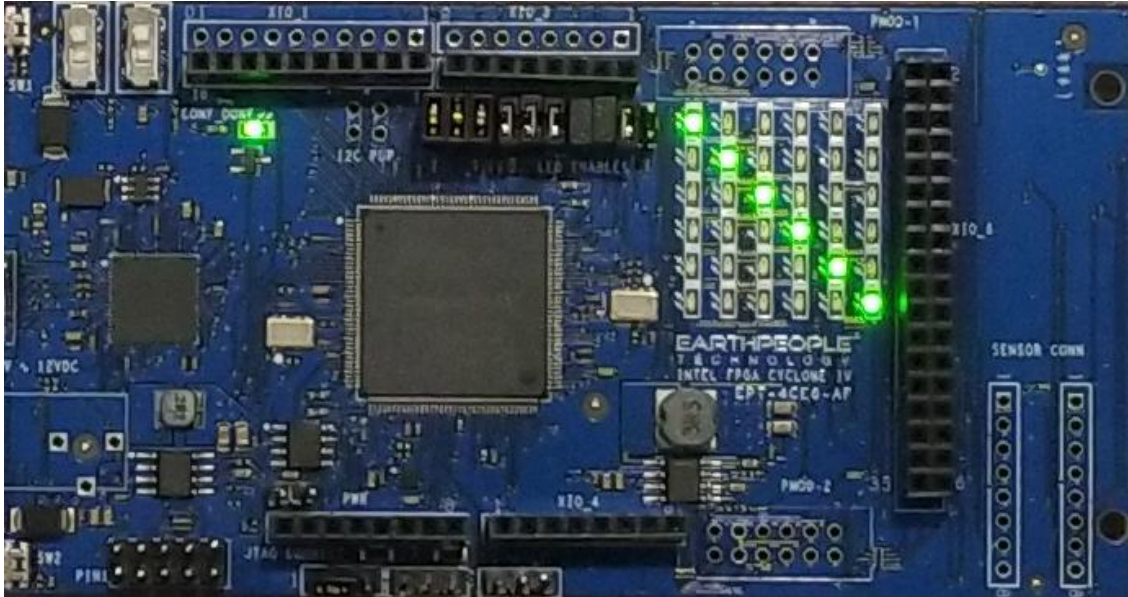
Each input at the board edge connector is floating, it has no solid connection to +3.3V or Ground. So, a mixture of Hi and Lo indications is normal.

Next, click on the “Tri-Wave” button under the LED Controls Group.

## FPGA Development System User Manual

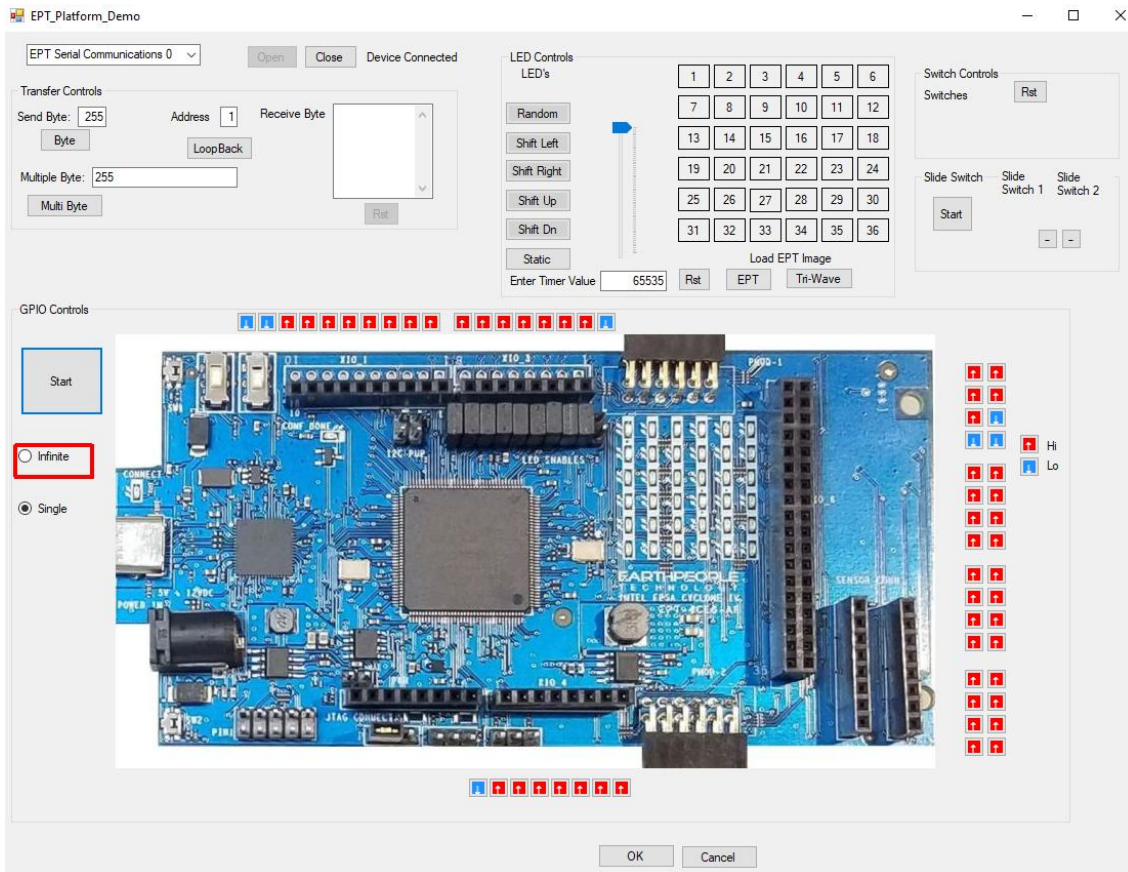


## FPGA Development System User Manual



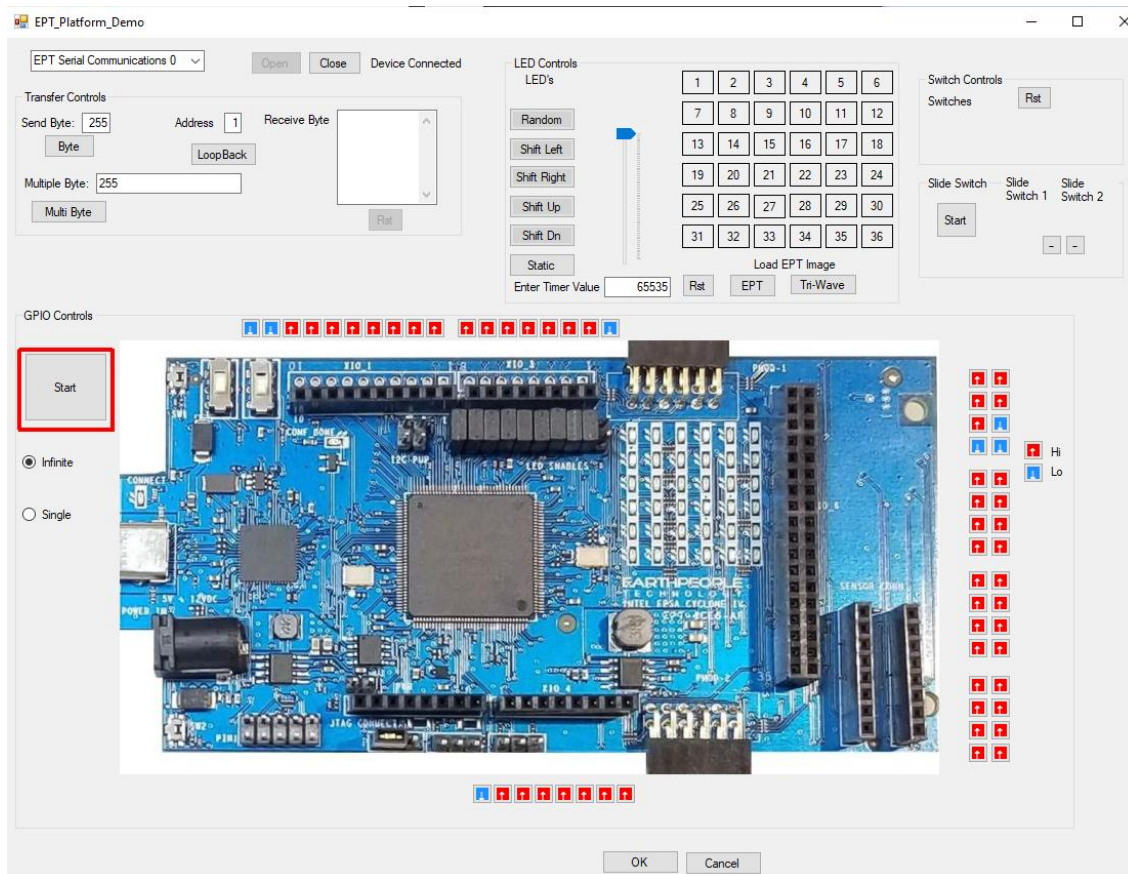
Then, click on the “Infinite” Radio button in the “GPIO Controls” Group Box.

## FPGA Development System User Manual



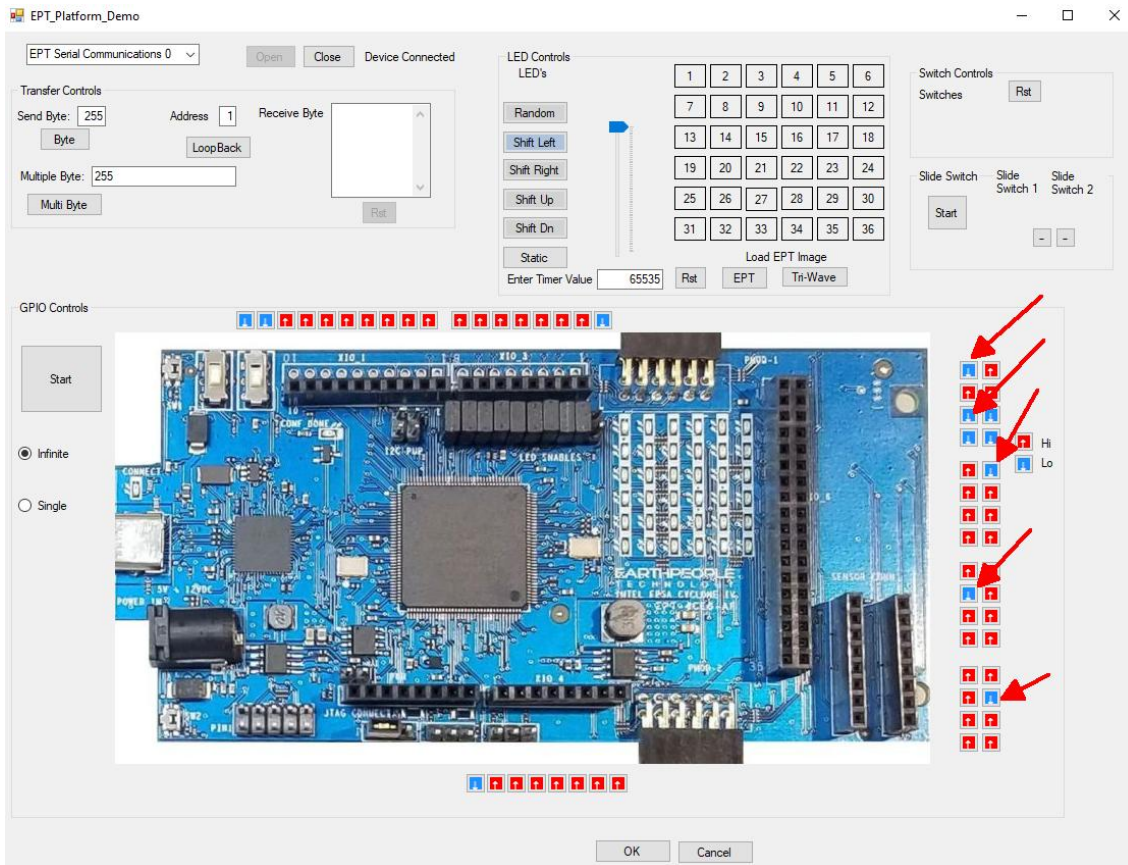
Then, click on the “Start” Button.

## FPGA Development System User Manual



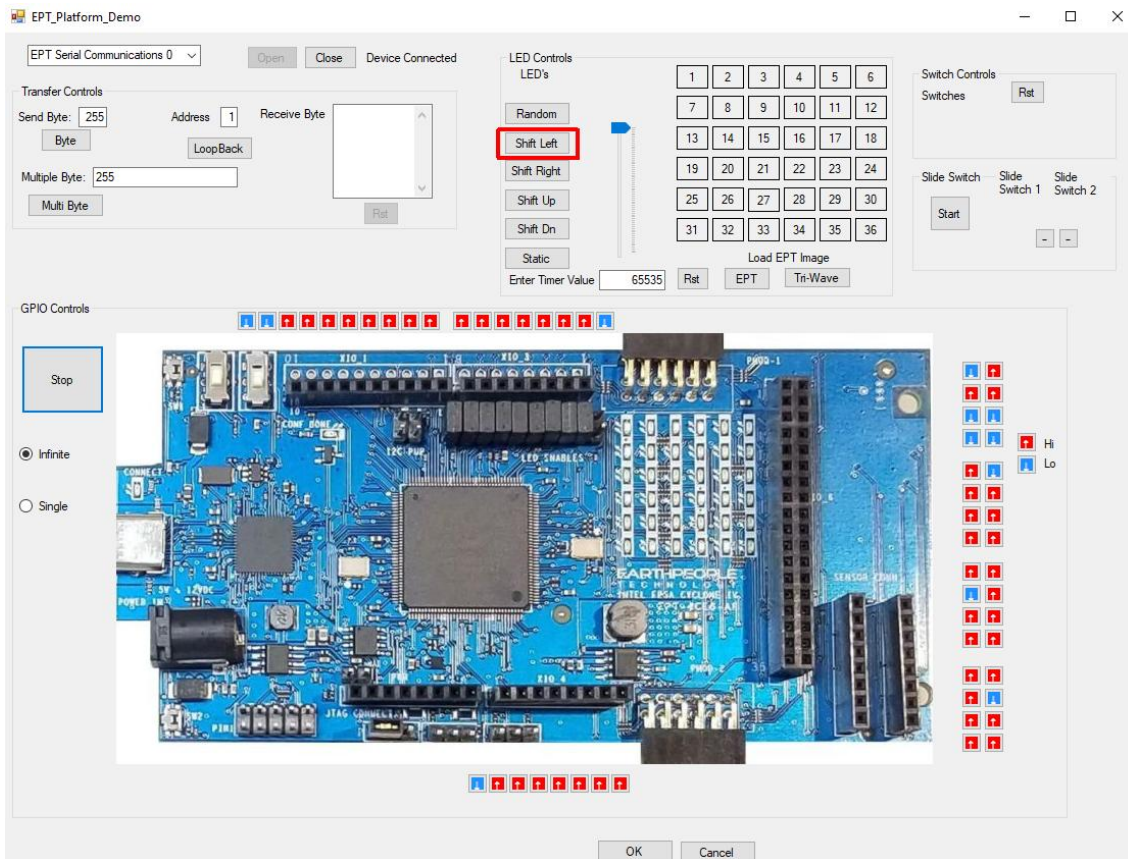
Notice that the GPIO's attached to the LED's in the 6x6 LED Array are now Lo instead of Hi.

## FPGA Development System User Manual



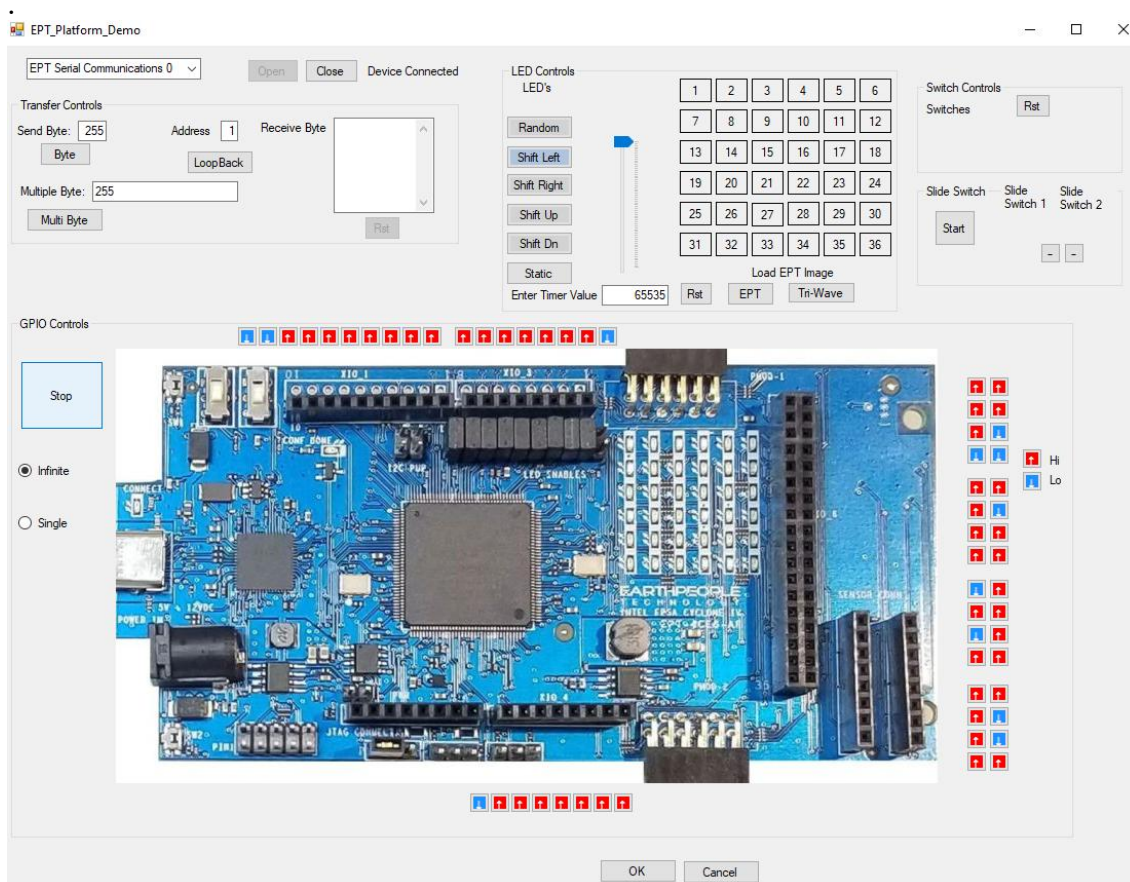
Click on the “Shift Left” button.

## FPGA Development System User Manual



Notice the GPIO's connected to the LEDs will change as the image on the 6x6 LED image shifts.

## FPGA Development System User Manual



The user can also connect +3.3V or Ground using a jumper to any of the GPIO's in the board edge connector. Be sure NOT to connect +3.3V to a Ground pin or Ground to a +3.3V pin. Consult the Data Sheet for the location of the Inputs on the connectors.

### 1.2 Hardware Description

The DueProLogic board is equipped with an Altera EP4CE6E22C8 FPGA; which is programmed using the Altera Quartus Prime software. The FPGA has 6672 Logic Elements and 276480 Total RAM Bits. An on board 66 MHz oscillator is used by the EPT Active Transfer Library to provide serial transfer using the UART method. Fifty Four I/O's from the FPGA are attached to six separate user connectors. The user connectors are organized to fit the Due Arduino platform. There is a separate 36 pin dual row connector at the rear of the board arranged to mate with standard Bread



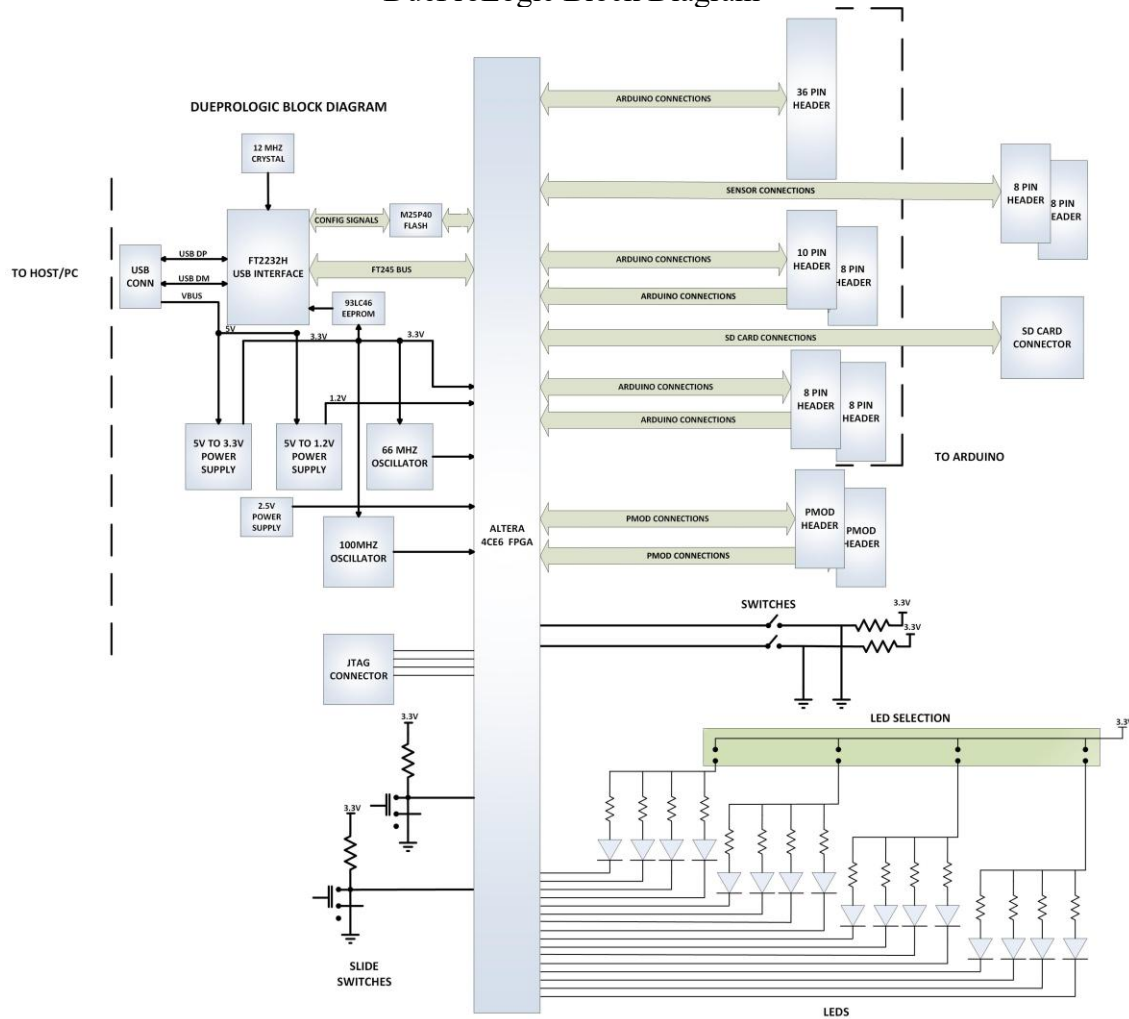
## FPGA Development System User Manual

boards. There are 36 green User LED's, two Slide Switches and two Push Buttons that are controllable by the user code. The hardware features are as follows.

- Altera EP4CE6 FPGA with 6272 Logic Cells
- Dual Channel High Speed USB FT2232H
- 66 MHz oscillator for driving USB data transfers and users code
- 100MHz oscillator for scaling up/down for users needs
- Standard SD Card interface for memory expansion
- 54 user Input/Outputs (+3.3V only)
- 36 Green LED Array accessible by the user
- Two PCB switches accessible by the user
- Two Slide switches
- Two PMOD Connectors

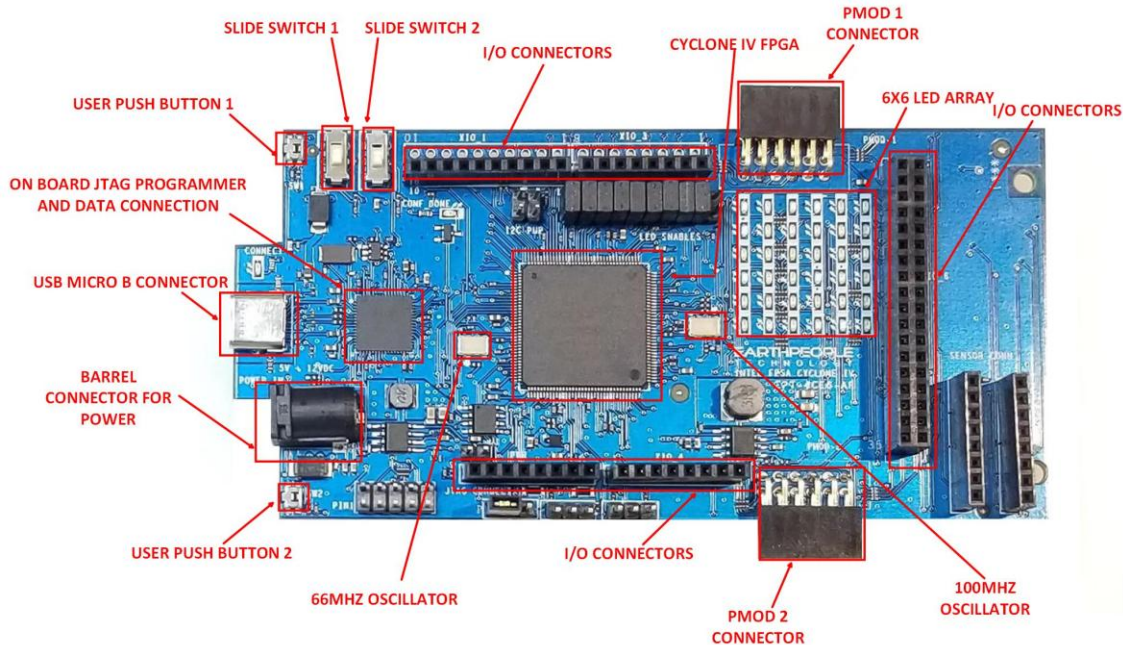
## FPGA Development System User Manual

### DueProLogic Block Diagram



## FPGA Development System User Manual

### DueProLogic Hardware

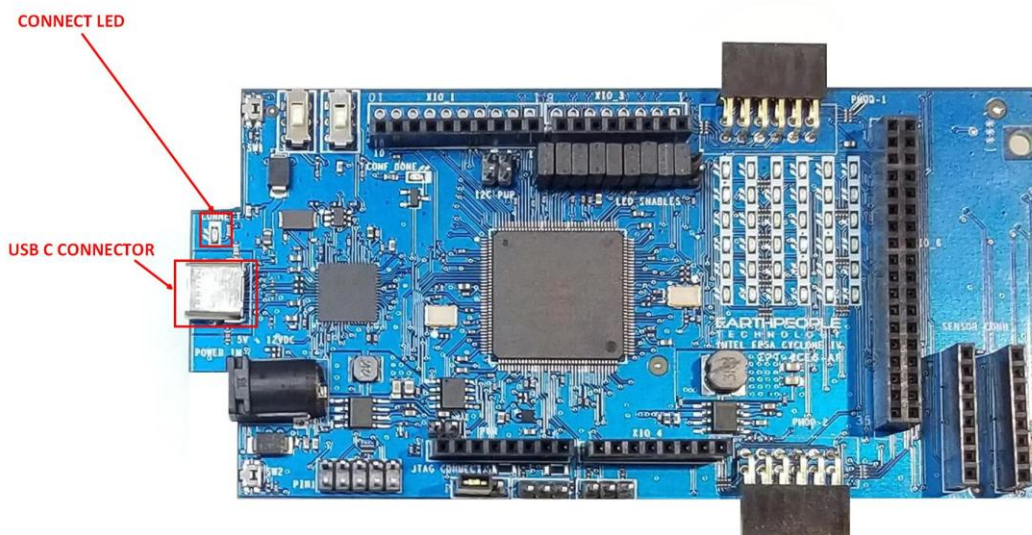


#### 1.2.1 Host PC Connection

The DueProLogic 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 DueProLogic via USB
- The FT2232HQ chip is working properly
- The Host PC has found the appropriate driver and will communicate with the DueProLogic

## FPGA Development System User Manual



### 1.2.2 CONF DONE LED

The CONF\_DONE LED is connected directly to the FPGA CONF\_DONE signal.



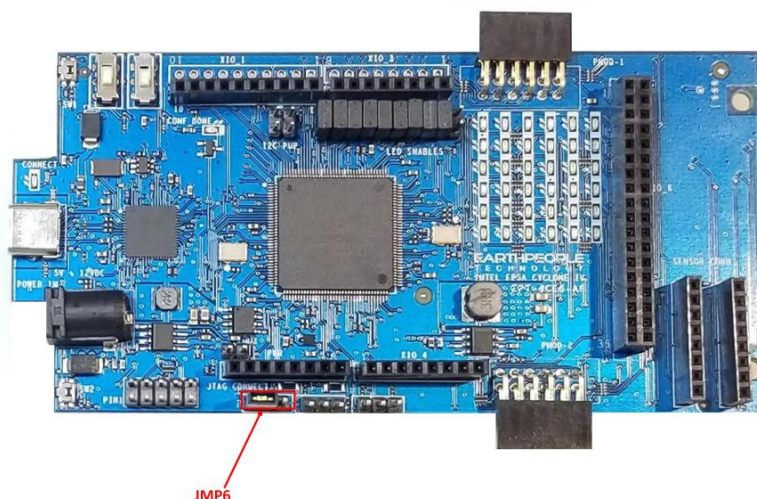
## FPGA Development System User Manual

### 1.2.3 Jumper Settings

There are six jumper selections on the DPL. These jumper selections will set up the DPL for proper operation and also some optional settings.

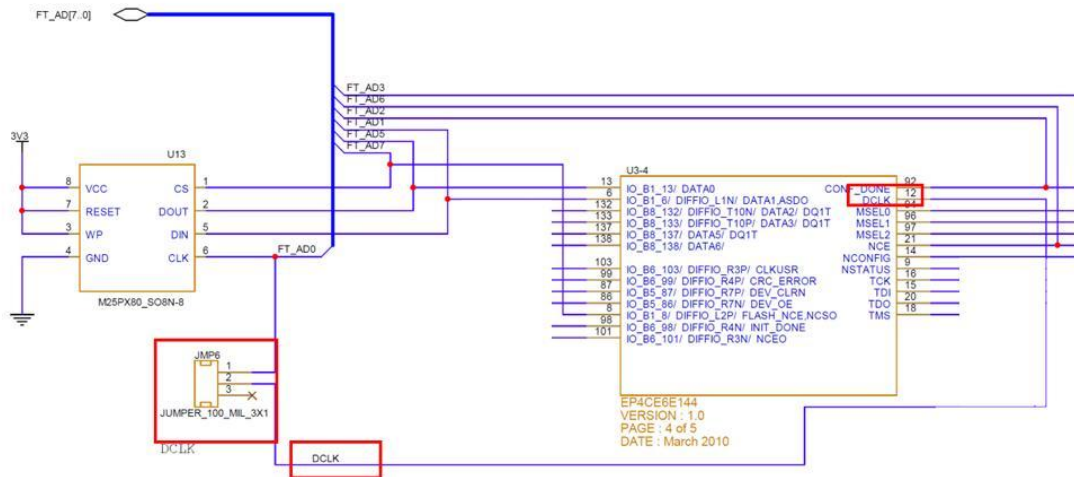
#### 1.2.3.1 JMP6

The jumper setting JMP6 is used for proper operation of configuring the on board flash for the FPGA.



JMP6 is used to allow DCLK to pass from FPGA to the configuration flash. This jumper must have a shunt install on pins 1 and 2 for proper operation of the configuration flash and FPGA.

## FPGA Development System User Manual



When using the JTAG only operation of the FPGA (using the \*.sof file to program the FPGA), this jumper should be removed.

### 1.2.3.2 JMP3

JMP3 provides a pullup/pulldown to the “NCONFIG” signal of the FPGA. When using the configuration flash to configure the FPGA,



# EARTHPEOPLE

T e c h n o l o g y

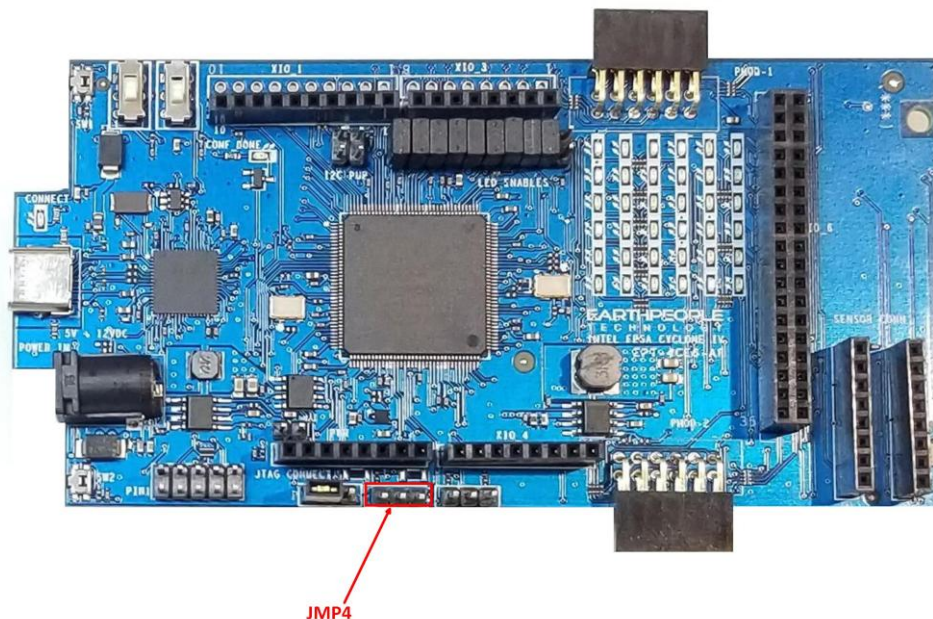


**JMP3**

### 1.2.3.3 JMP4

JMP4 provides a pullup/pulldown to the “CONF\_DONE” signal of the FPGA. When using the configuration flash to configure the FPGA,

## FPGA Development System User Manual

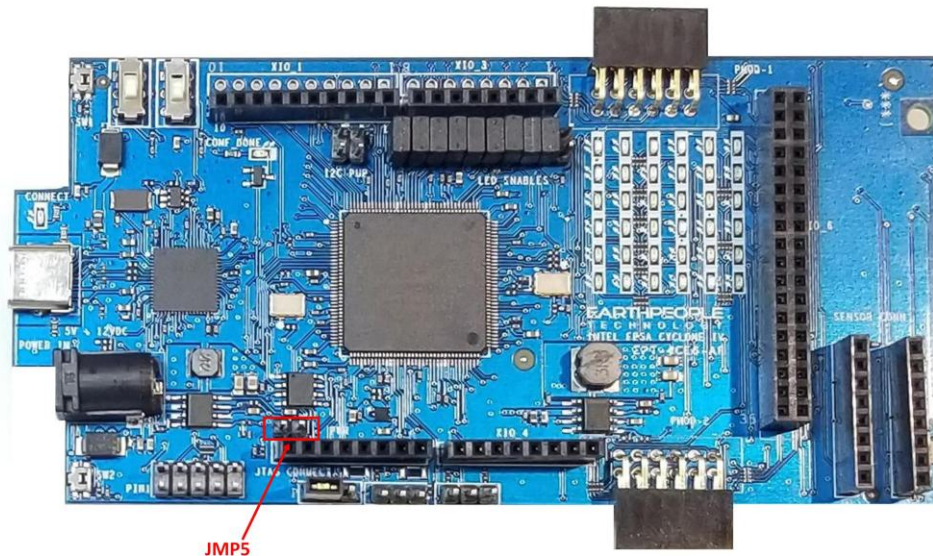
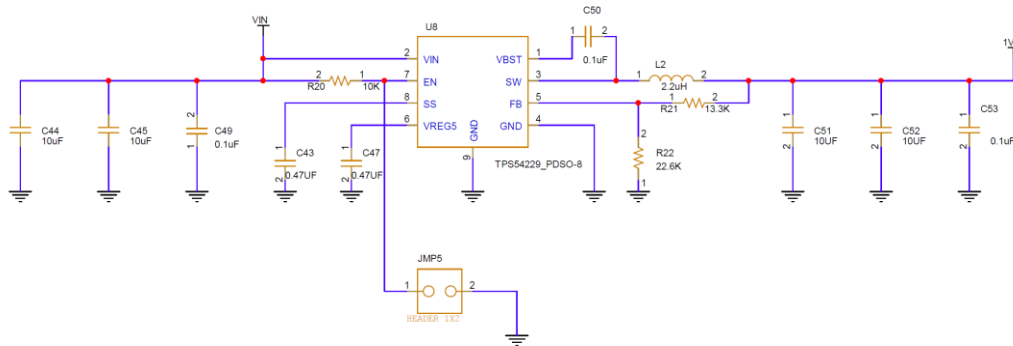


Do Not Add a jumper. When programming the FPGA using JTAG, connect a jumper between pins 1 and 2 to add a pullup to the “CONF\_DONE” signal.

### 1.2.3.4 JMP5

JMP5 provides a pulldown on the EN signal of the 1.2V Power Supply.

## FPGA Development System User Manual

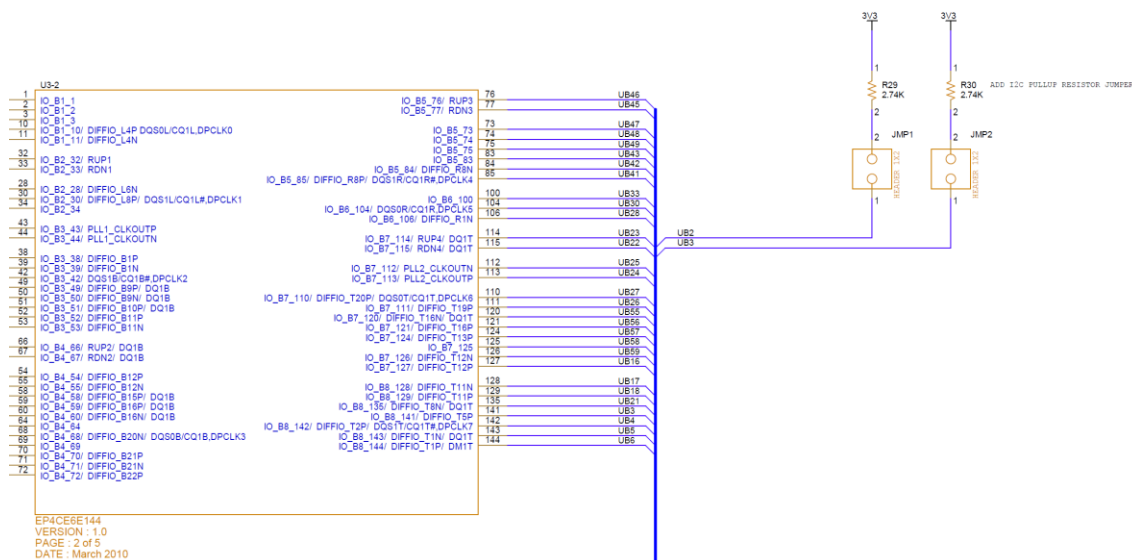


This jumper should remain unconnected for proper operation of the FPGA.

Only use this jumper if the configuration flash file has become corrupted and the only method to load a new configuration flash file is to disable the FPGA. For further instructions on the use of this jumper please email Earth People Technology directly.

### 1.2.3.5 JMP1 and JMP2

JMP1 and JMP2 are used to add pullups to the I2C communication signals.



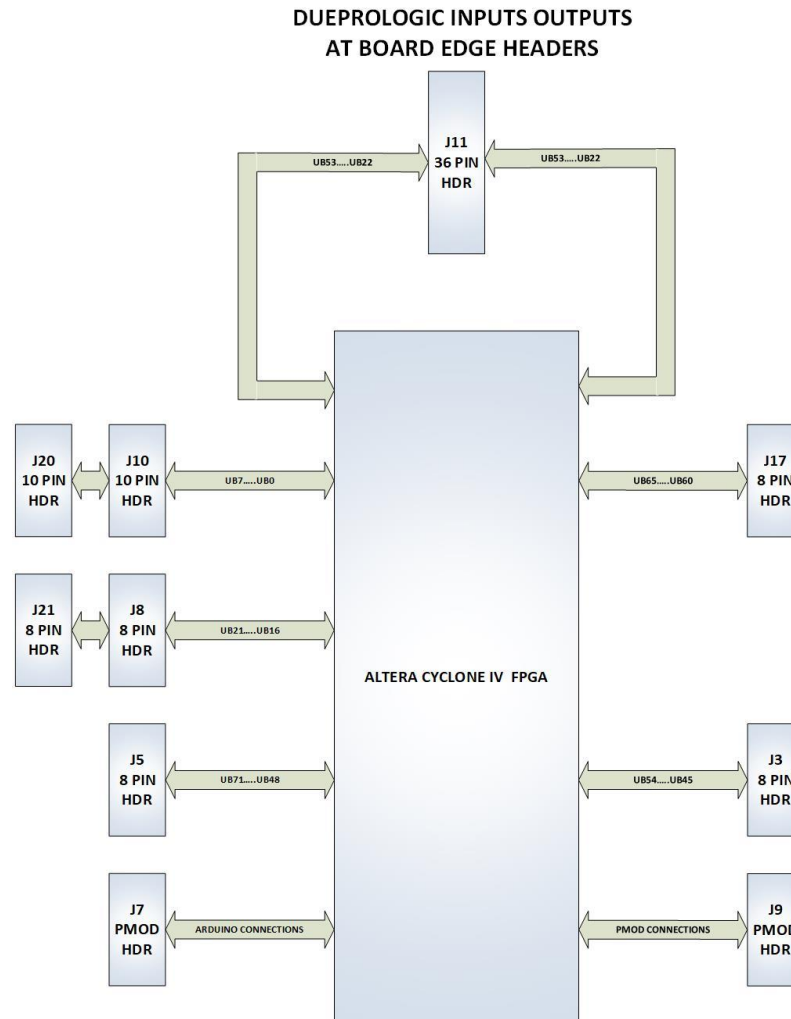
If signals UB2 and UB3 are needed for I2C signals SDA and SDC, connect a shunt to the jumpers to provide the 2.7K Ohm pullup resistors.

### 1.2.4 Inputs and Outputs

**Do not connect a 5Volt device to the DPL**

There are 54 Inputs/Outputs which are +3.3Volt only. The FPGA I/O's are organized as separate pins and connect to the Arduino connectors and separate connectors. Each I/O must be defined as input or output in the user code. Each pin of the connectors can behave as either input or output. Refer to the DueProLogic Data Sheet for exact pinout location.

## FPGA Development System User Manual

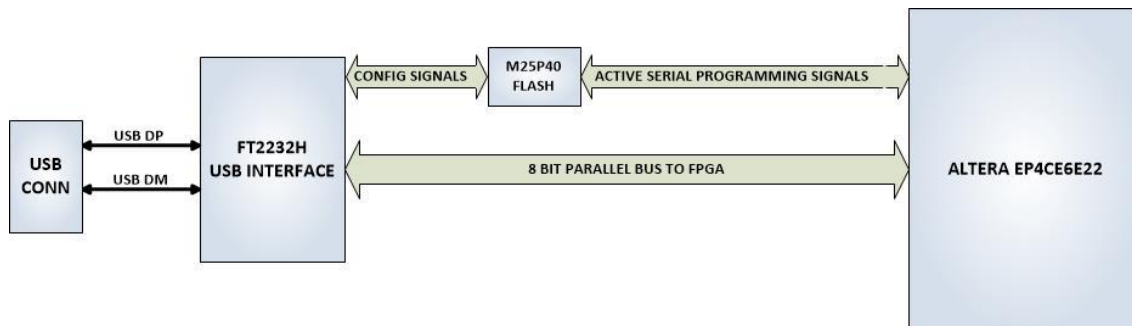
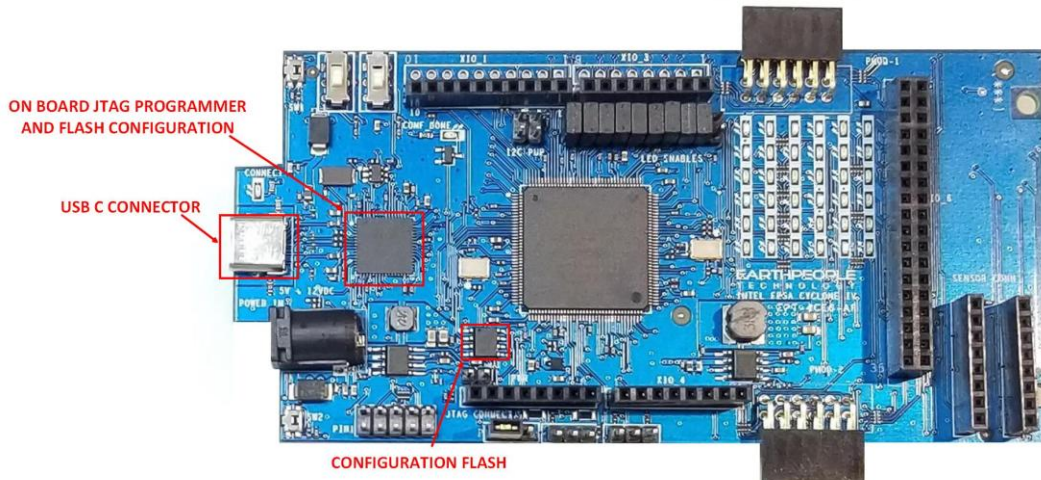


Each Header includes a Power and Ground connection which allows an easy way to power sensors or bread boards directly from the DPL.

