EARTH PEOPLE TECHNOLOGY (EPT), Inc

USB/PLD DEVELOPMENT SYSTEM FOR THE ARDUINO UNO

The EPT USB/PLD development system provides an innovative method of developing and debugging the the users microcontroller code. It can also provide a high speed data transfer mechanism between microcontroller and Host PC. The user's circuit board may be custom-built or an off-the shelf board (such as, for example, an Arduino open source prototyping platform) containing, typically, a USB connector and a microcontroller for which the user generates and loads appropriate code. The EPT USB/PLD development system provides a convenient, user-friendly method of devising and controlling code to test and debug the user's circuit board. The test code is loaded into a PLD (programmable logic device) on an EPT circuit board which connects to the user's board. The test code is generated and downloaded from a host Personal Computer (PC). The Hyper Serial Port software resides on the host PC and provides an interactive means of communicating with and controlling the EPT PLD code and thus the user's circuit board via the PC USB (universal serial bus). The EPT USB/PLD development system is a unique combination of proprietary hardware and software.

http://www.earthpeopletechnology.com/

Table of Contents

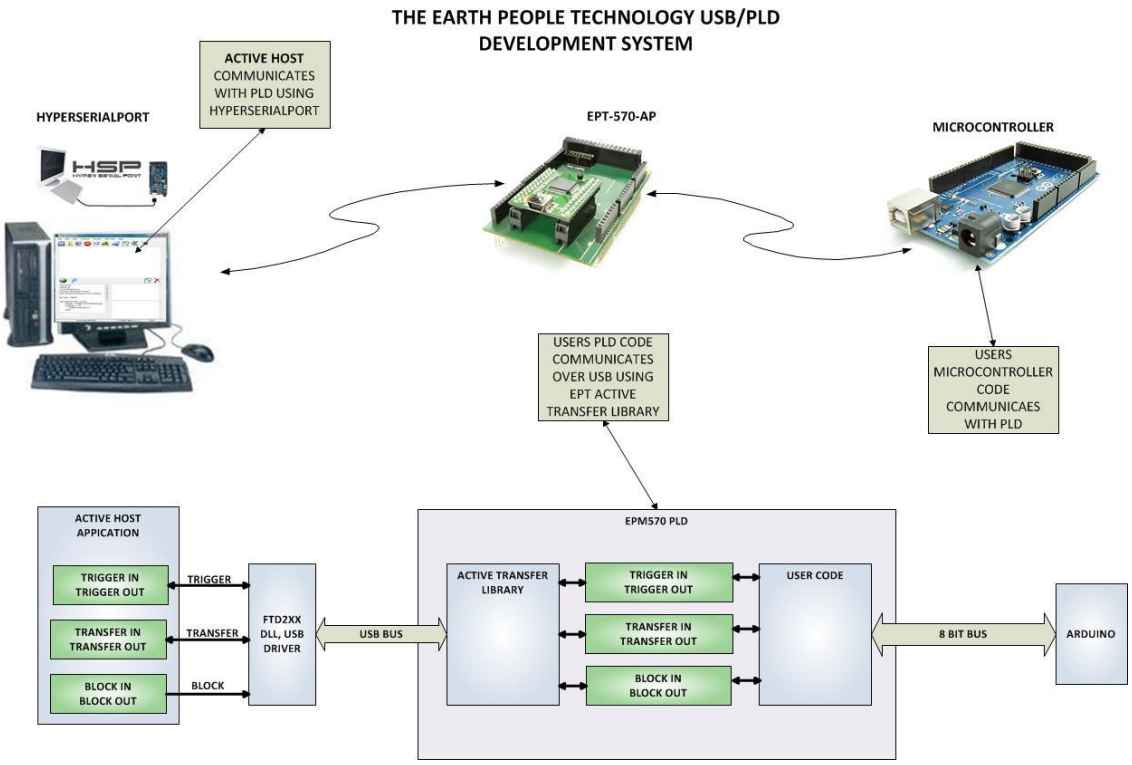
1 Introduction and General Description

The Earth People Technology USB/PLD development system comprises model number EPT-570-AP-XX. The Hardware consists of a High Speed (480 Mb/s) USB to parallel (8 bit) bus and PLD. The software consists of support from HyperSerialPort in the form of Active Host. The firmware, Active Transfer Library, is used on the PLD to provide advanced functions for control and data transfer to/from the Arduino.

1.1 Block Diagram

An overall block diagram of the EPT USB/PLD development system is shown in figure 1 below.

Figure 1.



The user's microcontroller code is being developed to perform particular functions required by the user. The code is downloaded to the device using the hardware/software system provided as part of the microcontroller development system. The EPT-570-AP USB/PLD Development System consists of a board base onto which is plugged the FT2232H board. These boards comprise the hardware components of the EPT USB/PLD development system. The PLD allows users to write HDL code that will implement any digital logic circuit. The users HDL code is compiled and synthesized and packaged into a programming file. The packaged file is programmed into the PLD using the side A channel of the USB to Serial chip, FT2232H.The Hyper Serial Port contains a software GUI (graphical user interface) and USB I/O (input/output) communication software. The Hyper Serial Port is available as a free download at www.hyperserialport.com . A block diagram of the board combination is shown in Figure 2.

1.2 EPT-570-AP Circuit Board

The EPT- 570-AP board is equipped with an Altera EPM570 PLD; which is programmed using the Altera Quartus II software. The PLD has 570 Logic Elements which is equivalent to 440 Macrocells. An on board 66 MHz oscillator is used by the EPT-Active-Transfer-Library to provide data transfer rates of 8 Mega Bytes per second. The EPT-Active-Transfer-Library provides control communication between the objective device and the PLD. Data transfer during the objective device checkout

between the PC and the PLD program is available via the Hyper Serial Port. The board also includes the following parts.

- Altera EPM570 in the TQFP 100 pin package
- 66 MHz oscillator for driving USB data transfers and users code
- Four 74LVC245 bidirectional voltage translator/bus transceiver
- 24 user Input/Outputs
- Four Green LED's accessible by the user
- Two PCB switches accessible by the user
- All connectors to stack into the Arduino Uno
- Two 13x2 connectors for accepting the FT2232H Mini Module.

Figure 3 contains an illustration of the EPT- 570-AP board.

Figure 2. EPT-570-AP Block Diagram

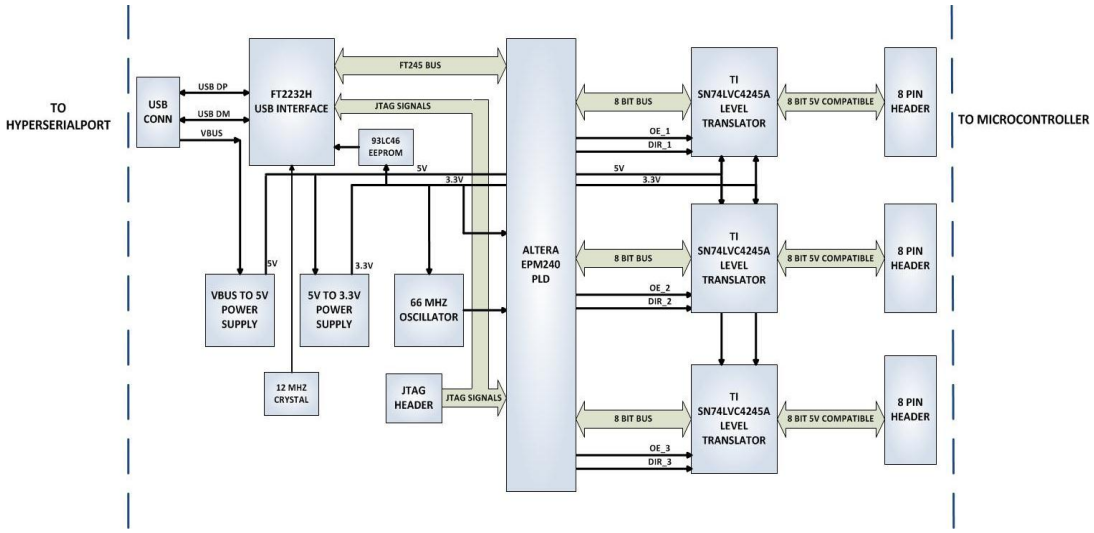
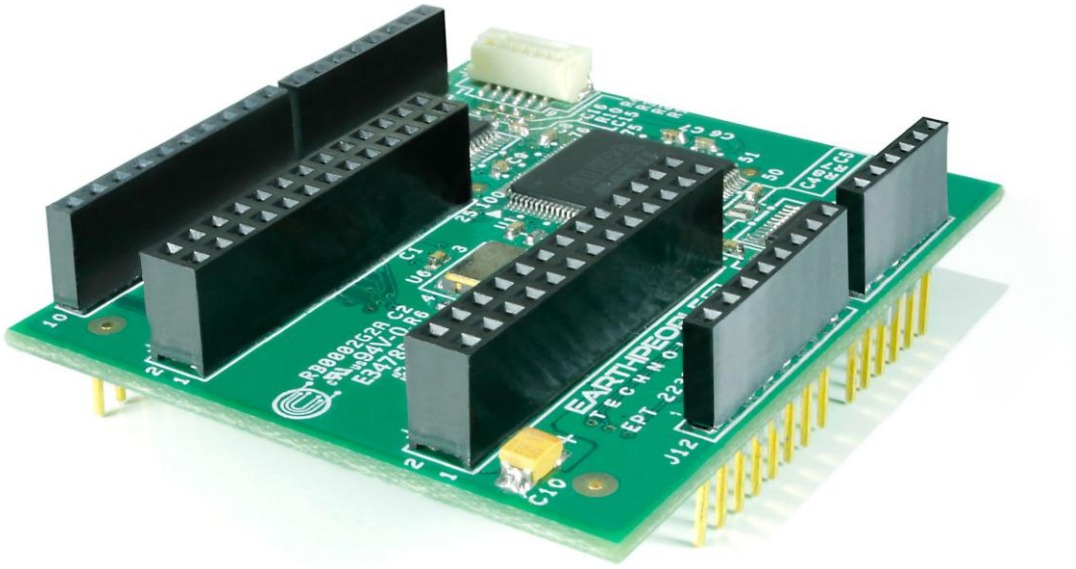


Figure 3. EPT- 570-AP BOARD FOR THE ARDUINO UNO



1.2.1 High Speed USB Communications

The EPT-570-AP-U2 USB/PLD Development system connects an FT2232H Dual High Speed USB (480 Mbits/sec) chip to the PLD. The PLD uses a dedicated channel on the FT2232H for high speed transfers to the PC. Using the EPT Active Transfer Library, sustained speeds of 8 Mbytes/sec can be achieved. The transfers are bi-directional.

1.2.2 Inputs and Outputs

There are 24 Inputs/Outputs which are +5 Volt tolerant. The I/O's are organized as three 8 bit directional ports. Each port must be defined as input or output. This means that all 8 bits of a port will point in the same direction, depending on the direction bit of the transceiver. The direction bit can be changed at any time, so that a port can change from input to output in minimum setup time of 6 nanoseconds. Each port also has an enable pin. This enable pin will enable or disable the bits of the port. If the port is disabled, the bits will "float".

1.2.3 JTAG

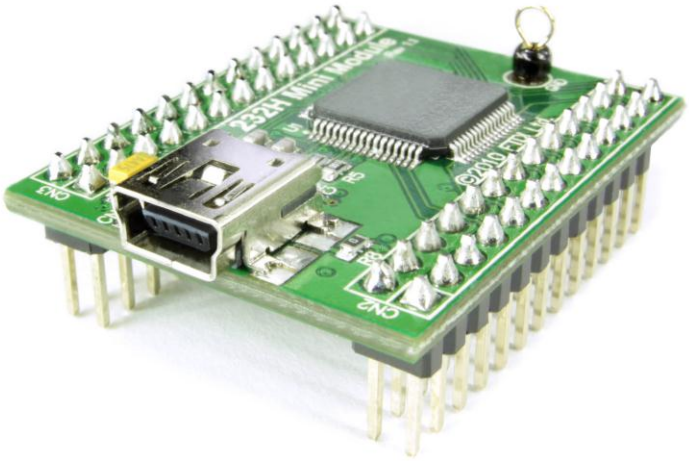
The EPT-570-AP-U2 uses the second channel of the FT2232H chip as a dedicated PLD programming port. The PLD must be programmed via JTAG signals and the FT2232H has built in JTAG signals. The PLD can be programmed directly from Quartus II 11.1 SP2 by using the "jtag_hw_mbftdi_blaster.dll". Just click on the Programmer button and select the MBFTDI-Blaster.

1.3 FT2232H Circuit Board

The FT2232H Mini Module board contains an FTDI 2232H dual channel high speed (480 Mb/s) USB to FIFO (first in-first out) integrated circuit chip to interface between the PC and the PLD. The FT2232H board provides a means

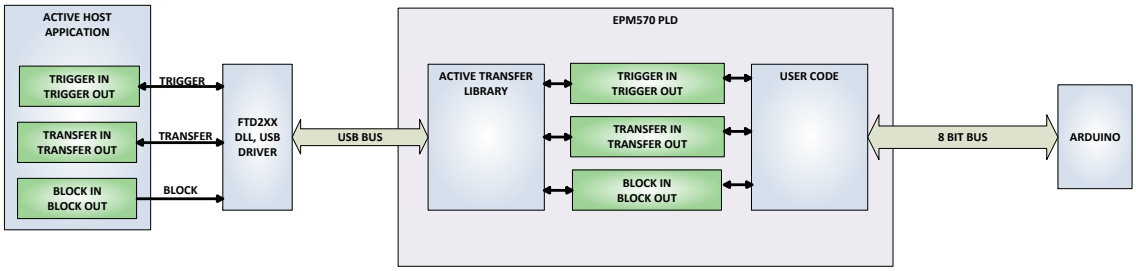
of data conversion from USB to serial and parallel FIFO for data derived from the PC and serial/parallel to USB for data being sent from the PLD to the PC. Channel A is configured as a JTAG bus and the Channel B is configured as an 8 bit parallel bus. PLD Programming commands are transmitted via the JTAG bus (channel A). Channel B has one dual port 4Kbyte FIFO for transmission from Host PC to the PLD, it also has one dual port 4Kbyte FIFO for receiving data from the PLD to the Host PC. The module uses the +5Vbus from the Host USB for self power. The FT2232H Mini Module provides its own 12 MHz clock and +3.3V and +1.8V power supplies. The +3.3V power supply output is used by the EPT-570-AP base board for all of its +3.3V power budget. Figure 4 contains an illustration of the FT2232H board.