### 1.2.5 FPGA Configuration

The Cyclone IV FPGA is configured for operation when the power is applied to the board. A dedicated Configuration Flash chip is included on the DueProLogic for the purpose of configuring the FPGA on power up. The DPL uses the second channel of the FT2232H chip as a dedicated Flash programming port. The Configuration Flash can be

## FPGA Development System User Manual



programmed directly from Quartus Prime by using the EPT-Blaster driver. Follow the instructions in the “EPT Drivers” section of this manual. Follow the instructions in the section “Setting up the Project and Compiling” to program the flash on the DPL.

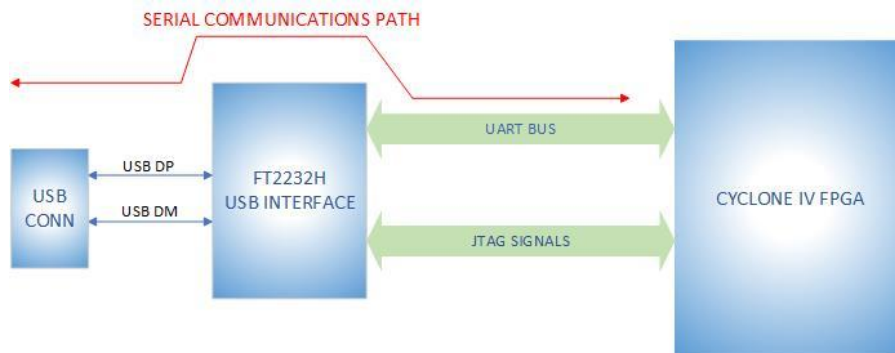
### 1.2.6 FT2232H Dual Channel USB to Serial Chip

The DueProLogic contains an FTDI 2232H dual channel high speed (480 Mb/s) USB to FIFO (first in-first out) integrated circuit to interface between the Host PC and the FPGA. The FT2232H provides a means of data conversion from USB to serial/ parallel data and serial/parallel to USB for data being sent from the FPGA to the PC. Channel A is configured as a Flash Configuration bus and Channel B is configured as a serial bus. FPGA Programming commands are transmitted via the channel A interface. Channel B

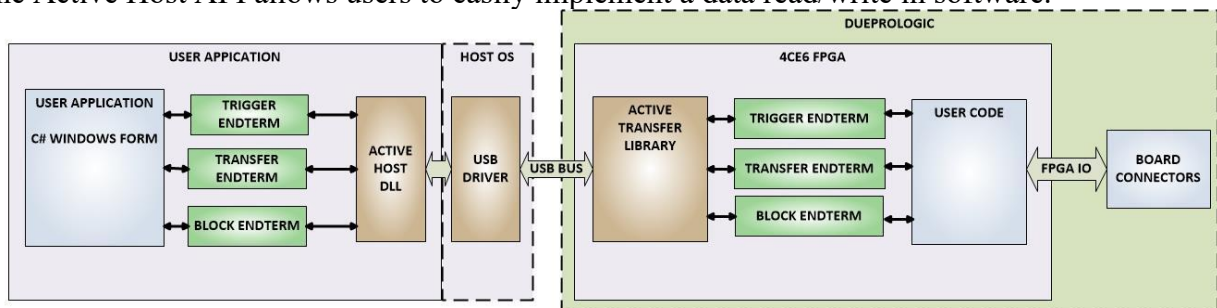
## FPGA Development System User Manual

has one dual port 4Kbyte FIFO for transmission from Host PC to the FPGA, it also has one dual port 4Kbyte FIFO for receiving data from the FPGA to the Host PC.

The serial communications pathway is used to send and receive data to/from the Host PC. The pathway comprises the USB-C Connector, USB to Serial chip and the Cyclone IV FPGA. The USB driver from FTDI is required for the FT2232H chip. Please see the section “EPT Drivers” for instructions on driver installation.



The serial communications pathway between host and FPGA is transparent to the user. The Active Host API allows users to easily implement a data read/write in software.

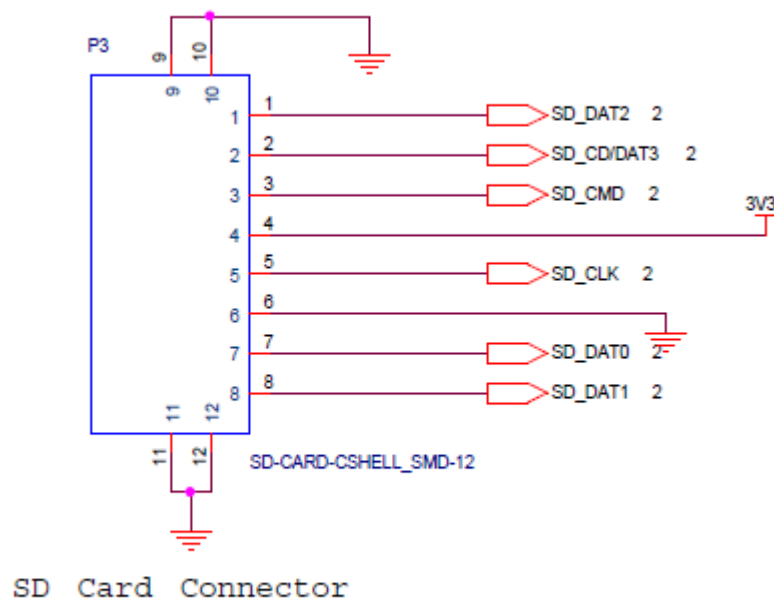
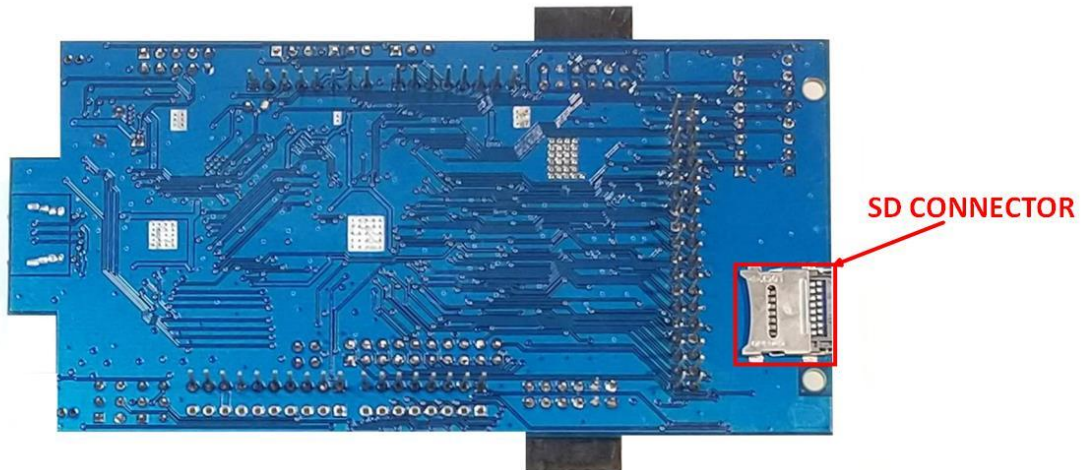


Please see the section “Active Host Application” on instructions for how to use the Active Host API to build an application for data read/write between Host and DPL.

### 1.2.7 SD Card Interface

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

## FPGA Development System User Manual



### 1.2.8 SD Card Protocol

There's a ton of information out there on using the MMC/SD SPI protocol to access SD cards but not much on the native protocol. This page hopes to rectify that with information helpful to those implementing a SD host or trying to understand what they're seeing on an oscilloscope.

#### 1.2.8.1 References

- The full MMC Specification is available as [JESD84-A44](#) from the [JEDEC website](#)
- The [SD Simplified Specification](#) from the SD Card Association covers most of the protocol
- The full SD specification is available after [joining the SD Card Association](#)

#### 1.2.8.2 Compatibility

MMC, SD, and SDHC cards are broadly compatible at the electrical and framing level. A properly designed controller should be able to handle them all. Some differences are:

- MMC cards are available in both High Voltage (2.7 - 3.6 V) and Dual Voltage (2.7 - 3.6 and 1.70 - 1.95 V)
- MMC is designed to support multiple cards on the same bus
- During initialisation MMC cards are clocked at 400 kHz or less
- MMC, SD, and SDHC all have different initialisation sequences

#### 1.2.8.3 Protocol

The protocol is a strict master/slave arrangement where data is clocked synchronously from the host to the card or from the card to the host over digital lines. Commands are sent from the host to the card and all commands have either no response, a 48 bit response, or a 136 bit response. Some commands may also start a data transfer to or from the card.

There are three types of signal:

- CLK, carrying the clock signal from the host
- CMD, carrying commands from the host and responses from the card
- DAT, carrying data from the host or data from the card



## FPGA Development System User Manual

There may be 1, 4, or 8 DAT lines. SD Cards can run at 0 - 25 MHz in Default Mode or 0 to 50 MHz in High-Speed Mode. MMC cards come in different grades that can run at up to 20, 26, or 52 MHz. No matter what all cards start up in 3.3 V, single DAT, and low speed mode with any other features negotiated during the initialisation.

### 1.2.8.4 Physical layer

All communication are at 3.3 V logic levels with 3.3 V being a high and 0 V being a low. CLK comes from the host and idles low. CMD and DAT are bidirectional and idle high. All are driven in a push/pull mode for speed.

Data is clocked into the host or card on the rising edge of CLK and changes on the falling edge. This is equivalent to the SPI (0, 0) mode.

### 1.2.8.5 Framing

The framing is a bit unusual. It feels like it was written by a embedded software engineer instead of a hardware or protocol engineer as the framing and use of CRCs is unusual and inconsistent. The advantage is that the framing maps through to a software only implementation pretty well.

All transfers start with a zero start bit and finish with a one stop bit. A card may signal that it is still working on the response by keeping the CMD line high until the response is ready.

All commands are 48 bits (6 bytes) long and all responses are either 48 bits (6 bytes) or 136 bits (17 bytes) long. The 48 bit transfers can be thought of as an 8 bit message ID, 32 bit argument, and 8 bit checksum.

Bytes are transferred most significant bit first. Words are transferred most significant byte first.

### 1.2.8.6 Commands and Responses

A command or response has the following format:

| Bit | # | Value                           | Name            |
|-----|---|---------------------------------|-----------------|
| 47  | 1 | 0                               | Start bit       |
| 46  | 1 | 1 for commands, 0 for responses | Transmitter bit |

## FPGA Development System User Manual

|       |    |   |            |
|-------|----|---|------------|
| 45-40 | 6  |   | Command ID |
| 39-8  | 32 |   | Argument   |
| 7-1   | 7  |   | CRC        |
| 0     | 1  | 1 | Stop bit   |

The CRC is a 7 bit CRC with polynomial  $x^7 + x^3 + 1$ . A table driven form can be found in the Linux kernel under [lib/crc7.c](http://lib/crc7.c). Bitwise forms may be generated using [pycrc](http://pycrc) with the parameters

```
--width=7 --poly=9 --reflect-in=0 --reflect-out=0 --xor-out=0 --xor-in=0 --
generate c --algorithm=bit-by-bit-fast
such as
```

```
for (int b = 0; b < 8; b++)
{
    uint bit = crc & 0x40;

    if ((data & 0x80UL) != 0)
    {
        bit ^= 0x40;
    }

    data <<= 1;
    crc <<= 1;

    if (bit != 0)
    {
        crc ^= 0x09;
    }
}
```

Note that the final CRC must be ANDed with 0x7F.

The CRC seed is zero and is calculated over the start, transmitter, command ID, and argument fields. The resulting CRC is compared for equality with the CRC from the message.

Some examples are:

*An APP\_CMD (55) command that prefixes a SD specific command*

Bytes: 0x770000000065

Bits: 0 1 110111 00000000000000000000000000000000 0110010 1



## FPGA Development System User Manual

Fields:

- Start bit = 0
- Transmitter = 1
- Command = 55 (decimal)
- Argument = 00000000
- CRC = 0x32
- Stop bit = 1

The CRC can be generated by feeding 0x77, 0x00, 0x00, 0x00, 0x00 into the CRC function above.

### *The response to the APP\_CMD*

Bytes: 0x370000012083

Bits: 0 0 110111 000000000000000000000000100100000 1000001 1

Fields:

- Start bit = 0
- Transmitter = 0 (this is a response)
- Response = 55 (decimal)
- Argument = 00000120
- CRC = 0x41
- Stop bit = 1

### *The response to CMD3 SEND\_RELATIVE\_ADDR*

Bytes: 0x03B368050019

Bits: 0 0 000011 10110011011010000000010100000000 0001100 1

Fields:

- Start bit = 0
- Transmitter = 0 (this is a response)
- Response = 3 (decimal)
- Argument = 0xB3680500
- CRC = 0x0C
- Stop bit = 1

Note that this cards Relative Card Address (RCA) is 0xB368

## FPGA Development System User Manual

### 1.2.8.7 Data

Data has the following format:

| Bit     | #     | Value | Name      |
|---------|-------|-------|-----------|
| 4113    | 1     | 0     | Start bit |
| 4112-17 | 512*8 |       | Data bits |
| 16-1    | 16    |       | CRC       |
| 0       | 1     | 1     | Stop bit  |

Note that this is for a typical transfer of a block of 512 bytes. The host knows from the command that was sent how many bytes to expect back. There is no other way of knowing the message length.

The CRC is is the ITU-T V.41 16 bit CRC with polynomial 0x1021. A table driven version can be found in the Linux kernel under [lib/crc-itu-t.c](http://lib/crc-itu-t.c). Note that there is no such thing as 'the' CRC16 so make sure you get the right one.

Unlike the commands or responses the CRC is calculated over all of the data bytes and does not include the start bit. The calculated CRC is checked for equality with the received CRC.

PENDING: Add an example data message with CRC.

### 1.2.8.8 Handover

The CMD and DAT lines are bidirectional. Handover occurs at the end of a command where both the host and the card switch to input mode for two clocks before the card starts driving in push/pull mode. Some commands must be responded to in a fixed number of clocks but most allow an arbitrary time before the response must start.

### 1.2.8.9 Initialisation

To initialise a SD or SDHC card, send the following:

- Write 74 clocks with CMD and DAT high
- Write CMD0 GO\_IDLE\_STATE. This will reset the card.
- Write CMD8 SEND\_IF\_COND for 3.3 V parts. If any SDHC cards are

## FPGA Development System User Manual

present then you will get a wired-OR response with 0x3F as the command and 0xFF as the CRC and stop bit. Note that this must be sent or SDHC cards will not respond to the following steps

- Write CMD55 APP\_CMD
- Receive a 55 response
- Write ACMD41 SD\_SEND\_OP\_COND
- Expect a wired-OR response with 0x3F as the command and 0xFF as the CRC and stop bit
- Check the ready bit in the previous response. If the card is not

ready then repeat the CMD55/ACMD41 until it is

- Write CMD2 ALL\_SEND\_CID
- Expect a wired-OR response with 0x3F as the command and 0xFF as the CRC and stop bit
- Write CMD3 SEND\_RELATIVE\_ADDR
- Expect a 3 response. The upper two bytes of the argument is the

Relative Card Address (RCA) which is used in the next step

- Write CMD7 SELECT\_CARD with the RCA
- Expect a 7 response

The card is now selected and ready to transfer data. See Figure 4-1 'SD Memory Card State Diagram' in the simplified spec for more information.

See section 4.7.4 'Detailed Command Description' in the simplified spec for more information on the commands and responses.

MMC cards are initialised using a similar but different method.

An example flow captured from Linux on a SC2440 is:

| Phase | Command          | Response     | Notes |
|-------|------------------|--------------|-------|
| CMD0  | 40000000009<br>5 | None         |       |
| CMD55 | 77000000006<br>5 | 370000012083 |       |

## FPGA Development System User Manual

|                                |                  |                                          |                          |
|--------------------------------|------------------|------------------------------------------|--------------------------|
| ACMD41<br>SEND_OP_COND         | 69001000005<br>F | 3F00FF8000FF                             | Card is<br>busy          |
| CMD55                          | 77000000006<br>5 | 370000012083                             |                          |
| ACMD41                         | 69001000005<br>F | 3F00FF8000FF                             | Card is<br>still<br>busy |
| CMD55                          | 77000000006<br>5 | 370000012083                             |                          |
| ACMD41<br>SEND_OP_COND         | 69001000005<br>F | 3F80FF8000FF                             | Card is<br>ready         |
| CMD2 ALL_SEND_CID              | 42000000004<br>D | 3F1D4144534420202010A0400BC10088AD<br>FF |                          |
| CMD3<br>SEND_RELATIVE_ADD<br>R | 43000100007<br>F | 03B368050019                             | RCA<br>of<br>0xB36<br>8  |

Note the missing CMD8 as this controller does not support SDHC. I didn't capture the final CMD7.

### 1.2.8.10 Reading

Once initialised reading from a card is straight forward.

To read a single page:

- Send CMD17 READ\_SINGLE\_BLOCK with the offset to read from as the argument
- Receive on the DAT lines
- Send CMD12 STOP\_TRANSMISSION once the block has been received

To read consecutive pages:

- Send CMD18 READ\_MULTIPLE\_BLOCK with the starting offset as the argument
- Receive as many blocks as you want on the DAT lines
- Send CMD12 STOP\_TRANSMISSION once the done

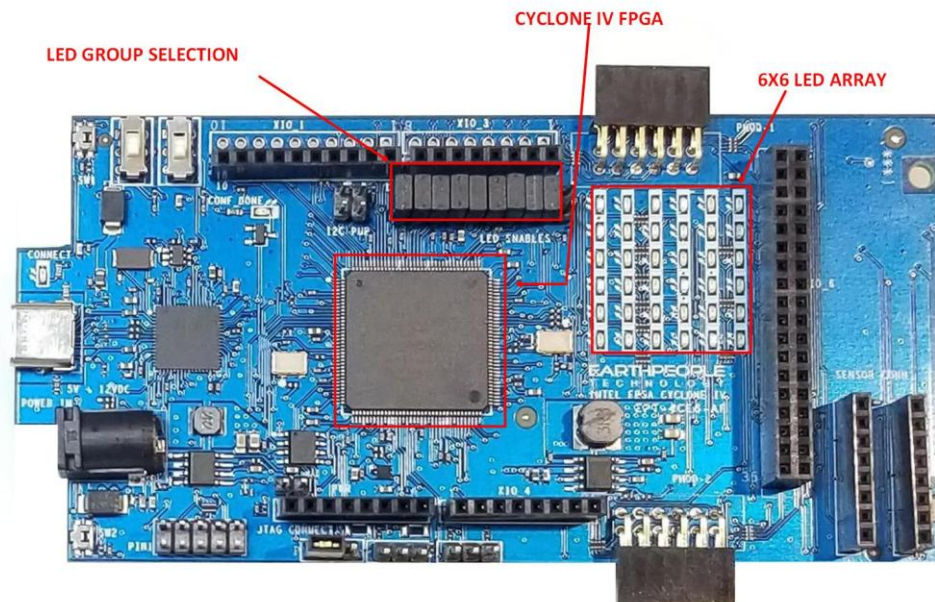
## FPGA Development System User Manual

Note that there will be a response to these commands and the response may be interleaved with the data. See Figure 3-3: (Multiple) Block Read Operation for more information.

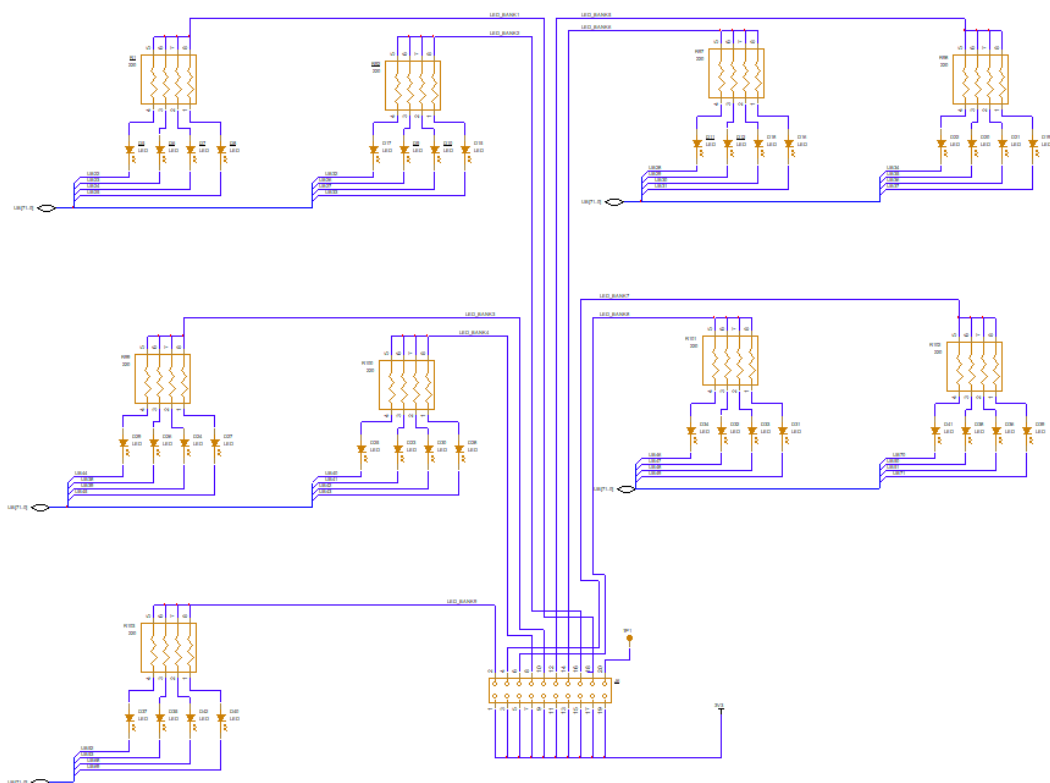
On SDHC cards the offset is in terms of 512 byte blocks. On SD cards the offset is in bytes and the number of bytes received depends on CMD16 SET\_BLOCKLEN

### 1.2.9 LEDs

The DueProLogic includes a 6x6 Green LED array. Each LED is sinked to an individual pin on the FPGA. Each LED is current limited to 6mA. The total current consumed for all 36 LEDs is 216mAs. The FPGA can easily sink this current. So, individually sinking all 36 LEDs makes easy control for User Code. The DueProLogic also contains a method to turn on/off the LEDs in four unit blocks. A jumper is used to control the state of each LED block.

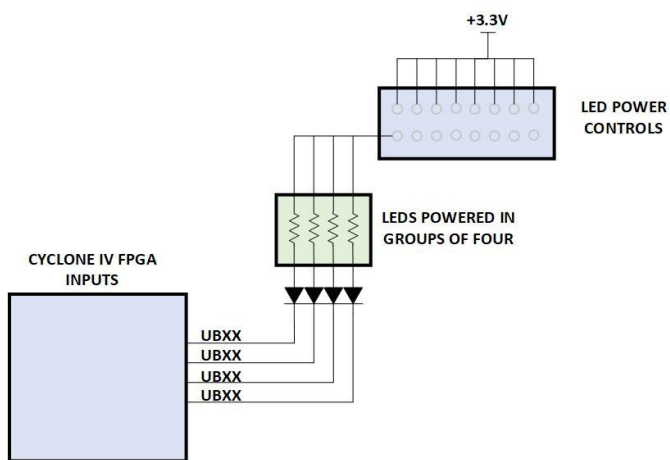


## FPGA Development System User Manual



LED Power Control allows a group of four LEDs to be powered by one jumper. When the jumper is connected, the LEDs are powered by +3.3V. When disconnected, the LEDs will not light up and the IO pin in the appropriate Board Edge Connector can be used without an LED.

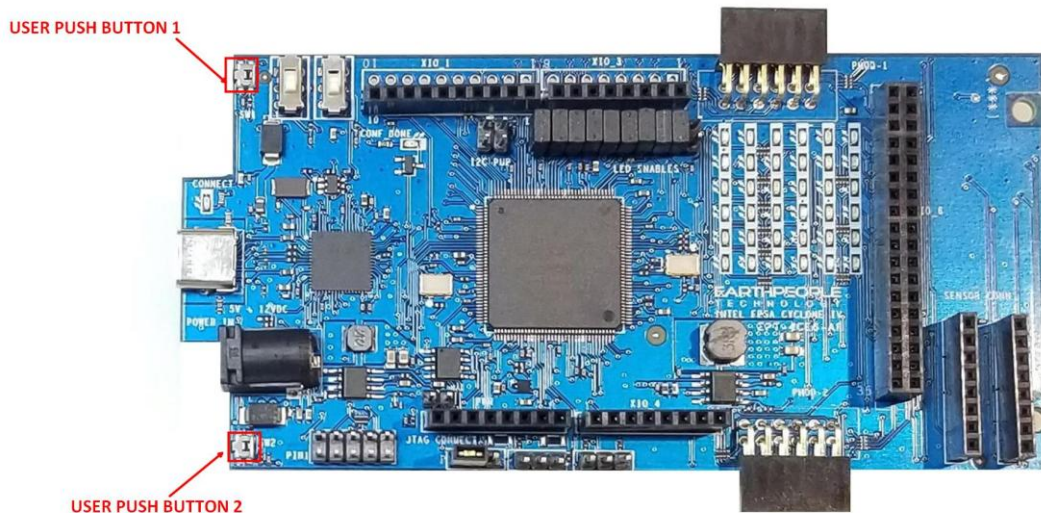
## FPGA Development System User Manual



### 1.2.10 Pushbuttons

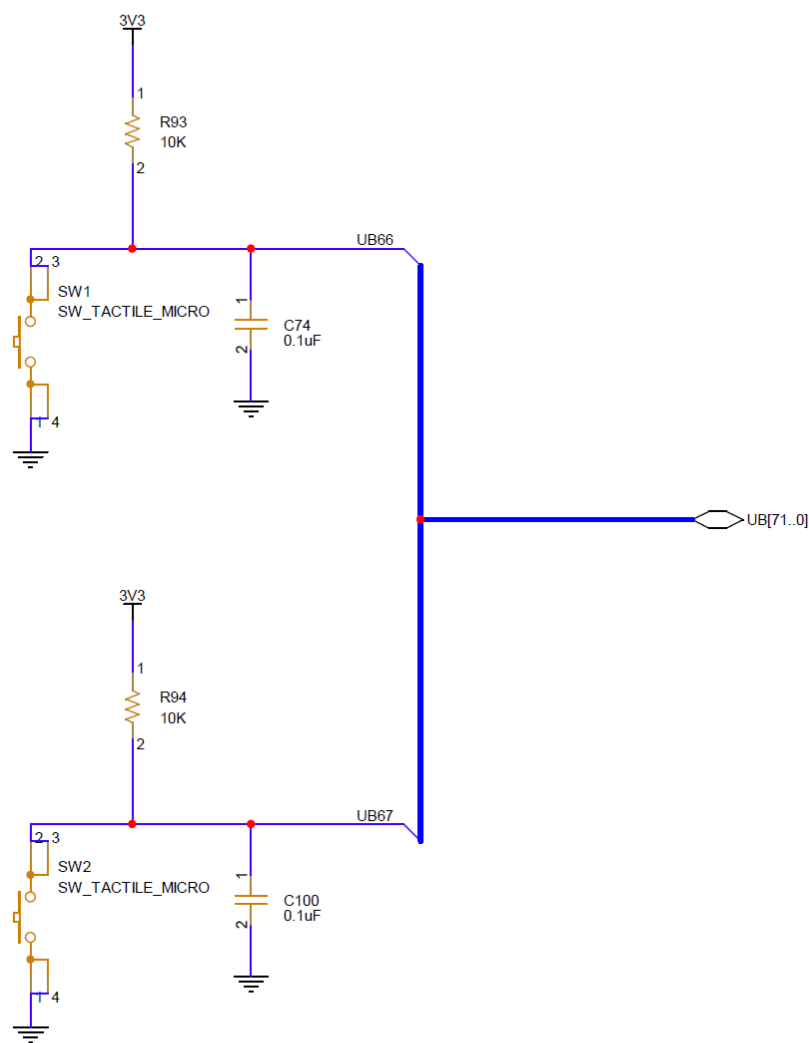
The DueProLogic includes two pushbuttons that can be used in a way the user would like. They are both attached to separate debounce circuits. These pushbuttons were designed to use as little PCB as possible. The switch is engaged by the plunger which is directed off the PCB.

## FPGA Development System User Manual



The switches will momentarily provide a ground connection to the appropriate pin on the FPGA. When Idle, the signals are pulled high through a 10K Ohm resistor.

## FPGA Development System User Manual



Push Button Switches

| Component | Net Name | Pin on Cyclone IV |
|-----------|----------|-------------------|
| SW1       | UB66     | 91                |
| SW2       | UB67     | 90                |

## FPGA Development System User Manual

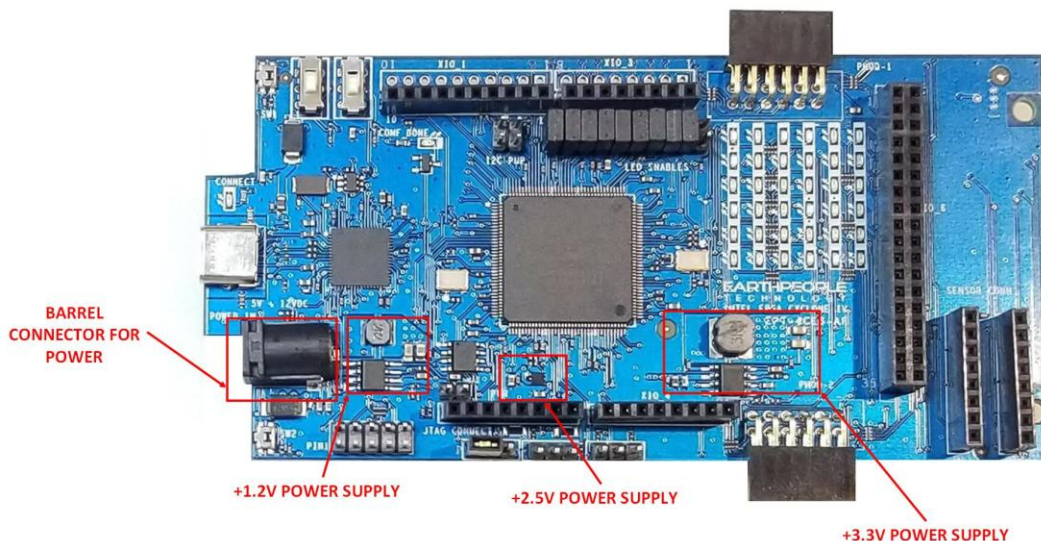
### 1.2.11 Power Options

There are two power options for the DueProLogic

- USB-C Connection to a PC (+5VDC)
- Barrel Connector (+5VDC to +12VDC)

The USB Micro B Connector is a the standard connector specified by the USB SIG.

The Barrel Connector is the located on the front of the board. It has 2.0mm inner diameter and 5.5mm outer diameter. The inner male connection is the positive connection and the outer connection is the ground. This connection can accept between +5VDC and +12VDC power. However, great care must be used when using this connection. This power is applied to several pins on the I/O headers. Consult the Data Sheet for these connections.

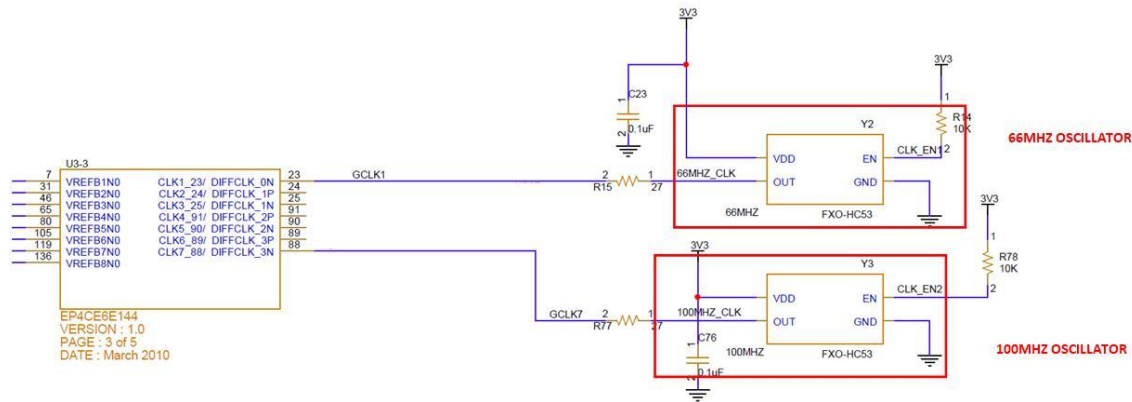


### 1.2.12 Clock Domains

There are two clock domains external to the Cyclone IV FPGA, 66 MHz and 100 MHz. The 66 MHz oscillator is a +3.3VDC device that provides a high speed clock to the FPGA. It is a CMOS device that provides a stable 66 MHz at  $\pm 50$  ppm. This clock can be used directly in the user code or use it as an input to one of the PLL's internal to the FPGA. It is intended that this clock will drive the logic of the user code. If a different

## FPGA Development System User Manual

clock frequency is required in the user code, use the PLL scale up/down to produce the desired clocking.



The 100MHz oscillator is a +3.3VDC device that provides a stable source for high speed clocking. This oscillator can be used for higher speed functions such as fast communications, memories and control.

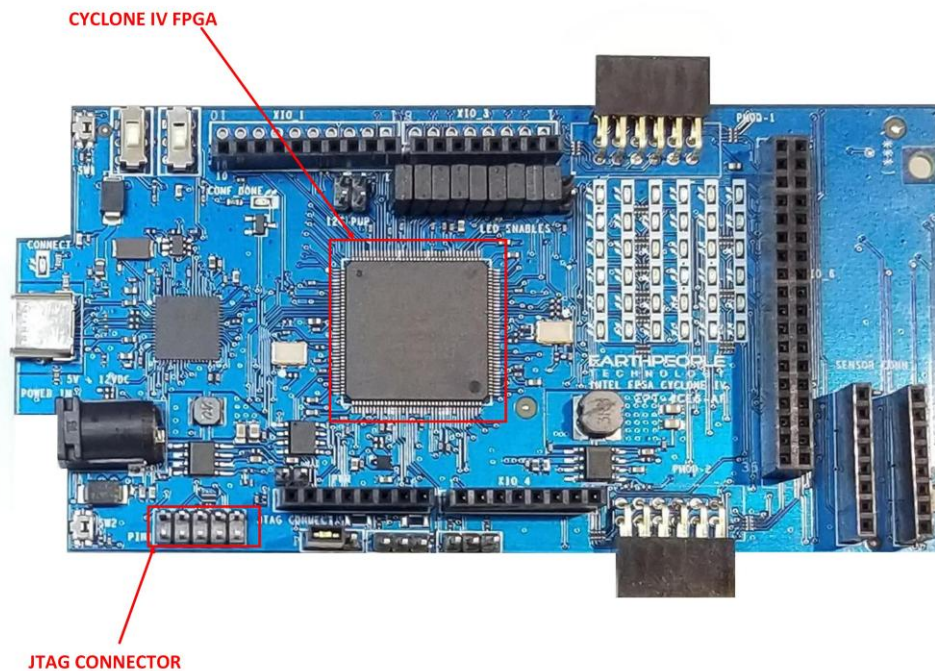
Both oscillators have an enable signal that are individually controlled by the FPGA. The oscillators are operational when the enable signal is high. The oscillators enable signals each have 10K pullups to keep them on at all times.

| Component              | Signal Name | FPGA Pin |
|------------------------|-------------|----------|
| Y2 (66MHz Oscillator)  | 66MHZ_CLK   | 23       |
| Y3 (100MHz Oscillator) | 100MHZ_CLK  | 88       |

## FPGA Development System User Manual

### 1.2.13 JTAG Header

The DueProLogic can be programmed by writing to the on board configuration flash or by directly accessing the FPGA JTAG connection.



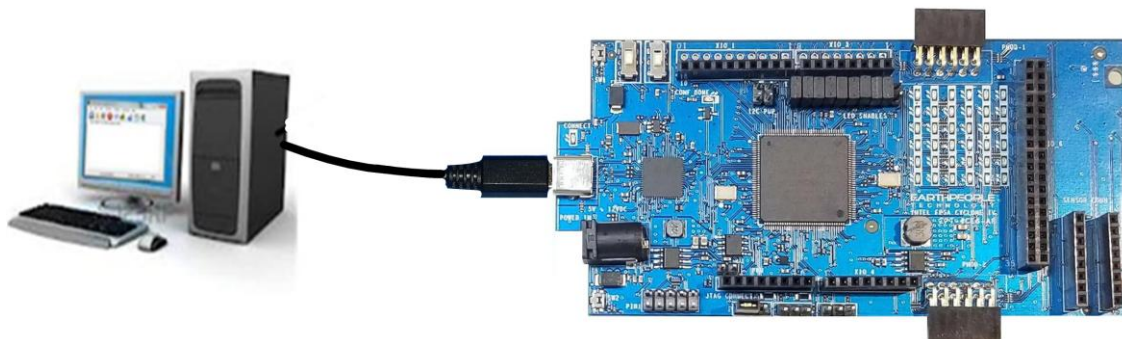
Connector J13 provides access to the FPGA JTAG connections. The pinout follows the USB Blaster arrangement which is standard on all Intel/Altera devices. Be sure to follow the pin 1 orientation.

This connector uses the standard Altera Blaster connector pinout.

|     |   |    |           |
|-----|---|----|-----------|
| TCK | 1 | 2  | GND       |
| TDO | 3 | 4  | VCC(TRGT) |
| TMS | 5 | 6  | NC        |
| NC  | 7 | 8  | NC        |
| TDI | 9 | 10 | GND       |

The VCC(TRGT) is set to +3.3V on the DueProLogic. There are no jumper settings to make in order to program the Cyclone IV FPGA. Just connect a compatible Blaster to the connector and the PC, then use the Quartus software to program the FPGA.

## FPGA Development System User Manual

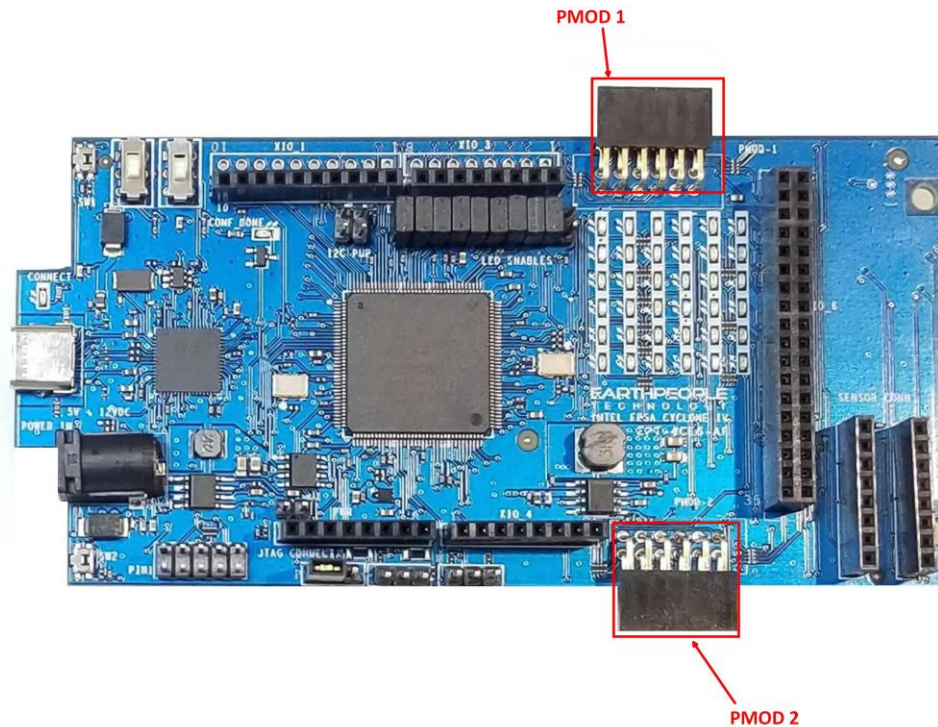


Connect an Intel/Altera compatible programmer to J13, and the FPGA can be programmed directly from Quartus Prime Lite.

### 1.2.14 PMOD Connectors

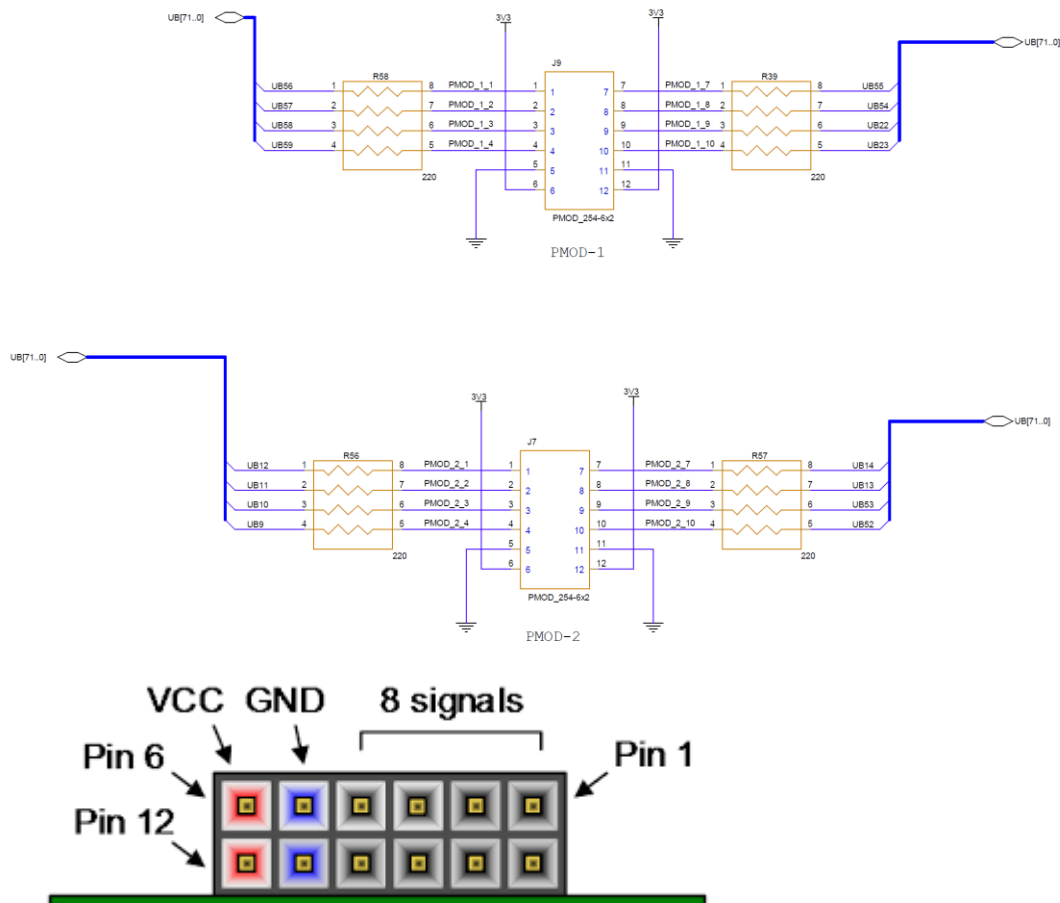
The DueProLogic includes two PMOD Connectors. These two connectors are located towards the rear of the board.

## FPGA Development System User Manual

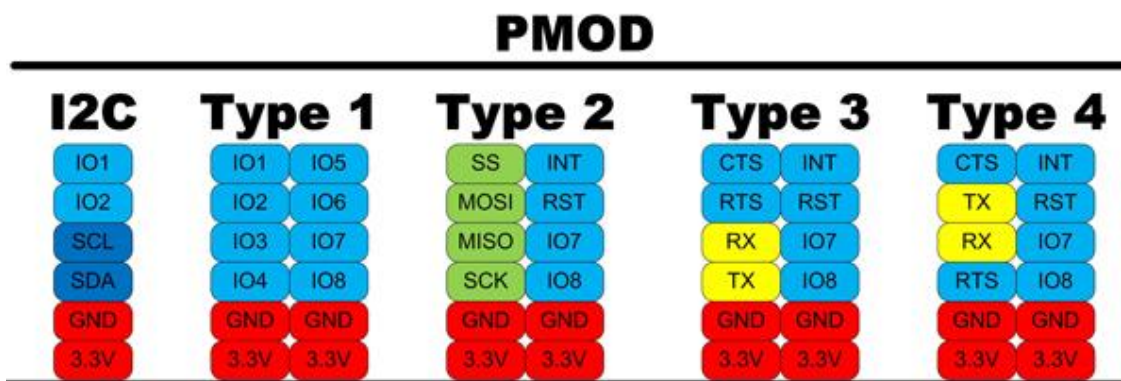


The PMOD pinouts follows the standard pinout. Pin 1 is located in the upper right when facing the connector.