Figure 4. FT2232H Mini Module



1.4 Active Host Application

HyperSerialPort has provided application software associated with the EPT USB/PLD development system. It is called Active Host and runs on the PC and is accessed through the HyperSerialPort software. The application is intended as a test platform for PLD to Active Transfer Library. However, due to the polling nature of the USB 2.0, Active Host can be used to transfer data to and from the Arduino. For the Host PC to receive data from the USB Device, it must periodically query the Device for any bytes it needs to transfer. If the Device reports back to the Host that it has bytes to transfer to the Host, the Active Host application will schedule a USB transfer to read the number of bytes into local memory. Since the polling of the Device occurs forever, the Arduino can write data into the PLD using one of the Active Transfer Library methods (Trigger, Byte Transfer, Block Transfer). Conversely, the Host PC can write to the Arduino using the Active Host at any time.



1.4.1 Active Host DLL

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

- EPT_List_Unopen_Devices()
- EPT_Open_Device()
- EPT_Close_Device()
- Trigger_In()
- Trigger_Out()
- Transfer_In()
- Transfer_Out()
- Block_In()
- Block_Out()

1.4.2 Active Host Open Device

To use the library functions for data transfer and triggering, an Earth People Technology device must be opened. Use the function EPT_List_Unopen_Devices() to detect all EPT devices connected to the PC. Select the device by part number, then use

```
private void EPT_FT2232_Interface_Load(object sender, System.EventArgs e)
{
    m_hPort = 0;
    fContinue = false;
    dwListDescFlags = FT_LIST_ALL;
    radNumber.Checked = true;
    btnClose.Enabled = false;
    btnOpen.Enabled = true;
    btnWrite.Enabled = false;
    radDescription.Enabled = true;
    radSerial.Enabled = true;
    radNumber.Enabled = true;
    if (EPT_List_Unopen_Devices() == false)
    {
        MessageBox.Show("Error Listing Devices");
    }
}
```

the function EPT_Open_Device() to get a handle from the device. This handle is used to read and write from the device. The handle is returned in the m_hPort_x variable. The _x is a number selected by the user, this will allow more than one device to be manipulated with handles. Use this handle when you want to read or write to device.

```
private void btnOpen_Click(object sender, System.EventArgs e)
{
    EPT_Open_Device();
}

private void btnClose_Click(object sender, System.EventArgs e)
{
    EPT_Close_Device();

    radNumber.Checked = true;
    btnClose.Enabled = false;
    btnOpen.Enabled = true;
    btnWrite.Enabled = false;
    radDescription.Enabled = true;
    radSerial.Enabled = true;
    radNumber.Enabled = true;

    timerUSB.Enabled = false;
}
```

1.4.3 Active Host Triggers

The user application can send a trigger to the Device (PLD) by using the Trigger_Out() function. First, open the EPT device to be used with EPTOpenDevice(). Then use the handle in the first parameter of the Trigger_Out() function. Add the bit or bits to assert high on the trigger byte as the second parameter. Then execute the function, the trigger bit or bits will momentarily assert high in the user code on the PLD.

```
private void btnTrigger1_Click(object sender, EventArgs e)
{
    send_trigger(0x01);
}
```

To detect a trigger from the Device, the user application must query the USB_IN() function. Any transfer that occurred during the USB polling iteration will be stored in the USB_IN() memory block.

```
private void btnTrigger4_Click(object sender, EventArgs e)
{
    send_trigger(0x0a);
}
```

1.4.4 Active Host Byte Transfers

To send a byte to the Device, the appropriate address must be selected. Up to eight modules can be instantiated in the user code on the PLD. Each module has it's own address.

```
private void btnWrite_Click(object sender, System.EventArgs e)
{
    int i;
    i = Convert.ToInt32(tbNumBytes.Text);
    uint TransferAddress = (uint)Convert.ToInt32(tbTransferAddress.Text)
    send_transfer_in((byte)i, (byte)TransferAddress);
}
```

Use the function Transfer_Out() to transfer a byte to the Device. First, open the EPT device to be used with EPTOpenDevice(). Then use the handle in the first parameter of the Transfer_Out() function. Enter the address of the transfer module as the second parameter. Enter the byte to be transferred on the third parameter. Then execute the function, the byte will appear in the ports of the Active Transfer module in the user code on the PLD.

```
private void btnWriteMultiByte_Click(object sender, EventArgs e)
{
    int i;
    uint TransferAddress = (uint)Convert.ToInt32(tbTransferAddress.Text);
    string[] result = tbWriteMultiByte.Text.Split(' ');
    for (i = 0; i < result.Length; i++)
    {
        send_transfer_in((byte)Convert.ToInt32(result[i]), (byte)TransferAddress);
        Thread.Sleep(1);
    }

}
```

To receive a byte transfer from the Device, use the Transfer_In() function to check for any bytes read for that address. To receive data from the USB, a polling technique must be used. 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 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. The Transfer_In() function can then read the local memory and check the registers for data received from the address of interest.

[Add code snippet showing the Active Transfer receiving bytes from the Host]

1.4.5 Active Host Block Transfers

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

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

```
private void btnBlkCompare32_Click(object sender, EventArgs e)
{

    uint BlockAddress = (uint)Convert.ToInt32(tbBlockRcvAddress.Text);
    uint BlockLength = (uint)Convert.ToInt32(tbBlockRcvLength.Text);
    uint BlockRepitions = (uint)Convert.ToInt32(tbRepitions.Text);
    device[BlockAddress].Address = BlockAddress;
    device[BlockAddress].Length = BlockLength;
    device[BlockAddress].Repititions = BlockRepitions;
    BlockCount = 0;
    BlockTransferStop = false;

    Thread t = new Thread(new ParameterizedThreadStart(BlockCompare));
    t.Start(BlockAddress);
}
```

```
public void BlockCompare(object data)
{
    uint BlockAddress = (uint)data;

    if ((device[BlockAddress].Repititions > 0) & !device[BlockAddress].TransferPending & !B
    {
        device[BlockAddress].TransferPending = true;
        send_block_in(block_8_in_payload, device[BlockAddress].Address, device[BlockAddress
        Thread.Sleep(1);
        send_trigger(0x20);
        Thread.Sleep(1);
        stopWatch.Start();
        send_trigger(0xa0);
        Thread.Sleep(1);
        send_trigger(0x10);

        if (BlockTransferInfinite)
            device[BlockAddress].Repititions = 1;
        else
            device[BlockAddress].Repititions--;
    }
}
```

To send a block, first, open the EPT device to be used with EPTOpenDevice(). Then use the handle in the first parameter of Block_Out() function. Enter the address of the transfer module as the second parameter. Next, place the address of the buffer to the third parameter of Block_Out(). Then execute the function, the entire buffer will be transferred to the USB chip. The data is available at the ports of the Active Block module in the user code on the PLD.

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

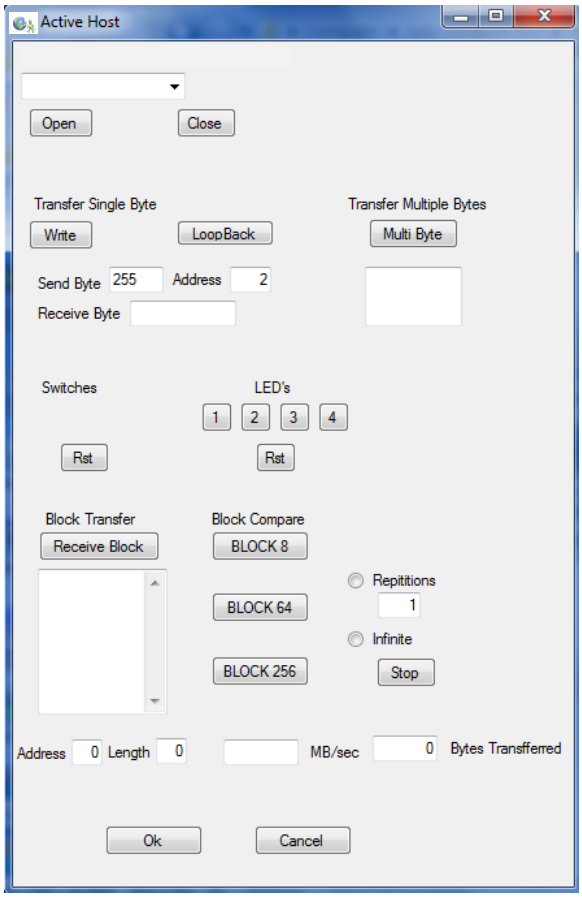
To receive a byte transfer from the Device, use the Block_In() function to check for any bytes read for that address. To receive data from the USB, a polling technique must be used. This is because the Bulk Transfer USB is a Host initiated bus. The Device will not transfer any bytes until the Host commands 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 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. The Block_In() function can then read the local memory and check the registers for data received from the address of interest.

[Add code snippet showing the Active Host Block receiving bytes from the Device]

1.4.6 Active Host Test Window

EPT provides a test application to allow the user to understand the functioning of the USB/PLD Development System. The source code is provided for both the PC application side and the PLD Device side. The user can use this code as a starting point to create their own custom applications.The windows form is show in Figure 4a.

Figure 4a Active Host



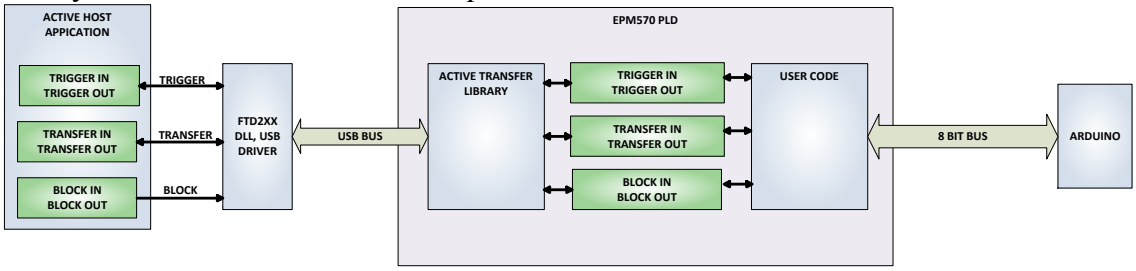
Features of the Active Host interface window include:

- Select Device
- Open Device
- Transfer Single Byte
- Transfer Single Byte Loopback
- Trigger In Switches
- Trigger Out LED’s
- Block Transfer
- Transfer Block Compare

The Active Host can be used as an integral part of the Arduino microcontroller code development process. The entire development process software consists of the Hyper Serial Port, the EPT Active Transfer Library, Python scripting using the HyperSerial Port Embedded Scripting Evaluator, Altera Quartus II software (freely available from the Altera website), the Arduino microcontroller development environment, and the Notepad++ code editor (available free from www.notepad-plus-plus.org). The procedures for using these software systems and their interaction with the hardware described above are detailed in section 3 and subsequent sections.

1.5 Active Transfer Library

The Active Transfer Library is an HDL library designed to transfer data to and from the EPT-570-AP via High Speed (480 MB/s) USB. It is a set of pre-compiled HDL files that the user will add to their project before building it. The description of what the library does and how to use it's components are described in this manual.



1.5.1 EPT Active Transfer System Overview

The Active Transfer Systme components consist of the following:

- ACTIVE_TRANSFER_LIBRARY.V
- ACTIVE_TRIGGER.V
- ACTIVE_TRANSFER.V
- ACTIVE_BLOCK.V

The Active_Transfer_Library.v provides the communication to the USB hardware. While separate Input and Output buses provide bi-directional communications with the plug in modules. See Figure 6 for an overview of the EPT Active_Transfer system.

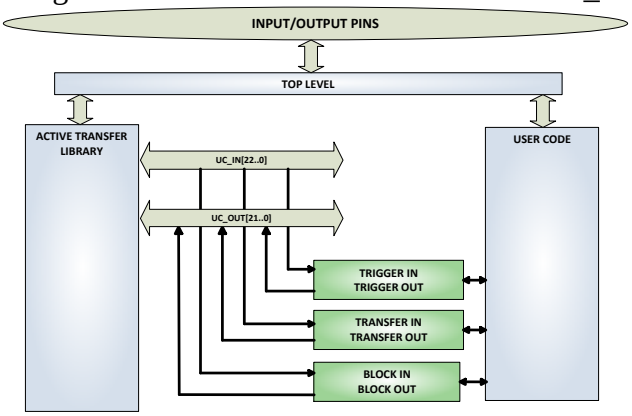


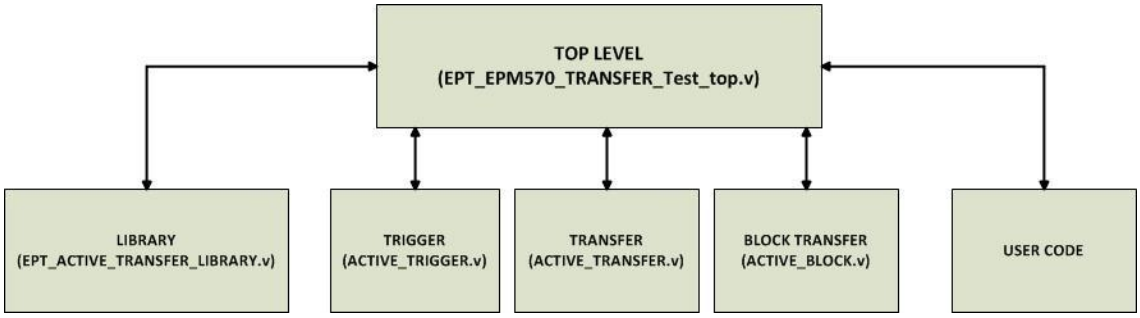
Figure 6 EPT Active Transfer Library Overview

Figure 6 shows how the modules of the EPT Active Transfer Library attach to the overall user project. The EPT Active_Transfer_Library.v, Active_Trigger.v, Active_Transfer.v and Active_Block.v modules are instantiated in the top level of the user project. The User_Code.v module is also instantiated in the top level. The Active_Transfer modules communicate with the User_Code through module parameters. Each module is a bi-directional component that facilitates data transfer from PC to PLD. 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.

1.5.2 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 PLD. 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.



```
1  //#############################################################################
2  //#
3  //# Copyright   Earth People Technology Inc. 2012
4  //#
5  //#
6  //# File Name:  EPT_FT2232_Transfer_Test_top.v
7  //#
8  //# Revision History:
9  //#         DATE        VERSION     DETAILS
10 //#         07/5/12     A           Created         RJJ
11 //#
12 //#
13 //#
14 //#############################################################################
15 `ifdef SIM
16    `include "../src/define.v"
17    `include "../Testbench/tb_define.v"
18 `endif
19
20 `timescale 1ns/1ps
21
22
23
24 //*****************************************************************************
25 //* Module Declaration
26 //*****************************************************************************
27
28 module ept_EPM570_Transfer_Test_top (
29
30
31     input  wire [1:0]          aa,
32     input  wire [1:0]          bc_in,
                                  •
                                  •
                                  •
687
688    //-----------------------------------------------
689    // Instantiate the EPT Library
690    //-----------------------------------------------
691
692     active_transfer_library         EPT_LIBRARY_TOP_INST
693     (
694     .aa                          (aa),
695     .bc_in                       (bc_in),
696     .bc_out                      (bc_out),
697     .bd_inout                    (bd_inout),
698
699     .UC_IN                       (UC_IN),
700     .UC_OUT                      (UC_OUT),
701
702     .TEST_SIGNAL_1               (data_byte_ready),
703     .STATE_OUT                   (ft_245_state_machine),
704     .TEST_BUS                    (register_decode),
705     .ENDPOINT_STATE_OUT          (endpoint_registers_state),
706     .ENDPOINT_TEST_BUS           (endpoint_write_to_host)
707     );
708
709    //-----------------------------------------------
710    // Instantiate the EPT Modules
711    //-----------------------------------------------
712  wire [22*3-1:0]  uc_out_m;
713  eptWireOR # (.N(3)) wireOR (UC_OUT, uc_out_m);
714      active_trigger              ACTIVE_TRIGGER_INST
715     (
716      .uc_clk                     (CLK_66),
```

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

The control buses, aa[1:0], bc_in[1:0], bc_out[2:0], and bd_inout[7:0] are used to channel data, and control signals to the USB interface chip. These signals are connected directly to input and output pins of the PLD.

1.5.3 Active Trigger

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

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

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

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

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

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

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

1.5.4 Active Transfer

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

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

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

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

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

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

1.5.5 Active Block

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

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

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

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

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

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

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

1.6 Timing Diagram for Active Transfer Methods

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

1.6.1 Active Trigger Timing

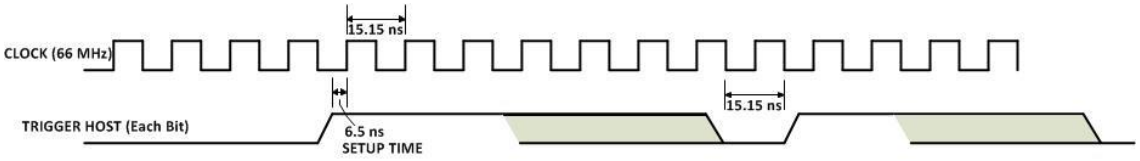


Figure xx Active Trigger to Host Timing

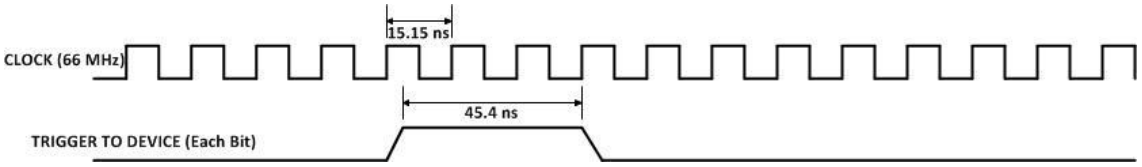


Figure xx Active Trigger to Device Timing

1.6.2 Active Transfer Timing

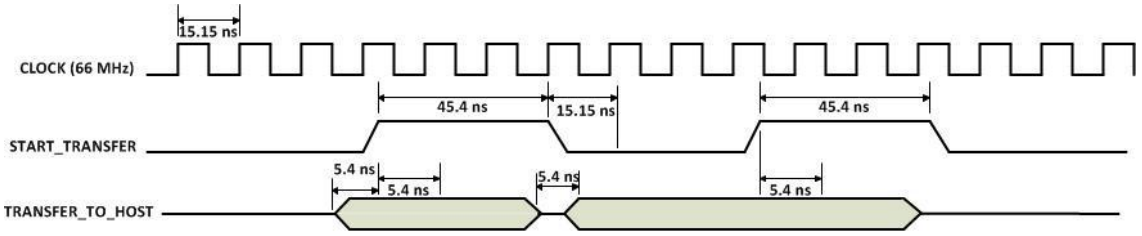


Figure xx Active Transfer To Host Timing

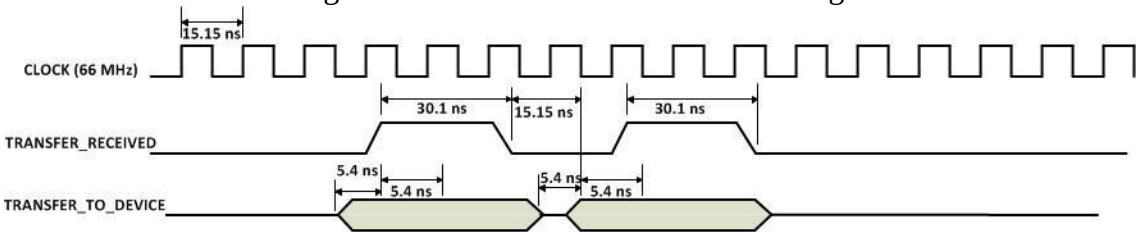


Figure xx Active Transfer To Device Timing

1.6.3 Active Block Timing

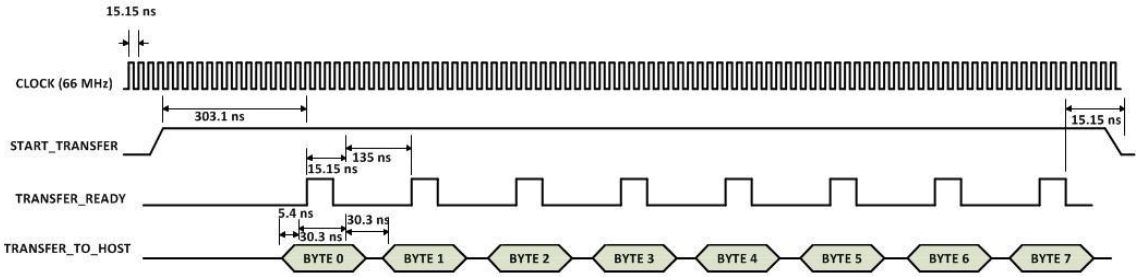


Figure xx Active Block To Host Timing

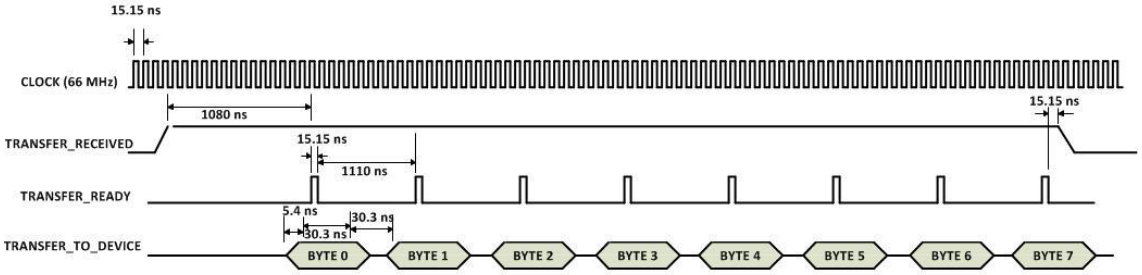


Figure xx Active Block To Device Timing

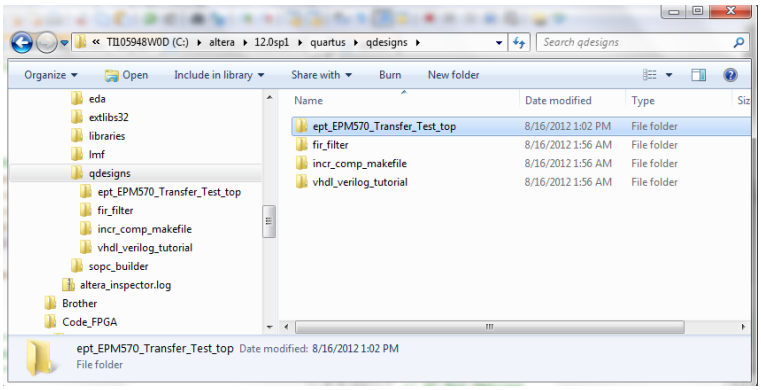
1.7 Compiling, Synthesizing, and Programming PLD

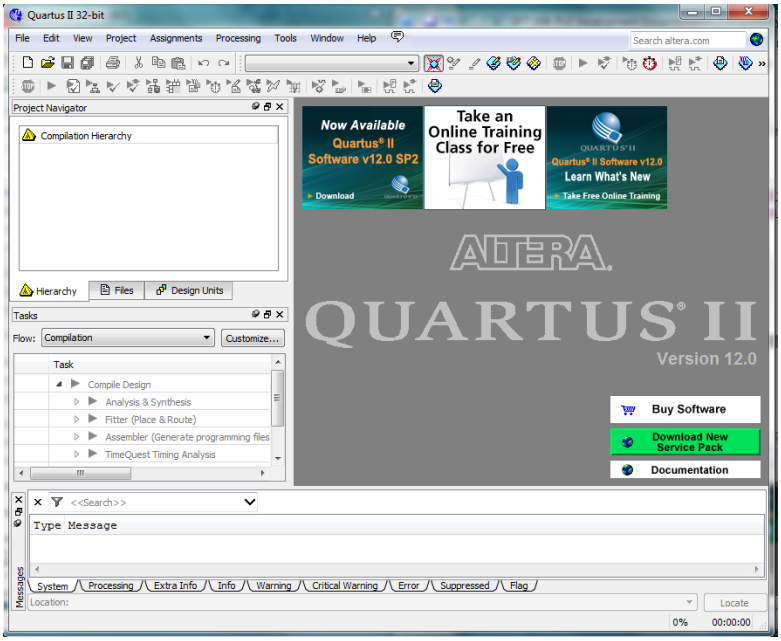
The PLD on the EPT-570-AP-U2 can be programmed with the Active Transfer Library and custom HDL code created by the user. Programming the PLD requires the use of the Quartus II software and a standard USB cable. There are no extra parts to buy, just plug in the USB cable. Once the user HDL code is written according to the syntax rules of the language (Verilog and VHDL) it can be compiled and synthesized using the Quartus II 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 PLD.

1.7.1 Setting up the Project and Compiling

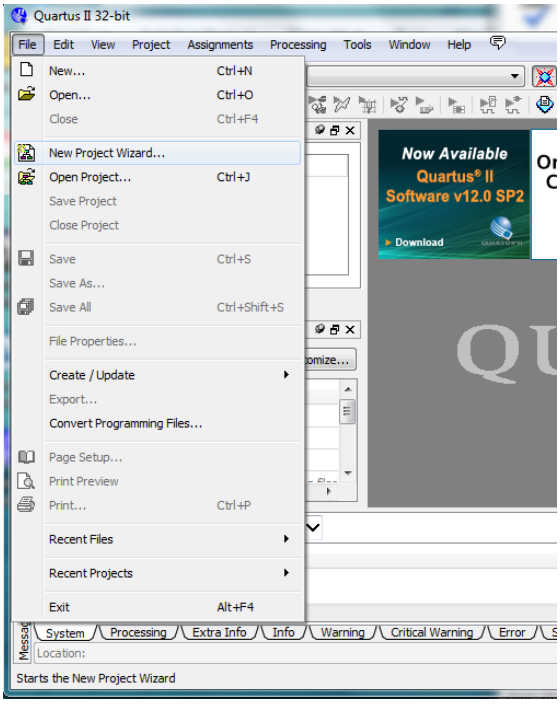
Once the HDL code (Verilog or VHDL) is written and verified using a simulator, a project can be created using Quartus II. Install the latest version of Quartus II on your PC. Bring up Quartus II, then use Windows Explorer to browse to

c:/altera/xxx/quartus/qdesigns and create a new directory called: “ept_EPM570_Transfer_Test_top”.