## FPGA Development System User Manual



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.



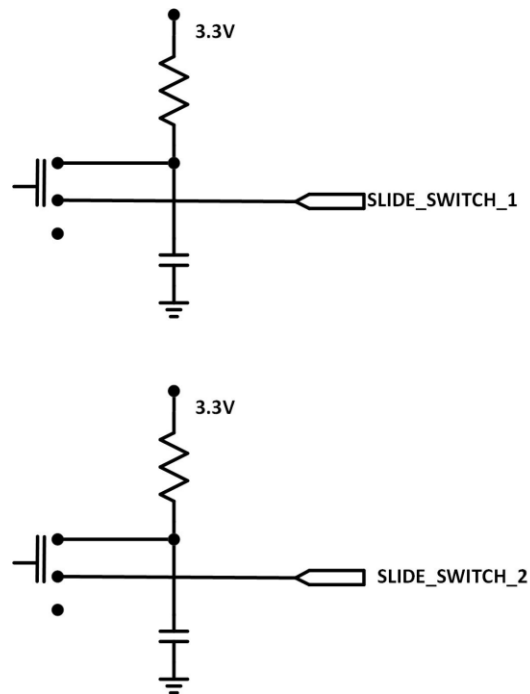
The PMOD have the following connections to the Cyclone IV chip:

| PMOD Pin Number | Signal Name | Cyclone IV Pin Number |
|-----------------|-------------|-----------------------|
| 1-1             | UB56        | 121                   |
| 1-2             | UB57        | 124                   |
| 1-3             | UB58        | 125                   |
| 1-4             | UB59        | 126                   |
| 1-7             | UB55        | 120                   |
| 1-8             | UB54        | 119                   |
| 1-9             | UB22        | 115                   |
| 1-10            | UB23        | 114                   |
| 2-1             | UB12        | 67                    |
| 2-2             | UB11        | 66                    |
| 2-3             | UB10        | 53                    |
| 2-4             | UB9         | 52                    |
| 2-7             | UB14        | 55                    |
| 2-8             | UB13        | 54                    |
| 2-9             | UB53        | 69                    |
| 2-10            | UB52        | 70                    |

### 1.2.15 Slide Switches

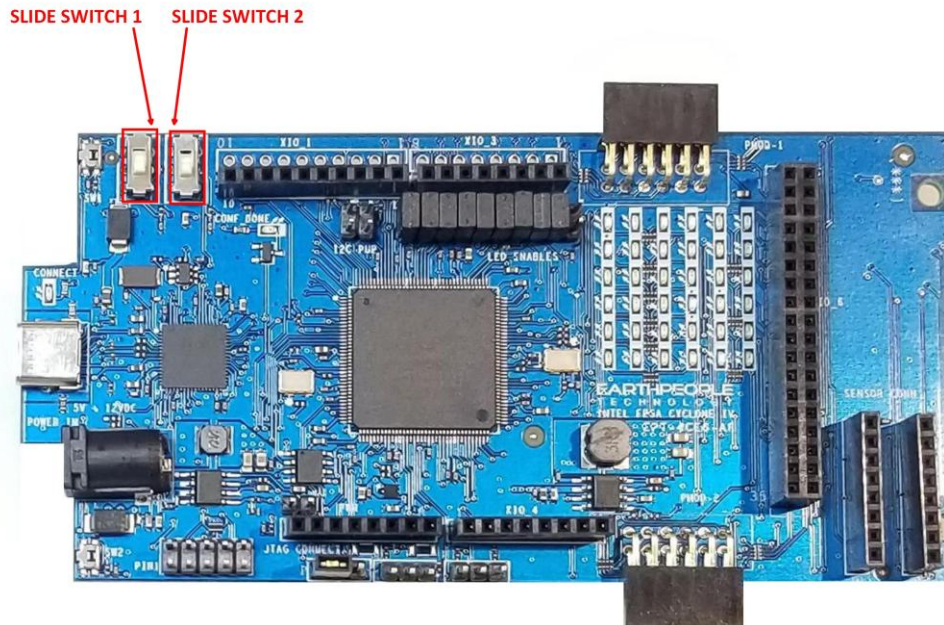
The DueProLogic includes two slide switches. Both are full contact switches. They include a 1uF cap to ground to debounce both switches.

## FPGA Development System User Manual



The slide switches provide a connection to either +3.3V in position 1 or Ground in position 2.

## FPGA Development System User Manual



| Component | Net Name       | Pin on Cyclone IV |
|-----------|----------------|-------------------|
| SW3       | SLIDE_SWITCH_1 | 91                |
| SW4       | SLIDE_SWITCH_2 | 90                |

### 1.3 FPGA Active Host Development

The DueProLogic comes complete with step by step instructions on building an entire communications system from FPGA to Windows Host. Using the EPT Active Host SDK provides an easy to use programming interface. The tools required are the free Visual Studio IDE and the free Quartus Prime.

The DueProLogic also comes with instructions on how to build a communications systems between the Arduino Due and the DPL. The DVD included in the kit has all source files, compiled projects and user manuals to assist the user in using the software. In order to assemble a full communications system between the PC and the DPL, the following software items are required:

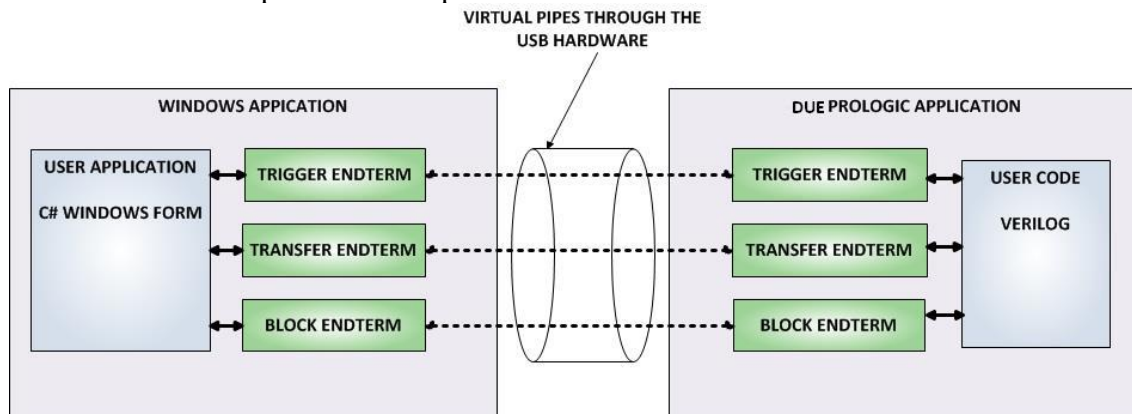
## FPGA Development System User Manual

- Active Host SDK – This library provides the communication mechanism on the PC
- Active Transfer library – This library is included and synthesized into the users code on the FPGA

These two software components provide a communication mechanism by talking to each other over an item called “EndTerms”. EndTerms is the name given to the virtual “pipes” provided by the above libraries. The next two sections provides a brief introduction to EndTerms. Following sections will provide greater detail on using EndTerms.

### 1.4 Active Host EndTerms

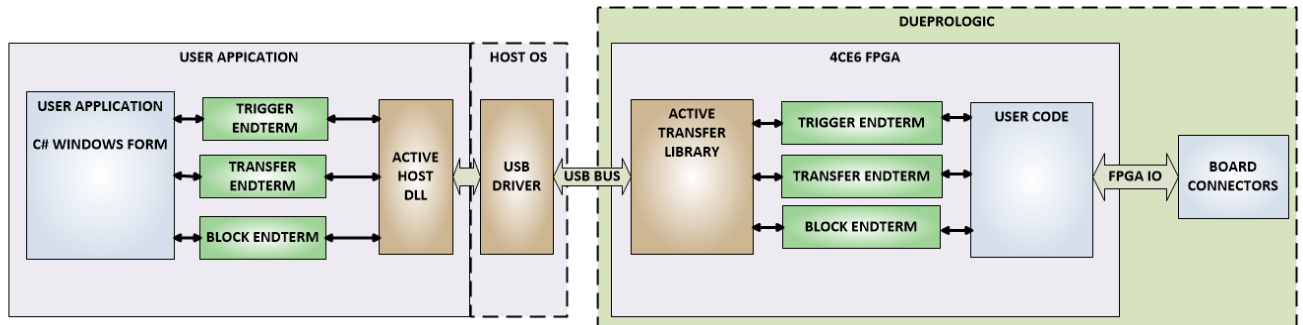
The Active Host SDK is provided as a set of functions, including the COM Port functionality, which easily interfaces to application software written in C#, C++ or C. It runs on the PC and provides transparent connection



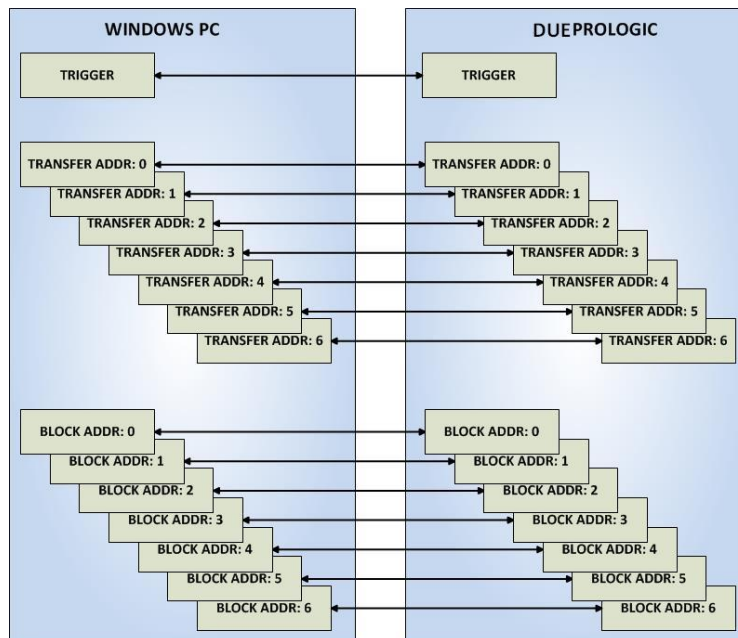
from PC/Windows application code through the USB driver to the user FPGA code. The user code connects to “Endterms” in the Active Host SDK. These Host “Endterms” have complementary HDL “Endterms” in the Active Transfer Library. Users have seamless bi-directional communications at their disposal in the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

- Trigger Endterm
- Transfer Endterm
- Block Endterm

## FPGA Development System User Manual

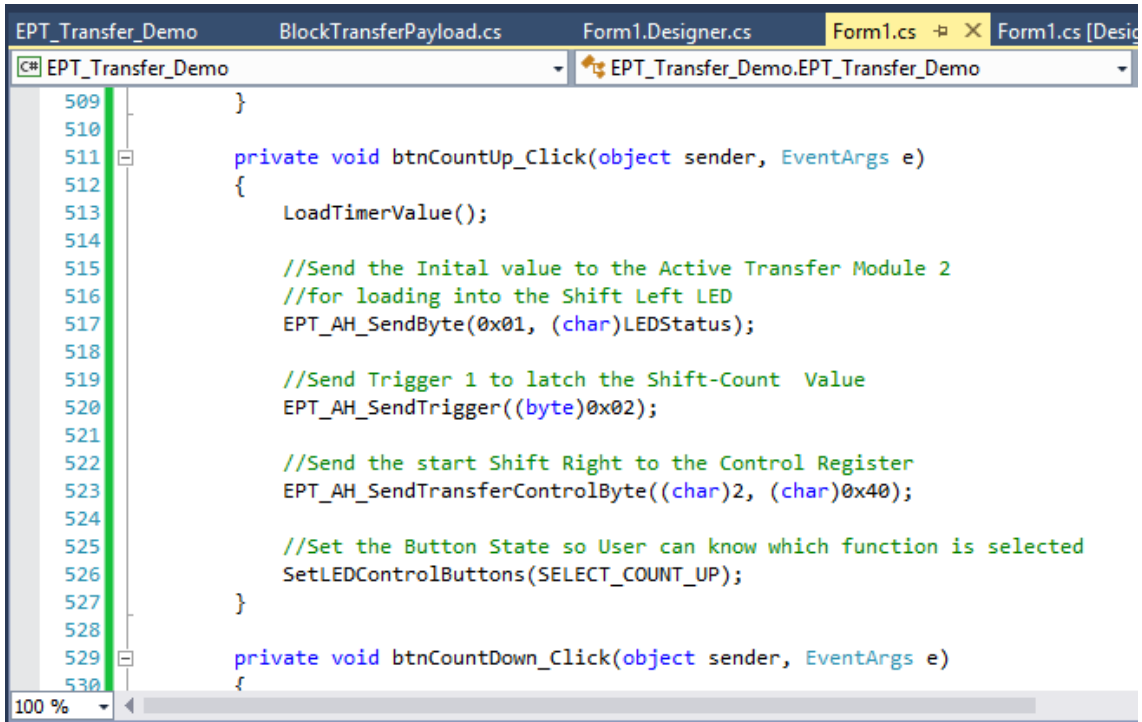


User code writes to the Endterms as function calls. Just include the address of the individual module (there are eight individually addressable modules of each Endterm).



Immediately after writing to the selected Endterm, the value is received at the HDL Endterm in the FPGA.

## FPGA Development System User Manual



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

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

## FPGA Development System User Manual

```

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

```

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

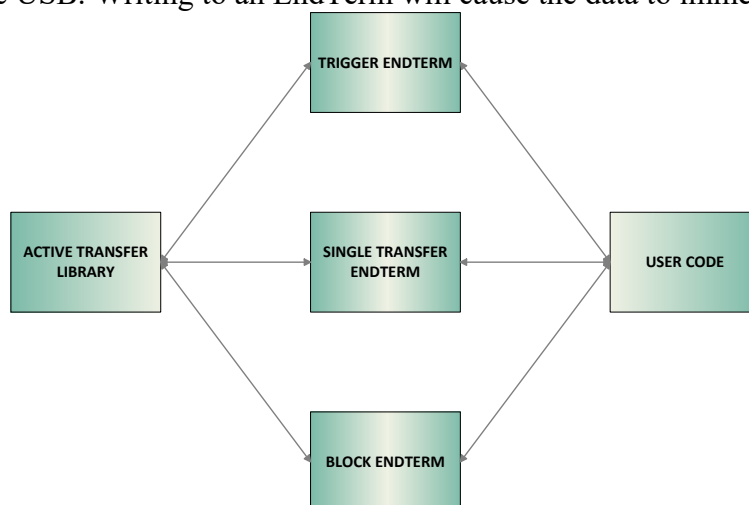
Receiving data from the FPGA is made simple by Active Host. Active Host transfers data from the FPGA as soon as it is available. It stores the transferred data into circular buffer. When the transfer is complete, Active Host invokes a callback function which is registered in the users application. This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

### 1.5 Active Transfer EndTerms

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

- Trigger Endterm
- Transfer Endterm
- Block Endterm

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



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

## 2 EPT Drivers

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

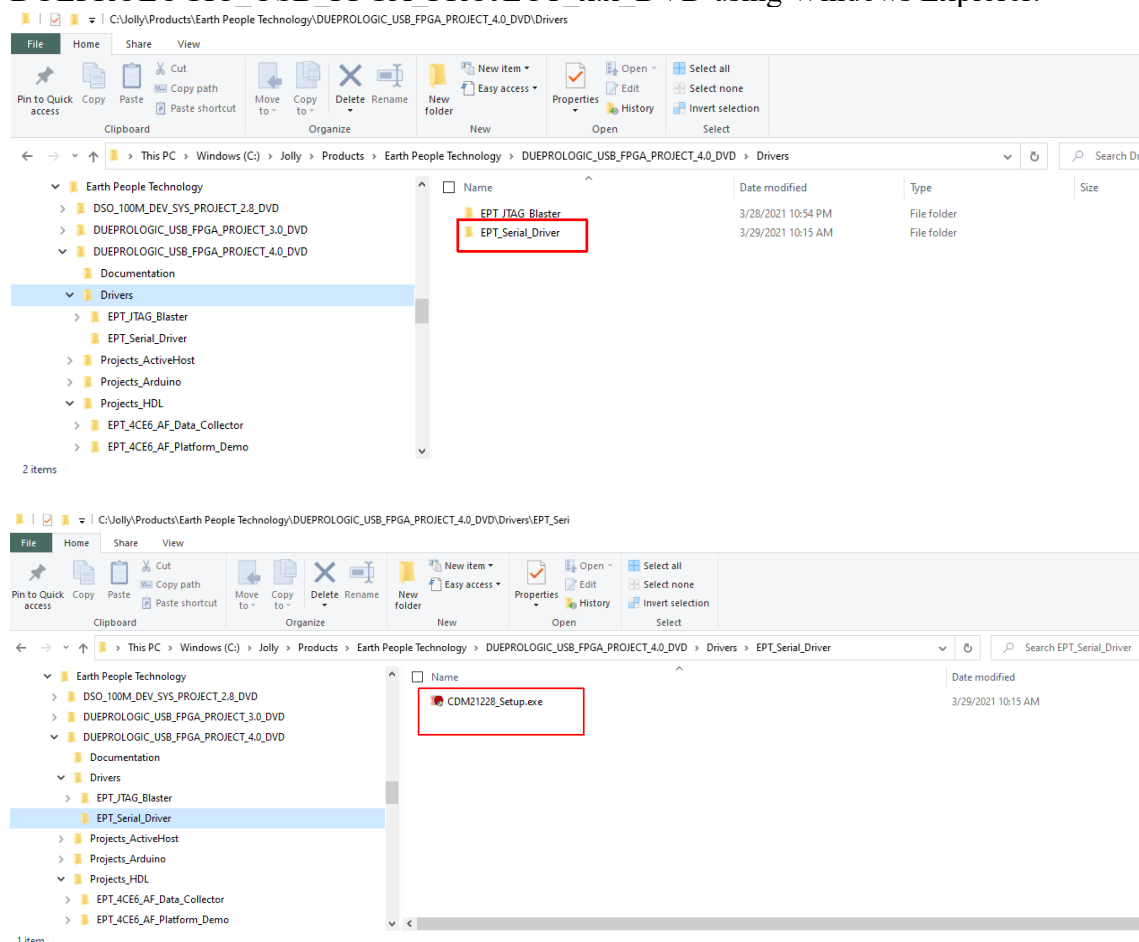


## FPGA Development System User Manual

### 2.1 USB Driver

The DueProLogic uses an FTDI FT2232H USB to Serial chip. This chip provides the USB interface to the PC and the serial/FIFO interface to the FPGA. The FT2232H requires the use of the FTDI USB driver. To install the driver onto your PC, use the EPT\_Serial\_Driver Folder. The installation of the FTDI 2.12.28 driver is easily accomplished by double clicking the CDM21228\_Setup.exe.

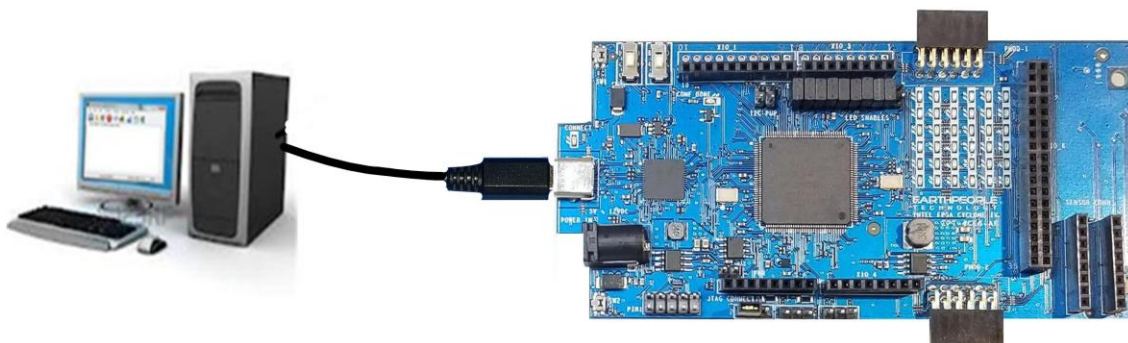
Locate the EPT\_Serial\_Driver folder in the Drivers folder of the DUEPROLOGIC\_USB\_FPGA\_PROJECT\_x.x\_DVD using Windows Explorer.



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

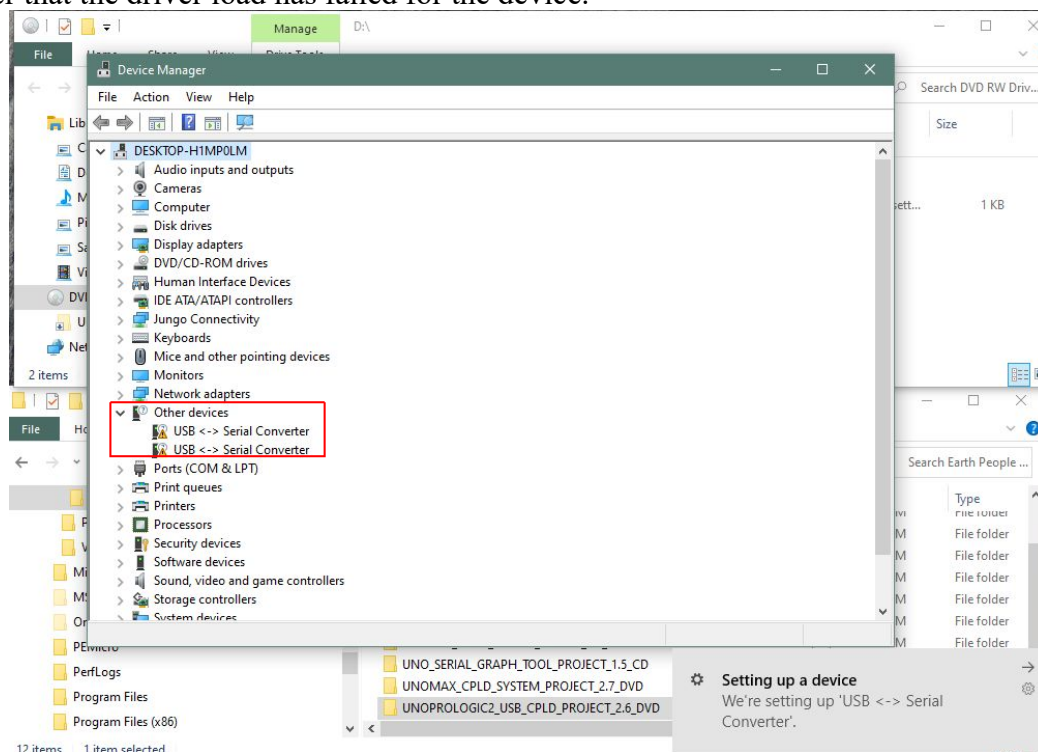
## FPGA Development System User Manual

Plug in the DueProLogic into an available USB port.



Windows will attempt to locate a driver for the USB device. Allow Windows to install the driver for the DueProLogic.

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





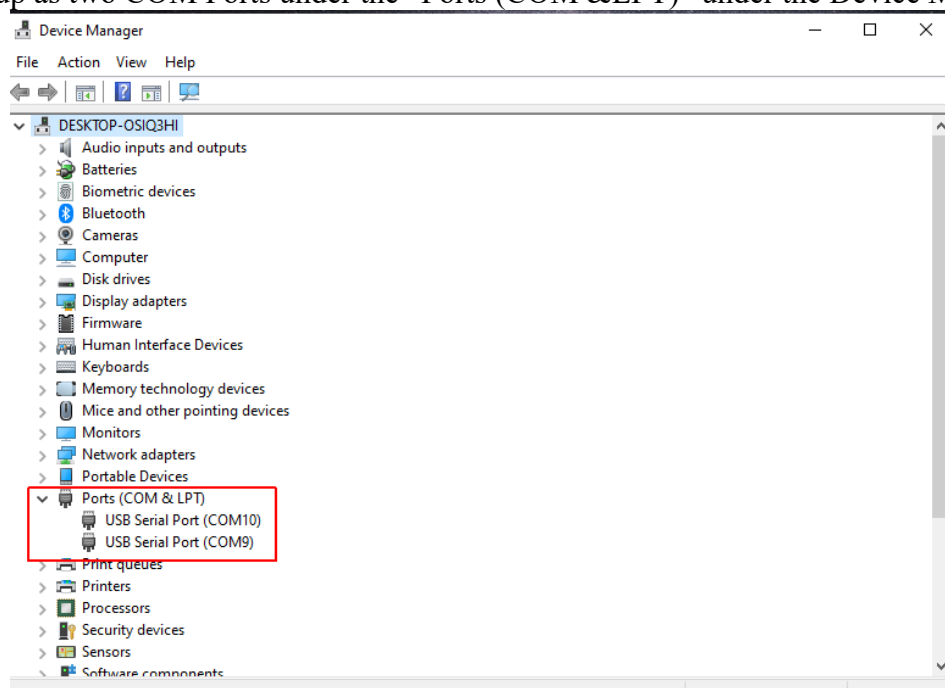
## FPGA Development System User Manual

Right click on the “USB <-> Serial Converter” icon that failed to load. Then select the “Load driver from Known Location”. Navigate to the DUEPROLOGIC\_USB\_FPGA\_PROJECT\_x.x\_DVD/drivers/EPT\_Serial\_Driver folder. Allow the Windows PC to locate and install the driver.

If this method continues to fail, contact support at Earth People Technology using the following methods:

[www.earthpeopletechnology.com/Forums](http://www.earthpeopletechnology.com/Forums)  
[support@earthpeopletechnology.com](mailto:support@earthpeopletechnology.com)  
[sales@earthpeopletechnology.com](mailto:sales@earthpeopletechnology.com)

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 DPL. The DPL 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 DueProLogic can be used for programming and USB data transfers.



## FPGA Development System User Manual

### ***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 DueProLogic. The DueProLogic can then be selected and perform programming of the FPGA. The file, jtag\_hw\_mbftdi\_blaster.dll must be placed into the folder that hosts the jtag\_server for Quartus.

The following are the steps to use Quartus with the DueProLogic:

- Install Quartus (Download from Altera)
- Install \*.dll (Use the software installer from Earth People Technology)

#### **2.2.1 Installing Quartus**

You must install Quartus Prime to configure the DueProLogic. Altera Quartus Prime must be downloaded from the Altera website.

Download the Quartus Prime by following the directions in the Section Downloading Quartus.

#### **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](#)

**\*\*Please be advised, Intel is prone to changing their website graphics every few months. The guide below is several months out of date. Please follow the latest instructions on the Quartus Prime Lite Download website. The instructions here are only meant as a guide to getting the software downloaded.\*\***

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.



## FPGA Development System User Manual

intel

PRODUCTS SUPPORT SOLUTIONS DEVELOPERS PARTNERS FOUNDRY

Search Intel.com

FPGA Software Download Center

### Intel® Quartus® Prime Lite Edition Design Software Version 22.1 for Windows

| ID     | Date      | Software Type    | Software Package    | Version | Operating Systems |
|--------|-----------|------------------|---------------------|---------|-------------------|
| 757262 | 11/4/2022 | FPGA Development | Quartus® Prime Lite | 22.1    | Windows           |

A newer version of this software is available, which includes functional and security updates. Customers should [click here](#) to update to the latest version.

Feedback

The Intel® Quartus® Prime Lite Edition Design Software, Version 22.1 includes functional and security updates. Users should keep their software up-to-date and follow the technical recommendations to help improve security. Additional security updates are planned and will be provided as they become available. Users should promptly install the latest version upon release.

Intel® Quartus® Prime Lite Edition Design Software, Version 22.1 is subject to removal from the web when support for all devices in this release are available in a newer version, or all devices supported by this version are obsolete. If you would like to receive customer notifications by e-mail, please subscribe to our [subscribe to our customer notification mailing list](#).

If you are using floating license server for Intel FPGA software, you need to upgrade to the latest license daemon software (v11.18.2.0). Intel FPGA software, version 22.1, doesn't work with old version of daemon software. You can download the daemon software from [this link](#).

To find software versions that support specific device families:

- Refer to the [Device Support List](#)
- Use the [Software Selector](#)

Critical Issues and Patches for the Intel® Quartus® Prime Lite Edition Software, Version 22.1.  
[Knowledge Base: Search for Errata.](#)

### Scroll down the page to the “Downloads” section.

Critical Issues and Patches for the Intel® Quartus® Prime Lite Edition Software, Version 22.1.  
[Knowledge Base: Search for Errata.](#)  
[Problems and Answers on specific IP or Products.](#)

Downloads

Multiple Download Individual Files Additional Software Copyleft Licensed Source

#### Multiple Download

Intel® Quartus® Prime Lite Edition Software (Device support included)

Download

Quartus-lite-22.1std.0.915-windows.tar

Size: 5.5 GB

SHA1: 86cd25b014999bbb4c2f0a38bfc3442438759d4

\*\* Nios® II EDS on Windows requires Ubuntu 18.04 LTS on Windows Subsystem for Linux (WSL), which requires a manual installation.  
\*\* Nios® II EDS requires you to install an Eclipse IDE manually.  
\*\* Total space required is 26.10 GB including tar file (5.48 GB), untarred files (5.48 GB) and installation (15.13 GB)

**What's Included?**

**Download and install instructions:**

1. Download the software .tar file and the appropriate device support files.
2. Extract the files into the same temporary directory.
3. Run the setup.bat file.

[Read Intel® FPGA Software Installation FAQ](#)

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

**Detailed Description**

**System Requirements:**  
[Operating System Support](#)

Minimum Disk Space for Intel® FPGA Software

Click on the “Download Quartus-lite-22.1 xxxxxx Windows Tar.



## FPGA Development System User Manual

Critical Issues and Patches for the Intel® Quartus® Prime Lite Edition Software, Version 22.1.  
Knowledge Base: Search for Errata.  
Problems and Answers on specific IP or Products.

### Downloads

[Multiple Download](#) [Individual Files](#) [Additional Software](#) [Copyleft Licensed Source](#)

#### Multiple Download

**Intel® Quartus® Prime Lite Edition Software (Device support included)**

**Download**  
Quartus-lite-22.1std.0.915-windows.tar

Size: 5.5 GB  
SHA1: 86cd25b014999bbb4c2f0a38bfc3442438759d4

\*\* Nios® II EDS on Windows requires Ubuntu 18.04 LTS on Windows Subsystem for Linux (WSL), which requires a manual installation.  
\*\* Nios® II EDS requires you to install an Eclipse IDE manually.  
\*\* Total space required is 26.10 GB including tar file (5.48 GB), untarred files (5.48 GB) and installation (15.13 GB)  
[What's Included?](#)

#### Download and install instructions:

1. Download the software .tar file and the appropriate device support files.
2. Extract the files into the same temporary directory.
3. Run the setup.bat file.

[Read Intel® FPGA Software Installation FAQ](#)

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

#### Detailed Description

##### System Requirements:

[Operating System Support](#)

[Minimum Disk Space for Intel® FPGA Software](#)

Click through the Legal Stuff.



## FPGA Development System User Manual



### Software License Agreement

#### Legal Disclaimer

**PLEASE NOTE:** This version of software ("Software") does not contain the latest functional and security updates. In order to use this version, you must first acknowledge the following term, which supplements and supersedes any inconsistent provision in the version of the Intel® FPGA Software License Subscription Agreement for the product (e.g., Intel® Quartus® Prime Software, Intel® HLS Compiler, Intel® FPGA SDK for OpenCL™, DSP Builder for Intel® FPGAs, or Advanced Link Analyzer) with which you use the Software:

Intel does not give or enter into any condition, warranty, or other term:

- i. with respect to any malfunctions or other errors in its Software caused by virus, infection, worm or similar malicious code not developed or introduced by Intel; or
- ii. to the effect that any Software will protect against all possible security threats, including intentional misconduct by third parties. Intel is not liable for any downtime or service interruption, for any lost or stolen data or systems, or for any other damages arising out of or relating to any such actions or intrusions or resulting from use of Software. Intel does not give or enter into any condition, warranty, or other term with respect to interoperability.

Intel does not warrant or assume responsibility for the accuracy or completeness of any information, text, graphics, links or other items within the Software. Please click "Accept" below to continue the download process.

Accept

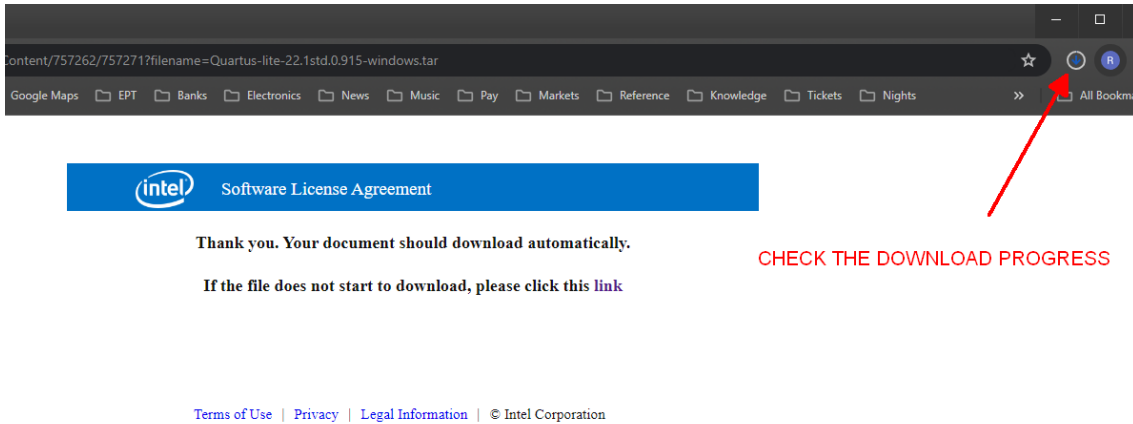
Reject

[Terms of Use](#) | [Privacy](#) | [Legal Information](#) | © Intel Corporation

This will start the download.



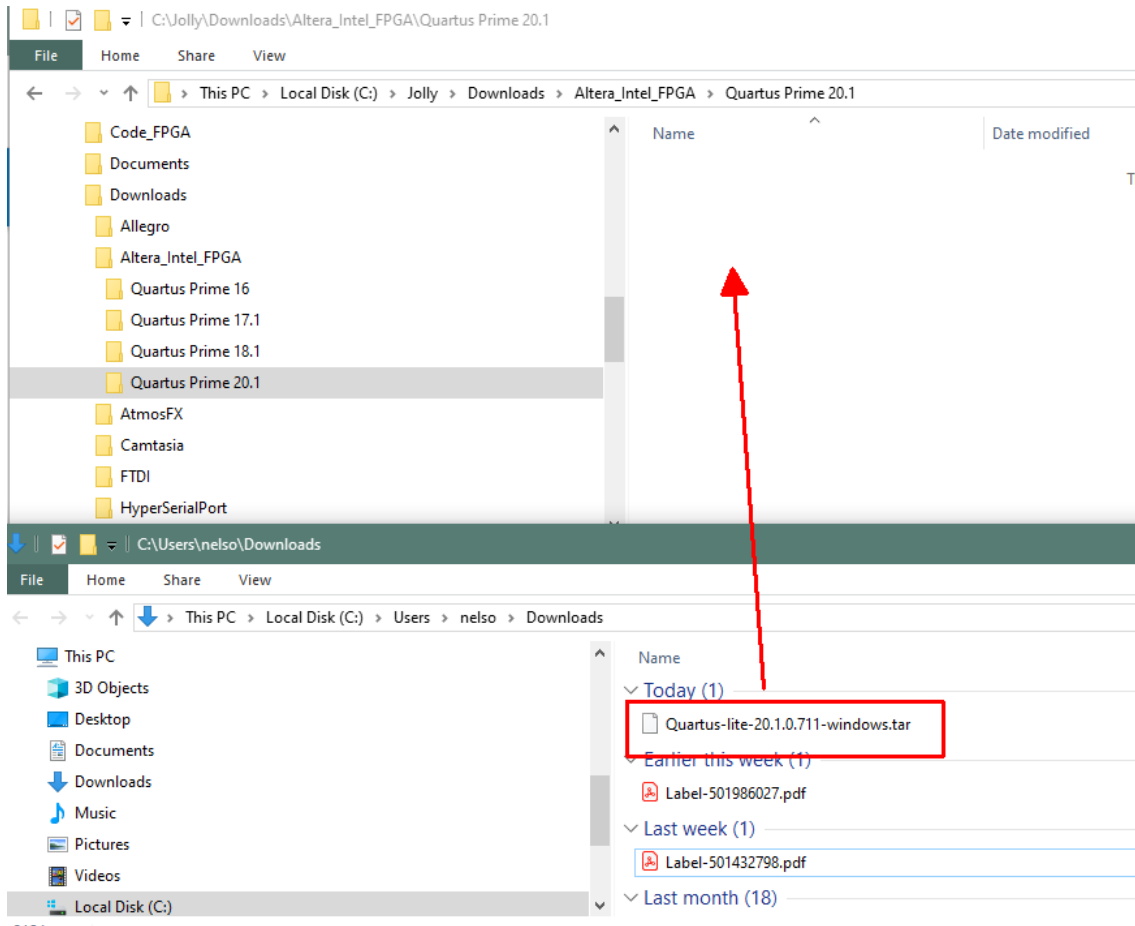
## FPGA Development System User Manual



CHECK THE DOWNLOAD PROGRESS

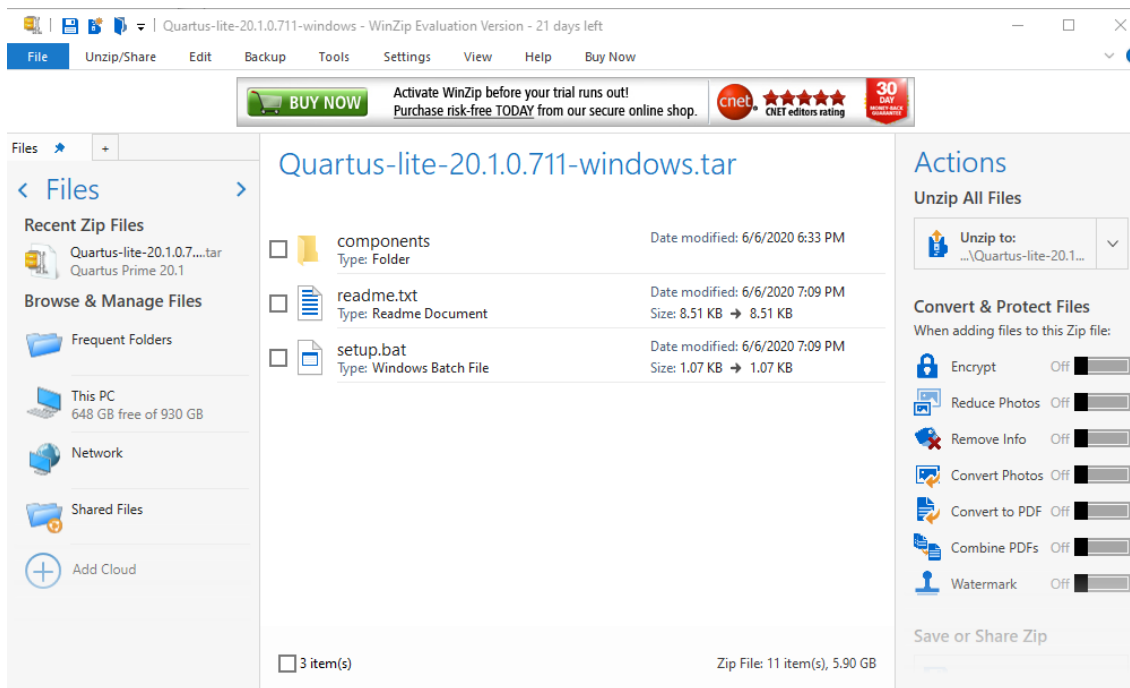
The file is 13.9 GB (or greater), so this could take a couple of hours depending on your internet connection. When download is complete, store the \*.tar file in a directory on your PC.

## FPGA Development System User Manual

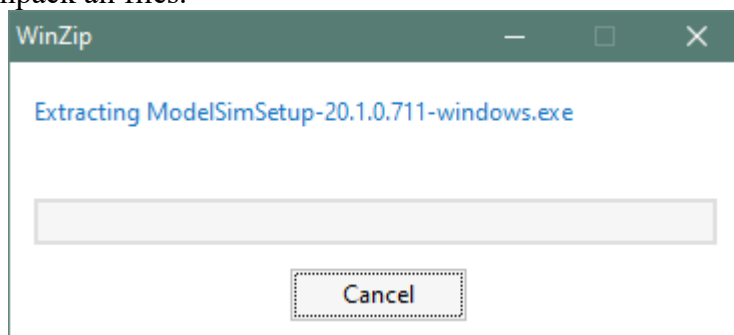


Use a tool such as WinZip to Extract the \*.tar file.

## FPGA Development System User Manual



The tool will unpack all files.

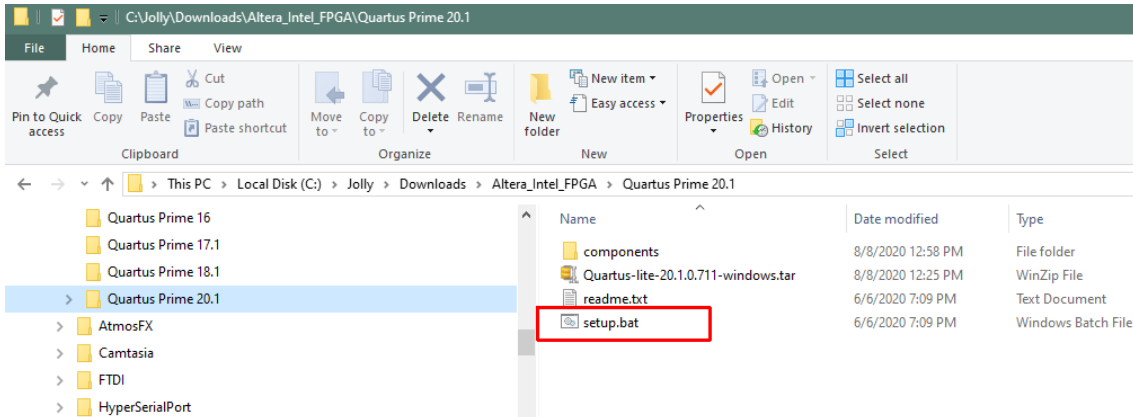


### 2.2.3 Quartus Installer

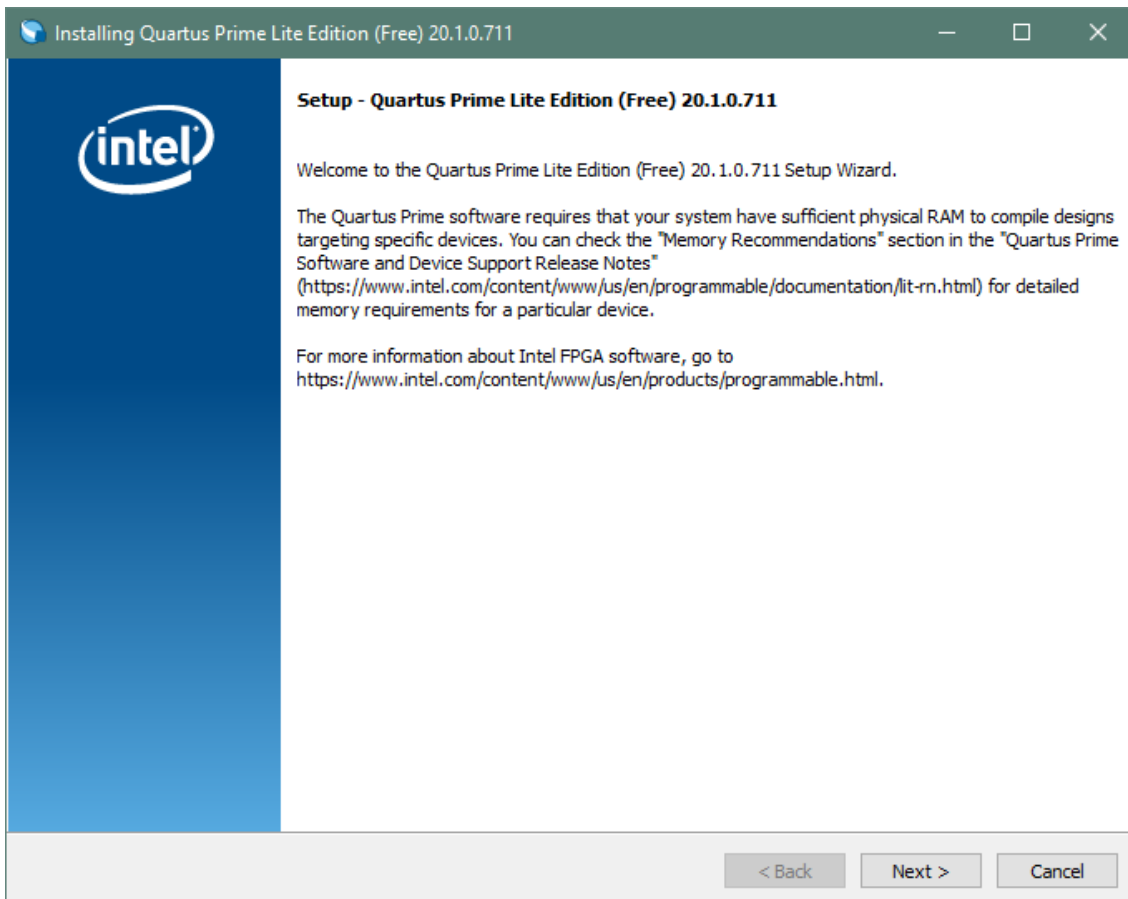
When the unpacking finishes from the previous section, double click the setup.bat file in the download folder.