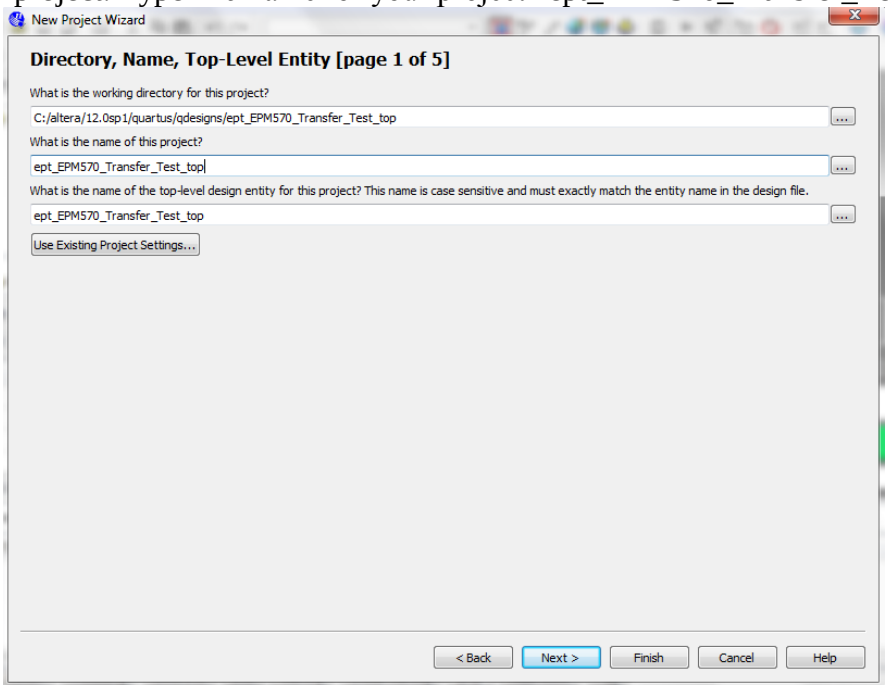




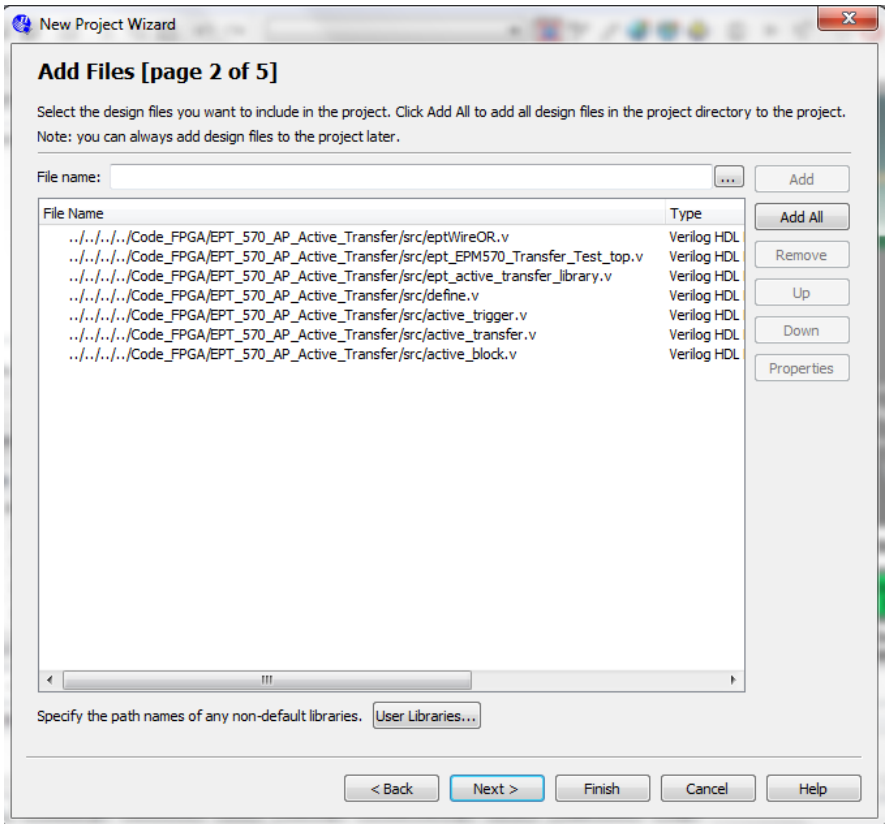
Under Quartus, Select File->New Project Wizard. The Wizard will walk you through setting up files and directories for your project.



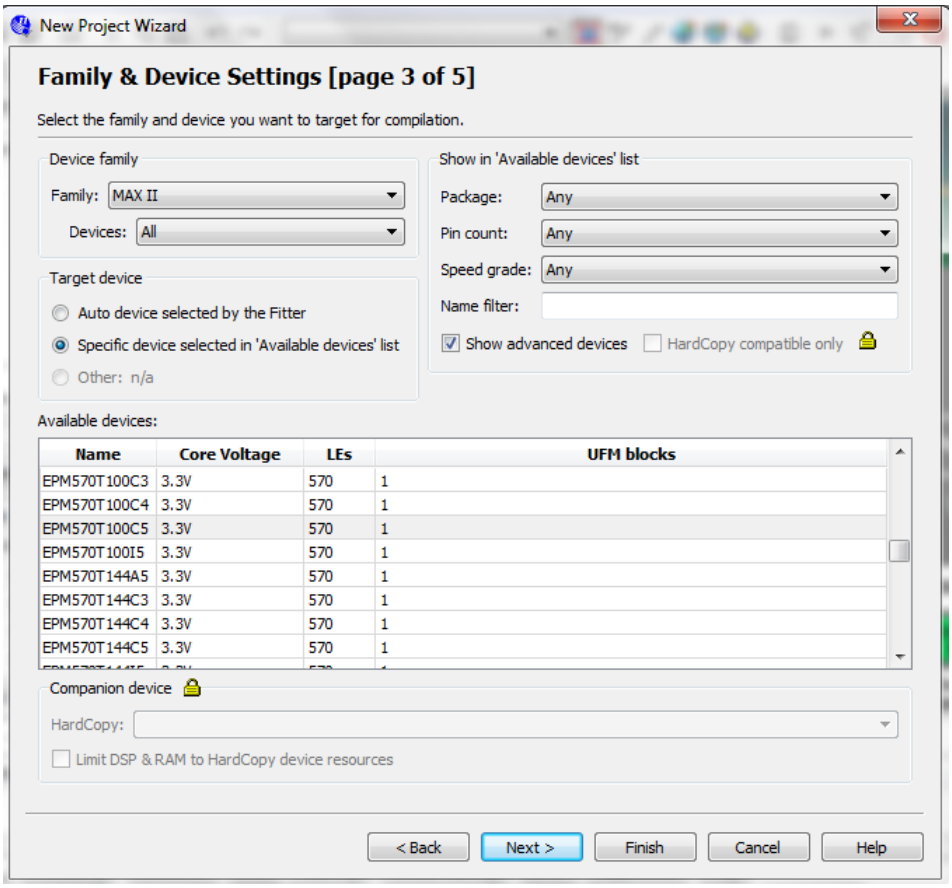
At the Top-Level Entity page, browse to the c:/altera/xxx/quartus/qdesigns directory to store your project. Type in a name for your project “ept_EPM570_Transfer_Test_top”.



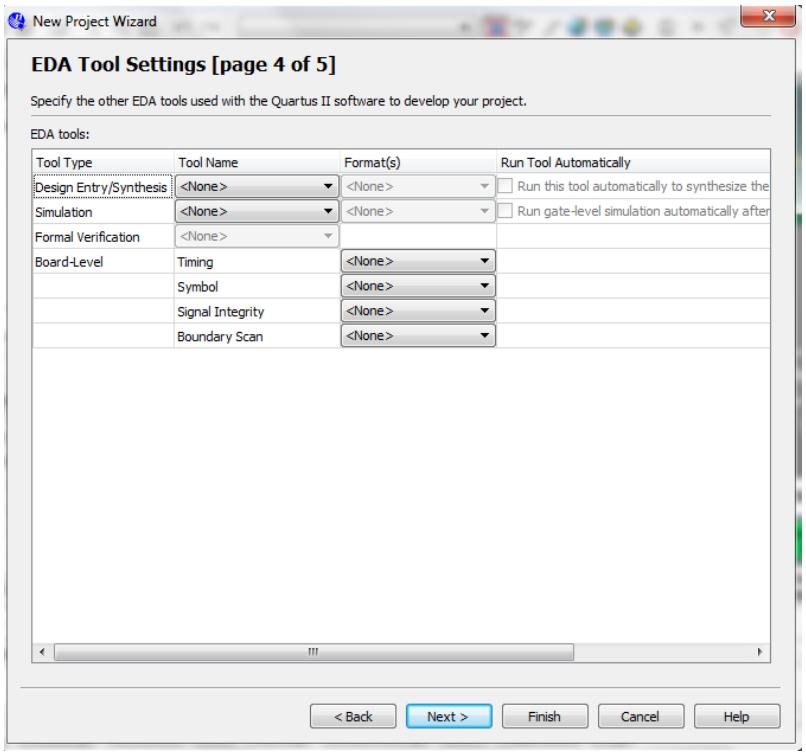
Select Next, at the File name: browse over to the directory where the source files are stored. Select all files for the project.



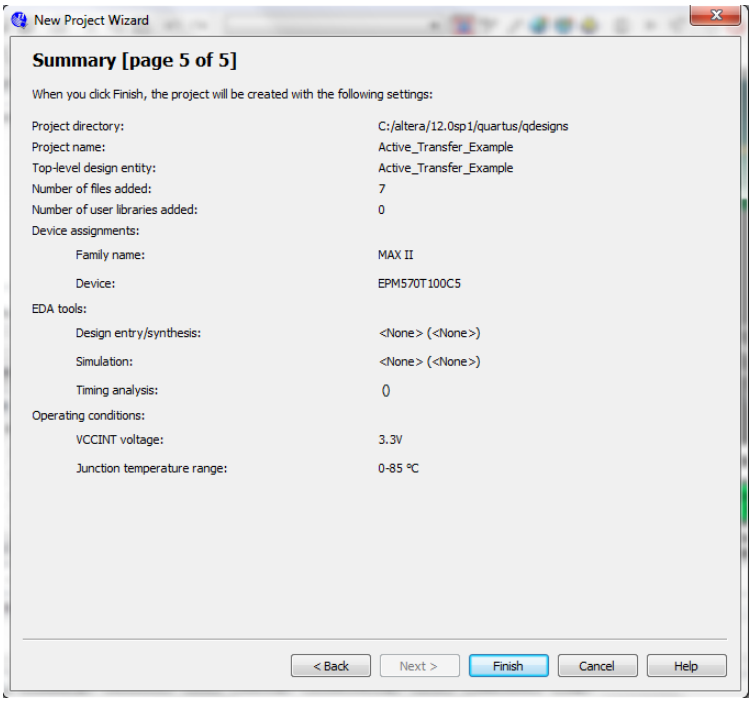
Select Next, at the Device Family group, select MAX II for Family. In the Available Devices group, browse down to EPM570T100C5 for Name.



Select Next, leave defaults for the EDA Tool Settings.



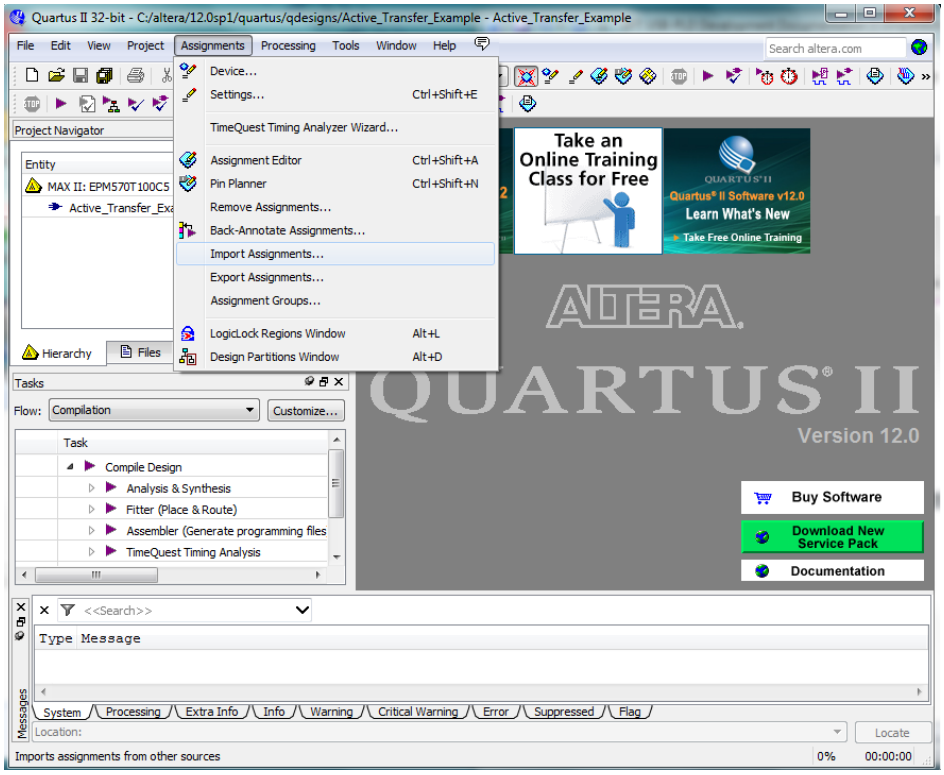
Select Next, then select Finish. You are done with the project level selections.