## FPGA Development System User Manual



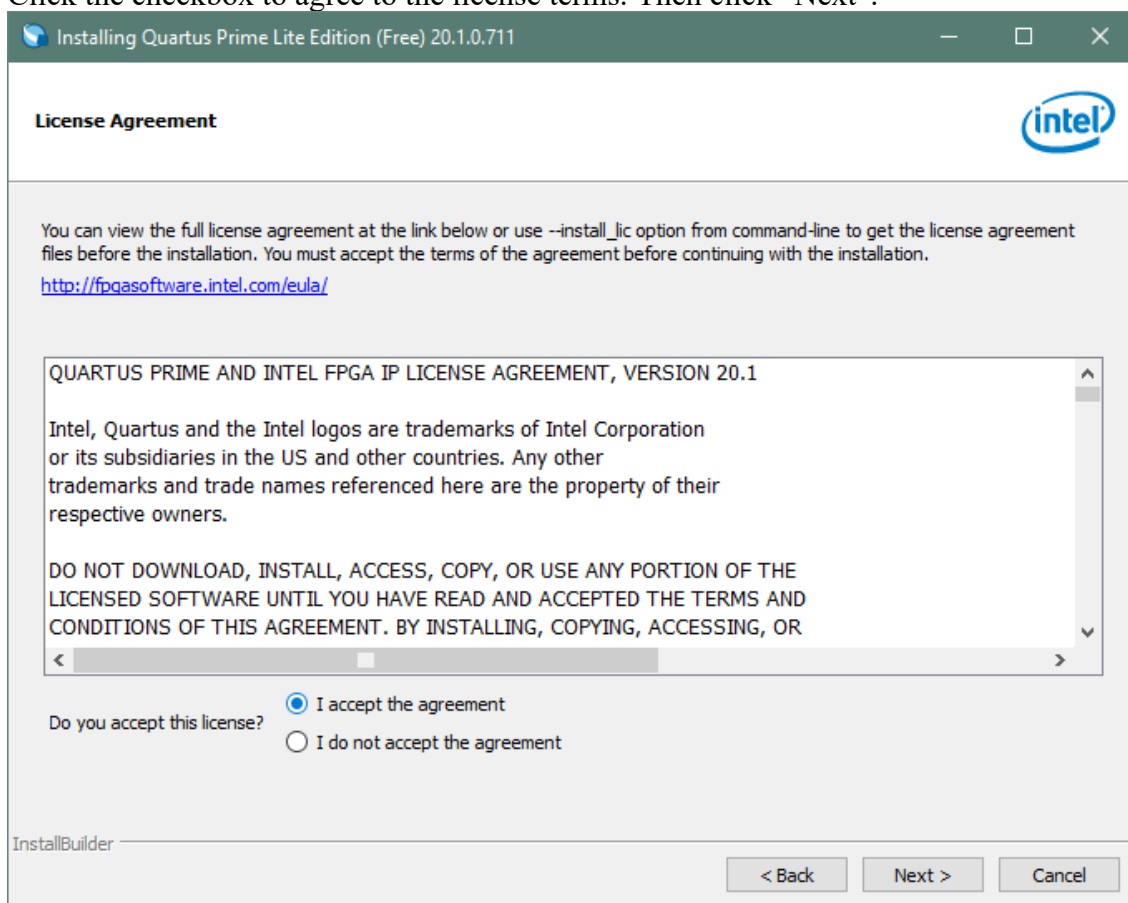
Click “Next” on the Introduction Window.





## FPGA Development System User Manual

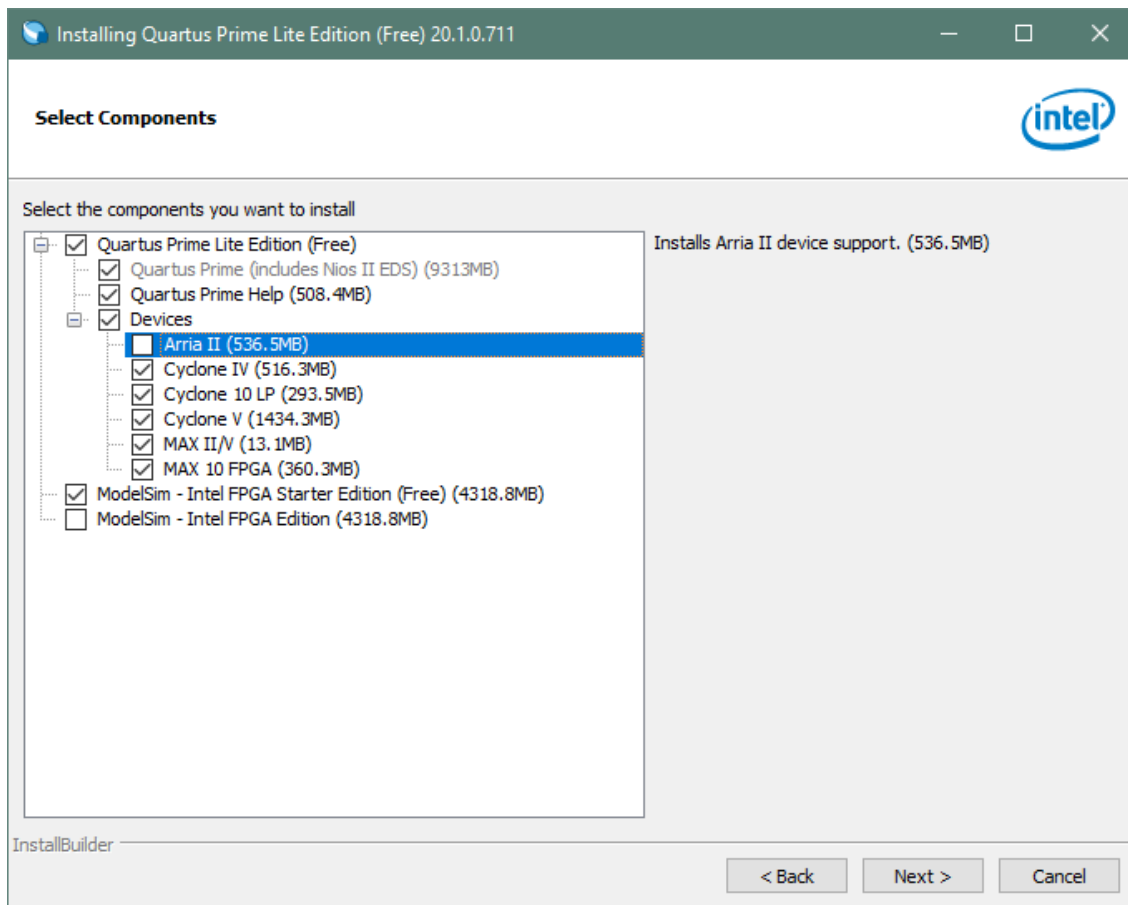
Click the checkbox to agree to the license terms. Then click “Next”.



Click “Next” and accept the defaults.

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

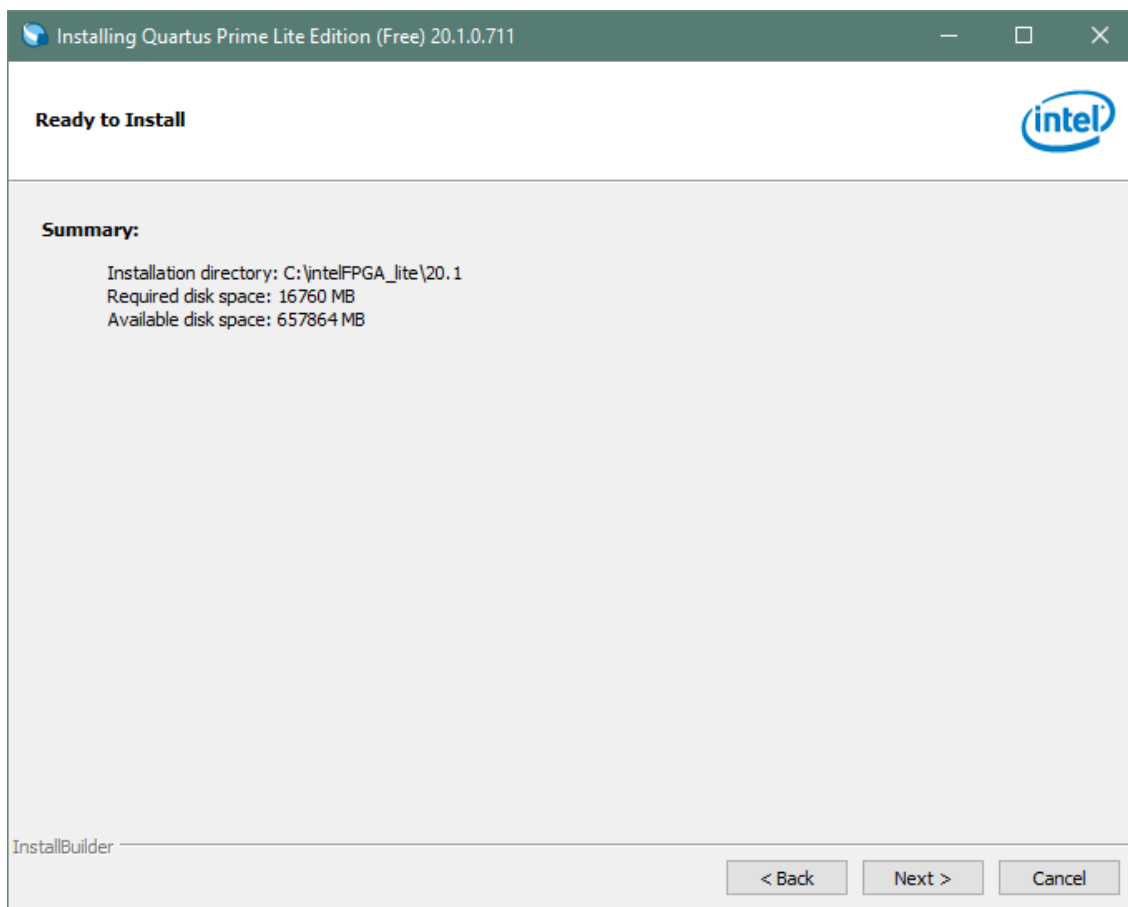
## FPGA Development System User Manual



Click “Next” to accept the defaults



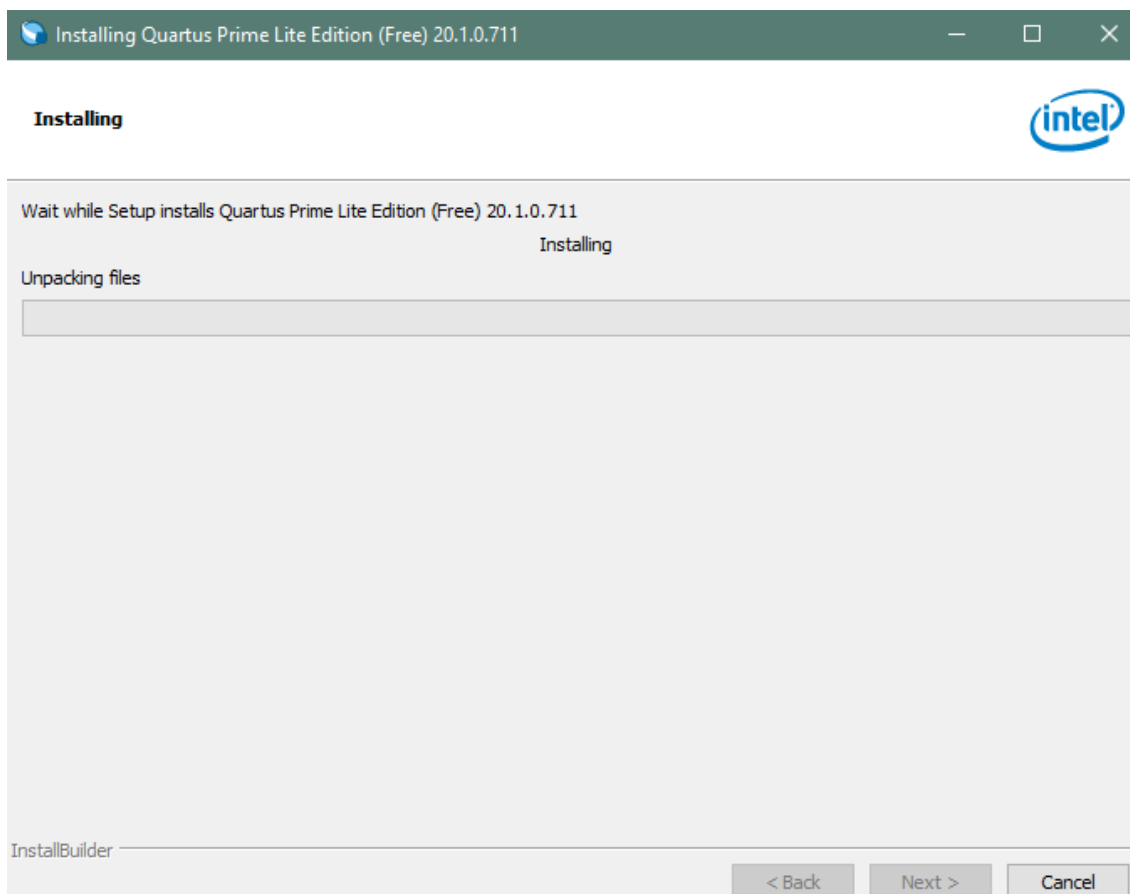
## FPGA Development System User Manual



Click “Next” to accept the defaults

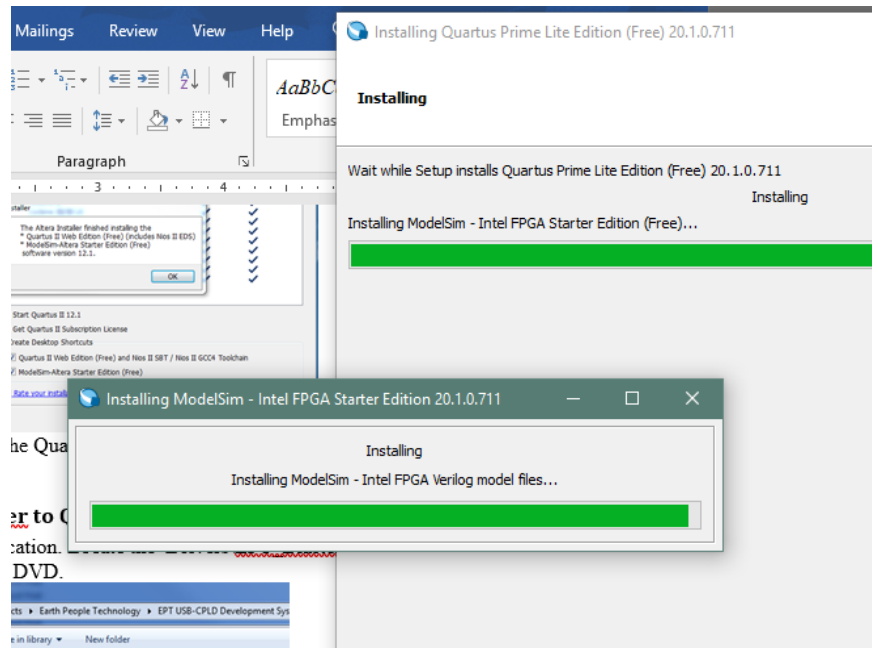


## FPGA Development System User Manual

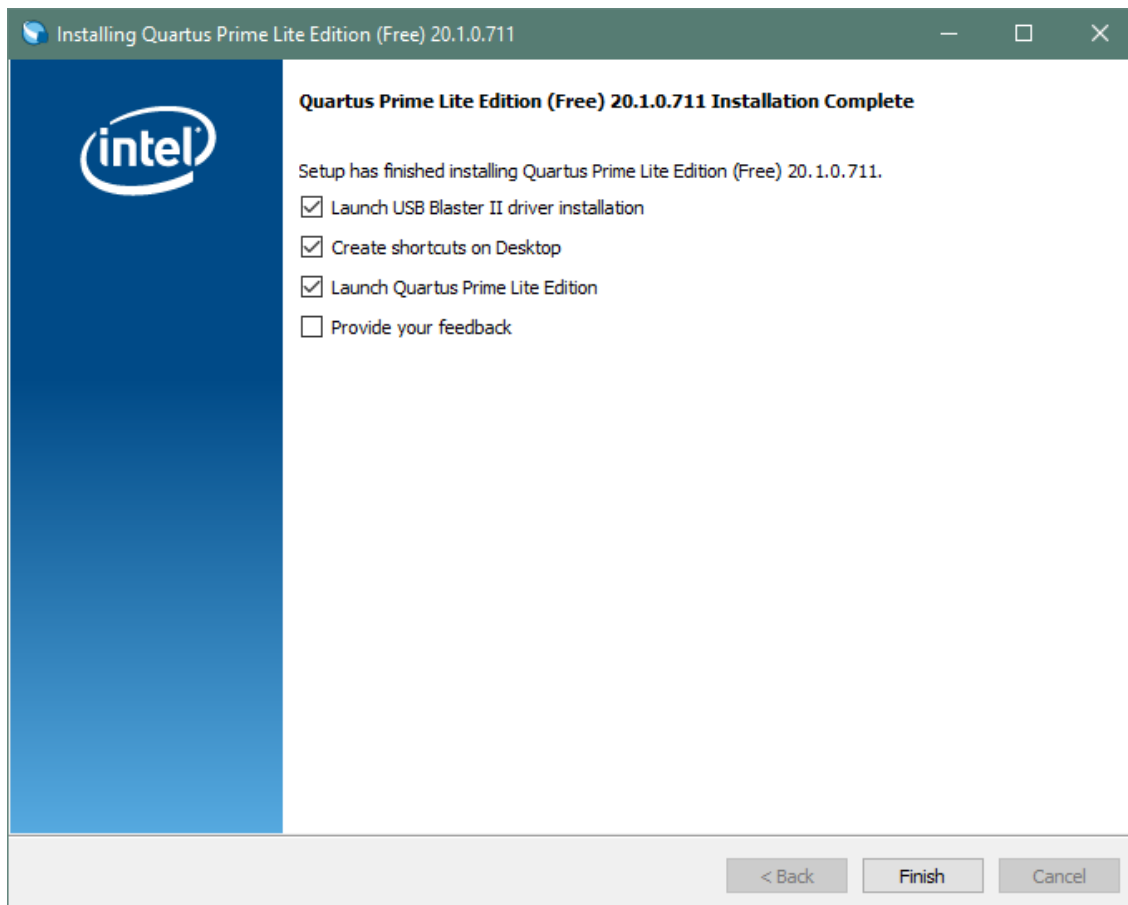


Wait for the installation to complete.

## FPGA Development System User Manual

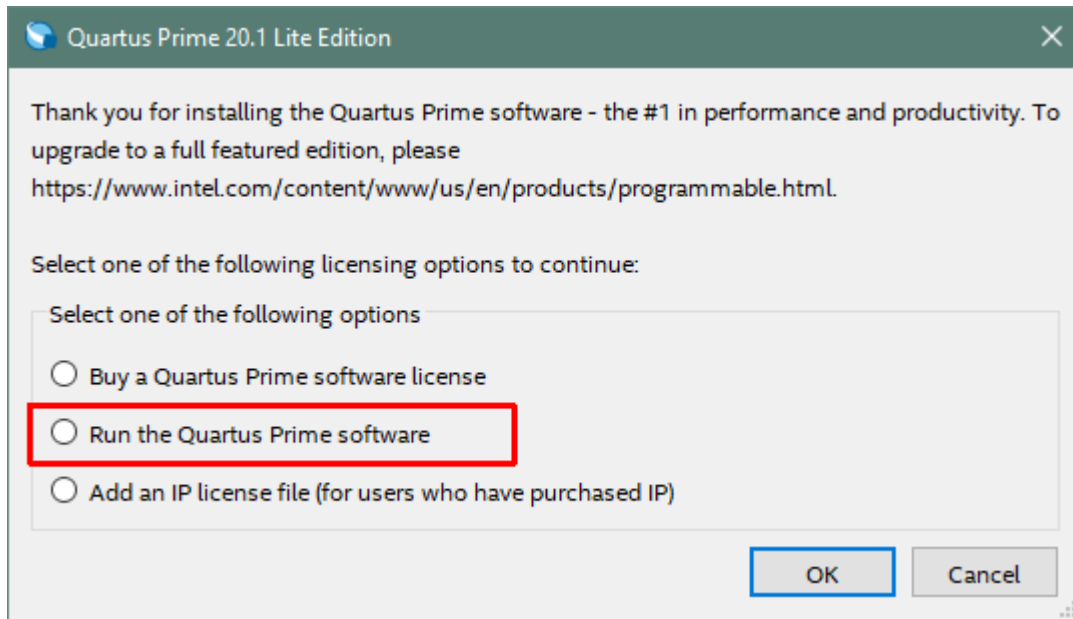


## FPGA Development System User Manual



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

## FPGA Development System User Manual



### 2.2.4 Adding the Terranlogic\_USB\_Blaster to Quartus Prime

Close out the Quartus Prime application.

To program the DueProLogic FPGA, the Terranlogic USB Blaster \*.dll must be added to the quartus64/bin folder of the root of the Quartus installation. Terranlogic has an automated installer that will add the \*.dll to the appropriate location.

To access the Terranlogic USB Blaster Installer, goto [www.earthpeopletechnology.com](http://www.earthpeopletechnology.com) website. Click on “Products” in the Banner



FPGA Development System User Manual

Click on “DueProLogic” under products

Locate the “Downloads” at the bottom of the page.

DueProLogic USB-FPGA Development System




**ALTERA  
CYCLONE IV FPGA  
DEVELOPMENT  
SYSTEM**

**Downloads**

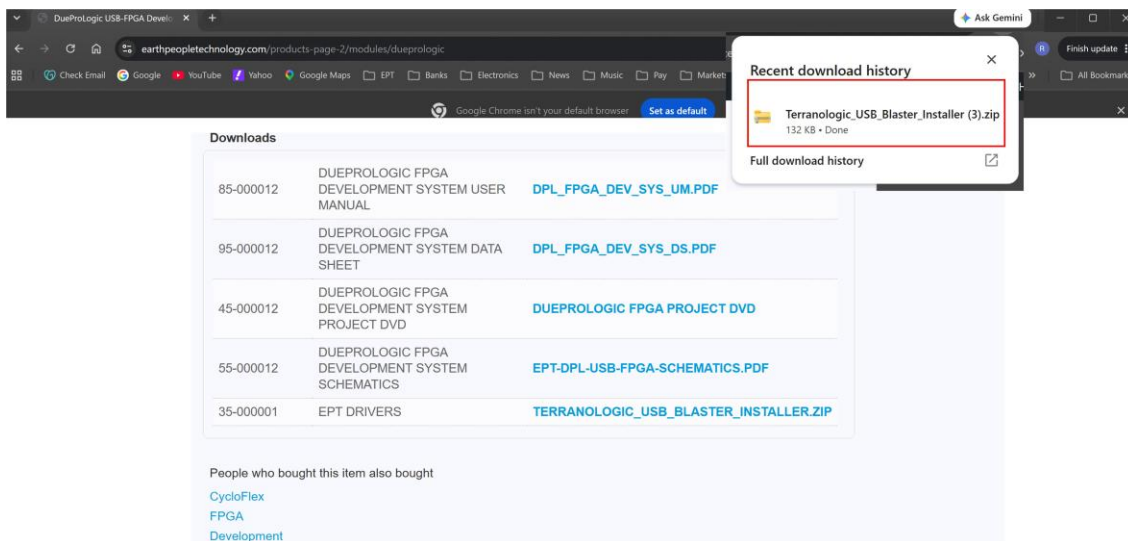
|           |                                                 |                                                        |
|-----------|-------------------------------------------------|--------------------------------------------------------|
| 85-000012 | DUEPROLOGIC FPGA DEVELOPMENT SYSTEM USER MANUAL | <a href="#">DPL_FPGA_DEV_SYS_UM.PDF</a>                |
| 95-000012 | DUEPROLOGIC FPGA DEVELOPMENT SYSTEM DATA SHEET  | <a href="#">DPL_FPGA_DEV_SYS_DS.PDF</a>                |
| 45-000012 | DUEPROLOGIC FPGA DEVELOPMENT SYSTEM PROJECT DVD | <a href="#">DUEPROLOGIC FPGA PROJECT DVD</a>           |
| 55-000012 | DUEPROLOGIC FPGA DEVELOPMENT SYSTEM SCHEMATICS  | <a href="#">EPT-DPL-USB-FPGA-SCHEMATICS.PDF</a>        |
| 35-000001 | EPT DRIVERS                                     | <a href="#">TERRANOLOGIC_USB_BLASTER_INSTALLER.ZIP</a> |

Double click on the \*.zip. The web browser will download the zip file.

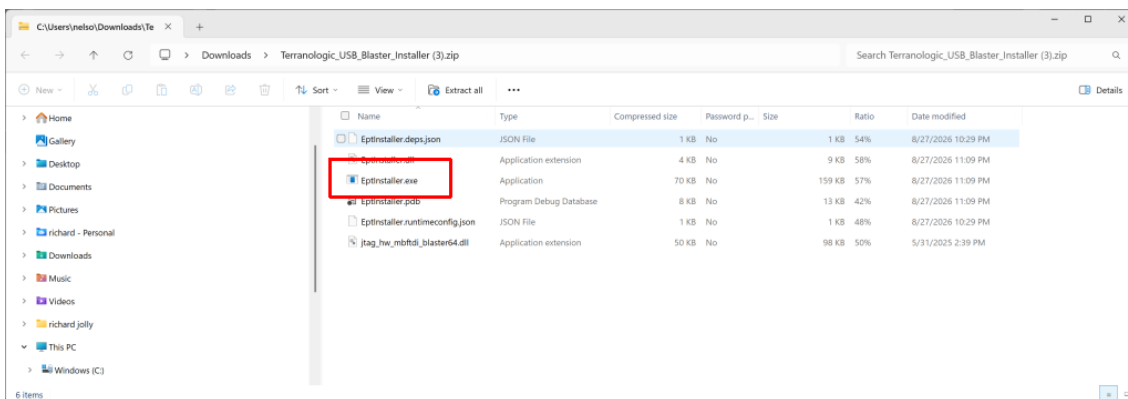


## FPGA Development System User Manual

The web browser will automatically download the zip file. When download is complete, double click on the zip file.



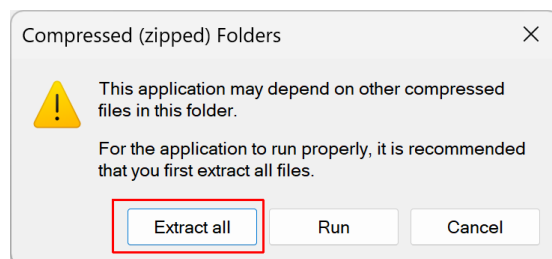
When the dialog box appears, double click on the EPTInstaller.exe file



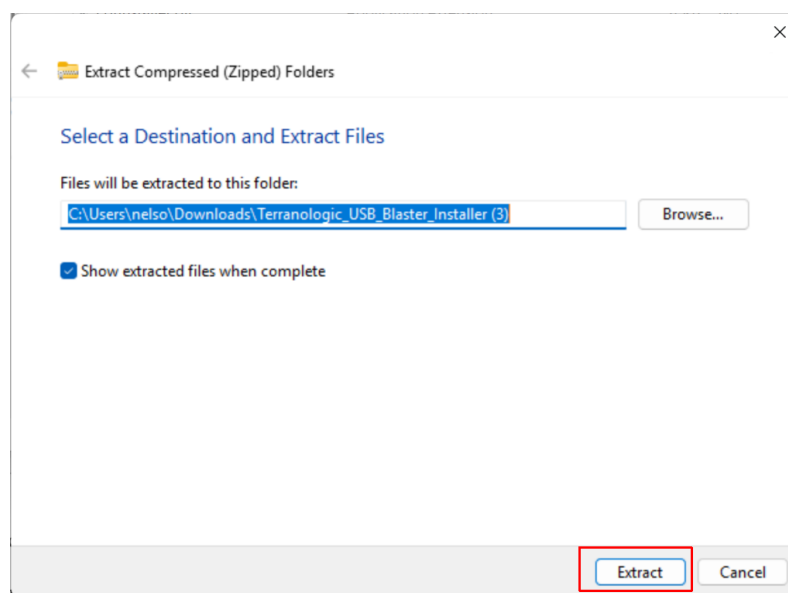
The following message will appear:



## FPGA Development System User Manual



Click on the “Extract all” button and the following message appears:

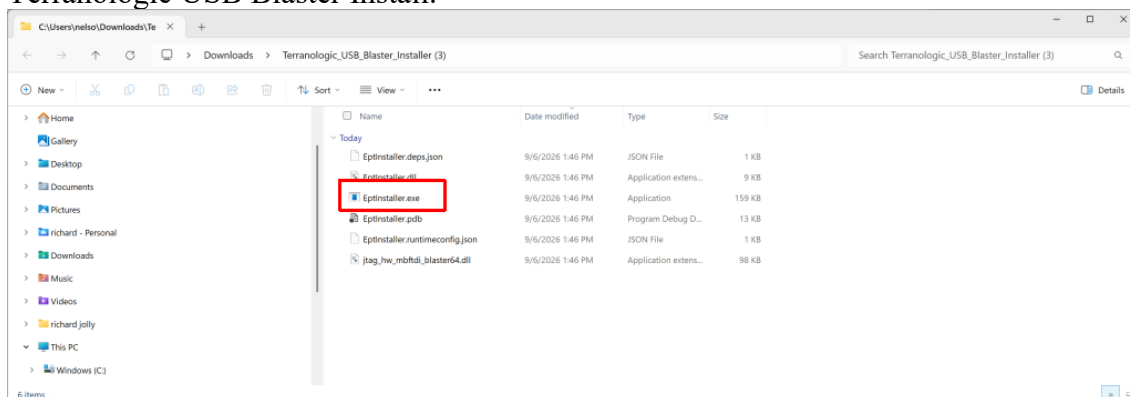


Click on the “Extract” button or select a different folder than the default folder and click the “Extract” button.

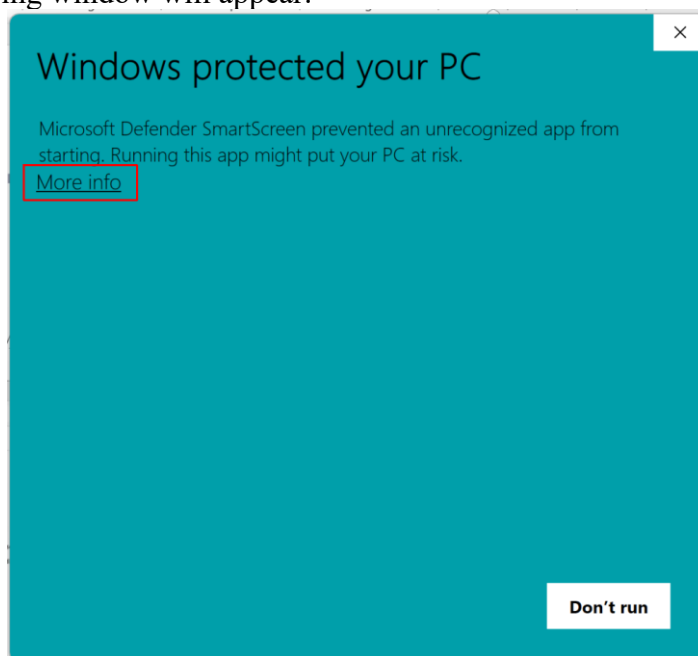


## FPGA Development System User Manual

Once the extraction is complete, double click on the “EptInstaller.exe” file to start the Terranologic USB Blaster Install.



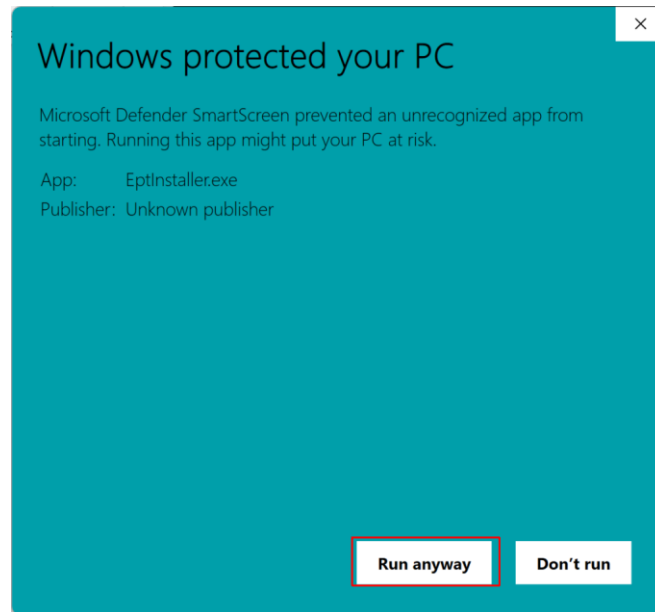
Next, the following window will appear:



Terranologic has not yet had the installer application certified by Microsoft. So, click on the “More Info” hyperlink.

Next, the following window will appear:

## FPGA Development System User Manual



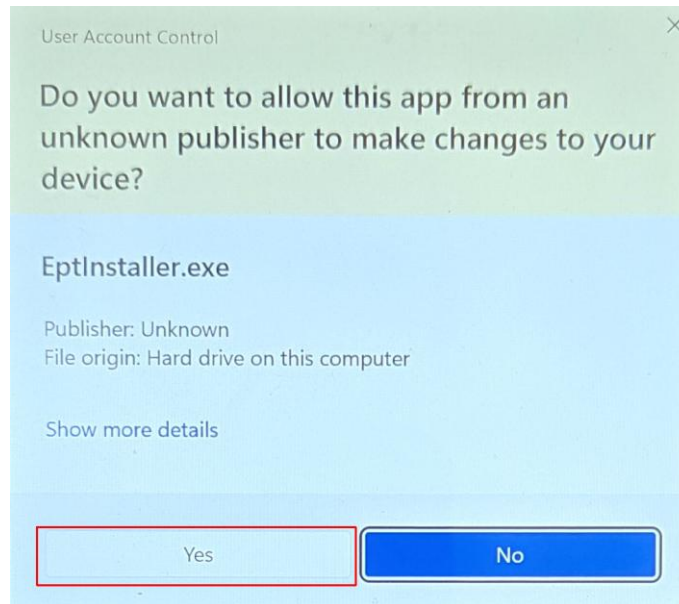
Since the Terranologic USB Blaster Installer is not yet certified, click the “Run anyway” button. If there are any questions about the installer software you can contact us at:

[sales@earthpeopletechnology.com](mailto:sales@earthpeopletechnology.com)

And we can provide a copy of the installer source code to ensure that there are no malicious entries in the program.

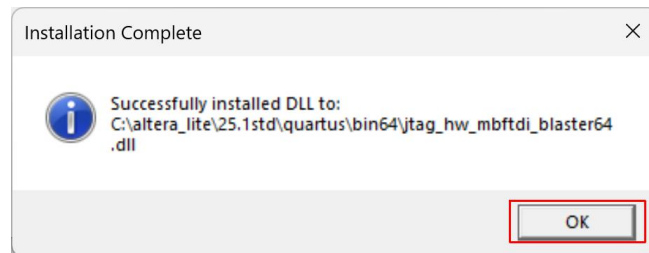
Next, the following window appears. This window will ensure that the user transfers the \*.dll file into the appropriate Quartus folder with full Administrator Privileges.

## FPGA Development System User Manual



Click “Yes”. If the user has full Administrator Privileges, the \*.dll file will be copied correctly. If the user does not have Administrator Privileges, then you will have to determine how to get these Privileges.

Once the installation is complete, you will see the following message.



Click “OK”. Open the Quartus Application.

The DLL is installed and the Quartus 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, contact Earth People Technology for support. There are three methods to contact EPT for support:



## FPGA Development System User Manual

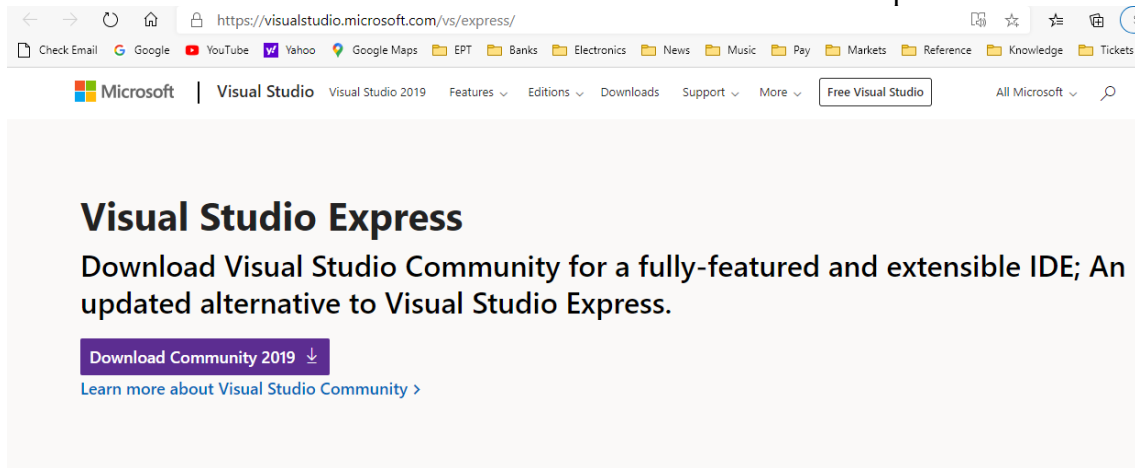
<https://www.earthpeopletechnology.com->Forums>  
[support@earthpeopletechnology.com](mailto:support@earthpeopletechnology.com)  
[sales@earthpeopletechnology.com](mailto:sales@earthpeopletechnology.com)

### ***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/>

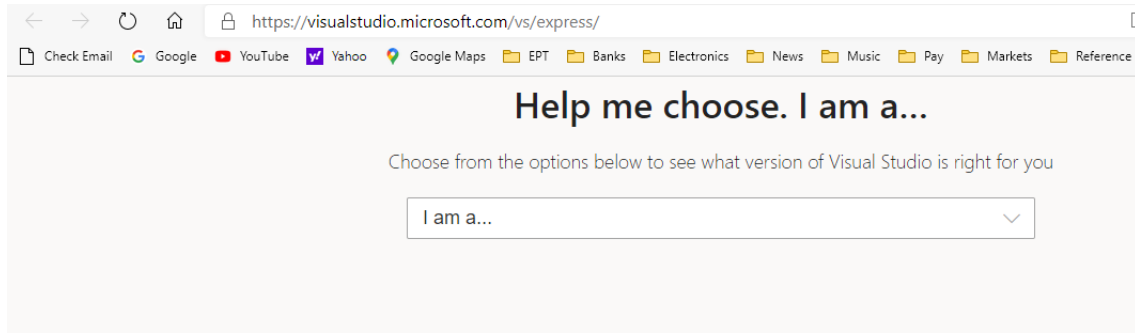
Go to the website and click on the “+” icon next to the Visual C# Express.



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



## FPGA Development System User Manual



### Still want Visual Studio Express?

#### [Express 2017 for Windows Desktop](#)

Supports building managed and native desktop applications.\*

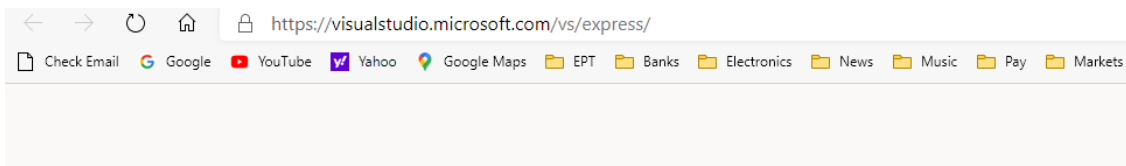
#### [Express 2015 for Windows Desktop](#)

Supports the creation of desktop applications for Windows.

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



## FPGA Development System User Manual



### Still want Visual Studio Express?

#### [Express 2017 for Windows Desktop](#)

Supports building managed and native desktop applications.\*

#### [Express 2015 for Windows Desktop](#)

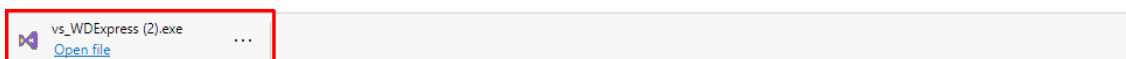
Supports the creation of desktop applications for Windows.

#### [Express 2015 for Web](#)

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

#### [Express 2015 for Windows 10](#)

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



Right click on the WDExpress.exe.

## Still want Visual Studio Express?

### [Express 2017 for Windows Desktop](#)

Supports building managed and native desktop applications.\*

### [Express 2015 for Windows Desktop](#)

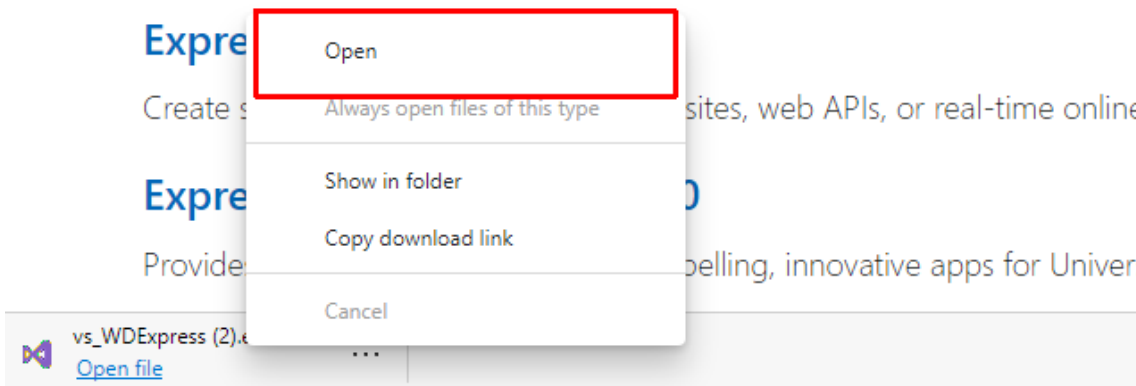
Supports the creation of desktop applications for Windows.

### [Express](#)

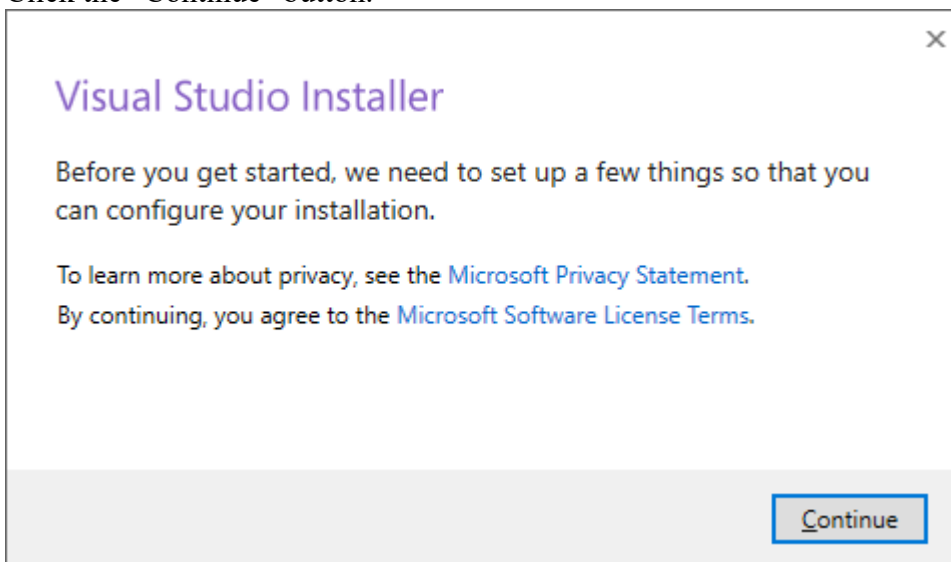
Create s... Always open files of this type ... sites, web APIs, or real-time online

### [Express](#)

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



Click the “Continue” button.