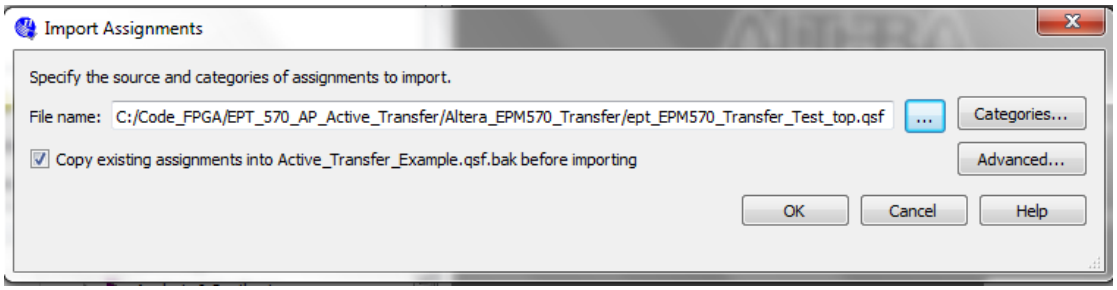
Next, we will select the pins and synthesize the project.

1.7.2 Selecting Pins and Synthesizing

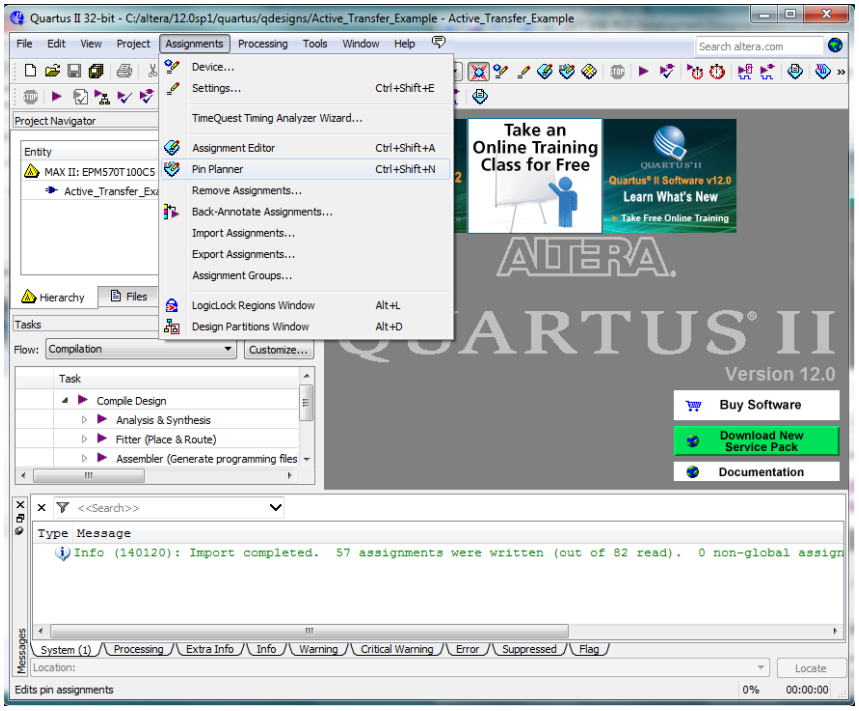
With the project created, we need to assign pins to the project. The signals defined in the top level file (in this case: ept_EPM570_Transfer_Test_Top.v) will connect directly to pins on the PLD. The Pin Planner Tool from Quartus II will check to verify that our pin selections do not violate any restrictions of the device. In the case of this example we will import pin assignments that created at an earlier time. Under Assignments, Select Import Assignments.

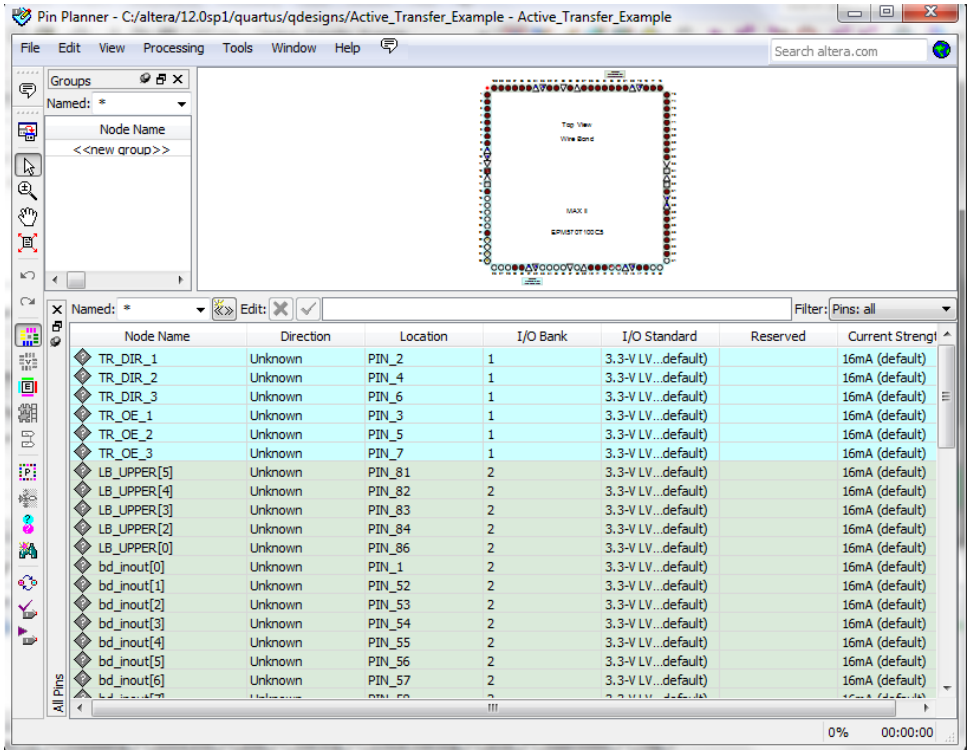


At the Import Assignment dialog box, browse to the folder with the Quartus Specification File. Select the "ept_EPM570_Transfer_Test_top.qsf" file.

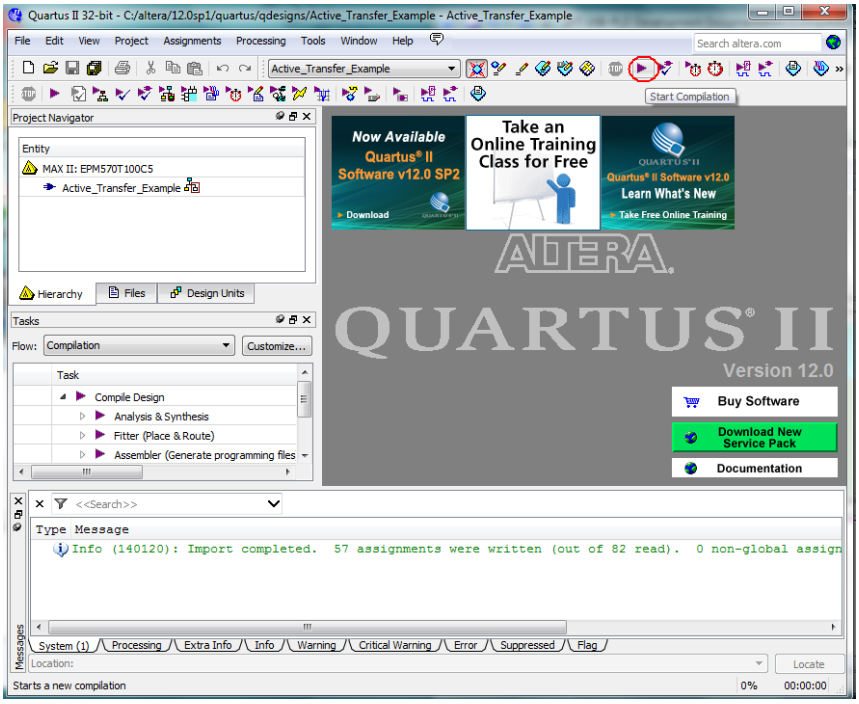


Click Ok. Under Assignments, Select Pin Planner. Verify the pins have been imported correctly.

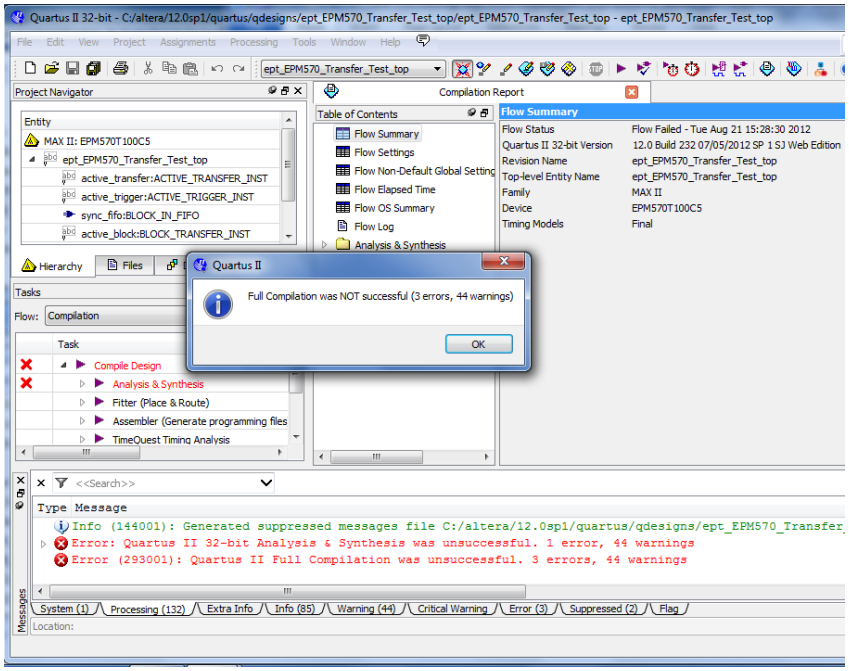




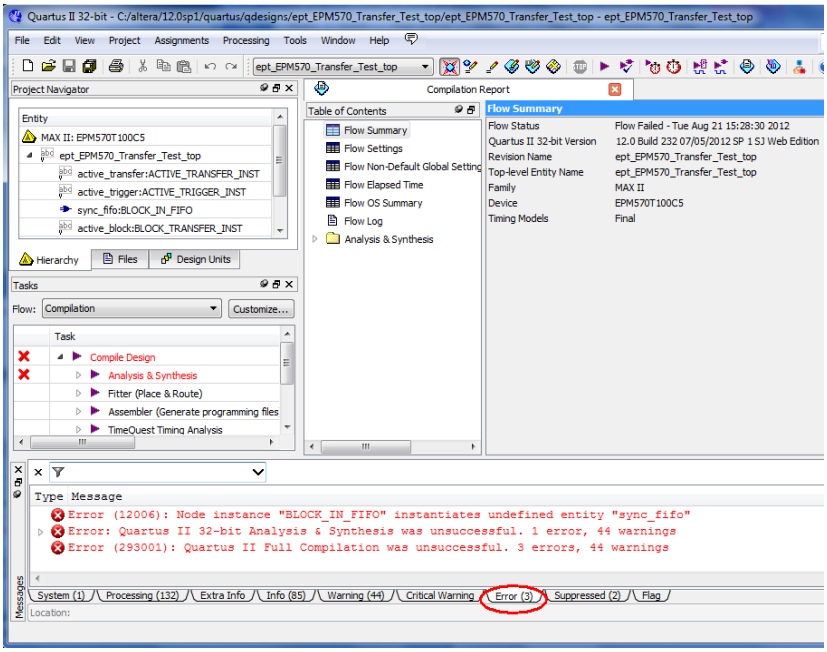
Exit the Pin Planner, and select the Start Compilation button.



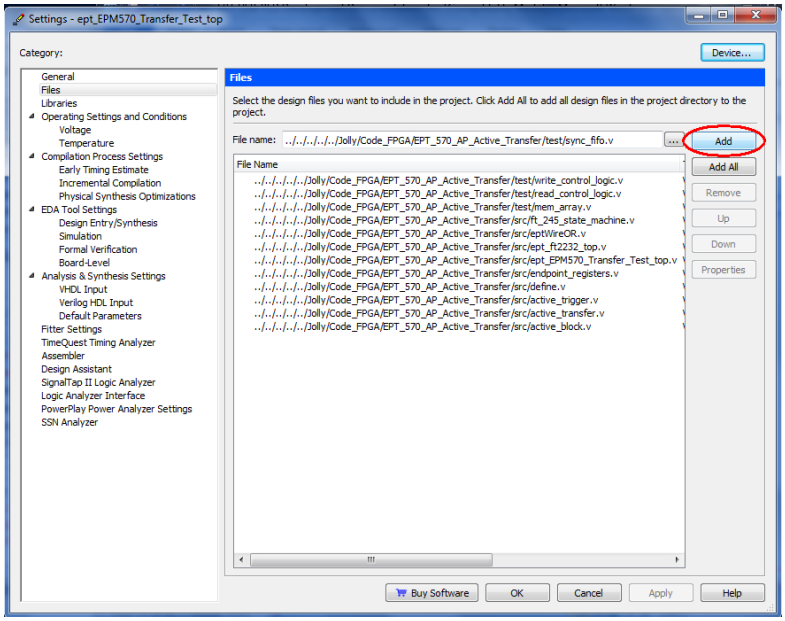
If you forget to include a file or some other error you should expect to see a screen similar to this:



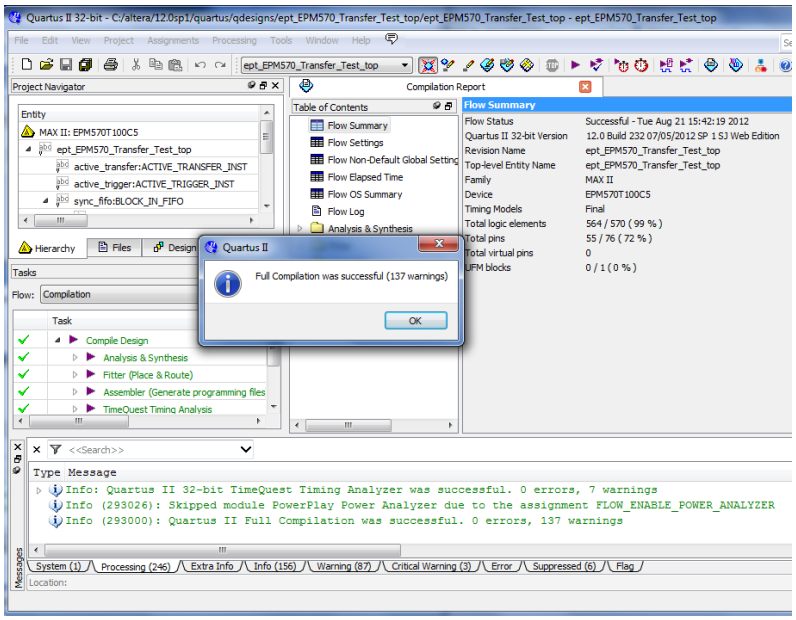
Click Ok, the select the “Error” tab to see the error.



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



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

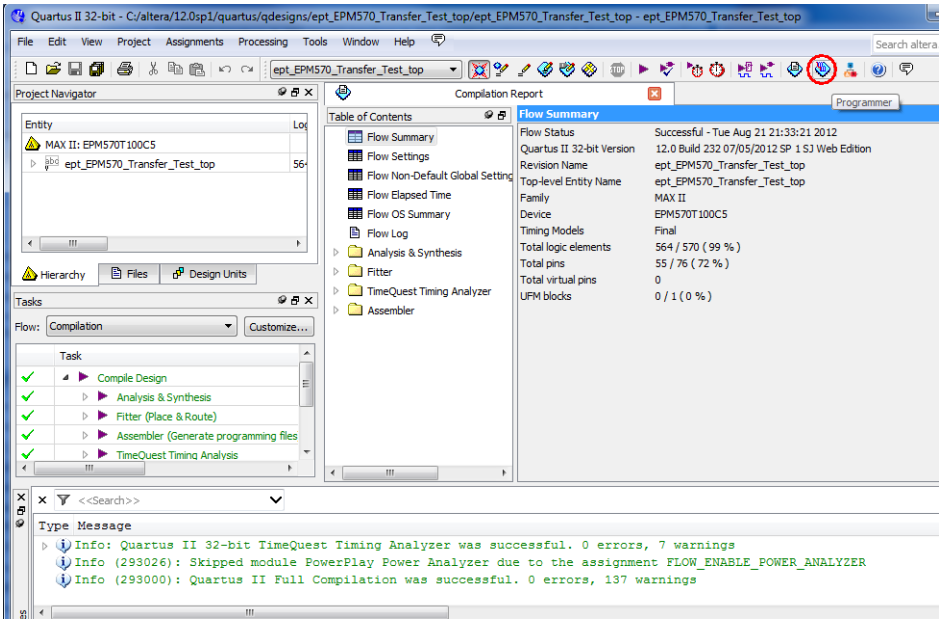


At this point the project has been successfully compiled, synthesized and a programming file has been produce. See the next section on how to program the PLD.

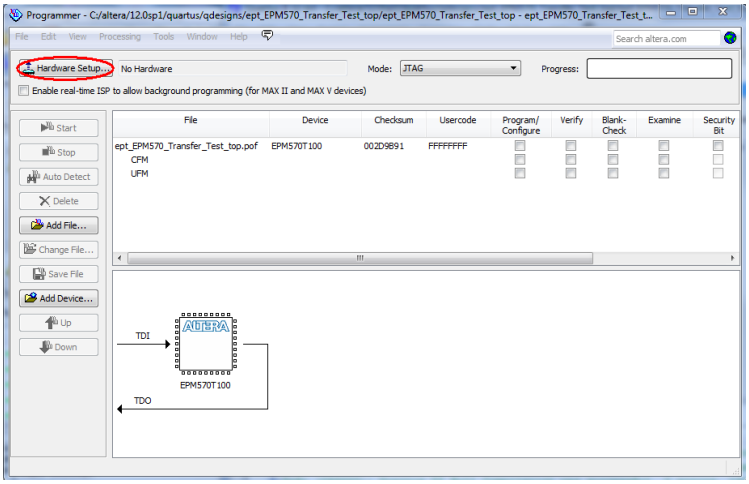
1.7.3 Programming the PLD

Programming the PLD is quick and easy. All that is required is a standard USB cable with a Mini Type B connector on one end and the EPT JTAG Driver DLL. Connect the EPT-570-AP to the PC, open up Quartus II, open the programmer tool, and click the Start button. To program the PLD, follow the steps to install the USB Driver and the JTAG Driver Insert for Quartus II.

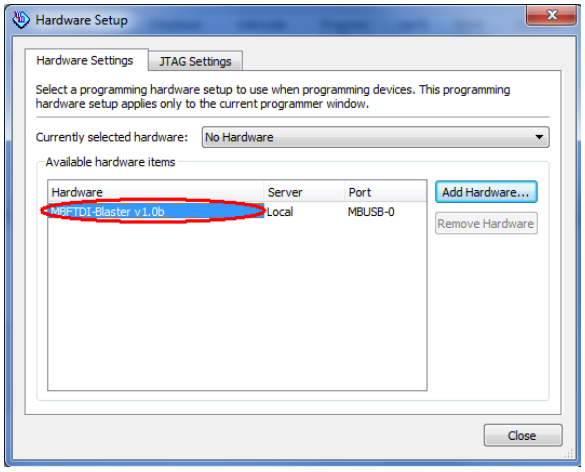
If the project created in the previous sections is not open, open it. Click on the Programmer button.



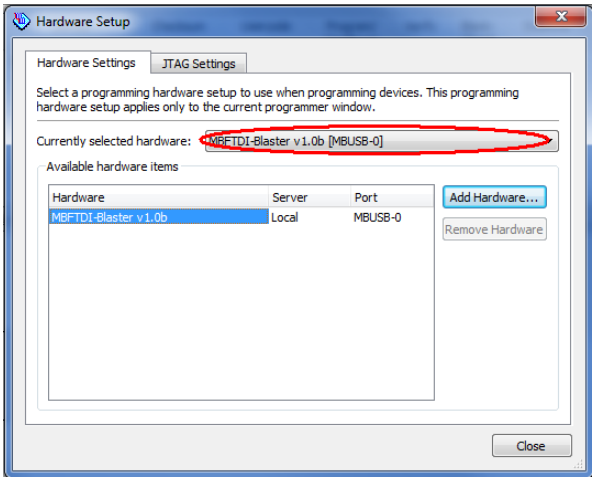
The Programmer Window will open up with the programming file selected. Click on the Hardware Setup button in the upper left corner.



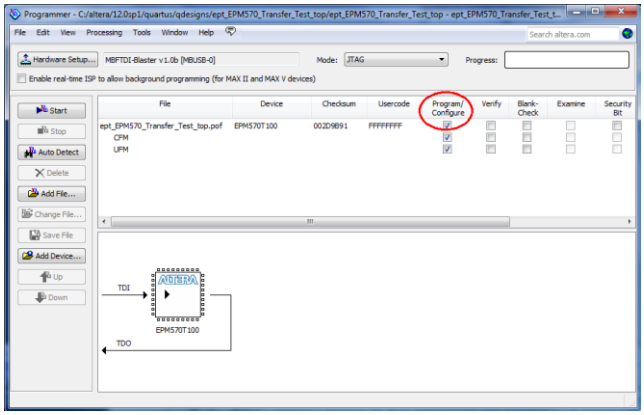
The Hardware Setup Window will open. In the “Available hardware items”, double click on “MBFTDI-Blaster v1.0b”.



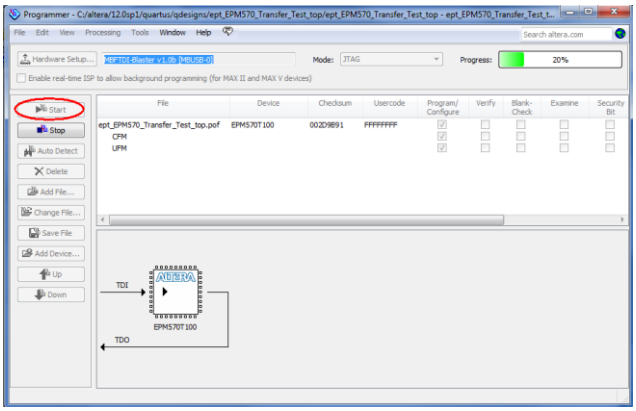
If you successfully double clicked, the “Currently selected hardware:” dropdown box will show the “MBFTDI-Blaster v1.0b”.



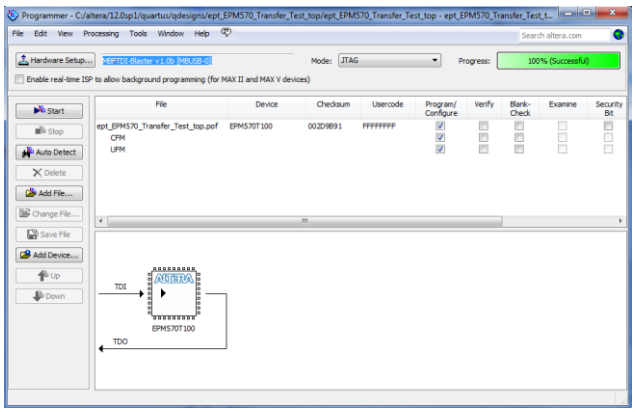
Next, selet the checkbox under the “Program/Configure” of the Programmer Tool. The checkboxes for the CFM and UFM will be selected automatically.



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



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



At this point, the EPT-570-AP is programmed and ready for use. To test that the PLD is properly programmed, bring up the Active Host Test Tool. Click on one of the LED's and verify that the LED selected lights up. Press one of the switches on the board and ensure that the switch is captured on the Active Host Test Tool. Now you are ready to connect to the Arduino Uno and write some code to transfer data between microcontroller and PC.

1.8 Assembling, Building, and Executing a Visual Project on the PC

The Active Host Application DLL is used to build a custom standalone executable on the PC that can perform Triggers and Transfer data to/from the EPT-570-AP. A standalone project can be range from a simple program to display and send data from the user to/from the Arduino Uno. Or it can more complex to include receiving data, processing it, and start or end a process on the Arduino. This section will outline the procedures to take an example project and Assemble it, Build it, and Execute it.

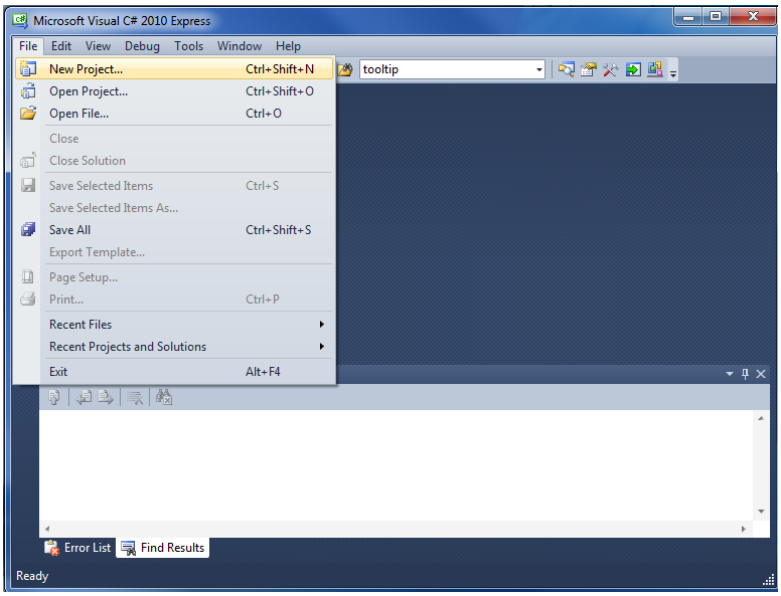
This guide will focus on writing a Windows Forms application using the C# language for the Microsoft Visual Studio with .NET Framework. This is due to the idea that beginners can write effective Windows applications with the C# .NET Framework. They can focus on a subset of the language which is very similar to the C language. Anything that deviates from the subset of the C language, presented as in the Arduino implication (such as events and controls), will be explained as the explanation progresses. Any language can be used with the Active Host Application DLL.

1.8.1 Creating a Project

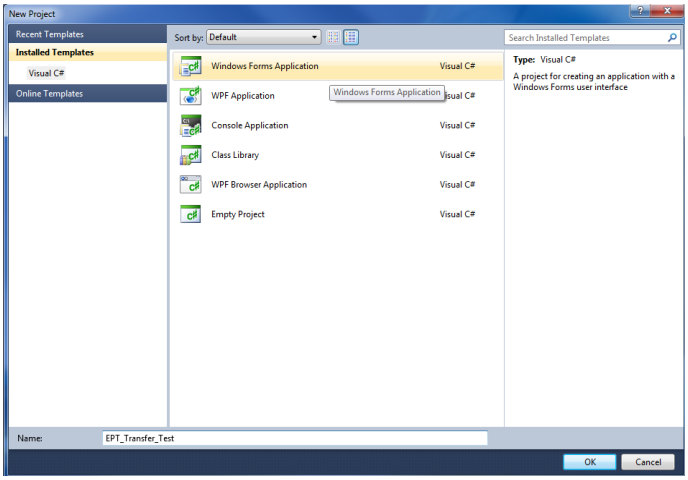
Download the Microsoft Visual C# 2010 Express environment from Microsoft. It's a free download.