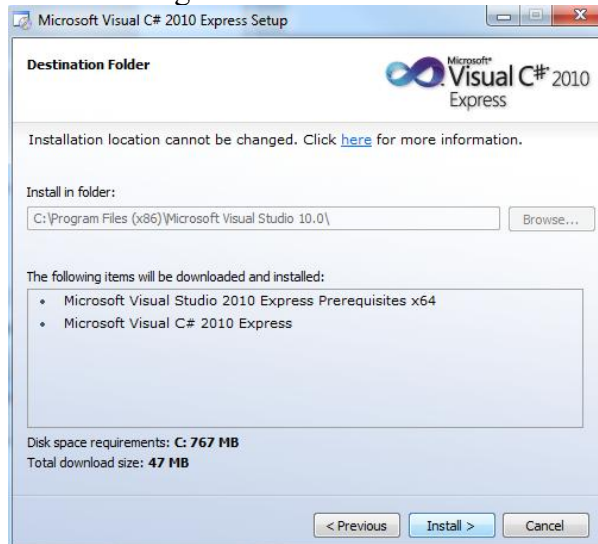


Next, follow the on screen windows and accept the default answers.

## FPGA Development System User Manual



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



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

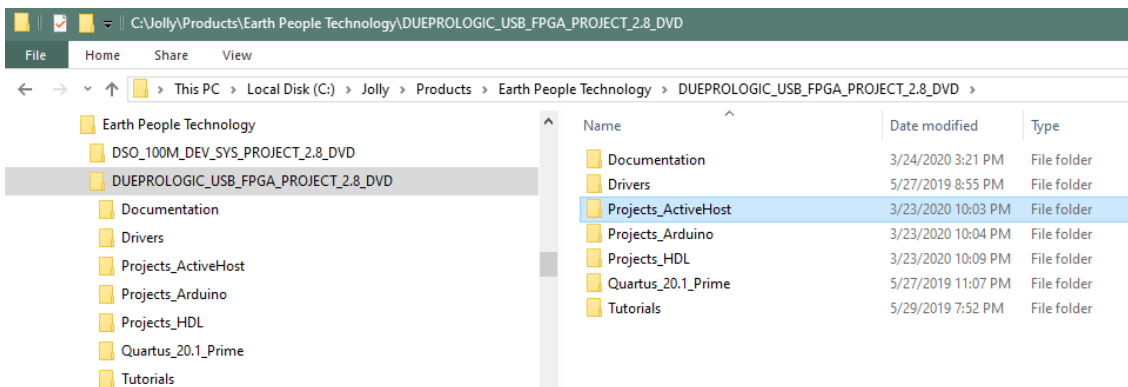


## FPGA Development System User Manual



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

The Active Host Application Software will allow the user to create a custom applications on the PC using the EndTerms to perform Triggers and Data Transfer to/from the DueProLogic. The methods and parameters of the Active Host library are explained in the Active Host Application section. Locate the \Projects\_ActiveHost folders on the EPT FPGA Development System DVD.

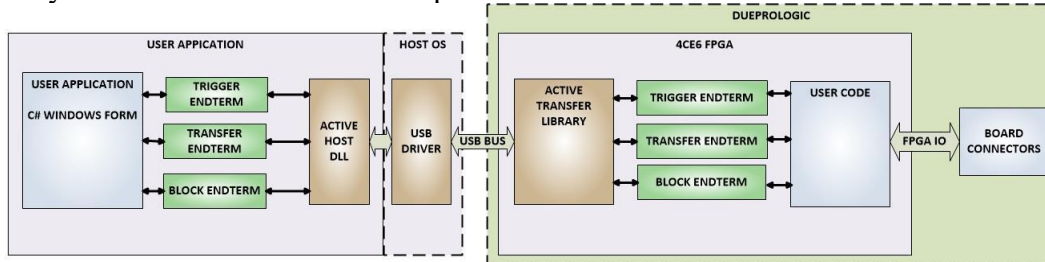


### 3 FPGA Active Transfer Library

The Active Transfer Library is an HDL library designed to transfer data to and from the DueProLogic via High Speed (480 MB/s) USB. It is a set of pre-compiled HDL files

## FPGA Development System User Manual

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.



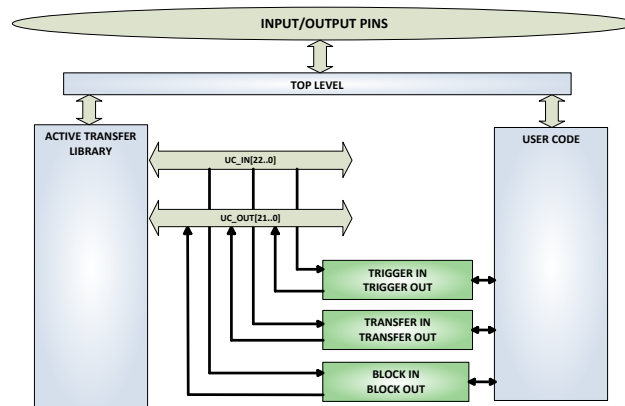
### 3.1 EPT Active Transfer System Overview

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

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. See the figure below for an overview of the EPT Active\_Transfer system.

#### EPT Active Transfer Library Overview

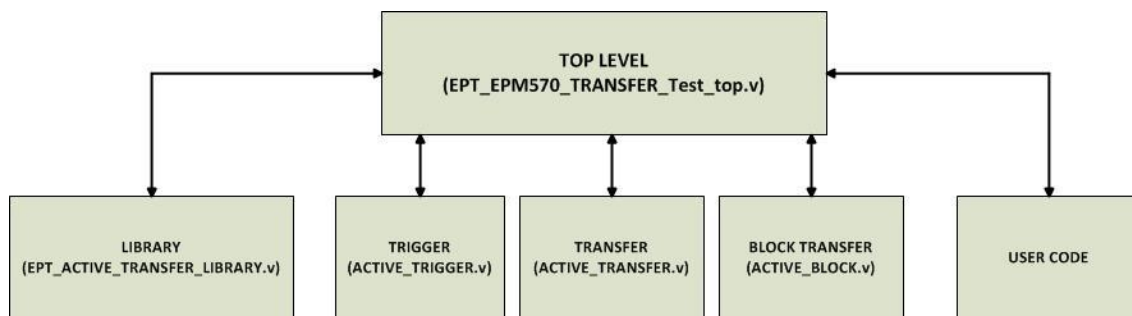


## FPGA Development System User Manual

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

### 3.2 Active Transfer Library

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





## FPGA Development System User Manual

```
//#####  
//#  
//# Copyright   Earth People Technology Inc. 2021  
//#  
//#  
//# File Name:   EPT_4CE6_AF_D1_Top.v  
//# Author:     Earth People Technology  
//# Date:       March 14, 2021  
//# Revision:    A  
//#  
//# Development: EPT Platform Demonstration Project  
//# Application: Altera Cyclone IV FPGA  
//# Description: This file contains verilog code which will allow access  
//#               to Active Transfer Library.  
//#  
//#  
//#  
//#*****  
//#  
//# Revision History:  
//#      DATE      VERSION    DETAILS  
//#      3/14/21    1          Created  
//#  
//#  
//#  
//#*****
```



## FPGA Development System User Manual

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

module EPT_4CE6_AF_D1_Top (

    input wire          CLK_66MHZ,
    input wire          RST,

    //UART COMMAND BUS
    output wire         UART_OUT, //To external
    input wire         UART_IN, //From external

    output wire [7:0]   XIO_1,      //XIO -- UB
    input wire  [5:0]   XIO_2,      //XIO -- UB
    output wire [5:0]   XIO_3,      //XIO -- UB
    input wire  [5:0]   XIO_4,      //XIO -- UB
    output wire [5:0]   XIO_5,      //XIO -- UB
    output wire [31:0]  XIO_6,      //XIO -- UB
    output wire [3:0]   XIO_7,      //XIO -- UB

    input wire         PB_SWITCH_1,
    input wire         PB_SWITCH_2

);
```

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.

## FPGA Development System User Manual

```
//-----  
// 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 UART signals are used to channel data, and control signals to the USB interface chip. These signals are connected directly to input and output pins of the FPGA.

### 3.2.1 Active Trigger EndTerm

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

## FPGA Development System User Manual

```

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

```

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

## FPGA Development System User Manual

```

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

```

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

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

## FPGA Development System User Manual

```

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

```

### 3.2.2 Active Transfer EndTerm

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

```

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

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

## FPGA Development System User Manual

```

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

```

## FPGA Development System User Manual

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

```

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

### 3.2.3 Active Block EndTerm

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

## FPGA Development System User Manual

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

```

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

```

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

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

## FPGA Development System User Manual

```

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

```

## FPGA Development System User Manual

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

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

### 3.3 Timing Diagram for Active Transfer EndTerms

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

#### 3.3.1 Active Trigger EndTerm Timing

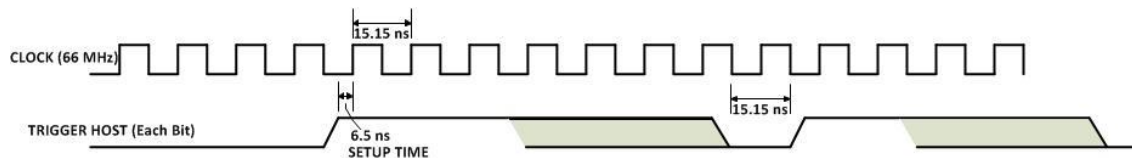


Figure xx Active Trigger to Host Timing

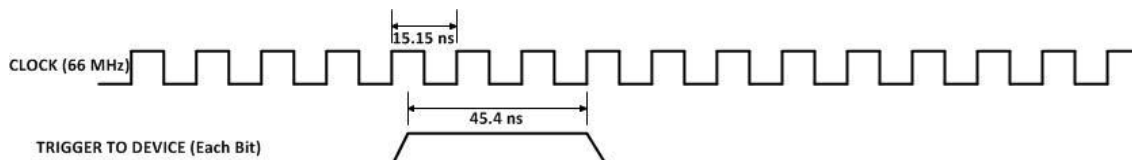


Figure xx Active Trigger to Device Timing

#### 3.3.2 Active Transfer EndTerm Timing

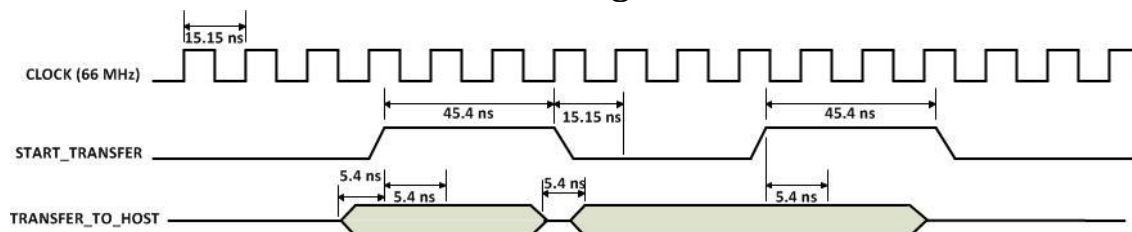


Figure xx Active Transfer To Host Timing

## FPGA Development System User Manual

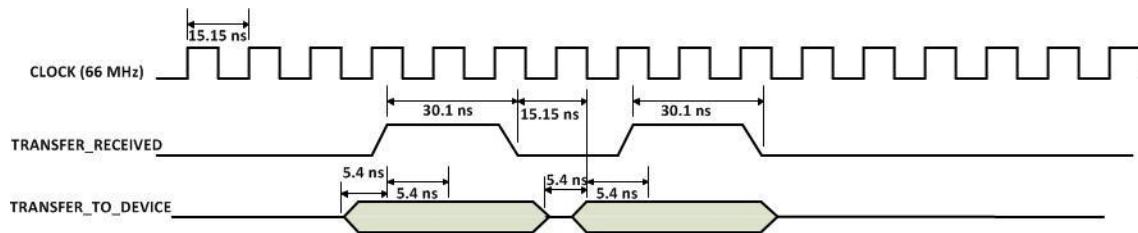


Figure xx Active Transfer To Device Timing

### 3.3.3 Active Block EndTerm Timing

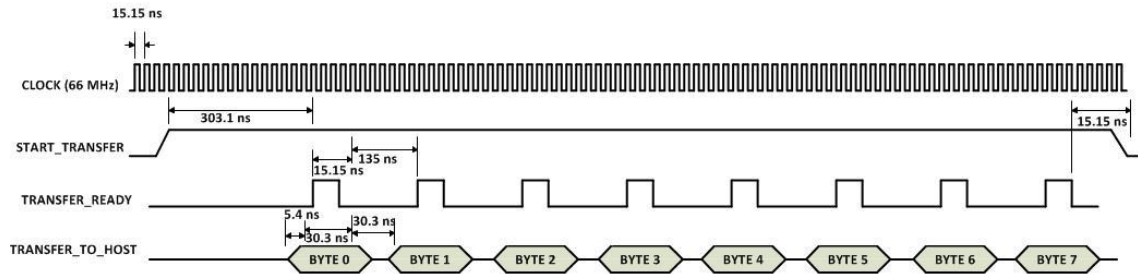


Figure xx Active Block To Host Timing

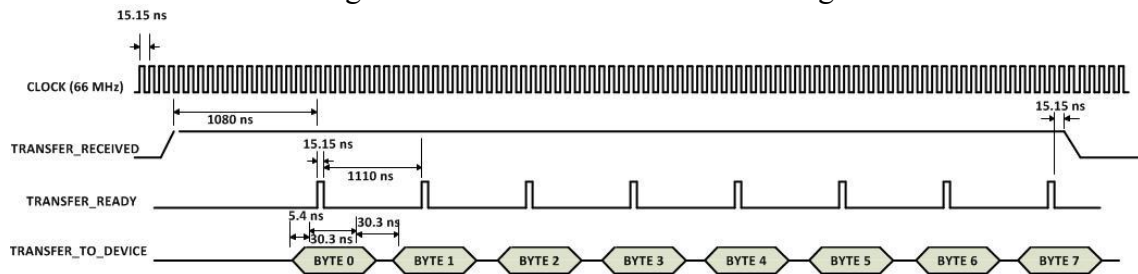


Figure xx Active Block To Device Timing

## 4 Compiling, Synthesizing, and Programming FPGA



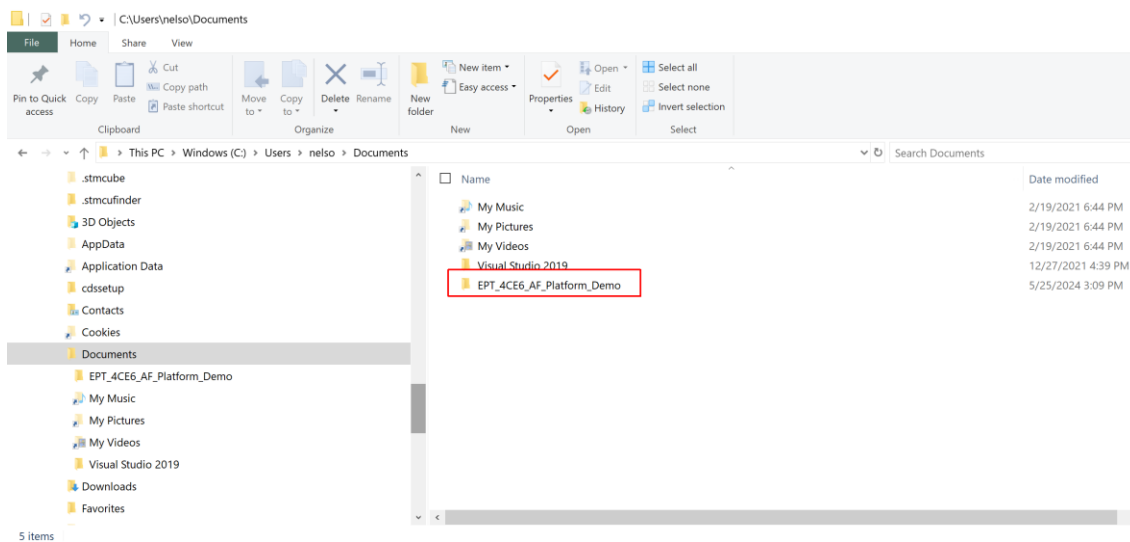
The FPGA on the EPT-4CE6-AF-D2 can be programmed with the Active Transfer Library and custom HDL code created by the user. Programming the FPGA requires the use of the Quartus Prime software and a standard USB cable. There are no extra parts to buy, just plug in the USB cable. Once the user HDL code is written according to the syntax rules of the language (Verilog and VHDL) it can be compiled and synthesized using the Quartus Prime software. This manual will not focus on HDL coding or proper coding techniques, instead it will use the example code to compile, synthesize and program the FPGA.

### ***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\ to create a new directory called: "EPT\_4CE6\_AF\_Platform\_Demo".



## FPGA Development System User Manual

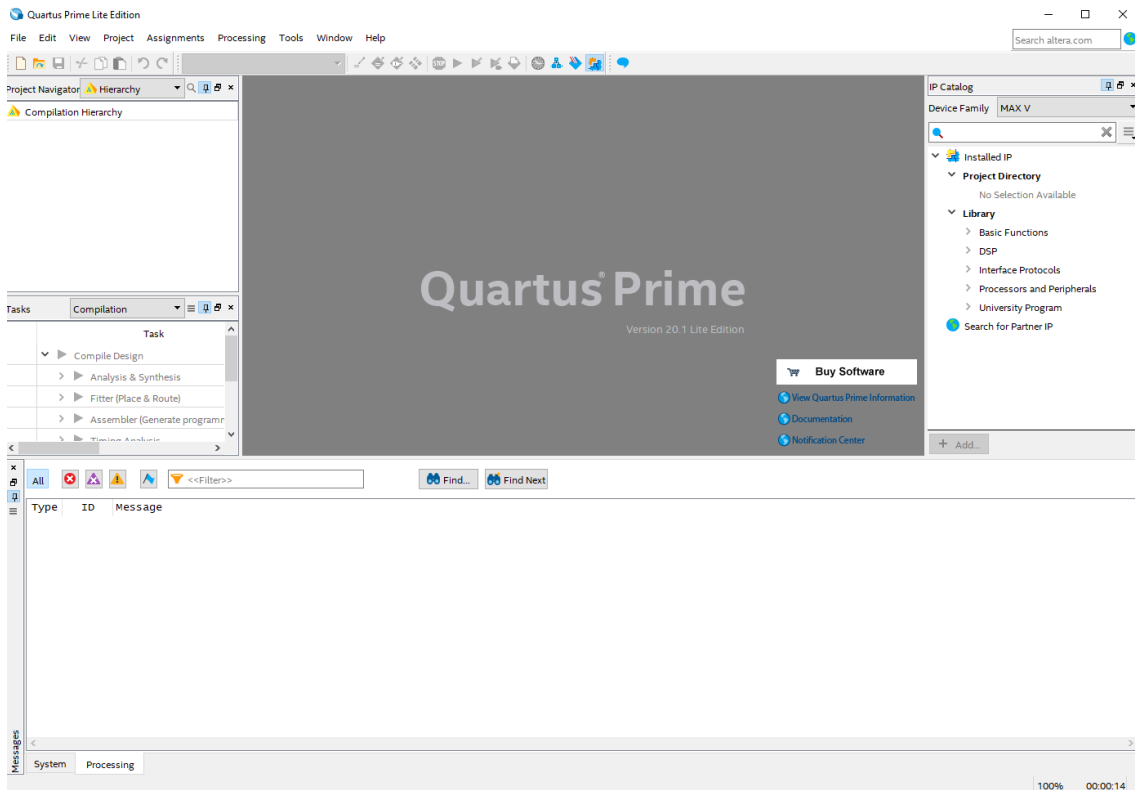


Open Quartus Prime by clicking on the icon.



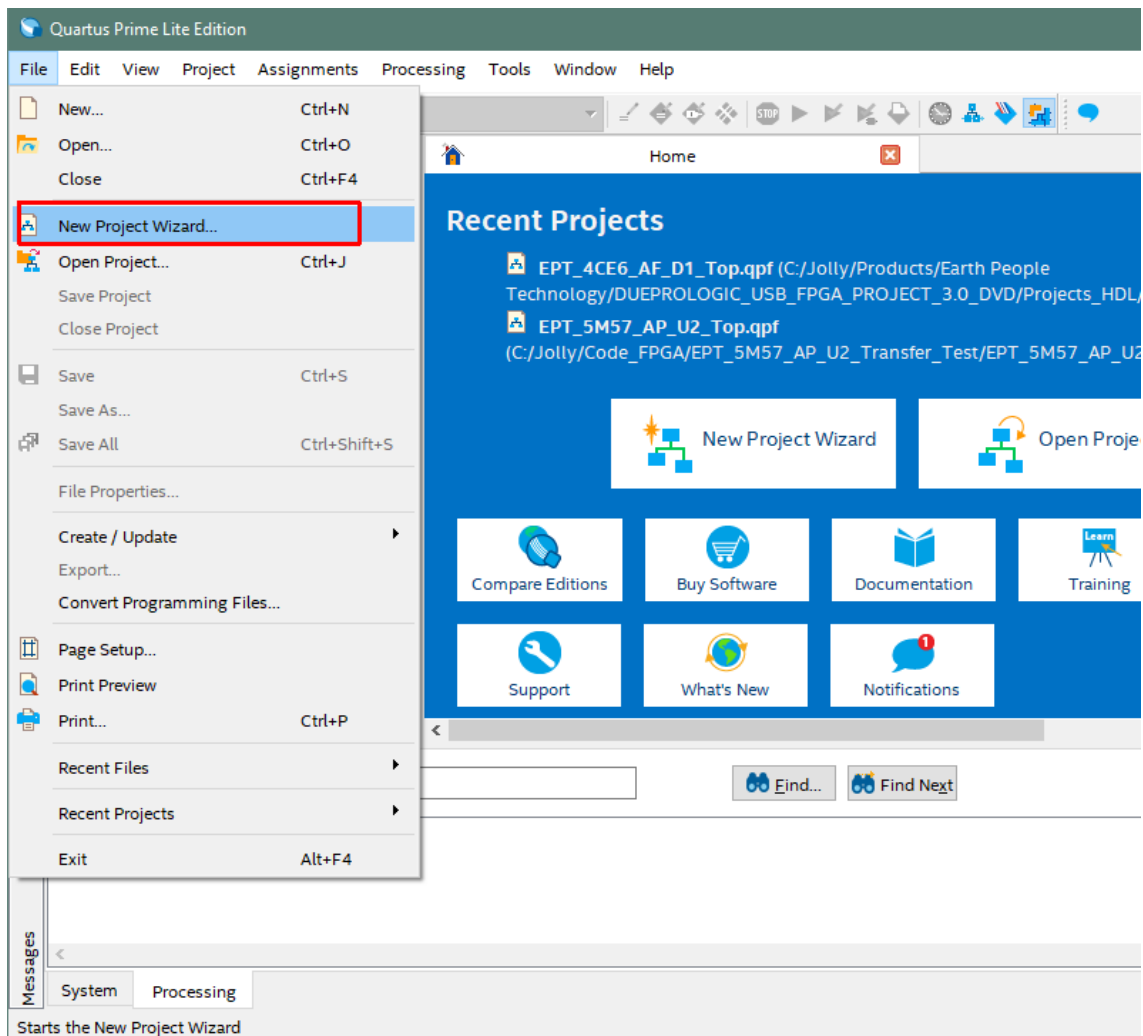


## FPGA Development System User Manual



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

## FPGA Development System User Manual



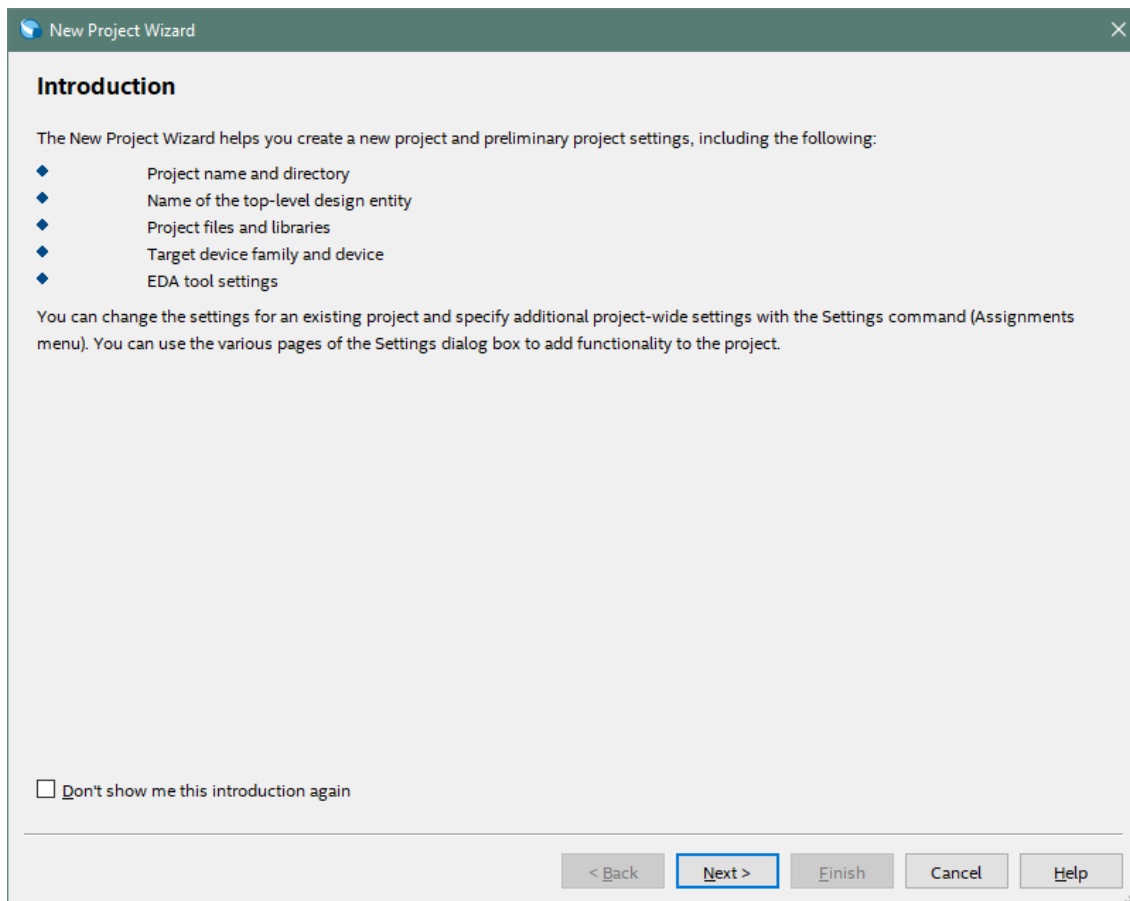
At the Top-Level Entity page, browse to the

C:\Users\nelso\Documents\EPT\_4CE6\_AF\_Platform\_Demo

directory to store your project. Type in a name for your project "EPT\_4CE6\_AF\_D1\_Top".

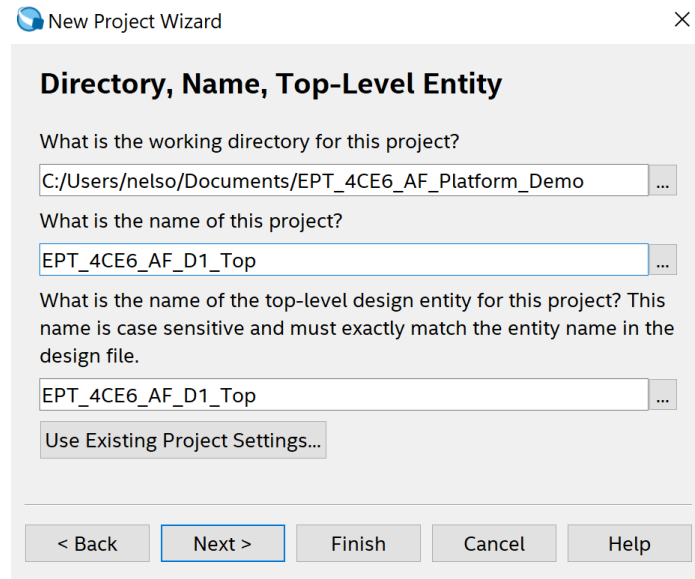


## FPGA Development System User Manual





## FPGA Development System User Manual



Select Next. At the Add Files window: Browse to the

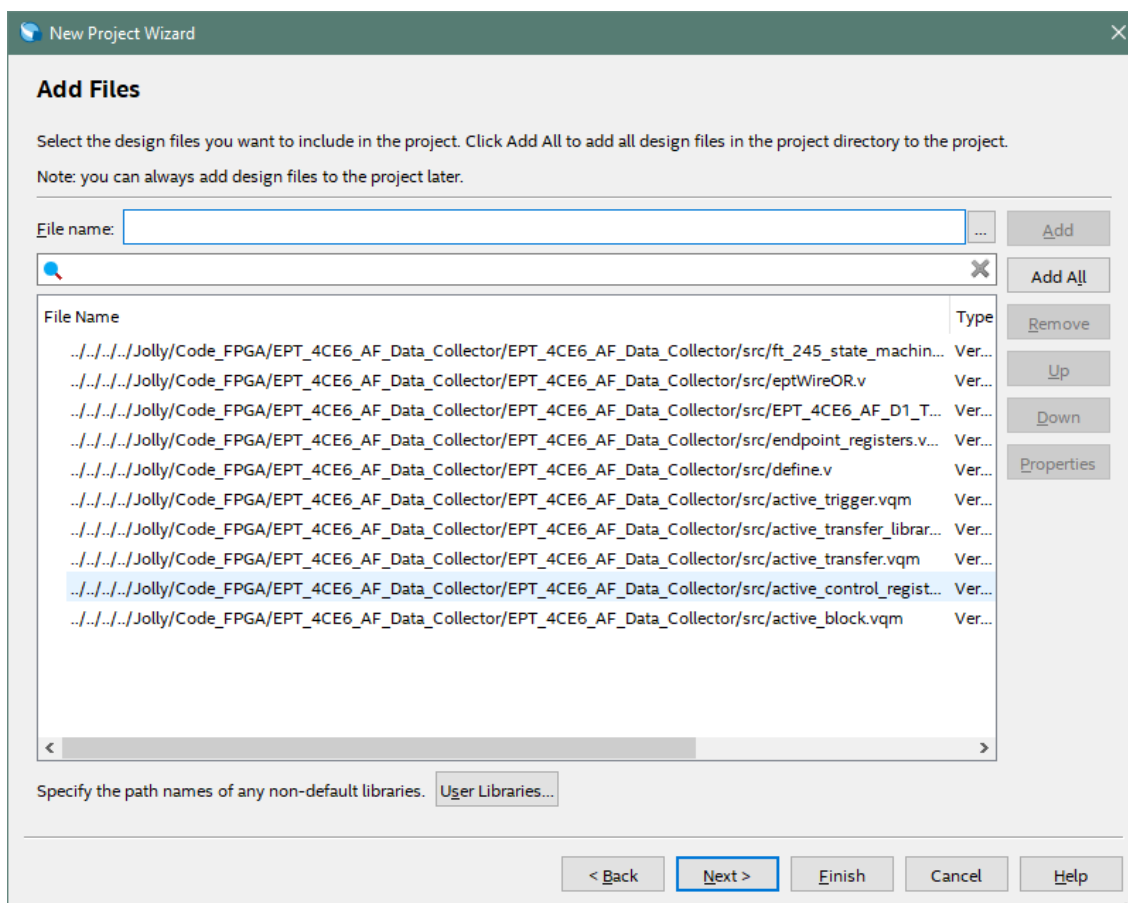
Projects\_HDL\EPT\_4CE6\_AF\_Platform\_Demo\src

folder of the EPT FPGA Development System DVD. Copy the files from the \src directory.

- active\_block.v
- active\_transfer.v
- active\_trigger.v
- active\_transfer\_uart.v
- active\_serial\_library.v
- define.v
- ft\_245\_state\_machine.v
- endpoint\_registers.v
- eptWireOr.v
- flipflop.v
- image\_block\_in.v
- image\_ept\_logo.v
- image\_face\_1.v
- image\_tri\_wave.v
- led\_update.v

## FPGA Development System User Manual

- lfsr.v
- mux.v
- read\_control\_logic.v
- serial\_clock.v
- sync\_fifo.v
- uart\_receiver.v
- uart\_top.v
- uart\_transmitter.v
- write\_control\_logic.v
- EPT\_4CE6\_AF\_D1\_Top.v



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

## FPGA Development System User Manual

New Project Wizard

### Family, Device & Board Settings

Device Board

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

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

Device family

Family: Cyclone IV E

Device: All

Target device

☐ Auto device selected by the Fitter

☒ Specific device selected in 'Available devices' list

☐ Other: n/a

Show in 'Available devices' list

Package: Any

Pin count: Any

Core speed grade: Any

Name filter:

☒ Show advanced devices

Available devices:

| Name         | Core Voltage | LEs  | Total I/Os | GPIOs | Memory Bits | Embedded multiplier 9-bi |
|--------------|--------------|------|------------|-------|-------------|--------------------------|
| EP4CE6E22C8  | 1.2V         | 6272 | 92         | 92    | 276480      | 30                       |
| EP4CE6E22C8L | 1.0V         | 6272 | 92         | 92    | 276480      | 30                       |
| EP4CE6E22C9L | 1.0V         | 6272 | 92         | 92    | 276480      | 30                       |
| EP4CE6E22I7  | 1.2V         | 6272 | 92         | 92    | 276480      | 30                       |
| EP4CE6E22I8L | 1.0V         | 6272 | 92         | 92    | 276480      | 30                       |

< Back Next > Finish Cancel Help

Select Next, leave defaults for the EDA Tool Settings.



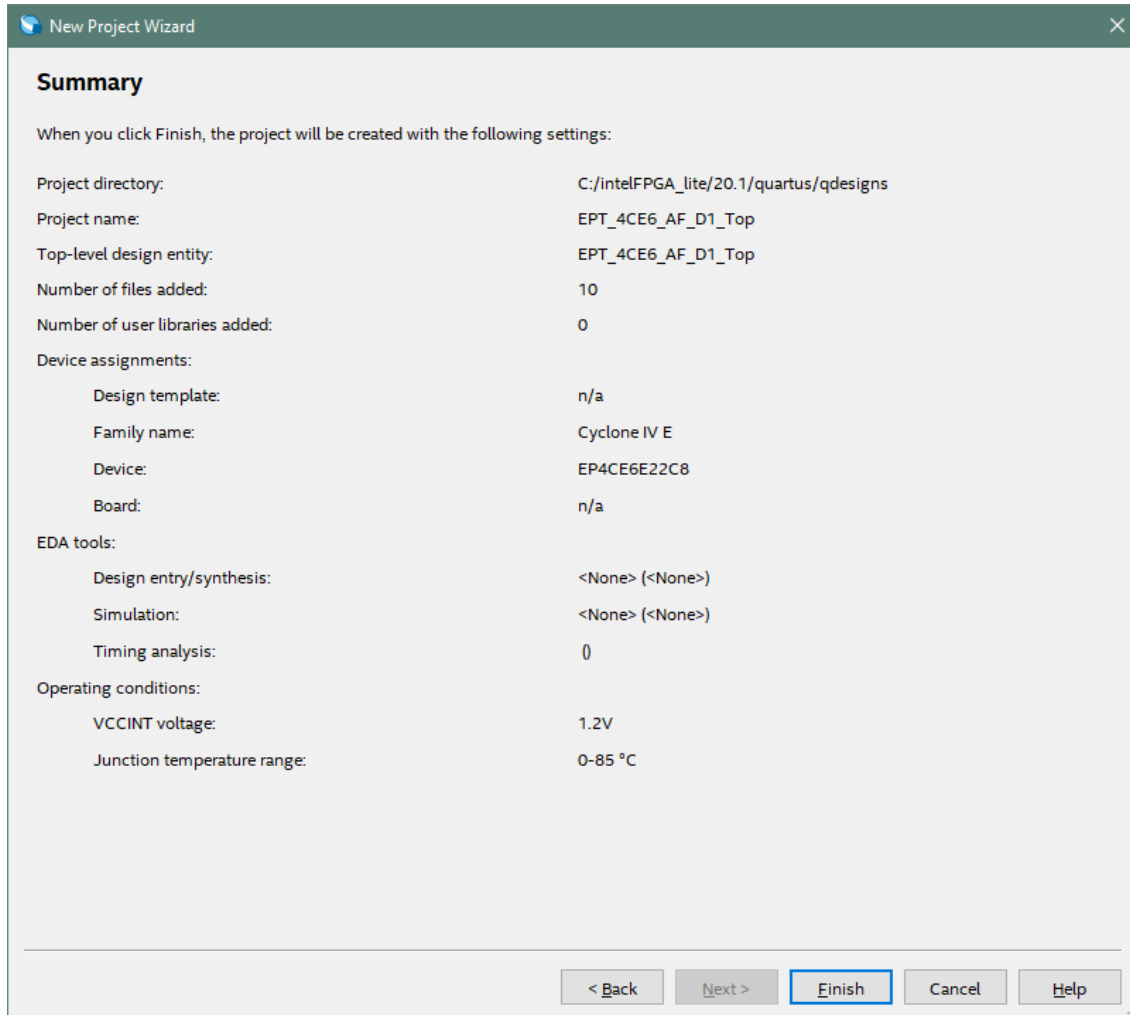
## FPGA Development System User Manual

The image shows a screenshot of the "New Project Wizard" dialog box, specifically the "EDA Tool Settings" tab. The dialog has a title bar with a close button (X) and a subtitle "EDA Tool Settings". Below the subtitle is a instruction: "Specify the other EDA tools used with the Quartus Prime software to develop your project." The main area is labeled "EDA tools:" and contains a table with four columns: "Tool Type", "Tool Name", "Format(s)", and "Run Tool Automatically". The table has five rows. The first row is for "Design Entry/Synth..." with "Tool Name" set to "<None>" and "Format(s)" set to "<None>". The second row is for "Simulation" with "Tool Name" set to "<None>" and "Format(s)" set to "<None>". The third row is for "Board-Level" with "Tool Name" set to "Timing" and "Format(s)" set to "<None>". The fourth row is for "Board-Level" with "Tool Name" set to "Symbol" and "Format(s)" set to "<None>". The fifth row is for "Board-Level" with "Tool Name" set to "Signal Integrity" and "Format(s)" set to "<None>". The sixth row is for "Board-Level" with "Tool Name" set to "Boundary Scan" and "Format(s)" set to "<None>". The "Run Tool Automatically" column has checkboxes for the first two rows, both of which are unchecked. At the bottom of the dialog are five buttons: "< Back", "Next >", "Finish", "Cancel", and "Help". The "Next >" button is highlighted with a blue border.

| Tool Type             | Tool Name        | Format(s) | Run Tool Automatically                                                                |
|-----------------------|------------------|-----------|---------------------------------------------------------------------------------------|
| Design Entry/Synth... | <None>           | <None>    | <input type="checkbox"/> Run this tool automatically to synthesize the current design |
| Simulation            | <None>           | <None>    | <input type="checkbox"/> Run gate-level simulation automatically after compilation    |
| Board-Level           | Timing           | <None>    |                                                                                       |
|                       | Symbol           | <None>    |                                                                                       |
|                       | Signal Integrity | <None>    |                                                                                       |
|                       | Boundary Scan    | <None>    |                                                                                       |

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

## FPGA Development System User Manual

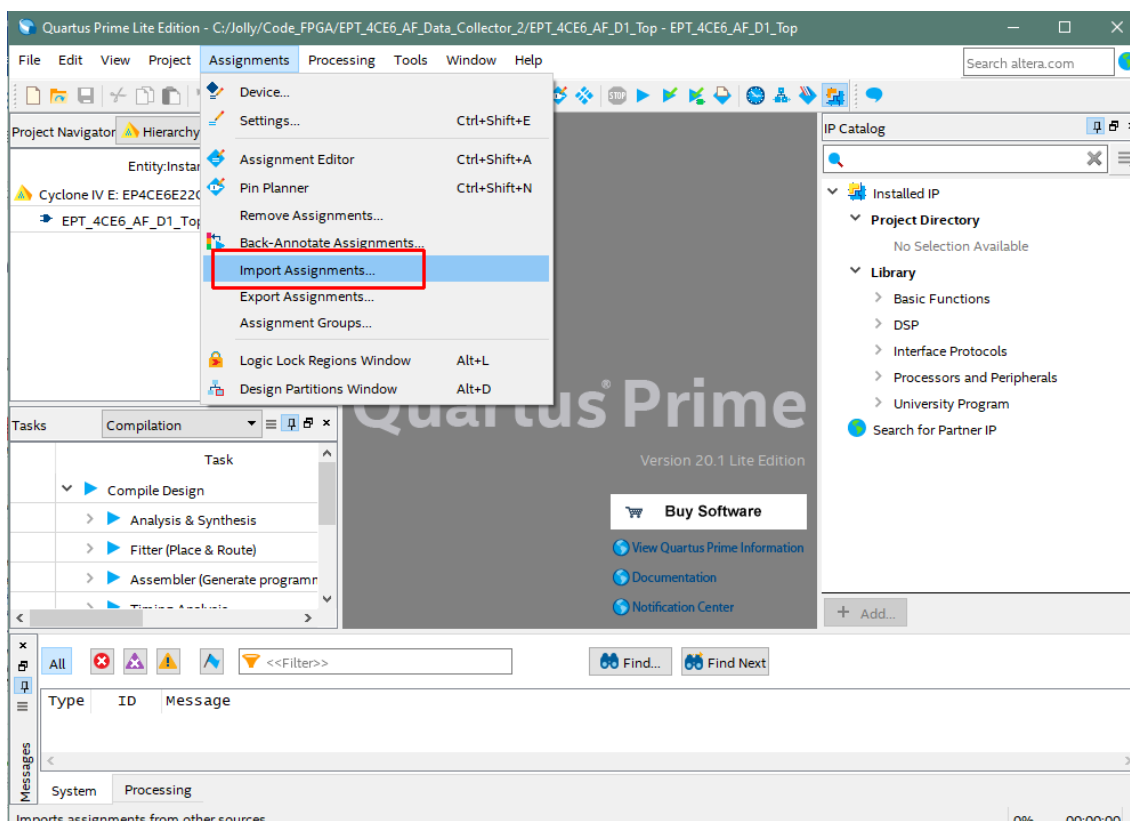


Next, we will select the pins and synthesize the project.

### 4.1.1 Selecting Pins and Synthesizing

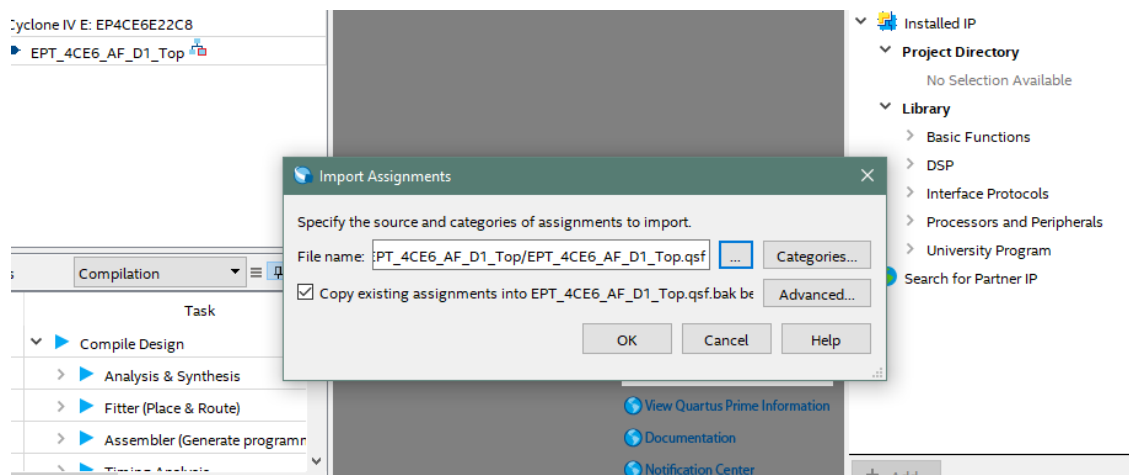
With the project created, we need to assign pins to the project. The signals defined in the top level file (in this case: EPT\_4CE6\_AF\_D1\_Top.v) will connect directly to pins on the FPGA. The Pin Planner Tool from Quartus Prime will add the pins and check to verify that our pin selections do not violate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

## FPGA Development System User Manual

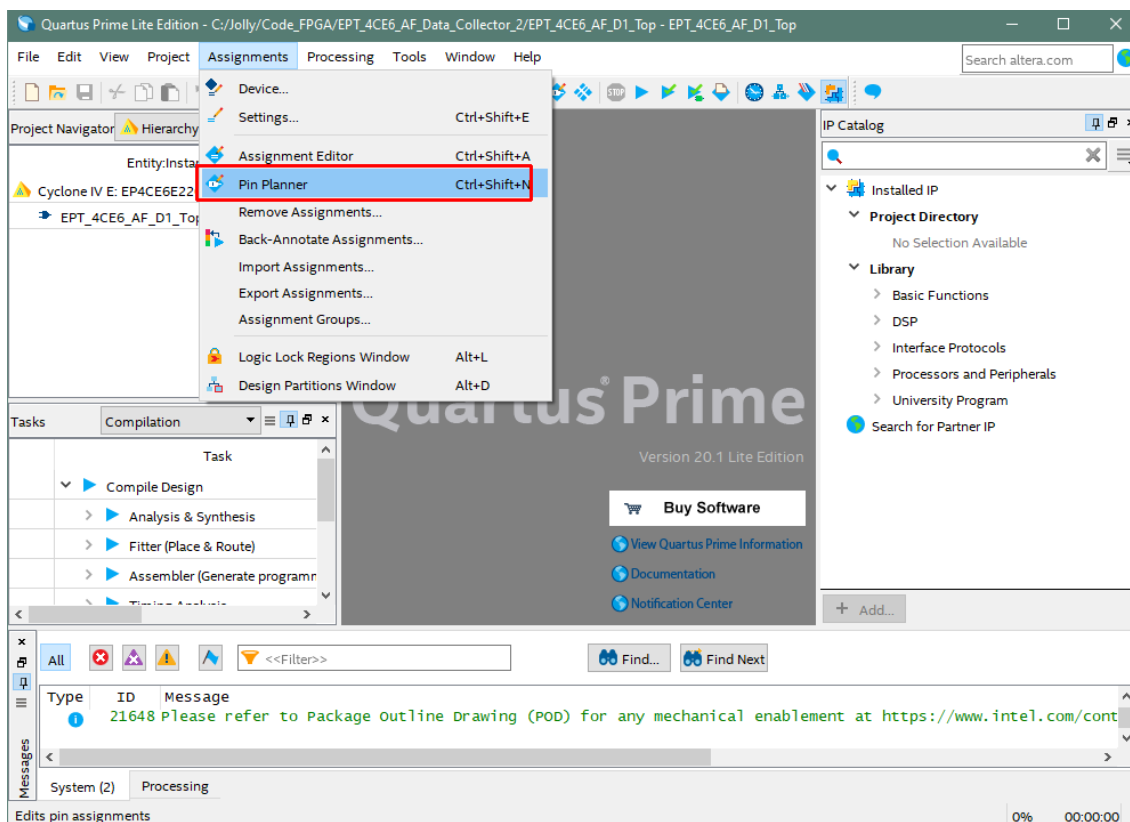


At the Import Assignment dialog box, Browse to the  
\\Projects\_HDL\\EPT\_Transfer\_Demo\\EPT\_4CE6\_AF\_D1\_TOP folder of the EPT  
FPGA Development System DVD. Select the “EPT\_4CE6\_AF\_D1\_Top.qsf” file.

## FPGA Development System User Manual



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





## FPGA Development System User Manual

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

The screenshot shows the Pin Planner interface for a Cyclone IV E device (EP4C6E22C8). The main window displays a top view of the device with wire bonds. Below the device view is a table of pin assignments. The table has columns for Node Name, Direction, Location, I/O Bank, VREF Group, I/O Standard, Reserved, and Current S. The table lists various nodes and their assigned pins and banks.

| Node Name   | Direction | Location | I/O Bank | VREF Group | I/O Standard    | Reserved | Current S |
|-------------|-----------|----------|----------|------------|-----------------|----------|-----------|
| bd_inout[0] | Unknown   | PIN_43   | 3        | B3_N0      | 2.5 V (default) |          | 8mA (defi |
| bd_inout[1] | Unknown   | PIN_38   | 3        | B3_N0      | 2.5 V (default) |          | 8mA (defi |
| bd_inout[2] | Unknown   | PIN_39   | 3        | B3_N0      | 2.5 V (default) |          | 8mA (defi |
| bd_inout[3] | Unknown   | PIN_34   | 2        | B2_N0      | 2.5 V (default) |          | 8mA (defi |
| bd_inout[4] | Unknown   | PIN_33   | 2        | B2_N0      | 2.5 V (default) |          | 8mA (defi |
| bd_inout[5] | Unknown   | PIN_32   | 2        | B2_N0      | 2.5 V (default) |          | 8mA (defi |
| bd_inout[6] | Unknown   | PIN_30   | 2        | B2_N0      | 2.5 V (default) |          | 8mA (defi |
| bd_inout[7] | Unknown   | PIN_28   | 2        | B2_N0      | 2.5 V (default) |          | 8mA (defi |
| bc_in[1]    | Unknown   | PIN_11   | 1        | B1_N0      | 2.5 V (default) |          | 8mA (defi |
| bc_in[0]    | Unknown   | PIN_10   | 1        | B1_N0      | 2.5 V (default) |          | 8mA (defi |
| bc_out[2]   | Unknown   | PIN_3    | 1        | B1_N0      | 2.5 V (default) |          | 8mA (defi |
| bc_out[1]   | Unknown   | PIN_2    | 1        | B1_N0      | 2.5 V (default) |          | 8mA (defi |
| bc_out[0]   | Unknown   | PIN_1    | 1        | B1_N0      | 2.5 V (default) |          | 8mA (defi |
| aa[1]       | Unknown   | PIN_23   | 1        | B1_N0      | 2.5 V (default) |          | 8mA (defi |
| aa[0]       | Unknown   | PIN_89   | 5        | B5_N0      | 2.5 V (default) |          | 8mA (defi |

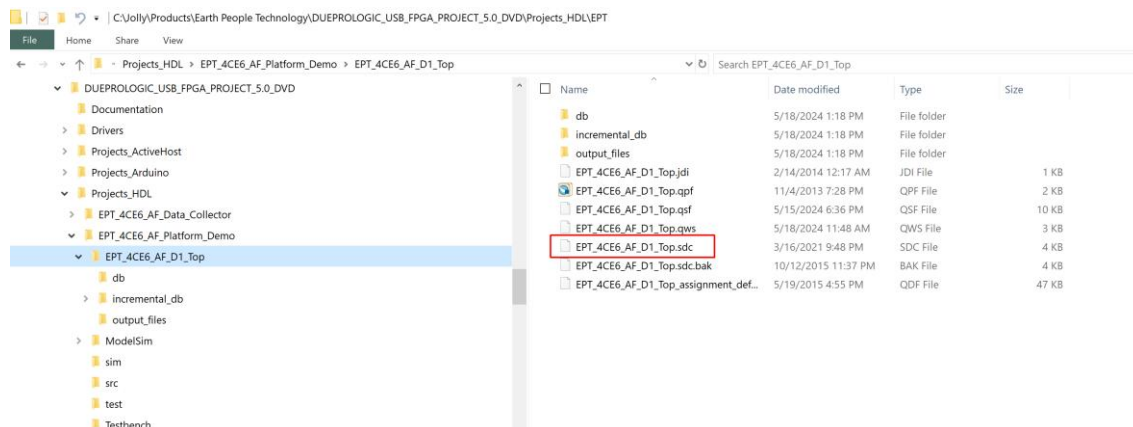


## FPGA Development System User Manual

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

### [Quest Timing Analyzer Quick Start Guide](#)

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

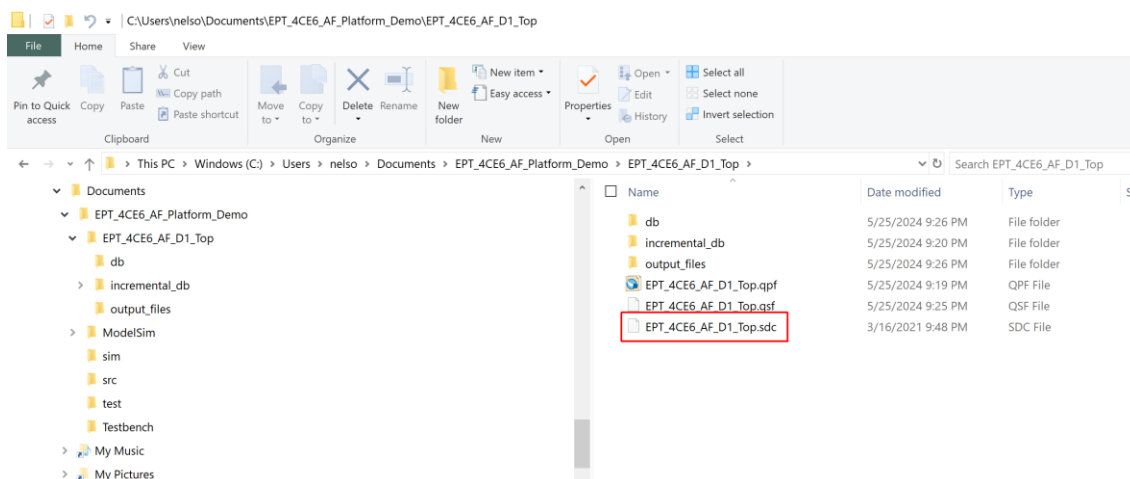


Copy the file and browse to

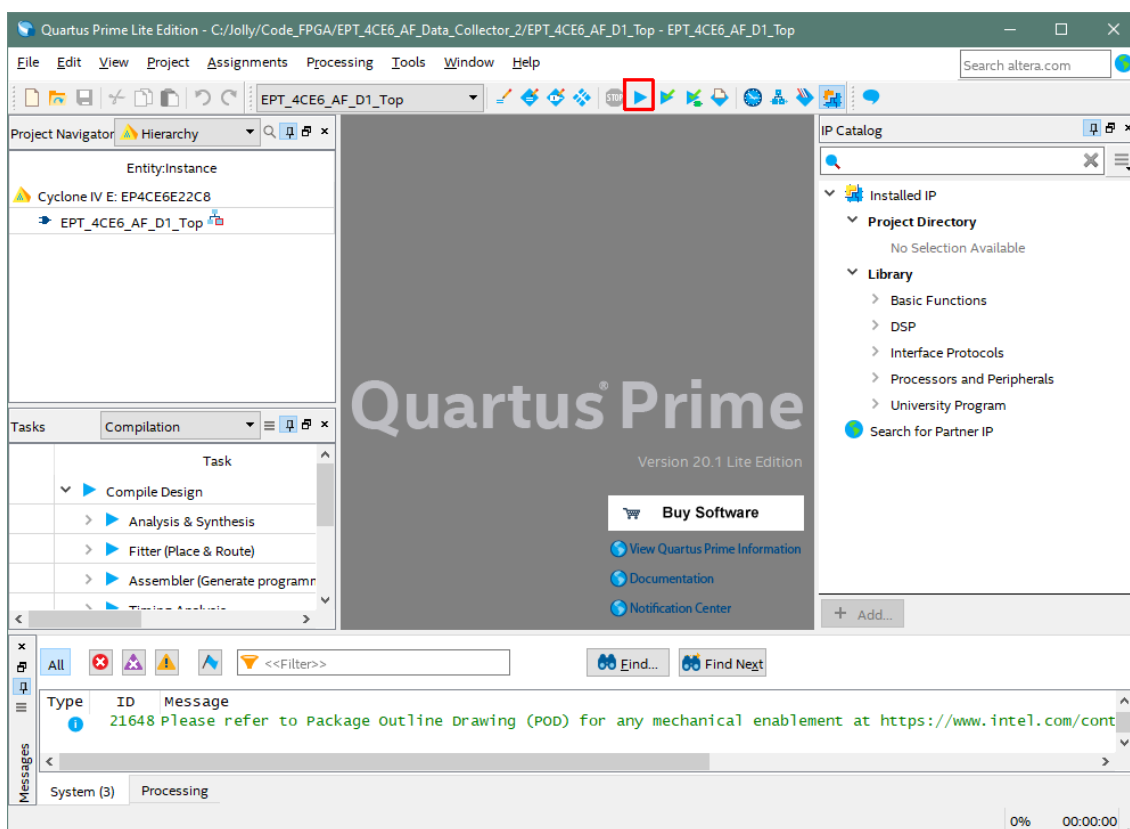
C:\Users\nelso\Documents\EPT\_4CE6\_AF\_Platform\_Demo \ EPT\_4CE6\_AF\_D1\_Top  
directory. Paste the file.



## FPGA Development System User Manual



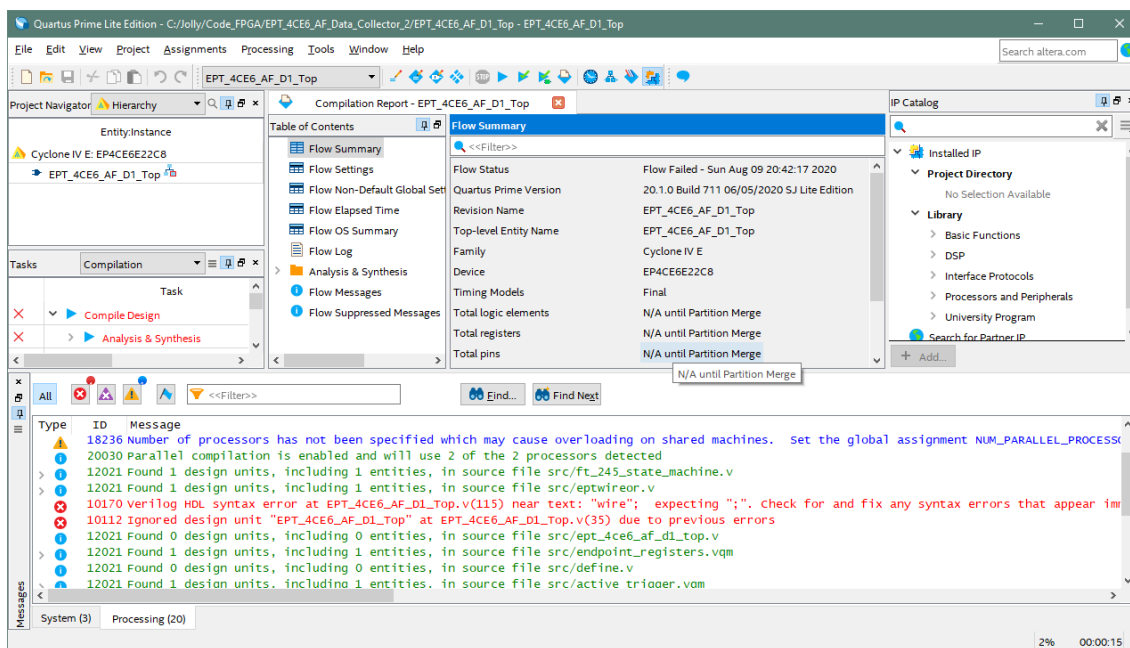
Select the Start Compilation button.





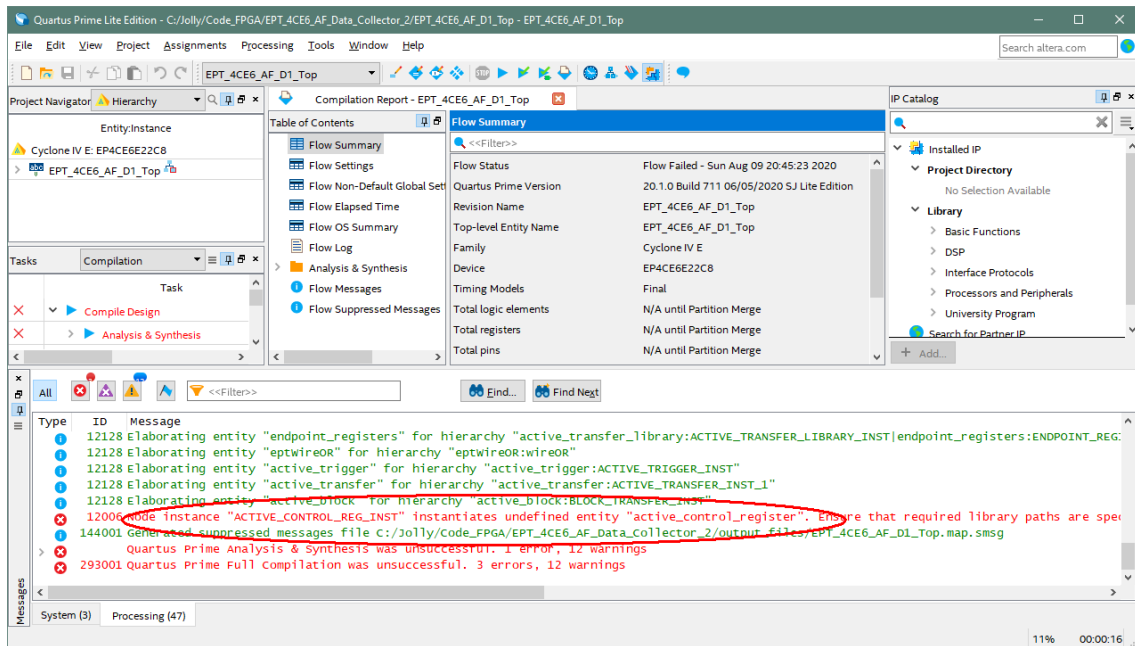
## FPGA Development System User Manual

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



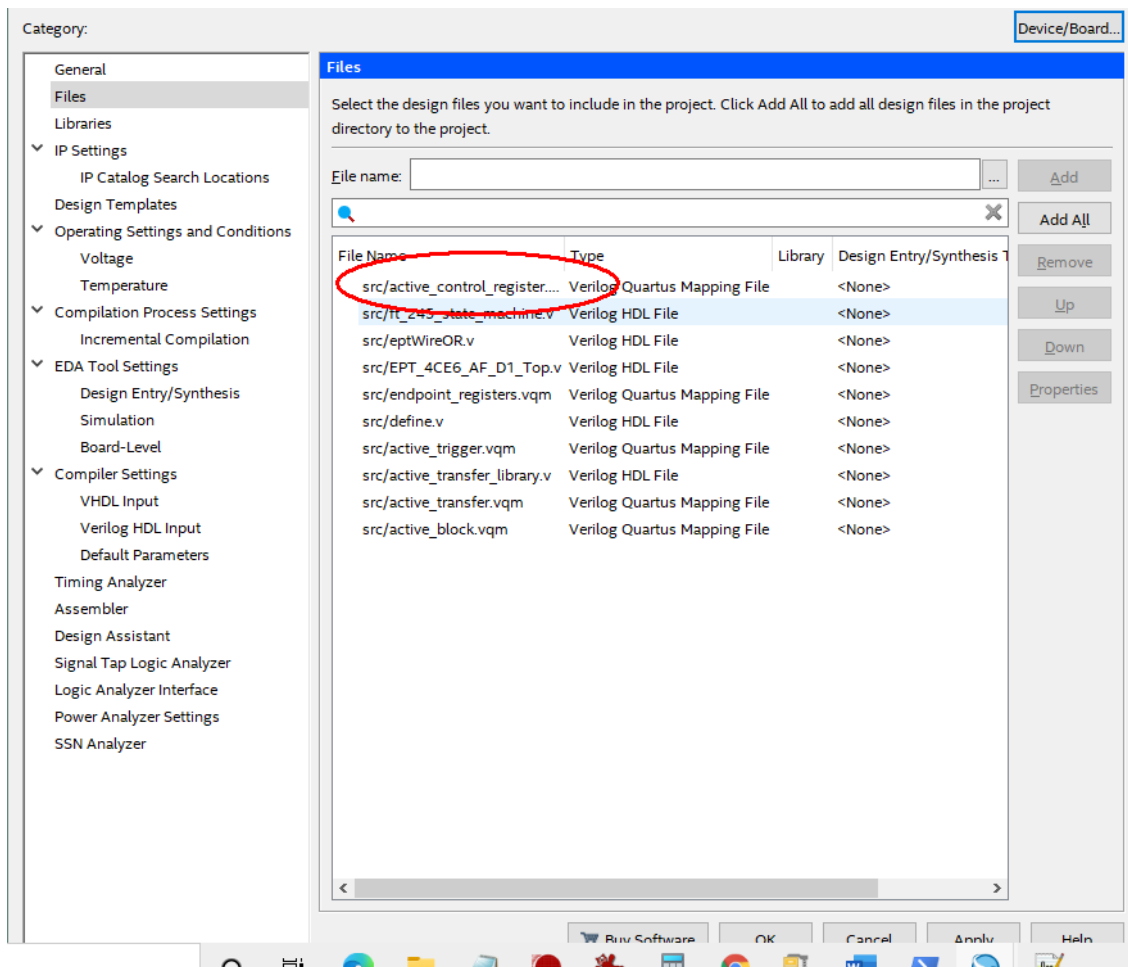
Click Ok, then select the "Error" tab to see the error.

## FPGA Development System User Manual



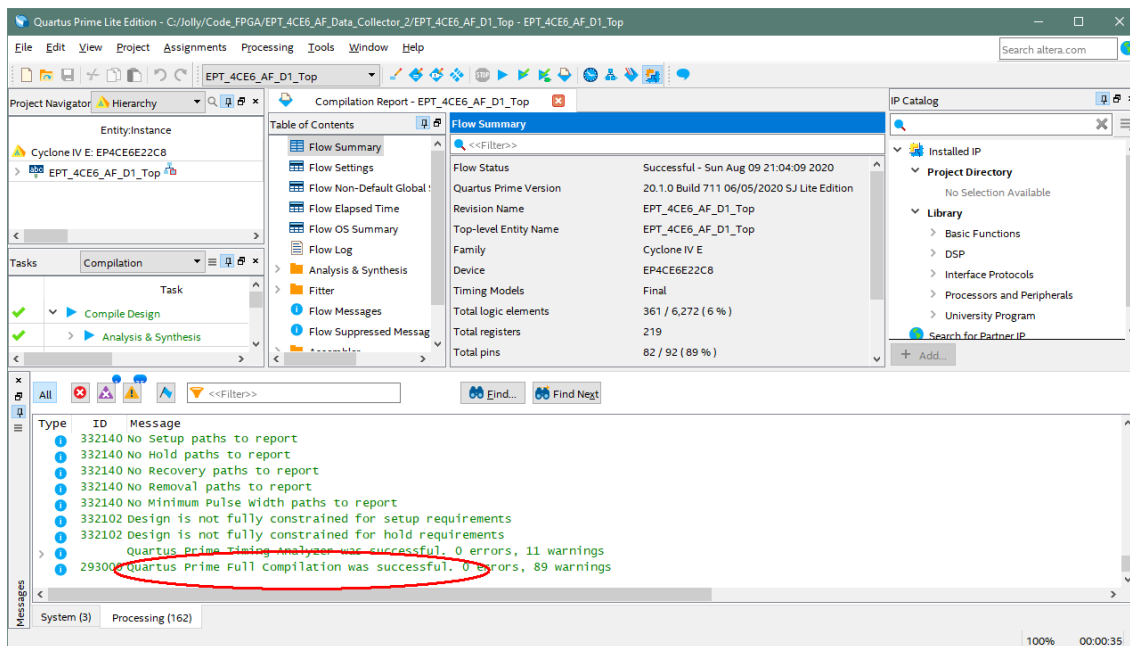
The error in this case is the missing file “active\_control\_register”. Click on the Assignment menu, then select Settings, then select Files. Add the “active\_control\_register.v” file from the database.

## FPGA Development System User Manual



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

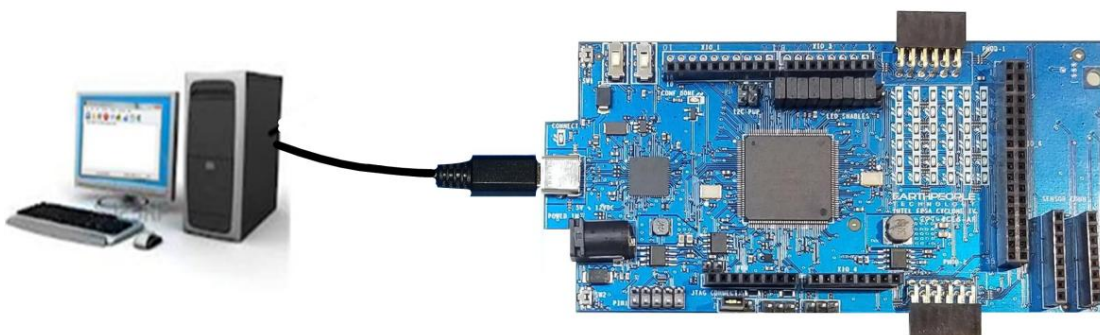
## FPGA Development System User Manual



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

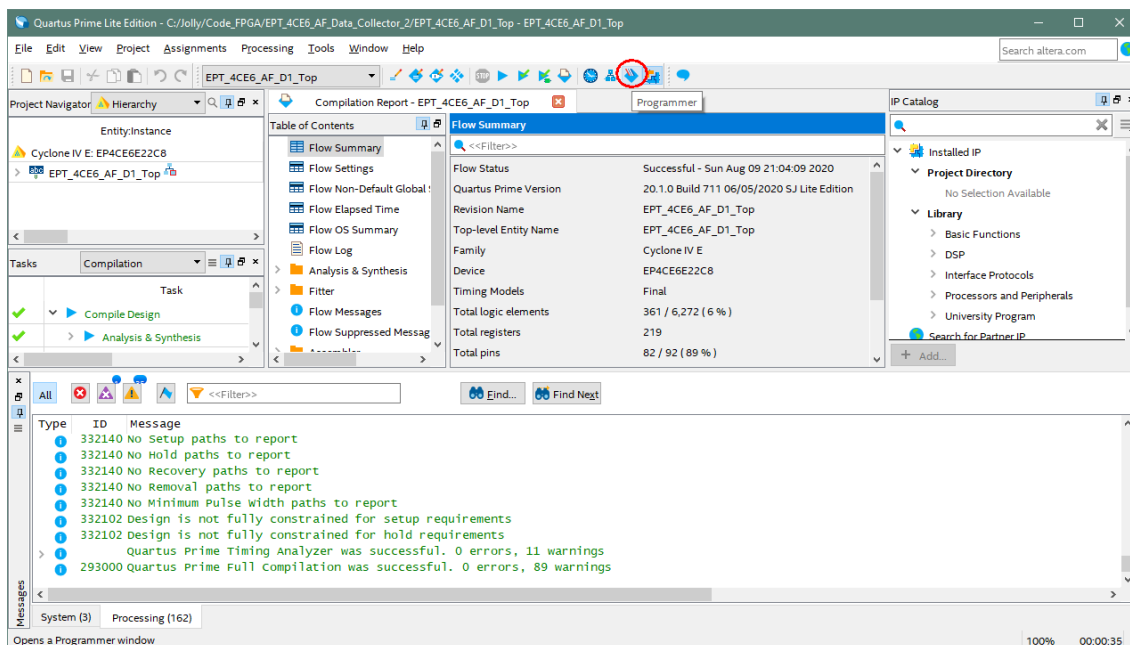
### 4.1.2 Configuring the FPGA

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

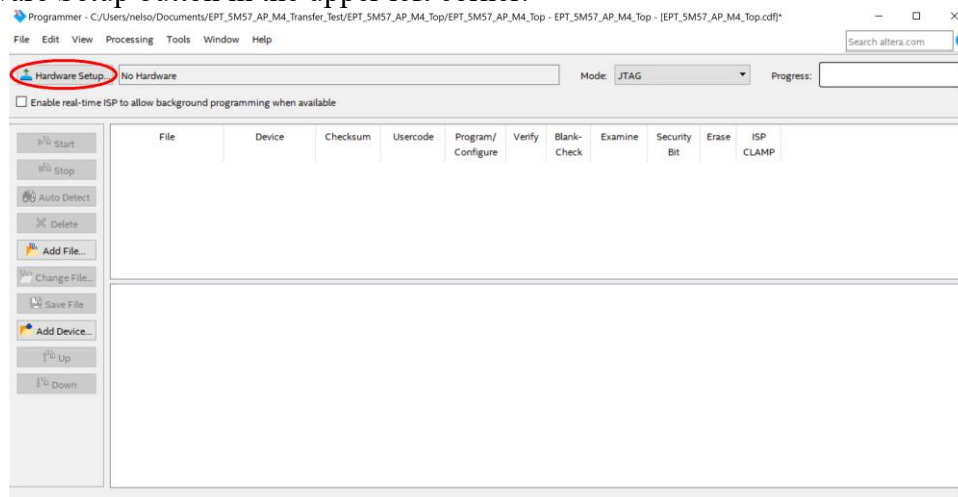


## FPGA Development System User Manual

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

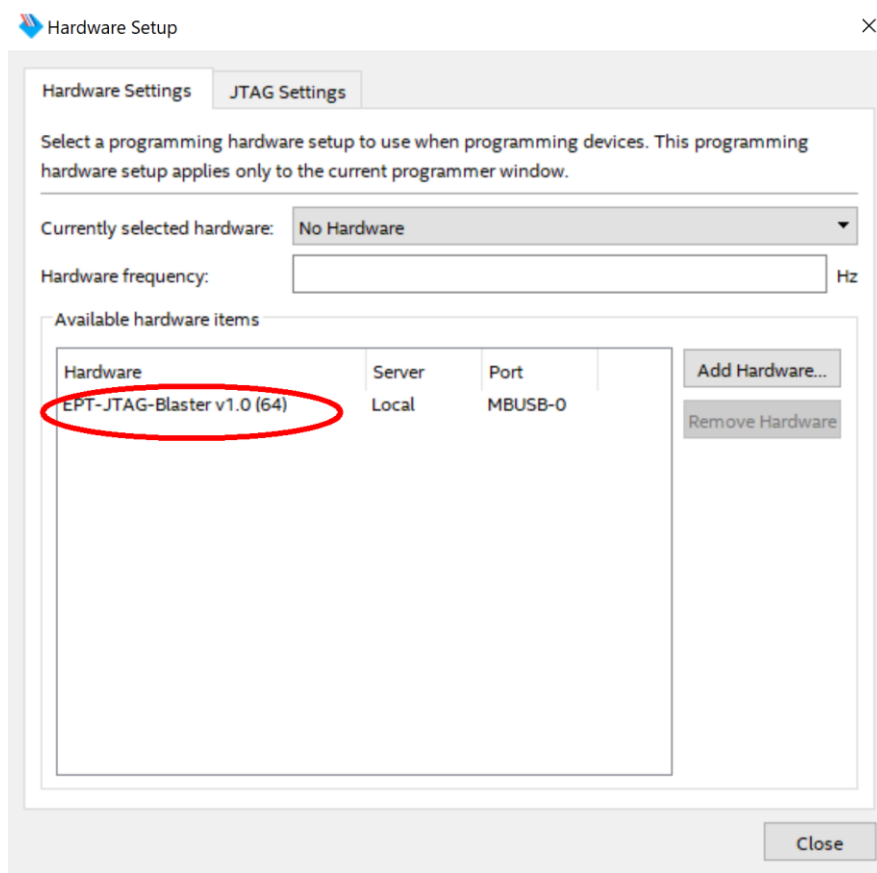


The Programmer Window will open up with the programming file selected. Click on the Hardware Setup button in the upper left corner.



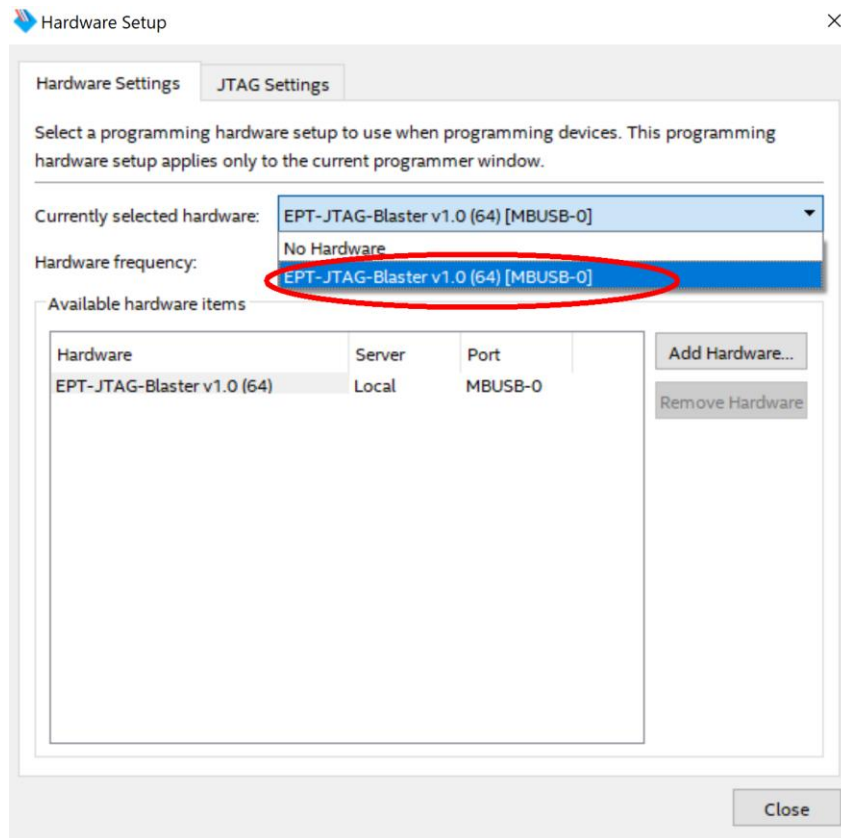
## FPGA Development System User Manual

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



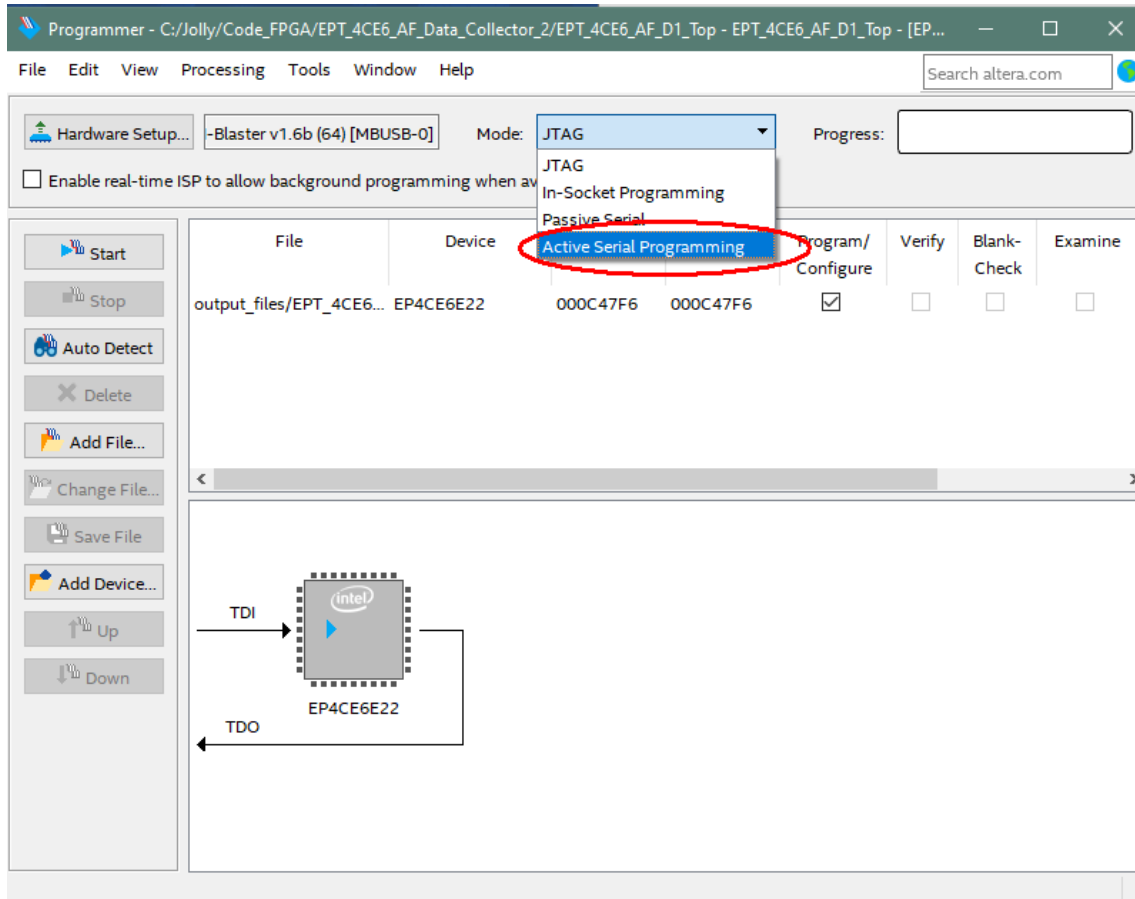
If you successfully double clicked, the “Currently selected hardware:” dropdown box will show the “EPT-Blaster v1.0 (64)”.

## FPGA Development System User Manual



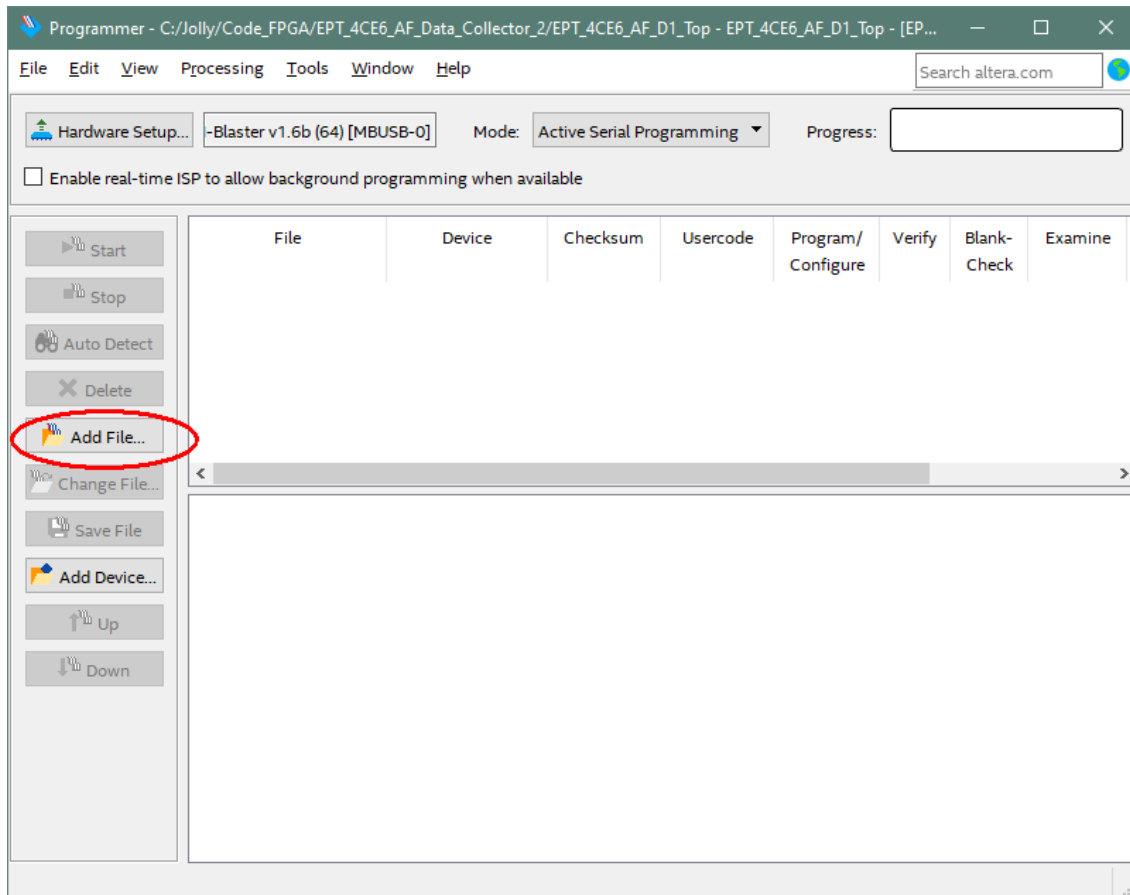
Click on the “Mode:” drop down box. Select the “Active Serial Programming” option.