[Add Microsoft Visual C# 2010 Express URL]

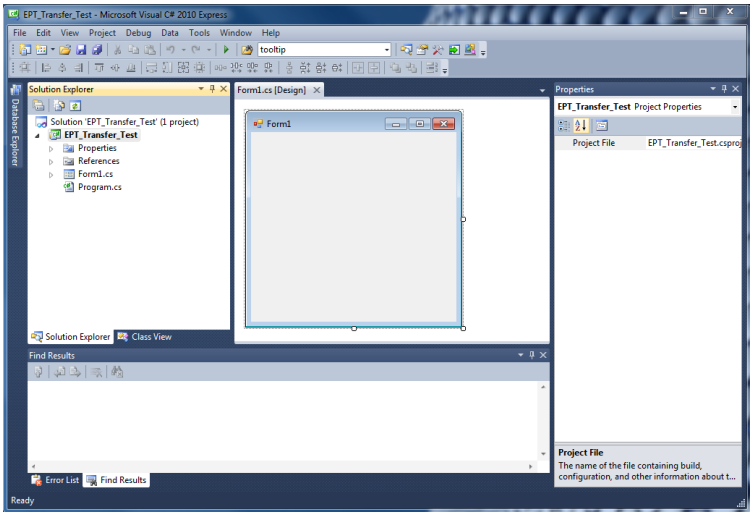
Install the application and select the defaults. Once the application is installed, open it up. Click on File->New Project.



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

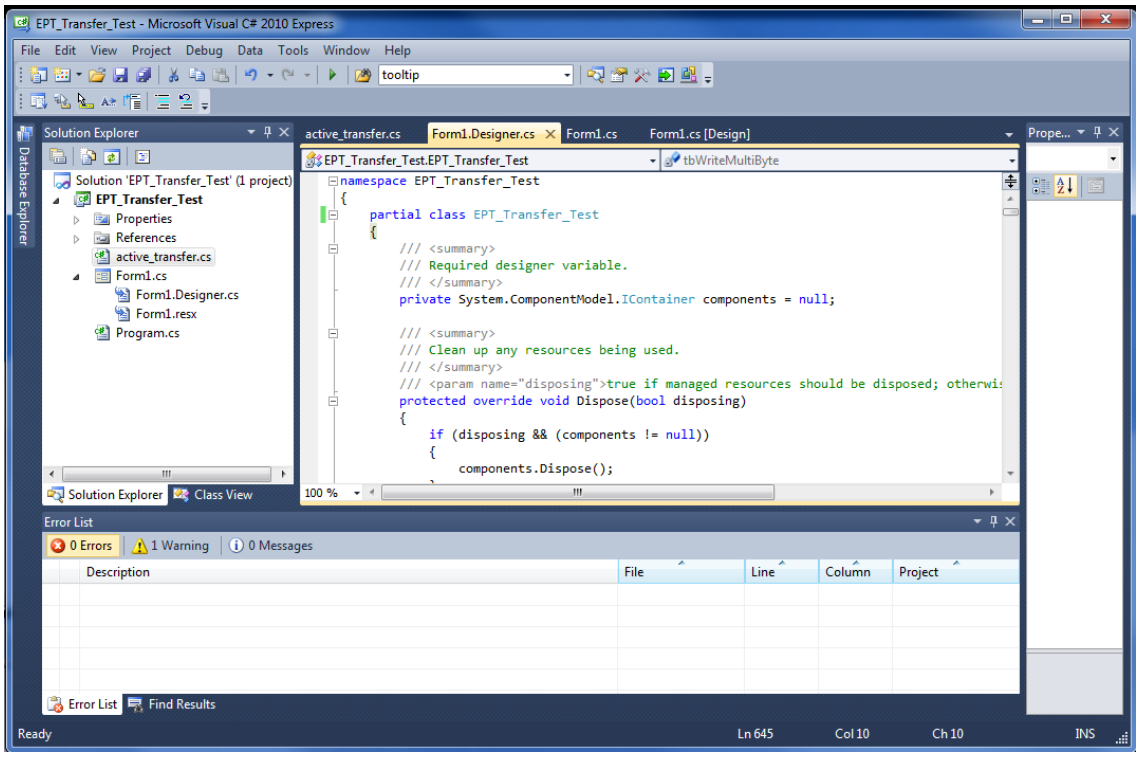


When the project creation is complete, the user files can be added.

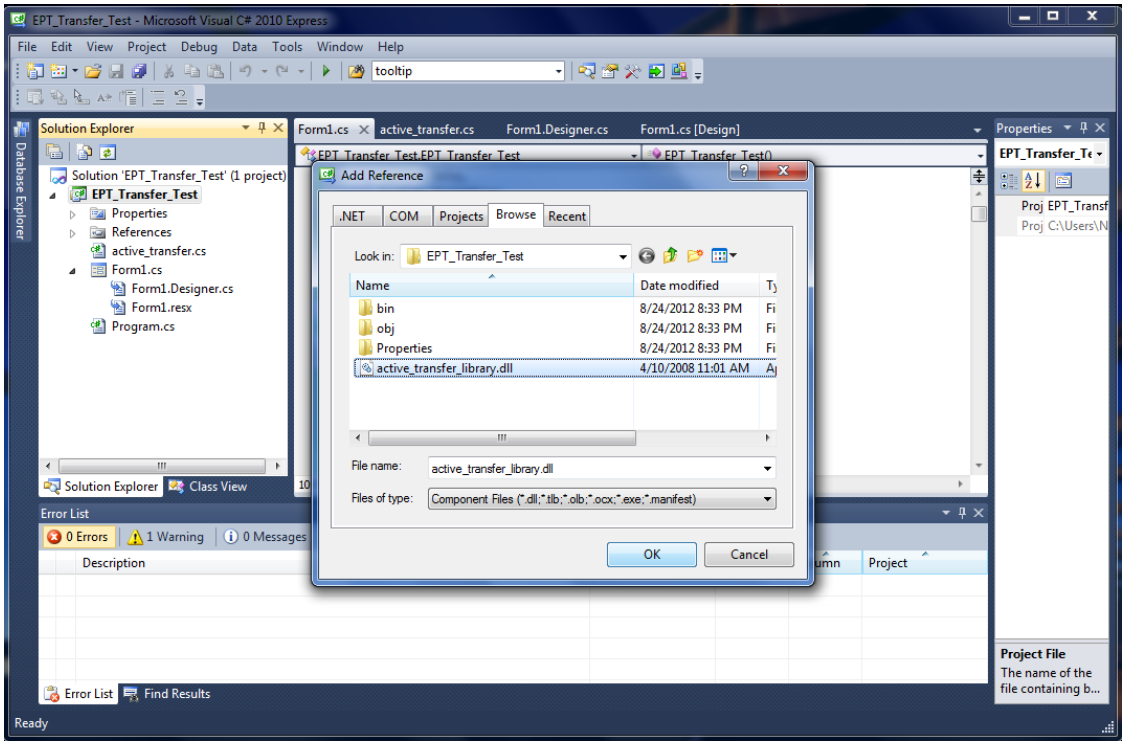


1.8.2 Assembling Files into the Project

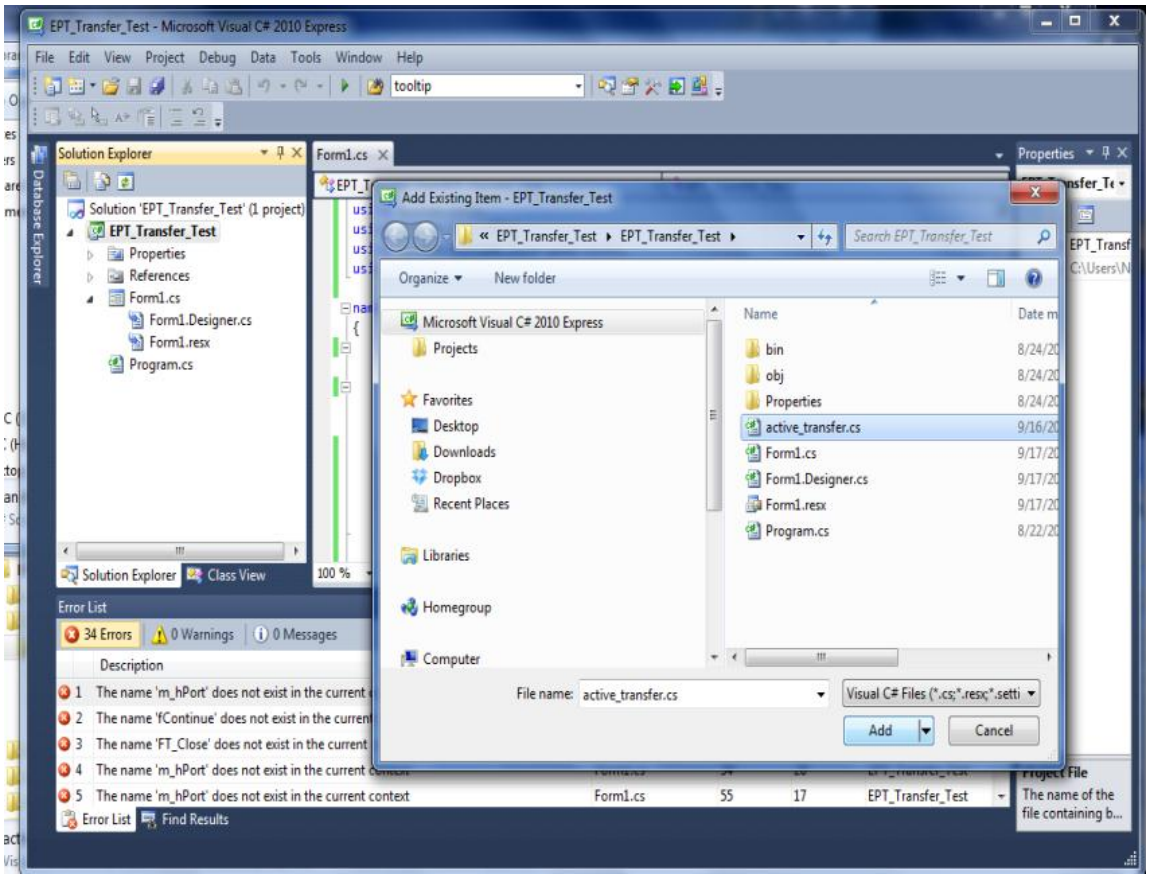
Download the EPT Active Host Application and install it on your PC. When the download unpacks itself, it will ask you where to place the user accessable items on your hard drive. Browse to the folder where the DLL resides, copy it, and install it in the top level folder of the EPT_Transfer_Test project.



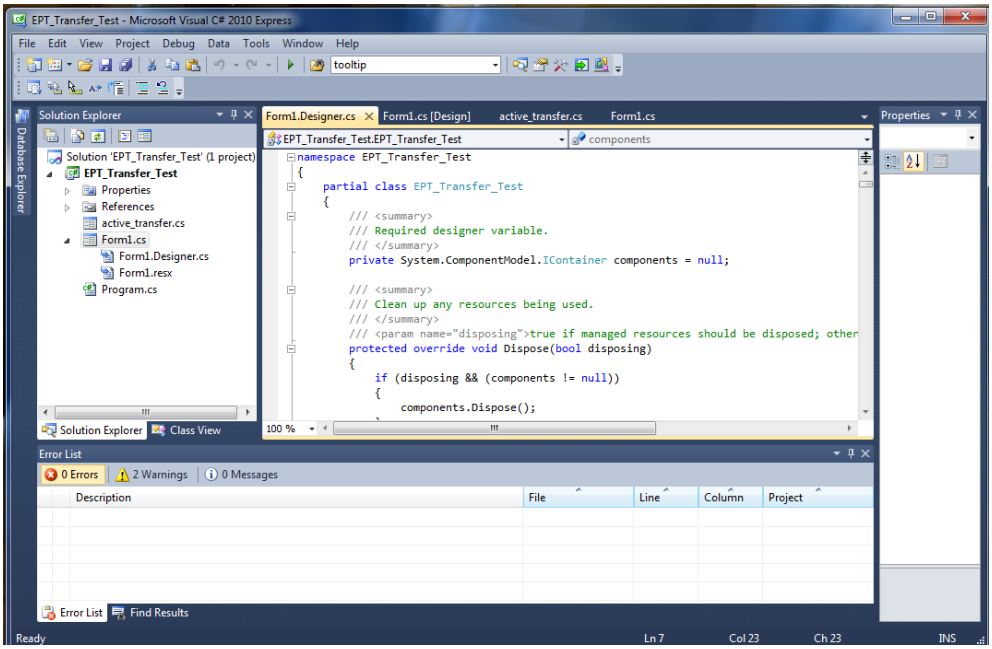
Right click on the project in Solution Explorer. Choose Add Reference. Click the Browse tab. Point to the DLL. Click Ok.



Browse the location where the EPT Active Host Application Suite is located, find the Data Transfer example folder, copy the *.cs files and place them in the project folder.



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.