## FPGA Development System User Manual

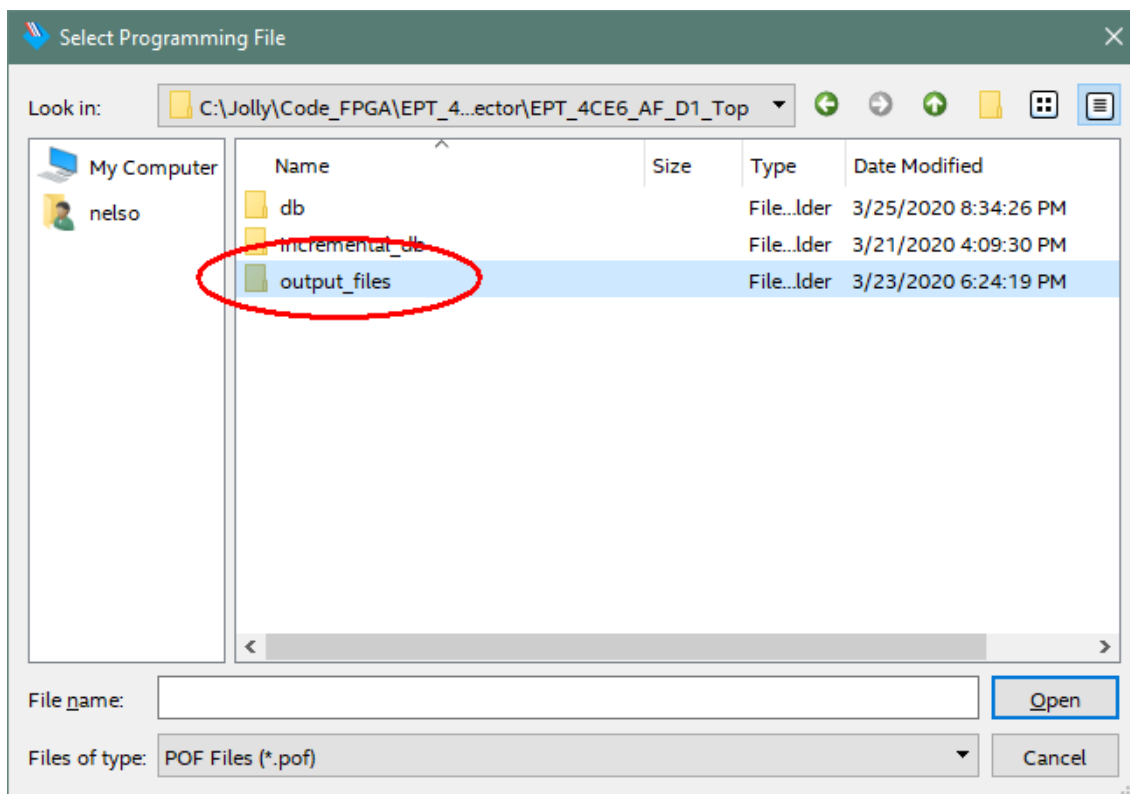


Click on the “Add File” button

## FPGA Development System User Manual

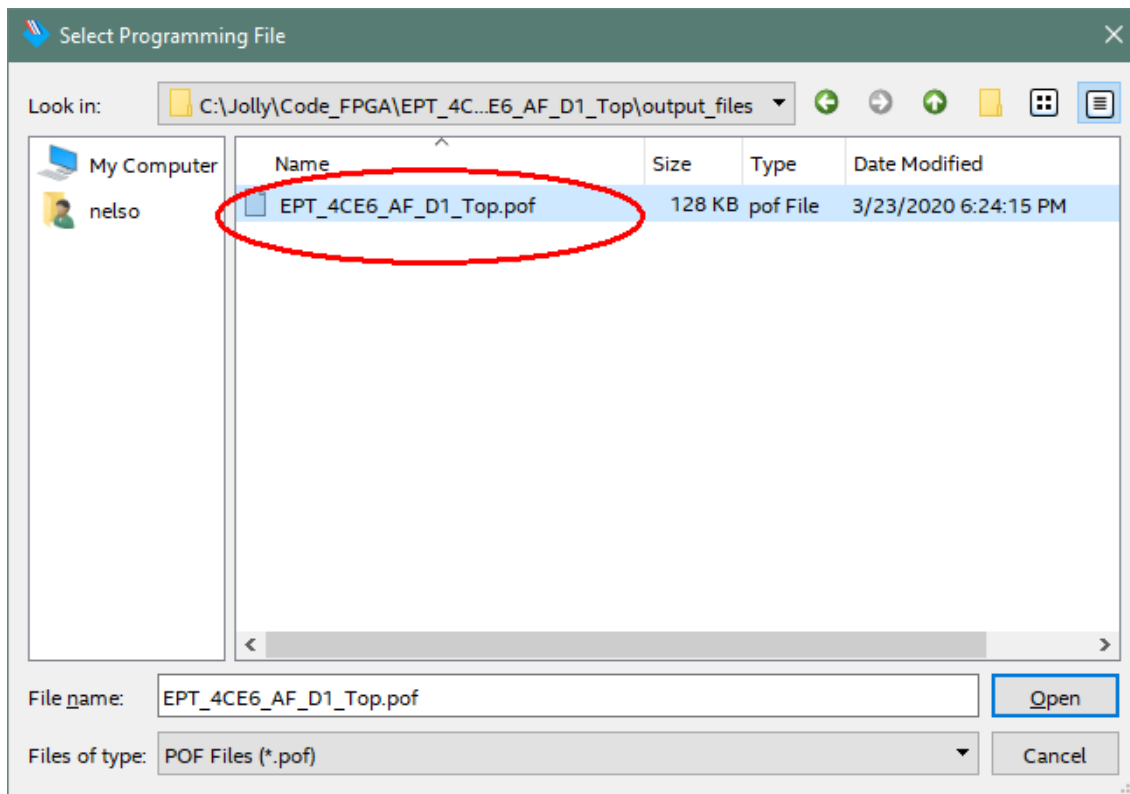


## FPGA Development System User Manual



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

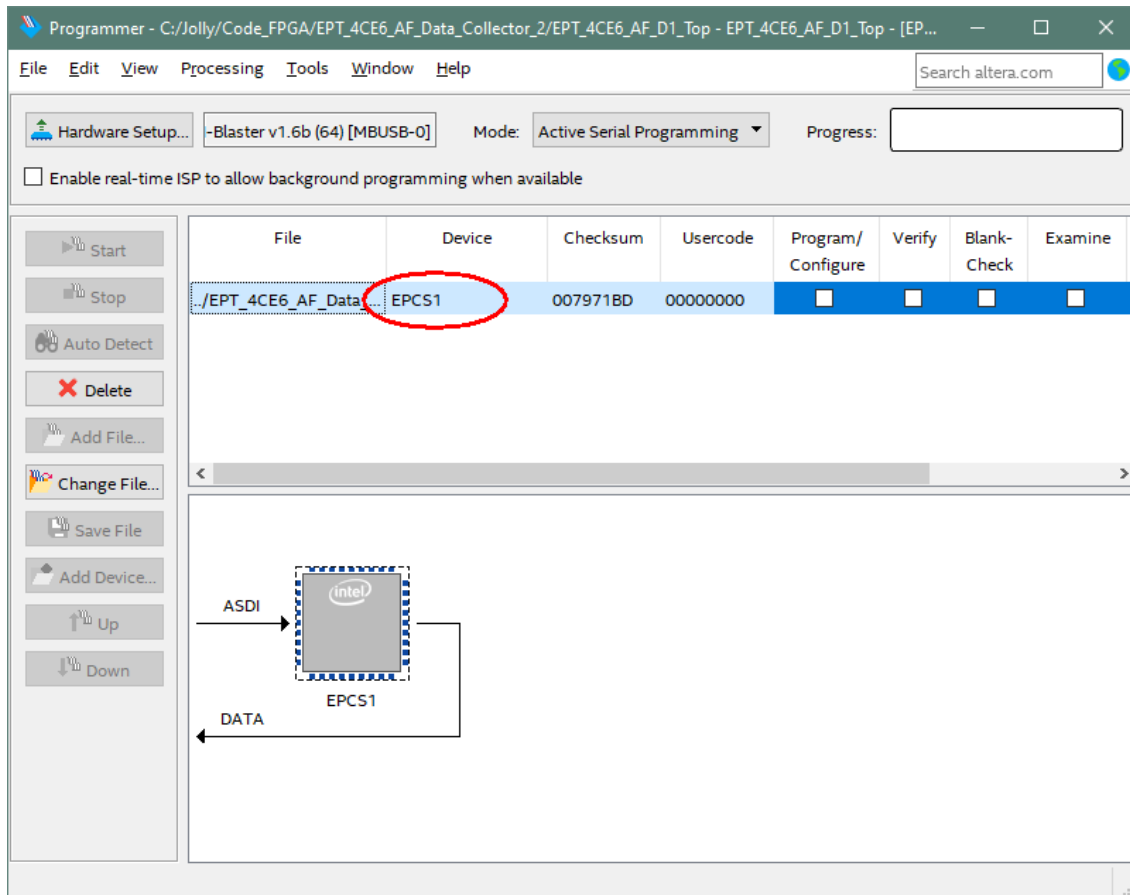
## FPGA Development System User Manual



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

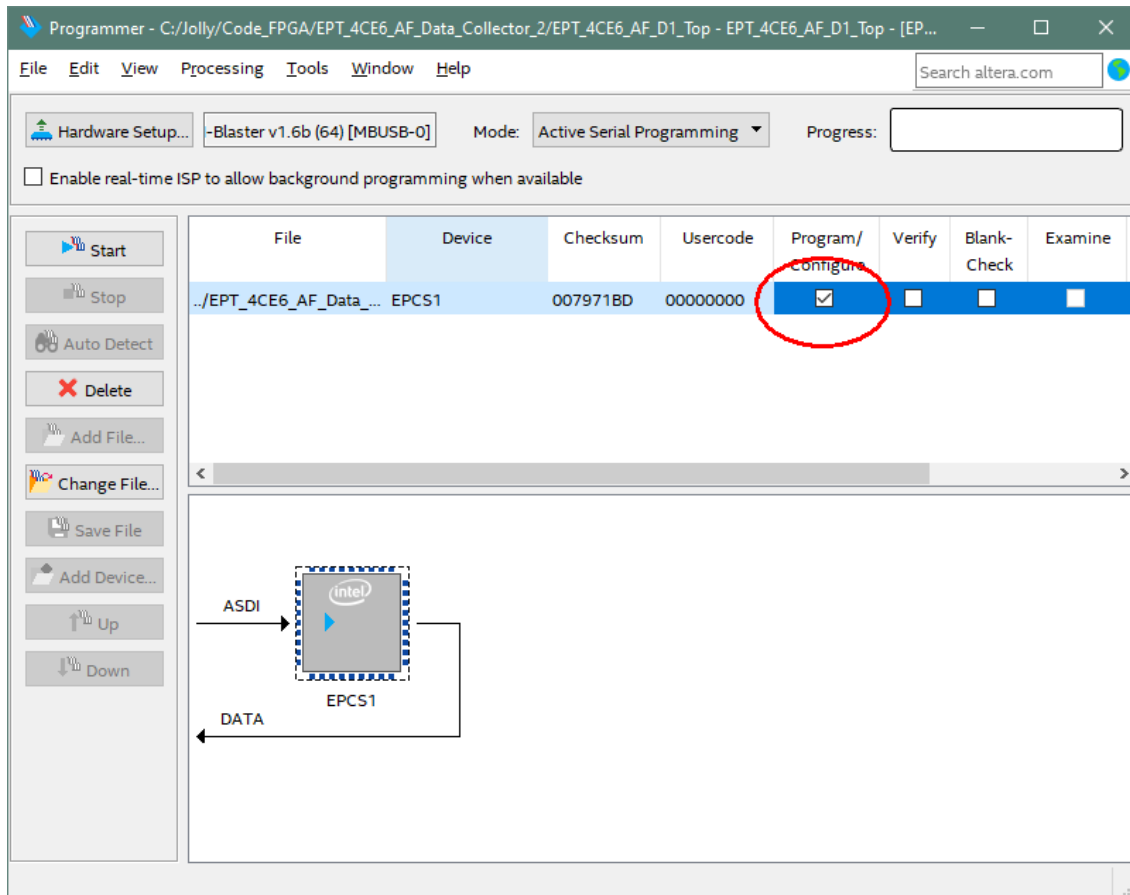
Select the EPCS1 under “Device”.

## FPGA Development System User Manual



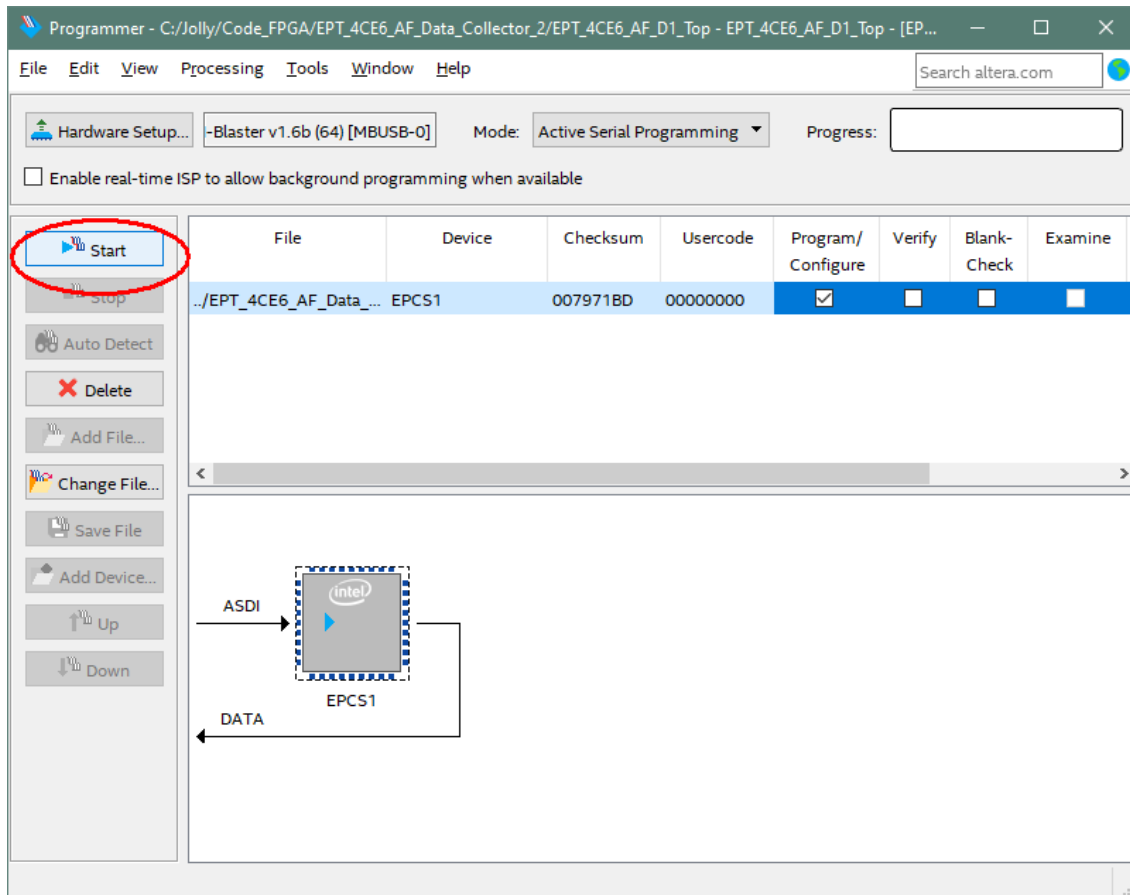
Next, select the checkbox under the “Program/Configure” of the Programmer Tool.

## FPGA Development System User Manual



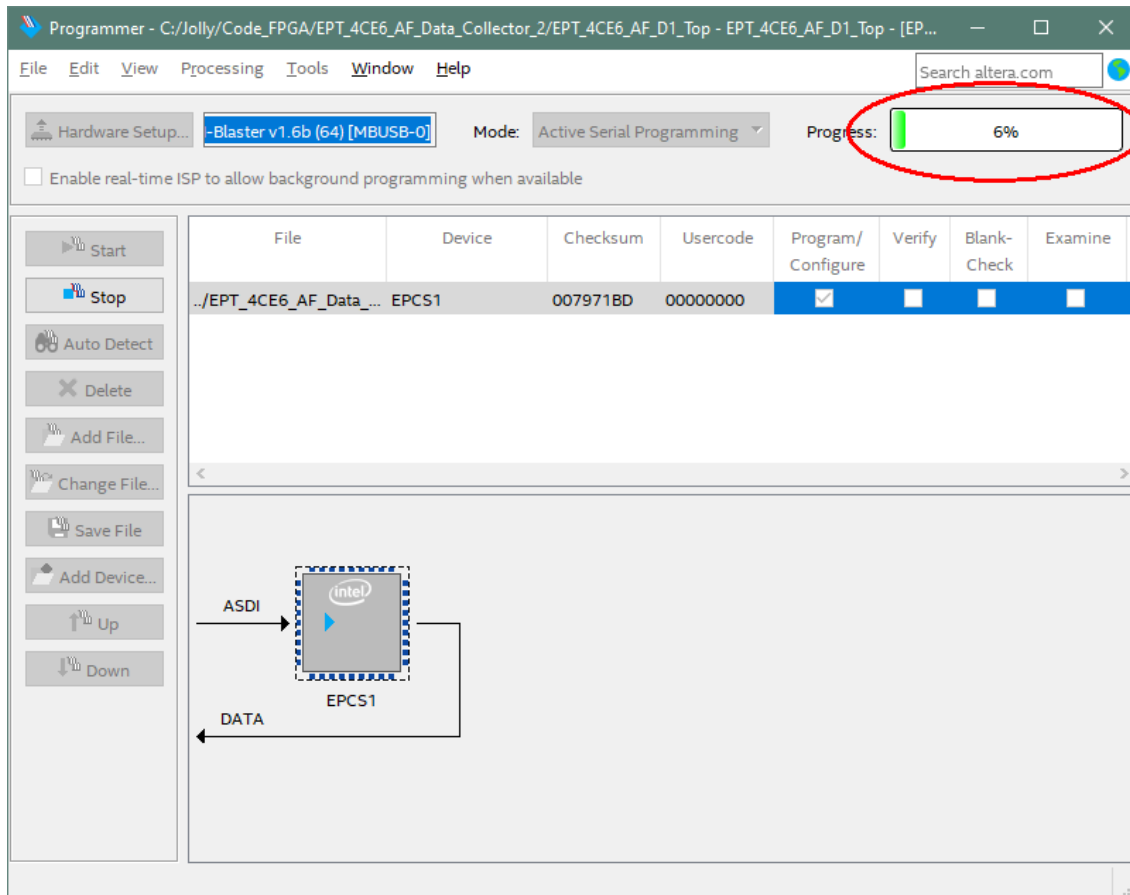
Click on the Start button to to start programming the FPGA. The Progress bar will indicate the progress of programming.

## FPGA Development System User Manual



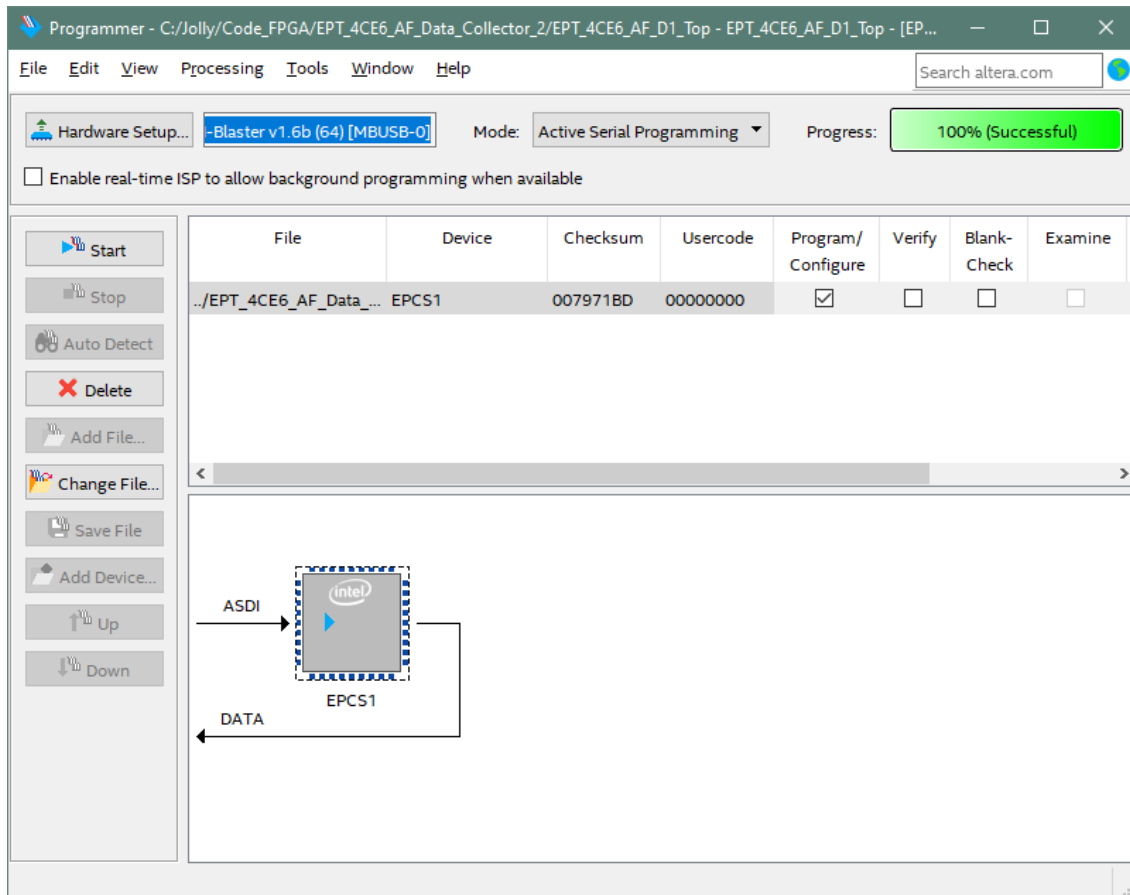
The programming of the DueProLogic will start and you can check the progress in the “Progress” Bar.

## FPGA Development System User Manual



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

## FPGA Development System User Manual



At this point, the DueProLogic is programmed and ready for use. To test that the FPGA is properly programmed, bring up the Platform Demo Tool. Click on one of the LED's and verify that the LED selected lights up. Press one of the switches on the board and ensure that the switch is captured on the Platform Demo Tool. Now you are ready to connect to the Arduino Due and write some code to transfer data between microcontroller and PC.

## 5 Active Host Application

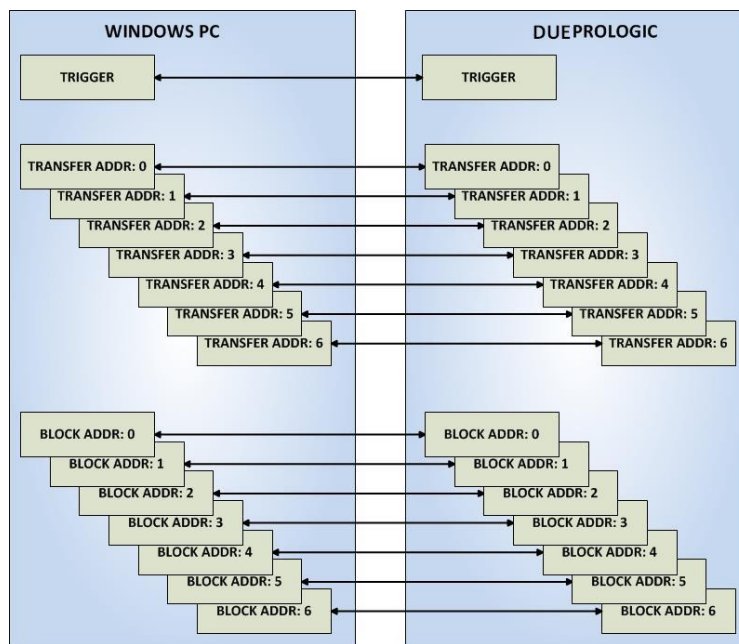
The Active Host SDK is provided which allows easy interface to application software written in C#. It runs on the PC and provides transparent connection from PC application code through the USB driver to the user FPGA code. The user code connects to "Endterms" in the Active Host SDK. These host "Endterms" have

## FPGA Development System User Manual

complementary HDL “Endterms” in the Active Transfer Library. Users have seamless bi-directional communications at their disposal in the form of:

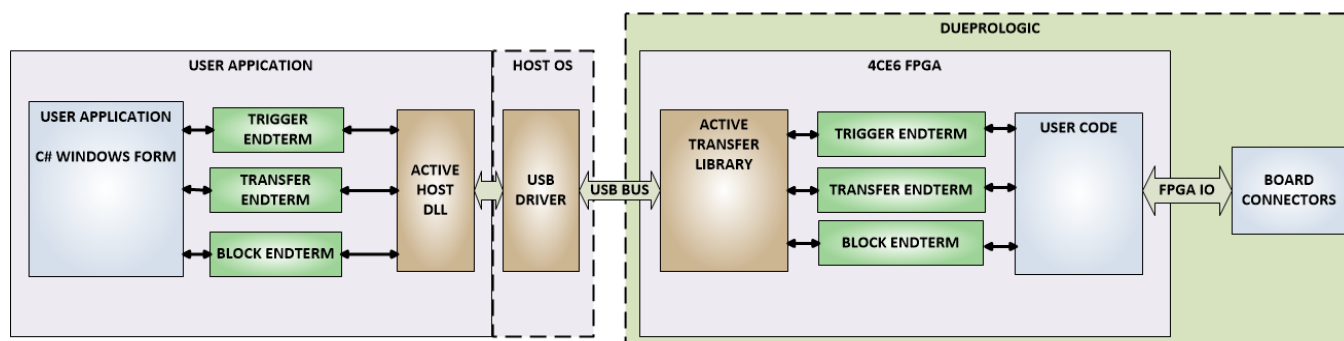
- Trigger Endterm
- Transfer Endterm
- Block Endterm

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



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

## FPGA Development System User Manual



### 5.1 Trigger EndTerm

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

### 5.2 Transfer(Byte) EndTerm

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

### 5.3 Block EndTerm

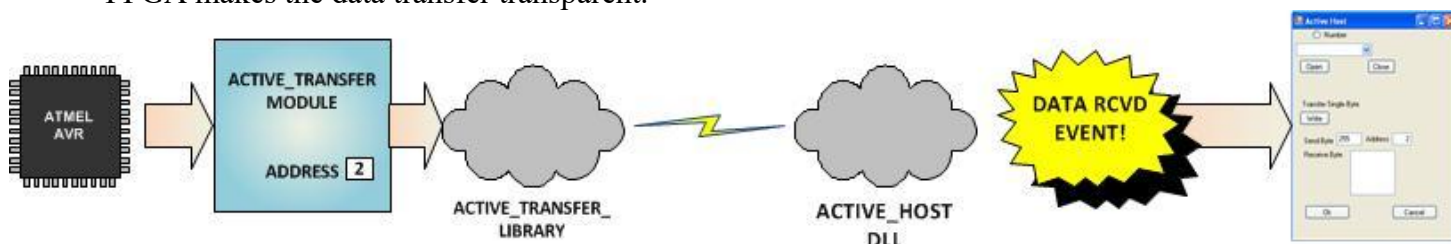
The Block EndTerm is a software component that provides a direct path from the users application to the commensurate Block EndTerm in the FPGA. The Block EndTerm is used to transfer a complete block to the FPGA. Block size is limited to 1 to 256 bytes. Eight separate Block EndTerm modules can be instantiated in the FPGA. Each module is addressed by the user application. Sending a block is easy, just use the function call

## FPGA Development System User Manual

with the address, block length, byte array. The block is buffered into a circular buffer in memory then transmitted via the USB bus to the Block EndTerm in the FPGA. Receiving a block is just as easy, a callback function is registered at initialization. When the FPGA transmits a block using its EndTerm, the callback function is called in the user application. The incoming Transfers are stored in a circular buffer in memory. This allows the user code to fetch the transfers with out losing bytes.

### 5.4 Active Host SDK

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



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

The user application will access the FPGA by use of functions contained in the Active Host SDK. The functions to access the FPGA 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. EPT Devices use the COM Port library of the Visual Studio to provide communications. The first function called when the Windows



## FPGA Development System User Manual

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 ComPortNames(). Use the function List Devices() to detect all EPT devices connected to the PC.

```
1 reference
private void EPT_Transfer_Demo_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!");

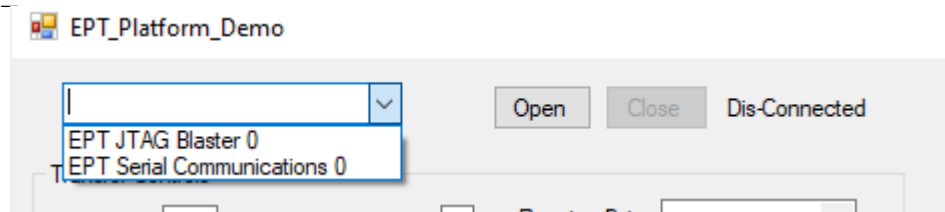
    SetButtonEnables_Close();
    //SetButtonEnables_Start();
}
```



## FPGA Development System User Manual

The ComPortNames() searches through all attached COM Ports and attempts to find the DueProLogic. Once it finds the DPL, a List is populated with each channel. The code in \*\_Load() function then adds the appropriate name to the COM Port channel and matches the COM Port number with the string in the list.

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 PortName is passed into the OpenSerialPort1 () function. If the function is successful, the device name is displayed in the label, labelDeviceCnt. Next, the device is made the active device the Open button is grayed out and the Close button is made active.

## FPGA Development System User Manual

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

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

### 5.4.2 Active Host Data Received Function

The serialPort\_AH\_DataReceived function is activated and assigned to the serial port received event. It resides in the active\_transfer.cs file. This function will be called when the serial port receives bytes from the DueProLogic. The function must determine if the incoming transfer is:

- Trigger Byte
- Transfer Byte
- Block Transfer

## FPGA Development System User Manual

```

1 reference
private void serialPort_AH_DataReceived(object sender, System.IO.Ports.SerialDataReceivedEventArgs e)
{
    uint c, 1;
    uint eptBytesToRead=0;

    if (serialPort_AH.BytesToRead < 2)
    {
        return;
    }
    int bytes = serialPort_AH.BytesToRead;
    int bytesStored = 0;
    byte[] buffer = new byte[bytes];
    serialPort_AH.Read(buffer, 0, bytes);

    //Check for incoming block
    c = (uint)buffer[0];
    c = c & 0xf8;
    if (c == 0xe0)
    {
        bytesStored = bytesStored + bytes;
        eptBytesToRead = (uint)buffer[1];
        byte[] blockBuffer = new byte[eptBytesToRead + 2];
        System.Buffer.BlockCopy(buffer, 0, blockBuffer, 0, bytes);
        while (bytesStored <= eptBytesToRead)
        {
            bytes = serialPort_AH.BytesToRead;
            buffer = new byte[bytes];
            serialPort_AH.Read(buffer, 0, bytes);

            System.Buffer.BlockCopy(buffer, 0, blockBuffer, bytesStored, buffer.Length);
            bytesStored = bytesStored + bytes;
        }

        this.Invoke(new MethodInvoker(delegate () { EPT_AH_Receive(blockBuffer); }));
    }
    else
    {
        //string WriteRcvChar = "";
        // WriteRcvChar = String.Format("{0}", (int)buffer[0]);
        this.Invoke(new MethodInvoker(delegate () { EPT_AH_Receive(buffer); }));
    }
}

```

Once the serialPort\_AH\_DataReceived() has stored all incoming bytes due to the transfer, it calls the EPT\_AH\_Receive(). This function will process the incoming transfer and populate the class:

- EPTReceiveDevice

## FPGA Development System User Manual

```

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.ToInt32(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;

            l = (uint)receiveBytes[1];
            EPTReceiveDevice.Length = l;
            //Display Length to text box
            string n = String.Empty;
            n += String.Format("EPTReceiveDevice Length = {0:x2}", EPTReceiveDevice.Length);
            n += "\r\n";
            //this.Invoke(new MethodInvoker(delegate () { textBox1.AppendText(n); }));
    }
}

```

## FPGA Development System User Manual

```

n += String.Format("EPTReceiveDevice Length = {0:X4}", EPTReceiveDevice.Length);
n += "\r\n";

EPTReceiveDevice.cBlockBuf = new byte[EPTReceiveDevice.Length];
//this.Invoke(new MethodInvoker(delegate () { textBox1.AppendText(n); }));
for (index = 2; index <= EPTReceiveDevice.Length+1; index++)
{
    EPTReceiveDevice.cBlockBuf[index - 2] = receiveBytes[index];
}
//Once EPTReceiveDevice has been populated, call the Receive Parser function
EPTParseReceive();
return;
//break;
default:
    //this.textBox1.AppendText(text);
    break;
}

//Write the payload into EPTReceiveDevice
p = (uint)receiveBytes[1];
EPTReceiveDevice.Payload = p;
//Display Payload to text box
string q = String.Empty;
q += String.Format("EPTReceiveDevice Payload = {0:X2}", EPTReceiveDevice.Payload);
q += "\r\n";

//this.Invoke(new MethodInvoker(delegate () { textBox1.AppendText(q); }));

//Once EPTReceiveDevice has been populated, call the Receive Parser function
EPTParseReceive();
}

//-----
// The Following Section of Visual Studio Code sends
// Triggers, Bytes and Block data to the EPT Device
//-----
5 references
private void EPT_AH_SendTrigger(byte trigger_value)
{
    PutBuff = new byte[1];

    // Load element 0 with the command.
    PutBuff[0] = (byte)(OUT_CONTROL_WORD + TRIGGER_OUT_COMMAND + 0x00);
    //Send byte to COM Port
    serialPort_AH.Write(PutBuff, 0, 1);
}

```

This object holds the command, address and payload for the transfer. Once the object is populated, the function calls the EPTParseReceive() function.

### 5.4.3 Active Host Triggers

The user application can send a trigger to the FPGA by using the EPT\_AH\_SendTrigger() function. First, open the EPT device to be used with 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 FPGA.

## FPGA Development System User Manual

```
private void EPT_AH_SendTrigger(byte trigger_value)
{
    PutBuff = new byte[1];

    // Load element 0 with the command.
    PutBuff[0] = (byte)(OUT_CONTROL_WORD + TRIGGER_OUT_COMMAND + 0x00);
    //Send byte to COM Port
    serialPort_AH.Write(PutBuff, 0, 1);

    // Load element 0 with the payload.
    PutBuff[0] = (byte)trigger_value;
    //Send byte to COM Port
    serialPort_AH.Write(PutBuff, 0, 1);
}
```

To detect a trigger from the FPGA, the user application will set up functions in the EPTParseReceive() function. A switch statement is used to decode which event should be called to handle the incoming received data.

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

## FPGA Development System User Manual

```
private void EPTParseReceive()
{
    // Parse the command
    switch (EPTReceiveDevice.Command)
    {
        // This is a trigger command
        case TRIGGER_IN_COMMAND:
            TriggerOutReceive();
            break;
        // This is a single-byte transfer command
        case TRANSFER_IN_COMMAND:
            TransferOutReceive();
            break;
        // This is a data block transfer command
        case BLOCK_IN_COMMAND:
            BlockInReceive();
            break;
        default:
            break;
    }
}
```

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

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

#### 5.4.4 Active Host Byte Transfers

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

```
private void EPT_AH_SendByte(byte device_channel, byte data_byte)
{
    //byte tempVar = 0;
    PutBuff = new byte[1];

    // Load element 0 with the command.
    PutBuff[0] = (byte)(OUT_CONTROL_WORD + BYTE_TRANSFER_OUT_COMMAND + device_channel);
    //Send byte to COM Port
    serialPort_AH.Write(PutBuff, 0, 1);

    // Load element 0 with the payload.
    PutBuff[0] = (byte)data_byte;
    //Send byte to COM Port
    serialPort_AH.Write(PutBuff, 0, 1);
}
```

Use the function `EPT_AH_SendByte()` to send a byte the selected module. Then add the address of the transfer module as the first parameter of the `EPT_AH_SendByte()` function. Enter the byte to be transferred in the second parameter. Then execute the function, the byte will appear in the ports of the Active Transfer module in the user code on the FPGA.

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

To receive a byte transfer from the Active host, user code must subscribe to the event created when the incoming byte transfer has arrived. 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.

## FPGA Development System User Manual

```
private void ETParseReceive()
{
    // Parse the command
    switch (EPTReceiveDevice.Command)
    {
        // This is a trigger command
        case TRIGGER_IN_COMMAND:
            TriggerOutReceive();
            break;
        // This is a single-byte transfer command
        case TRANSFER_IN_COMMAND:
            TransferOutReceive();
            break;
        // This is a data block transfer command
        case BLOCK_IN_COMMAND:
            BlockInReceive();
            break;
        default:
            break;
    }
}
```

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

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

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

### 5.4.5 Active Host Block Transfers

The Active Host Block Transfer is designed to transfer blocks of data between Host and FPGA and vice versa through the Block EndTerm. This allows buffers of data to be transferred with a minimal amount of code. The Active Host Block module (in the User



## FPGA Development System User Manual

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 `openSerialPort1()`, transfer module as the first parameter. Next, place the pointer to the buffer in the second parameter of `EPT_AH_SendBlock()`. Add the length of the buffer as the third parameter. Then execute the function, the entire buffer will be transferred to the USB chip. The data is available at the port of the Active Block module in the user code on the FPGA.

```
private void EPT_AH_SendBlock(byte device_channel, byte data_length, byte[] data_array)
{
    PutBuff = new byte[1];

    // Load element 0 with the command.
    PutBuff[0] = (byte)(OUT_CONTROL_WORD + BLOCK_TRANSFER_OUT_COMMAND + device_channel);
    //Send byte to COM Port
    serialPort_AH.Write(PutBuff, 0, 1);

    // Load element 0 with the length of data to follow in block out.
    PutBuff[0] = (byte)data_length;
    //Send byte to COM Port
    serialPort_AH.Write(PutBuff, 0, 1);

    foreach (byte b in data_array)
    {
        // Load element 0 with the payload.
        PutBuff[0] = b;
        //Send byte to COM Port
        serialPort_AH.Write(PutBuff, 0, 1);
    }
}
```

To receive a block transfer from the FPGA Device, a polling technique is used by the Active Host SDK. 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

## FPGA Development System User Manual

called from the Active Host SDK. 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()
{
    // Parse the command
    switch (EPTReceiveDevice.Command)
    {
        // This is a trigger command
        case TRIGGER_IN_COMMAND:
            TriggerOutReceive();
            break;
        // This is a single-byte transfer command
        case TRANSFER_IN_COMMAND:
            TransferOutReceive();
            break;
        // This is a data block transfer command
        case BLOCK_IN_COMMAND:
            BlockInReceive();
            break;
        default:
            break;
    }
}
```

The 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.



## FPGA Development System User Manual

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

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

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

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

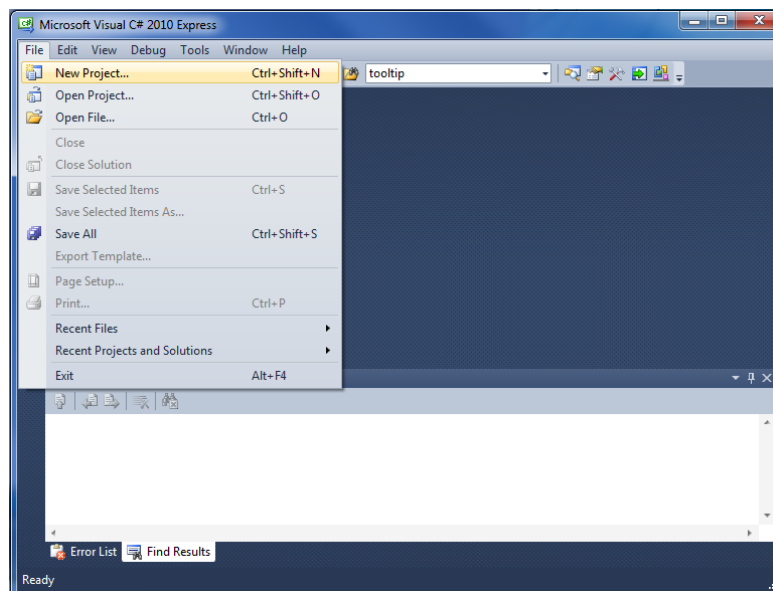
This guide will focus on writing a Windows Forms application using the C# language for the Microsoft Visual Studio with .NET Framework. This is due to the idea that beginners can write effective Windows applications with the C# .NET Framework. They can focus on a subset of the language which is very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

## FPGA Development System User Manual

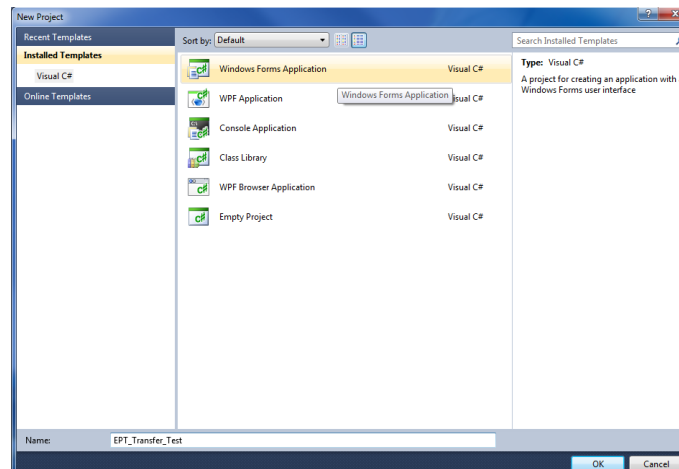
implication (such as events and controls), will be explained as the explanation progresses. Any language can be used with the Active Host Application SDK.

### **6.1 Creating a Project**

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

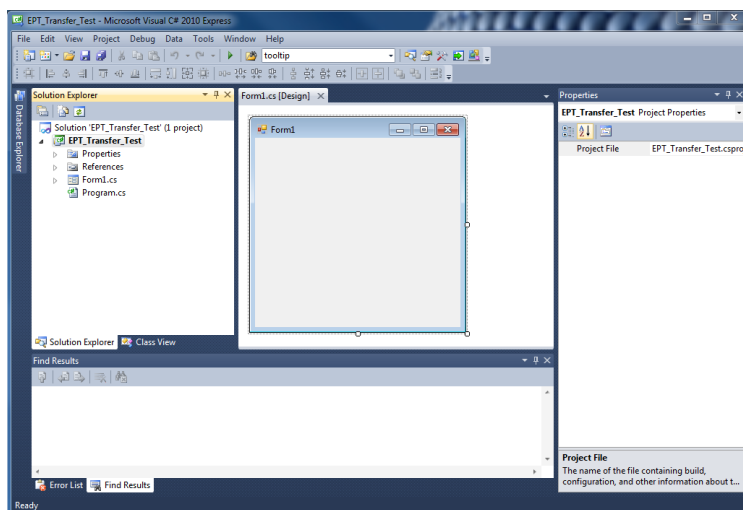


At the New Project window, select the Windows Forms Application. Then, at the Name: box, type in EPT\_Transfer\_Demo

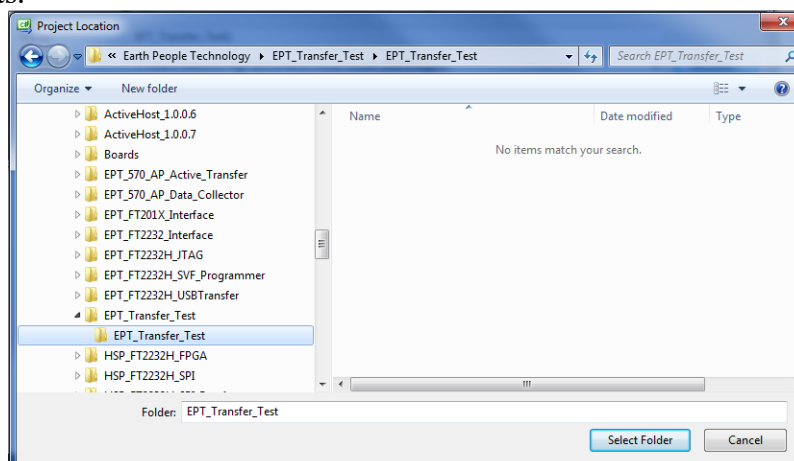


## FPGA Development System User Manual

The project creation is complete.



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

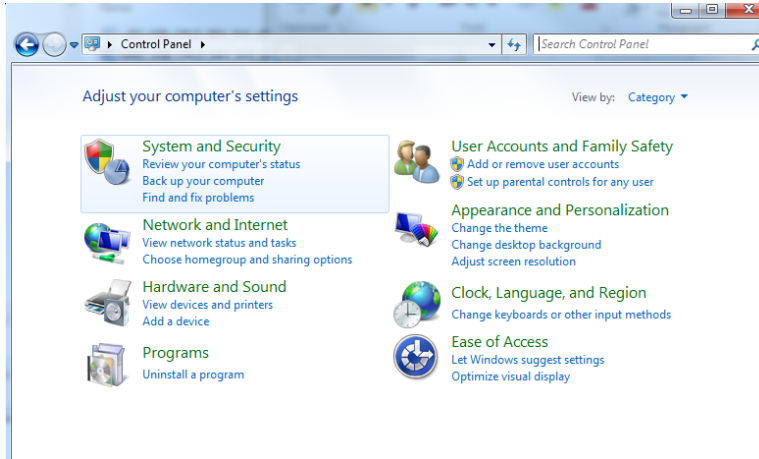


### 6.1.1 Setting up the C# Express Environment x64 bit

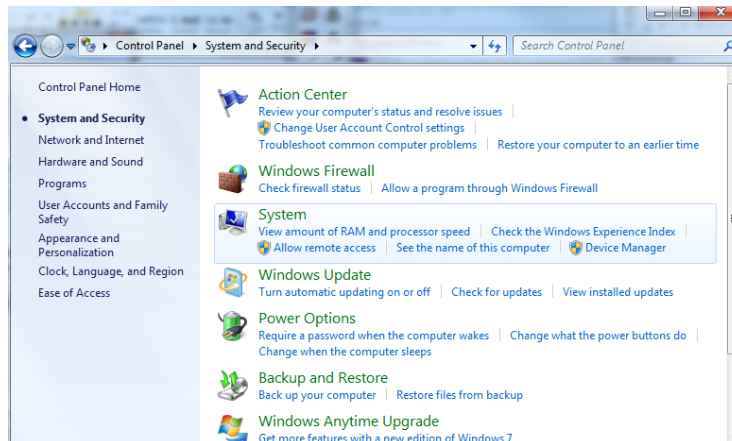
The project environment must be set up correctly in order to produce an application that runs correctly on the target platform. If your system supports 64 bit operation, perform the following steps. Otherwise if your system is 32 bit skip to the Section, Assembling Files into the Project. Visual C# Express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

## FPGA Development System User Manual

if your system supports, you can check it by going to Start->Control Panel->System and Security->System

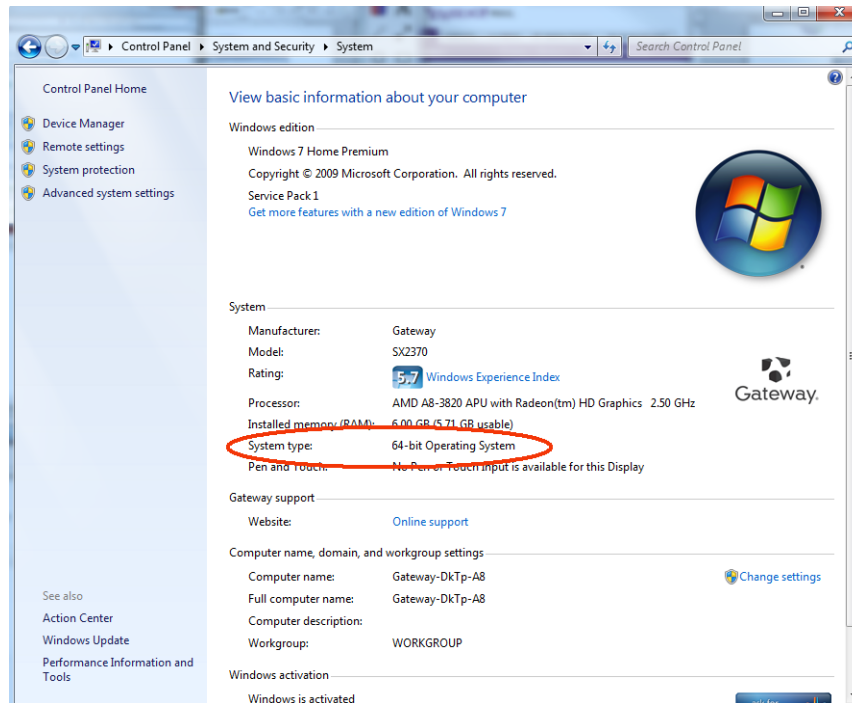


Click on System.

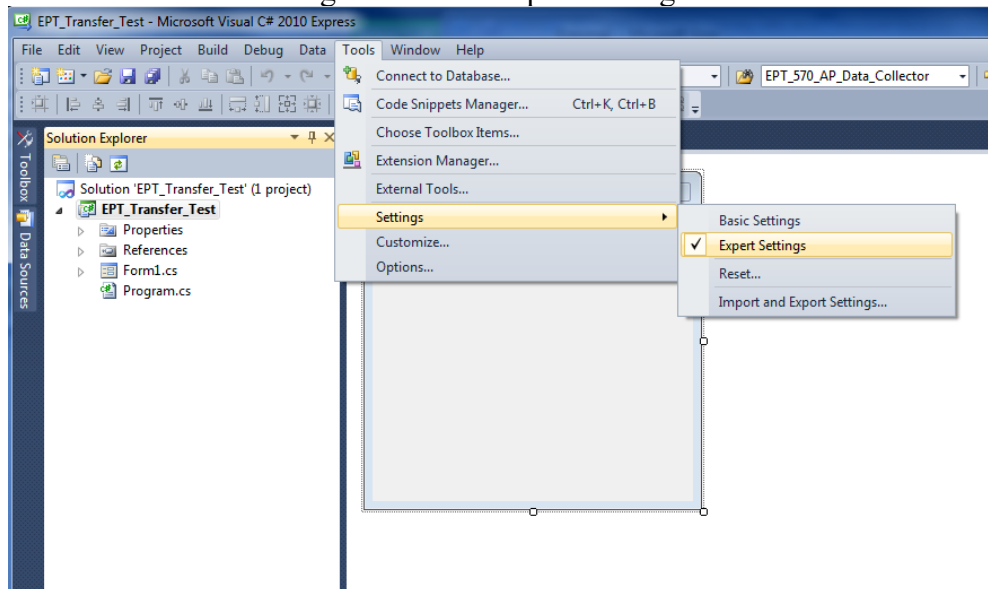


Check under System\System type:

## FPGA Development System User Manual

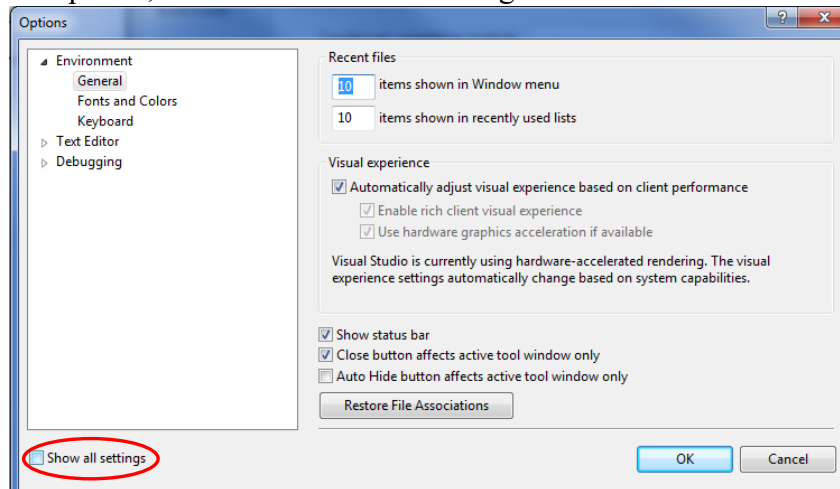


First, we need tell C# Express to produce 64 bit code if we are running on a x64 platform. Go to Tools->Settings and select Expert Settings

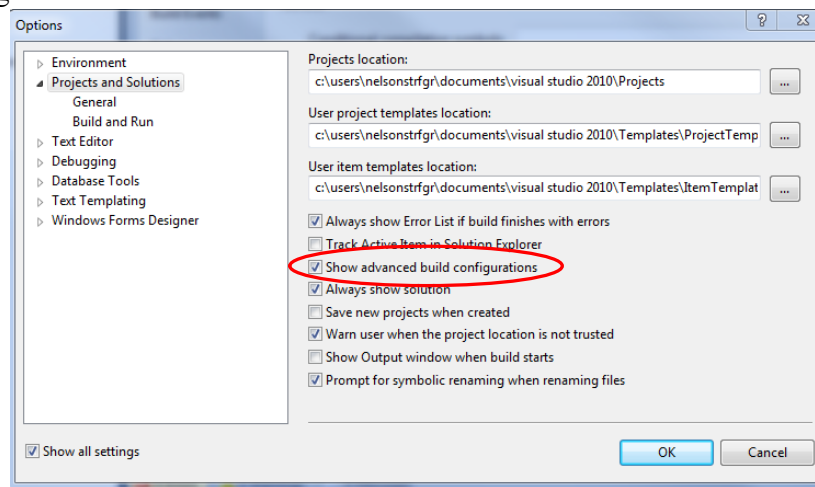


## FPGA Development System User Manual

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

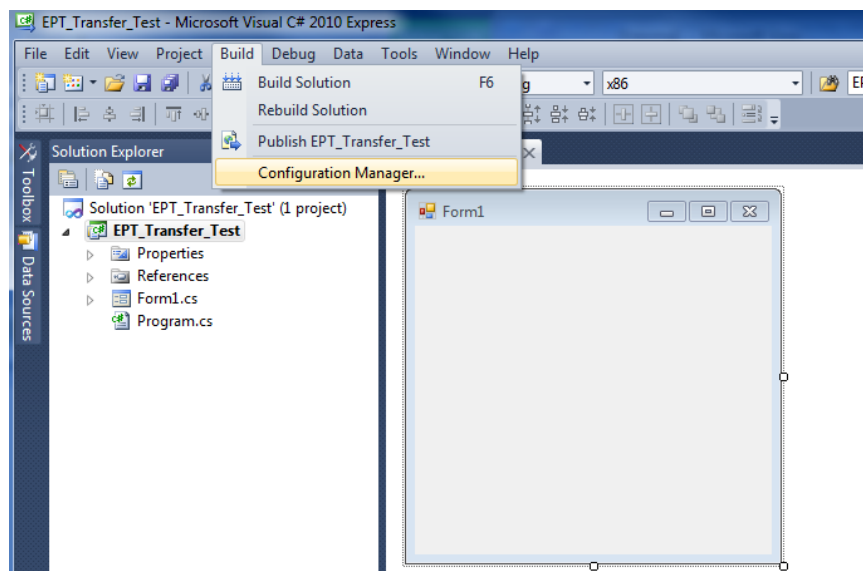


In the window on the left, go to “Projects and Solutions”. Locate the “Show advanced build configurations” check box. Check the box.

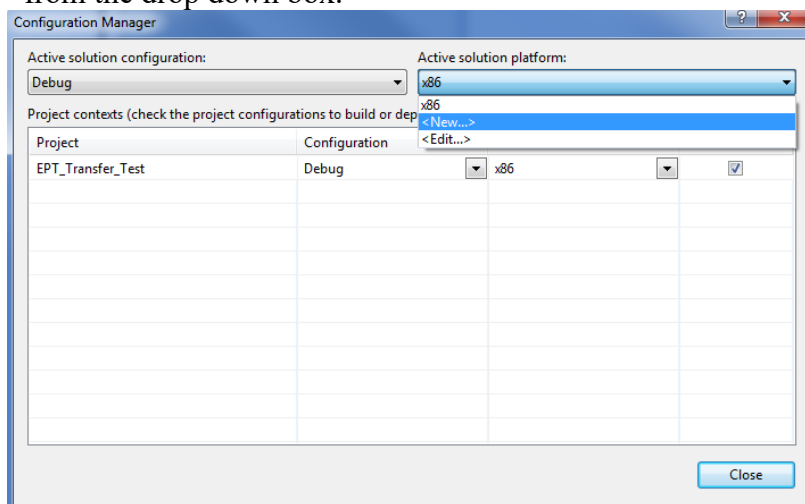


Go to Build->Configuration Manager.

## FPGA Development System User Manual

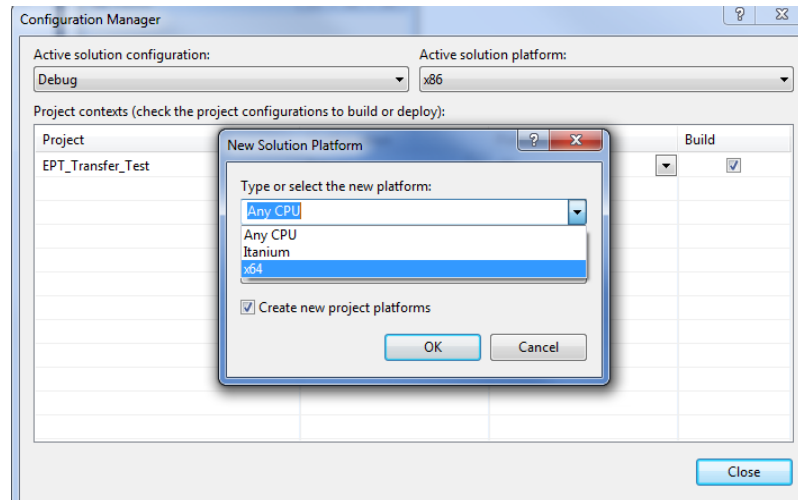


In the Configuration Manager window, locate the “Active solution platform:” label, select “New” from the drop down box.

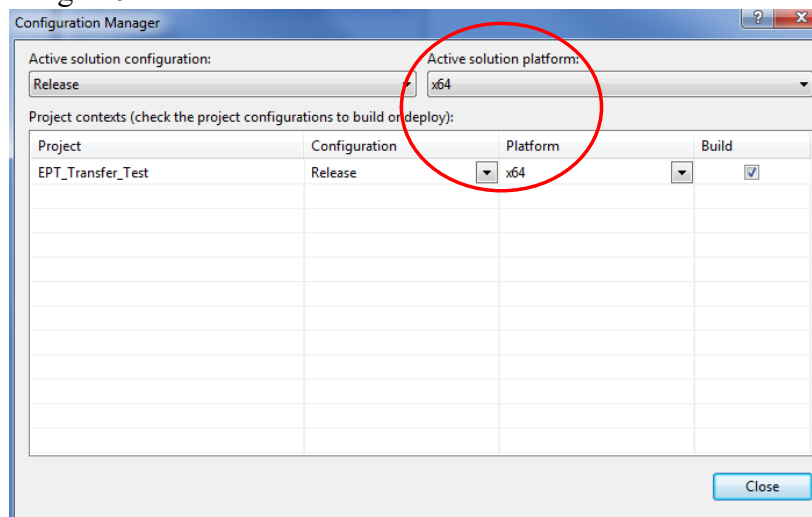


In the New Solution Platform window, click on the drop down box under “Type or select the new platform:”. Select “x64”.

## FPGA Development System User Manual

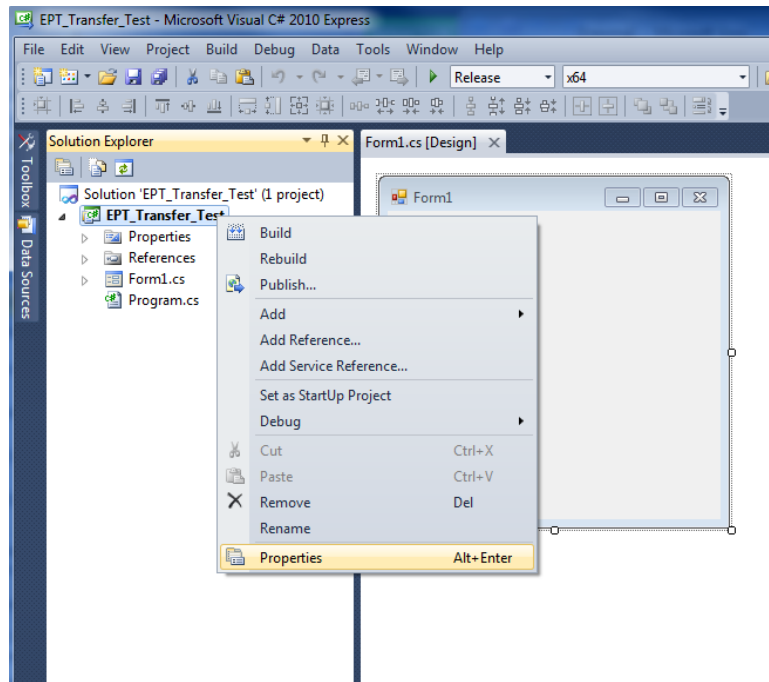


Click the Ok button. Verify that the “Active Solution Platform” and the “Platform” tab are both showing “x64”.

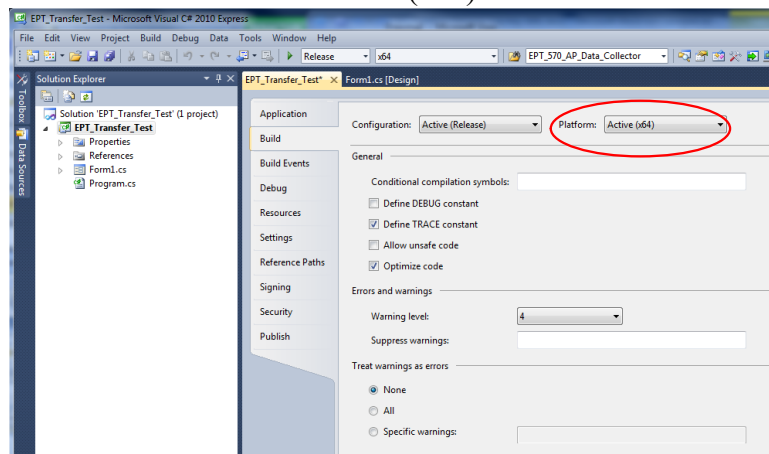


Also, select “Release” under “Active solution configuration”. Click Close. Then, using the Solution Explorer, you can right click on the project, select Properties and click on the Build tab on the right of the properties window.

## FPGA Development System User Manual



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



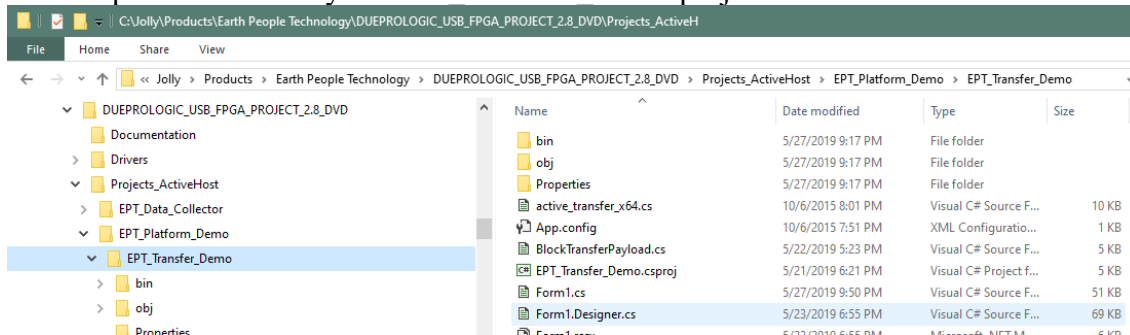
Click on the Save All button on the tool bar. The project environment is now setup and ready for the project files. Close the Project.



## FPGA Development System User Manual

### 6.2 Assembling Files into the Project

Locate the EPT FPGA Development System DVD installed on your PC. Browse to the EPT\_Platform\_Demo folder where the Project files, copy the \*.cs files, and install them in the top level folder of your EPT Platform Demo project.



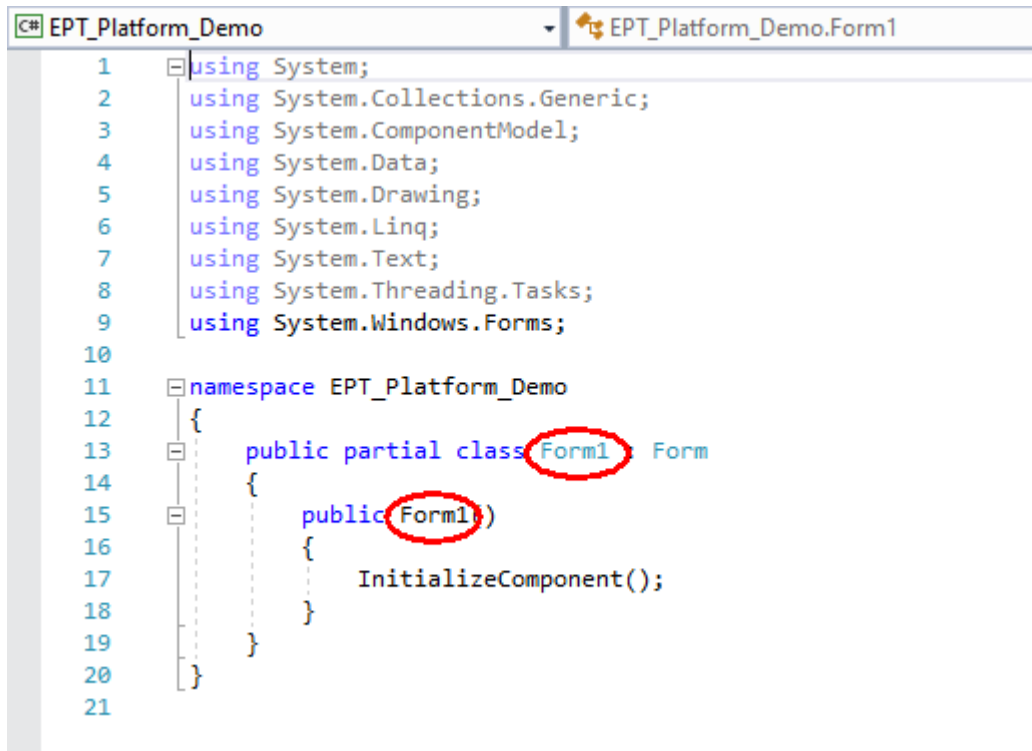
#### 6.2.1 Changing Project Name

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

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

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

## FPGA Development System User Manual



```
1  using System;
2  using System.Collections.Generic;
3  using System.ComponentModel;
4  using System.Data;
5  using System.Drawing;
6  using System.Linq;
7  using System.Text;
8  using System.Threading.Tasks;
9  using System.Windows.Forms;
10
11 namespace EPT_Platform_Demo
12 {
13     public partial class Form1 : Form
14     {
15         public Form1()
16         {
17             InitializeComponent();
18         }
19     }
20 }
21
```

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

## FPGA Development System User Manual

```
using System.Runtime.InteropServices;
using System.Diagnostics;

namespace EPT_Platform_Demo
{
    public partial class EPT_Platform_Demo : Form
    {
        public EPT_Platform_Demo()
        {
            InitializeComponent();
        }

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

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

```
namespace EPT_Platform_Demo
{
    partial class EPT_Platform_Demo
    {
        /// <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;
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }
    }
}
```

## FPGA Development System User Manual

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.btnOK);
this.Controls.Add(this.btnCloseDevice);
this.Controls.Add(this.btnOpenDevice);
this.Controls.Add(this.gbTransferControl);
this.Controls.Add(this.groupBox1);
this.Controls.Add(this.gbTriggerOut);
this.Name = "EPT Platform Demo";
this.Text = "EPT Platform Demo";
this.Load += new System.EventHandler(this.EPT_Platform_Demo_Load);
this.gbTriggerOut.ResumeLayout(false);
this.gbTriggerOut.PerformLayout();
this.gbTransferControl.ResumeLayout(false);
this.gbTransferControl.PerformLayout();
this.groupBox1.ResumeLayout(false);
this.groupBox1.PerformLayout();
this.LEDBox.ResumeLayout(false);
this.LEDBox.PerformLayout();
((System.ComponentModel.ISupportInitialize)(this.trkbrTimer)).EndInit();
```

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

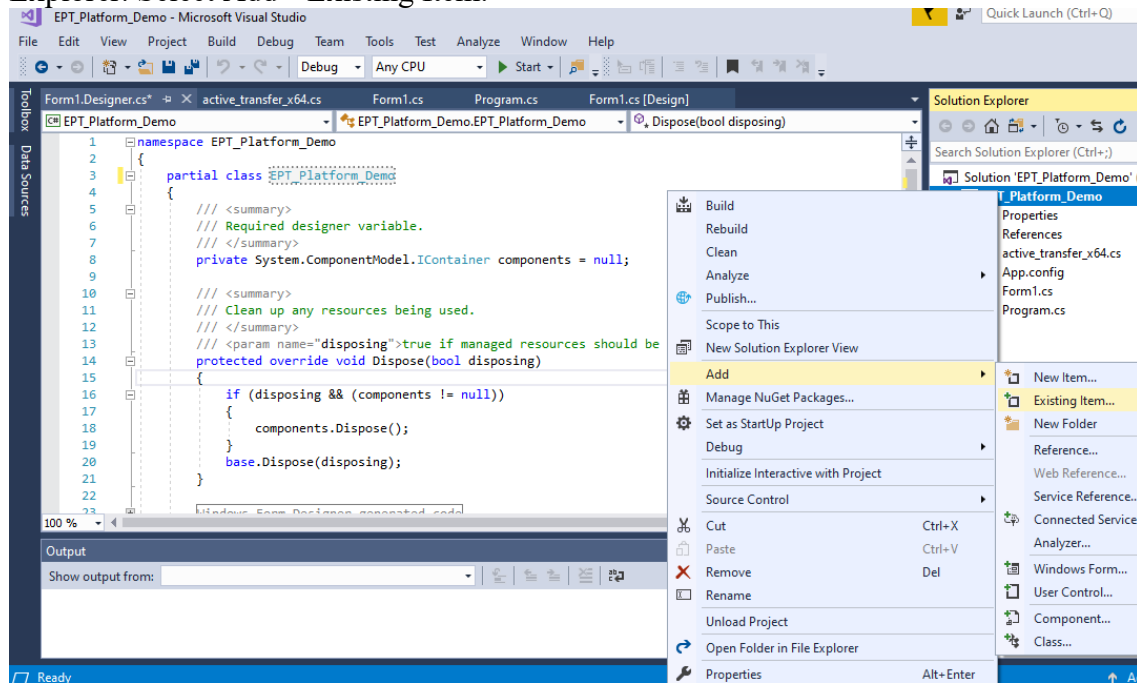
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;

namespace EPT_Transfer_Demo
{
    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_Transfer_Demo());
        }
    }
}
```

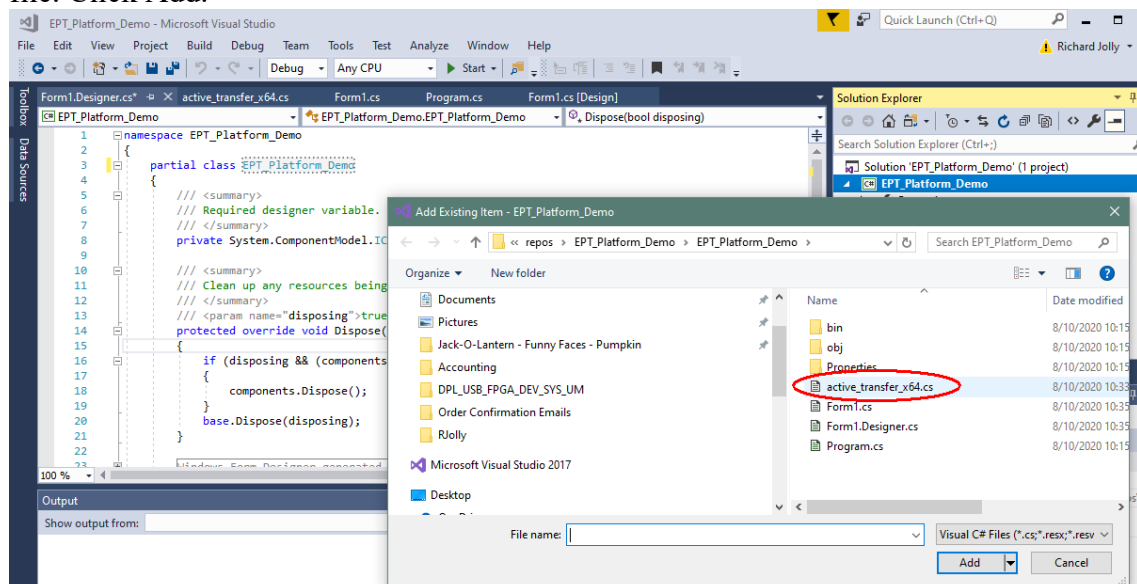
## FPGA Development System User Manual

### 6.2.2 Add Files to Project

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

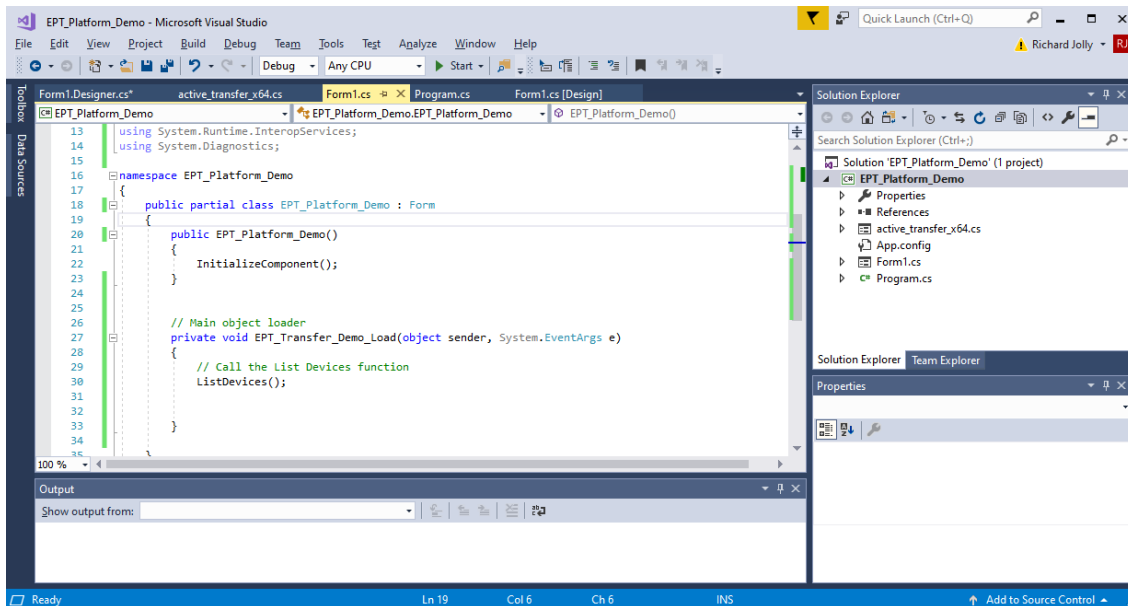


Browse to the EPT\_Platform\_Demo project folder and select the active\_transfer\_64.cs file. Click Add.



## FPGA Development System User Manual

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



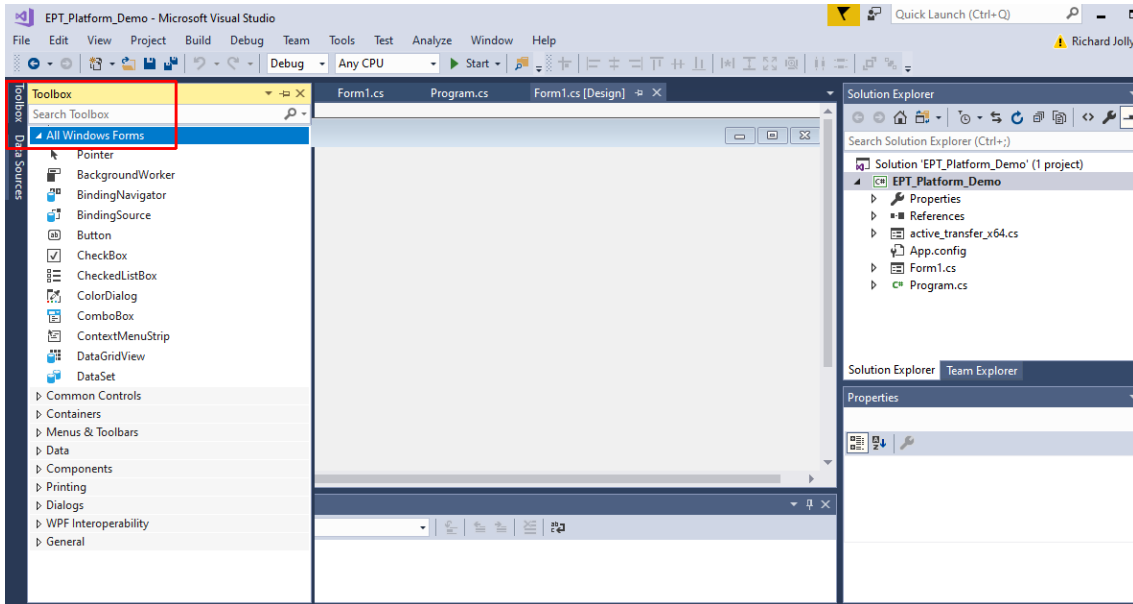
### 6.2.3 Adding Controls to the Project

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.

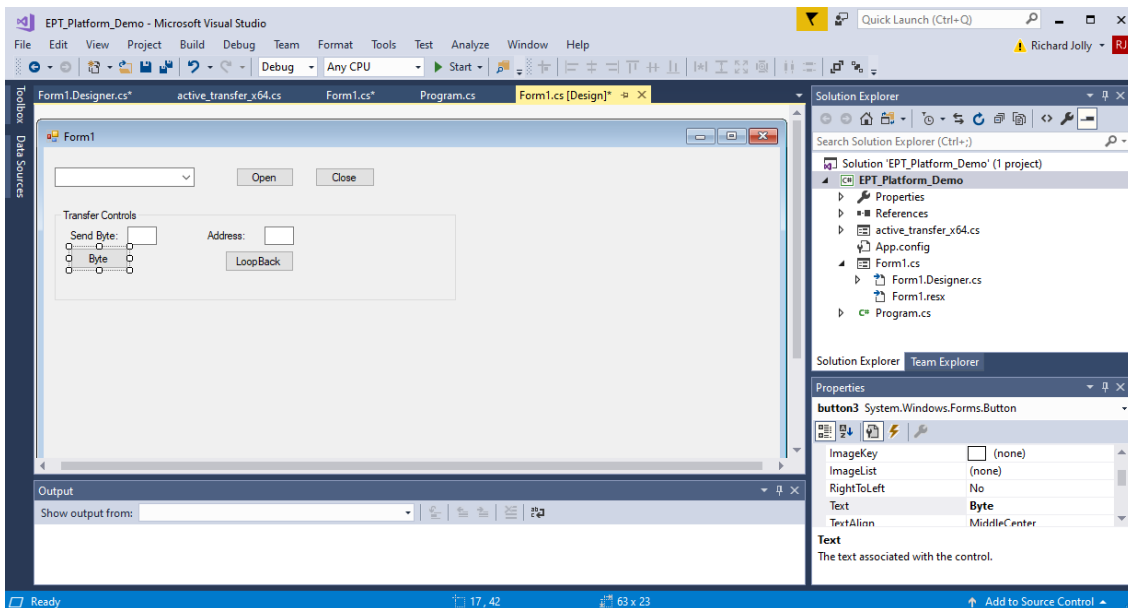
Event based programming means the software responds to events created by the user, a timer event, external events such as serial communication into PC, internal events such as the OS, or other events. The events we are concerned with for our example program are user events and the timer event. The user 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.

## FPGA Development System User Manual

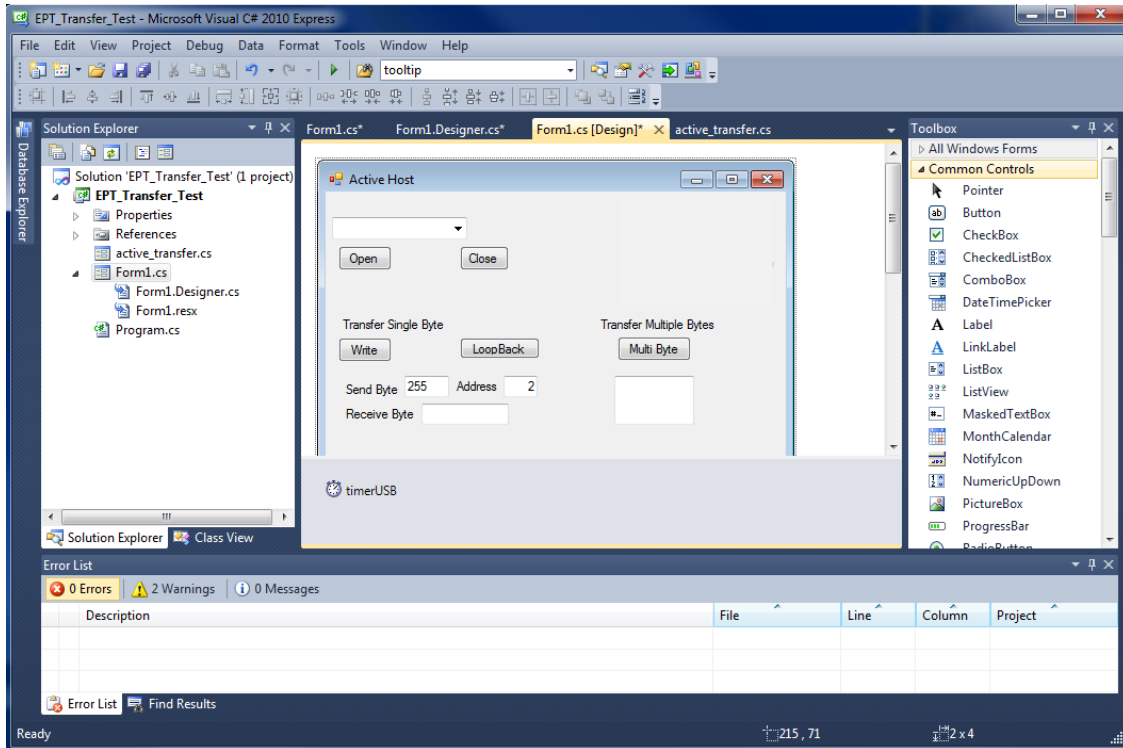


Locate the button on the Toolbox, grab and drag the button onto the Form1.cs [Design] and drop it near the top.



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.

## FPGA Development System User Manual



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

Private void btnOpenDevice\_click(object sender, System.EventArgs e)

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

## FPGA Development System User Manual

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

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

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

When btnOpenDevice\_Click is called, it calls the function “OpenDevice()”. This function 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

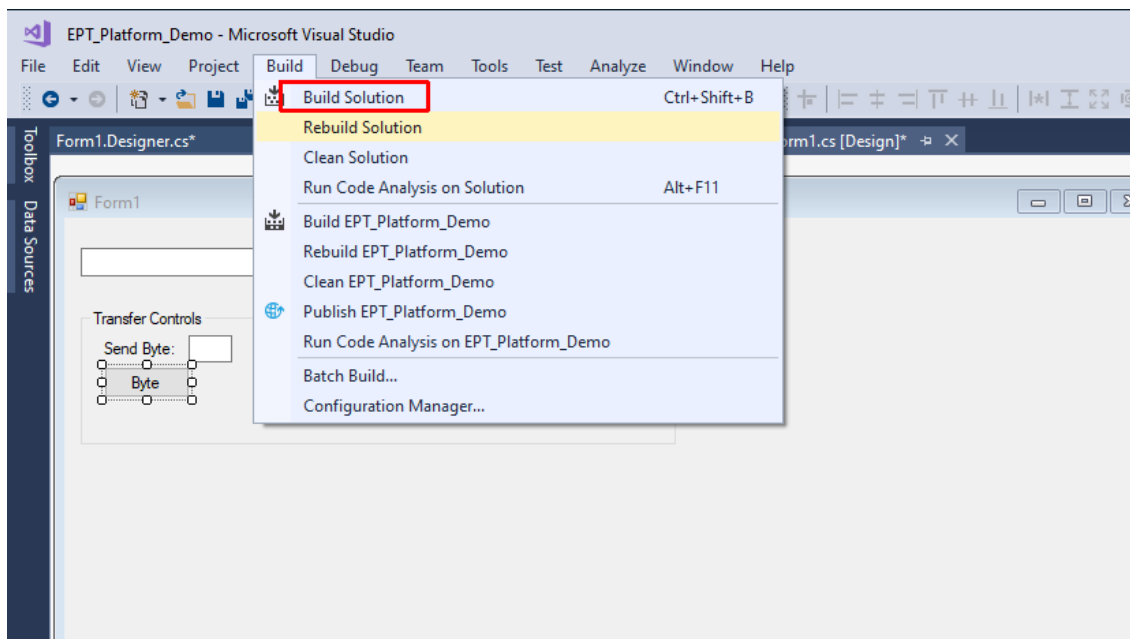
<http://www.homeandlearn.co.uk/csharp/csharp.html>

## FPGA Development System User Manual

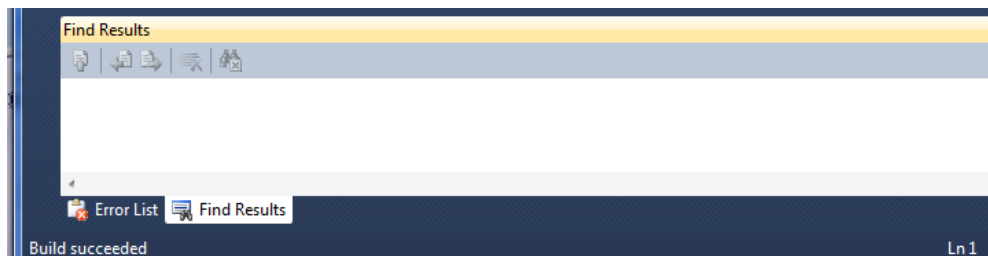
to become intimately familiar with Visual C# .NET programming. In the meantime, follow the examples from the Earth People Technology to perform some simple reads and writes to the EPT USB-FPGA Development System.

### 6.2.4 Building the Project

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



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

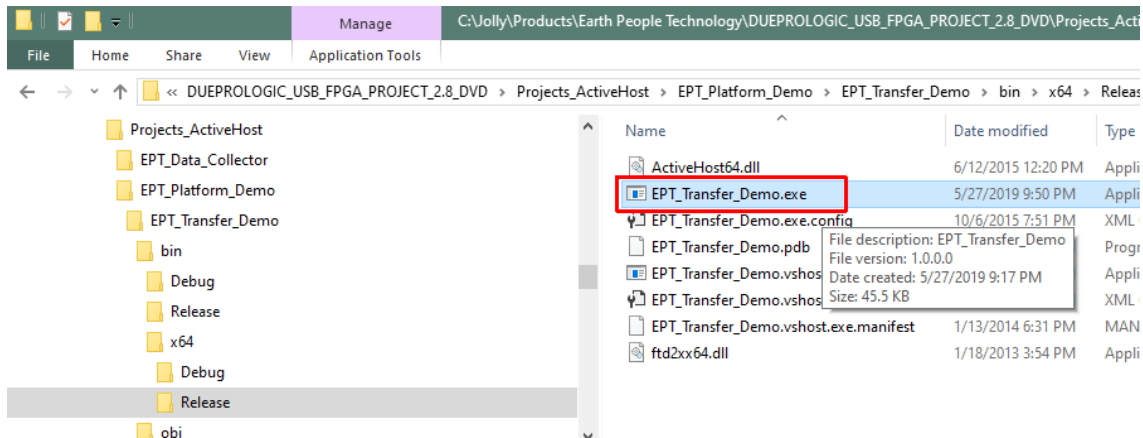




## FPGA Development System User Manual

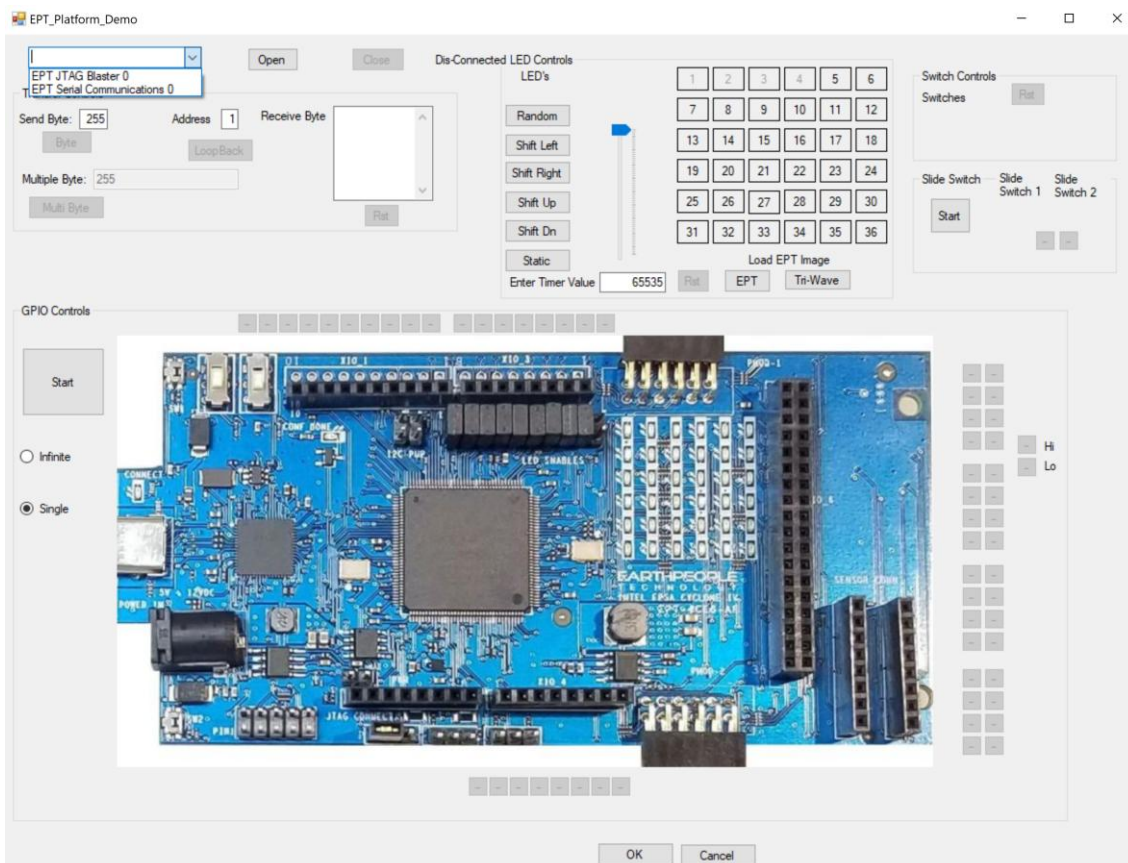
### 6.2.5 Testing the Project

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



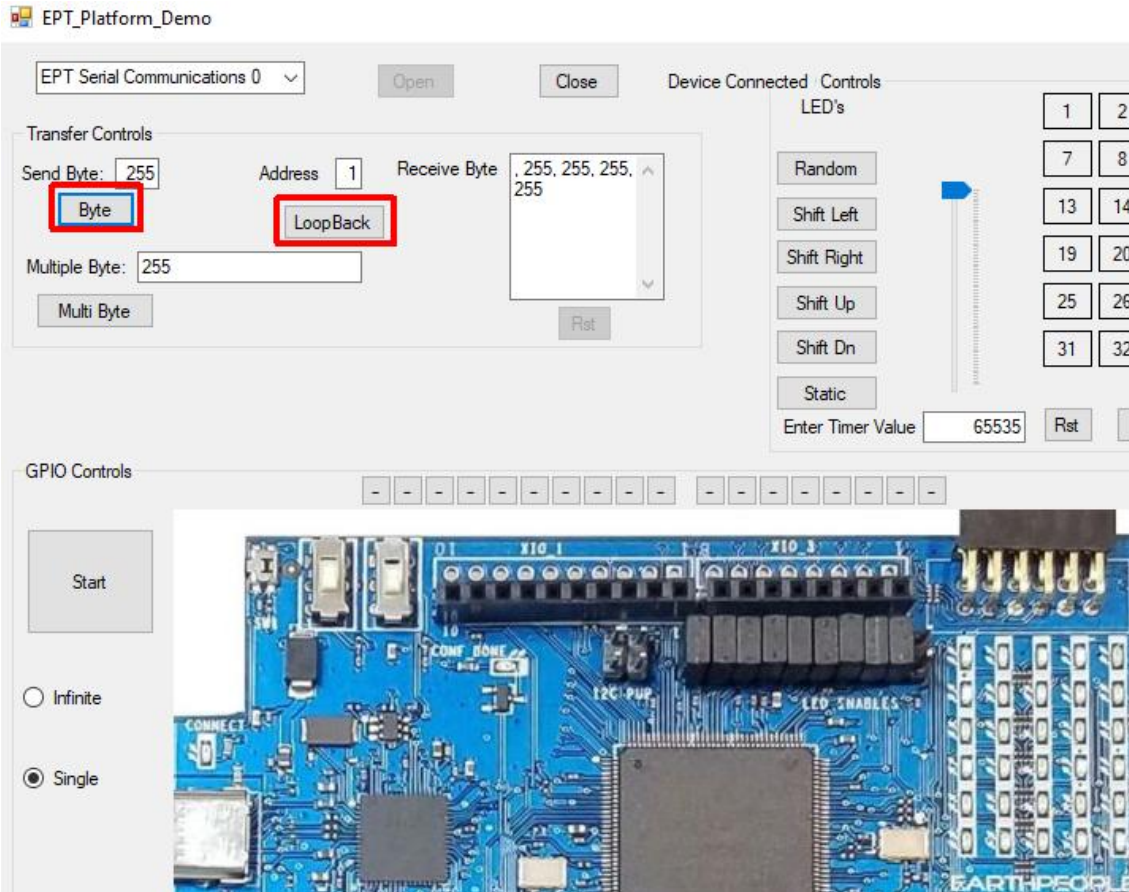
The EPT\_Platform\_Demo.exe file can now be tested using the DueProLogic board. To test the file, connect the DueProLogic to the Windows PC using Type A to USB-C cable. Make sure the driver for the board loads. If the USB driver fails to load, the Windows OS will indicate that no driver was loaded for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

## FPGA Development System User Manual



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

## FPGA Development System User Manual



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

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

If there are any questions about the hardware, software, drivers, user manual, please contact Earth People Technology. There are three methods to contact EPT for support:

<https://www.earthpeopletechnology.com->Forums>  
[support@earthpeopletechnology.com](mailto:support@earthpeopletechnology.com)  
[sales@earthpeopletechnology.com](mailto:sales@earthpeopletechnology.com)



## FPGA Development System User Manual