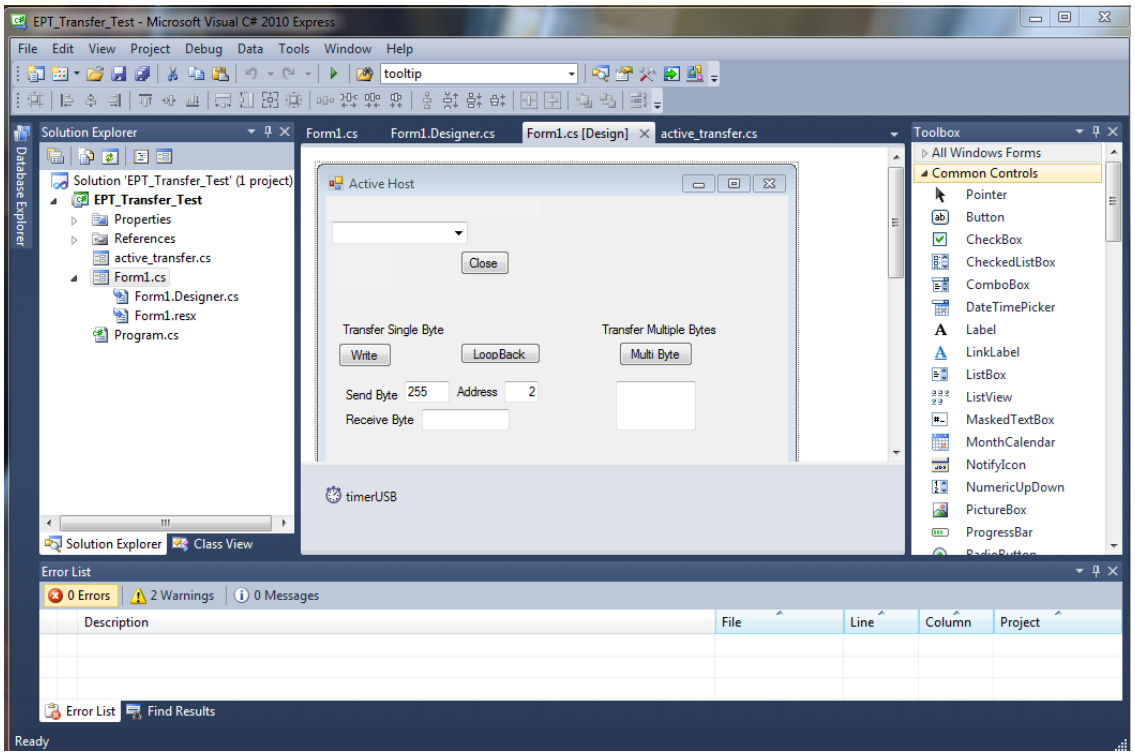


1.8.3 Adding Controls to the Project

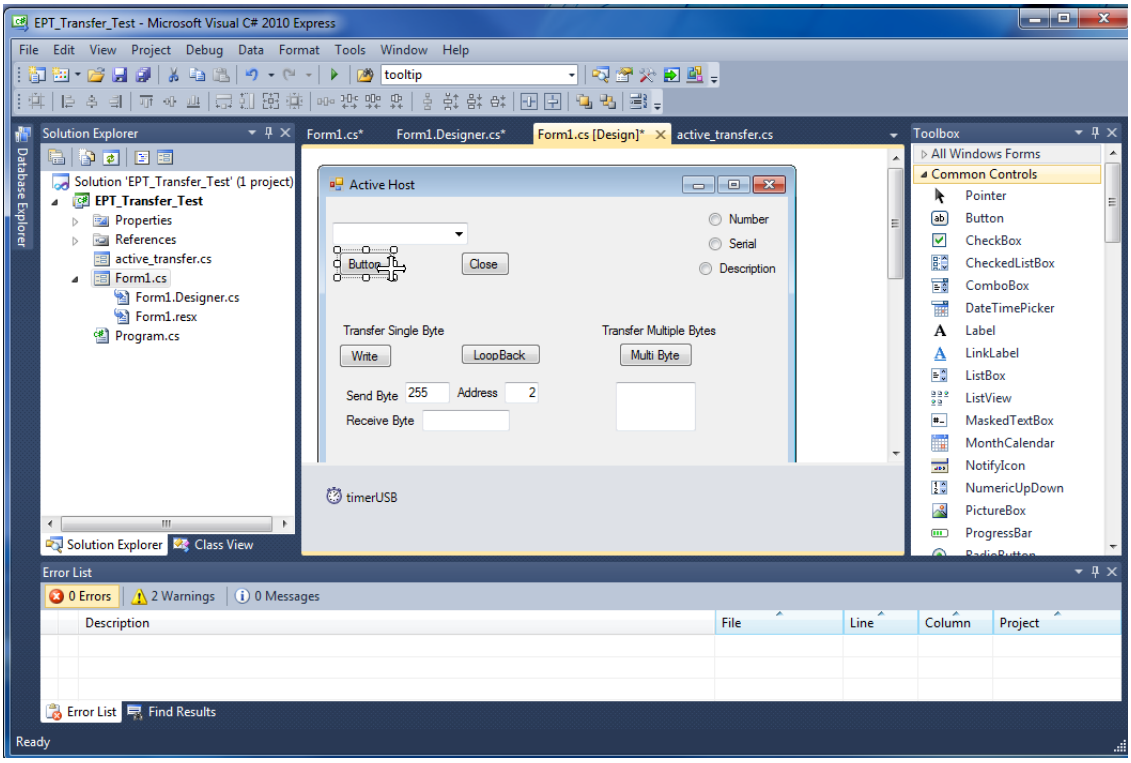
Although, the C# language is very similar to C Code, there are a few major differences. The first is C# .NET environment is event based. A second is C# utilizes classes. This guide will keep the details of these items hidden to keep things simple. However, a brief introduction to events and classes will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

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

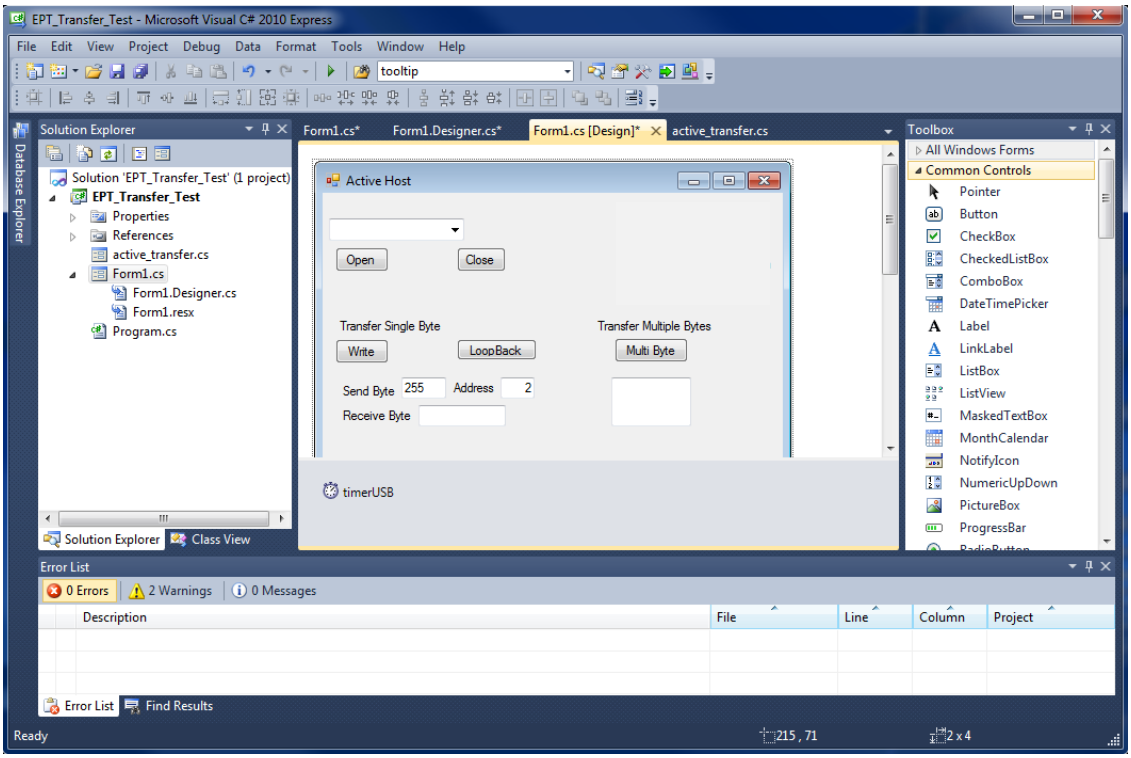
The easiest way to add a button to a form is to double click the Form1.cs in the Solution Explorer. Click on the  button to launch the Toolbox.



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



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



Double click on the Open button. The C# Explorer will automatically switch to the Form1.cs code view. The callback function will be inserted with the name of the button

along with "_click" appended to it. The parameter list includes (object sender, System.EventArgs e). These two additions are required for the callback function to initiate when the "click" event occurs.

Private void btnDeviceSel_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.

```
//
// btnOpen
//
      this. btnOpen.Location = new System.Drawing.Point(12, 51);
      this. btnOpen.Name = "btnDeviceSel";
      this. btnOpen.Size = new System.Drawing.Size(50, 23);
      this. btnOpen.TabIndex = 4;
      this. btnOpen.Text = "Init";
      this. btnOpen.UseVisualStyleBackColor = true;
      this. btnbtnOpenDeviceSel.Click += new System.EventHandler(this.
btnOpen_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 btnOpen button to call the EventHandler –btnOpen_Click. This is where the magic of the button click event happens.

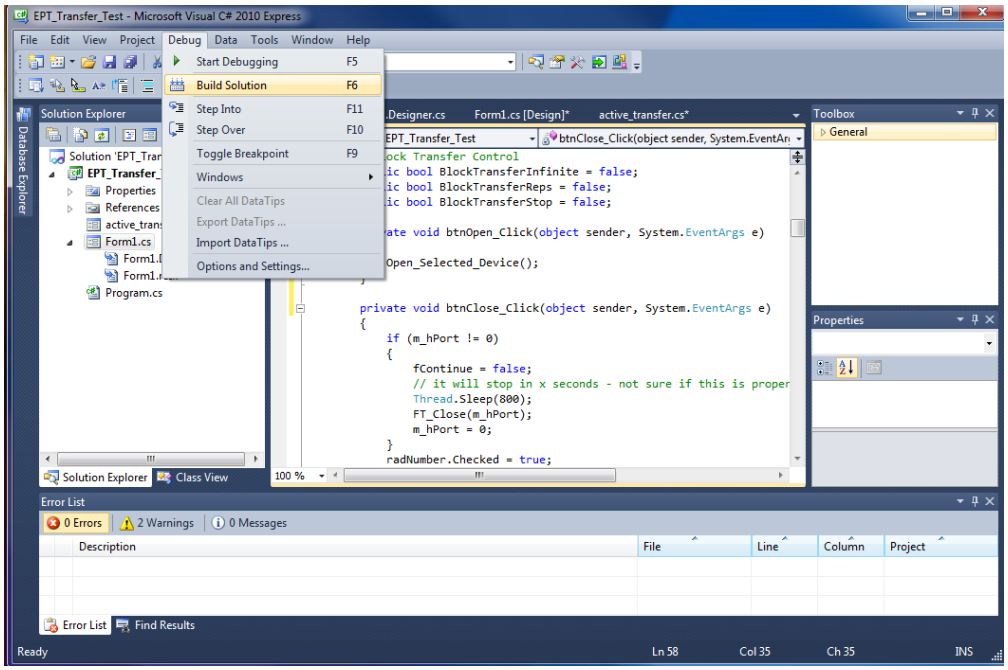
```
private void btnOpen_Click(object sender, System.EventArgs e)
{
    Open_Selected_Device();
}

private void btnClose_Click(object sender, System.EventArgs e)
{
    if (m_hPort != 0)
    {
        fContinue = false;
        // it will stop in x seconds - not sure if this is proper
        Thread.Sleep(800);
        FT_Close(m_hPort);
        m_hPort = 0;
    }
    radNumber.Checked = true;
```

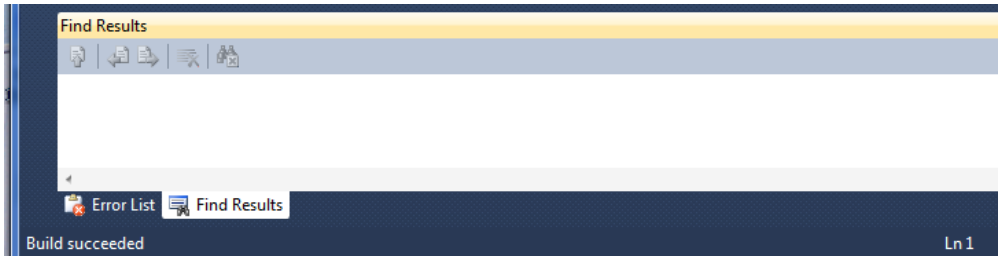
When btnOpen_Click is called, it calls the function "Open_Selected_Device()". This function is defined in the dll and will connect to the device selected in the combo box. This is a quick view of how to create, add files, and add controls to a C# project. The user is encouraged to spend some time reviewing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

Visual C# .NET programming. In the meantime, follow the examples from the Earth People Technology to perform some simple reads and writes to the EPT USB/PLD Development System.

1.8.4 Building the Project

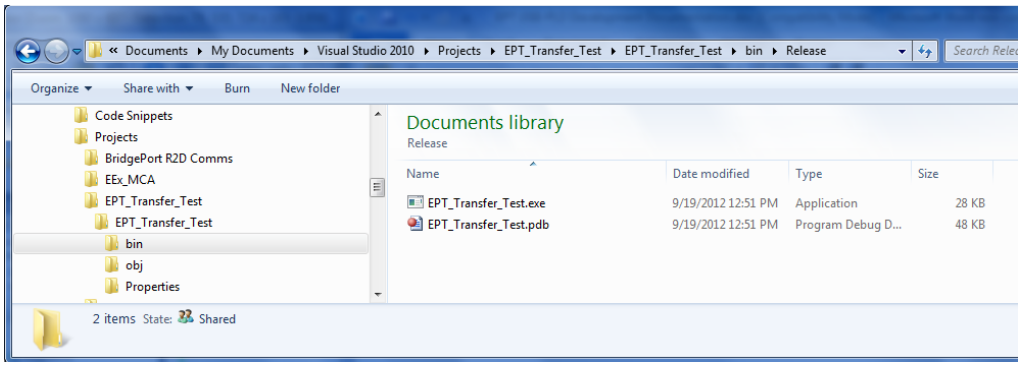
Building the EPT_Transfer_Test project will compile the code in the project and produce an executable file. To build the project, go to Debug->Build Solution.



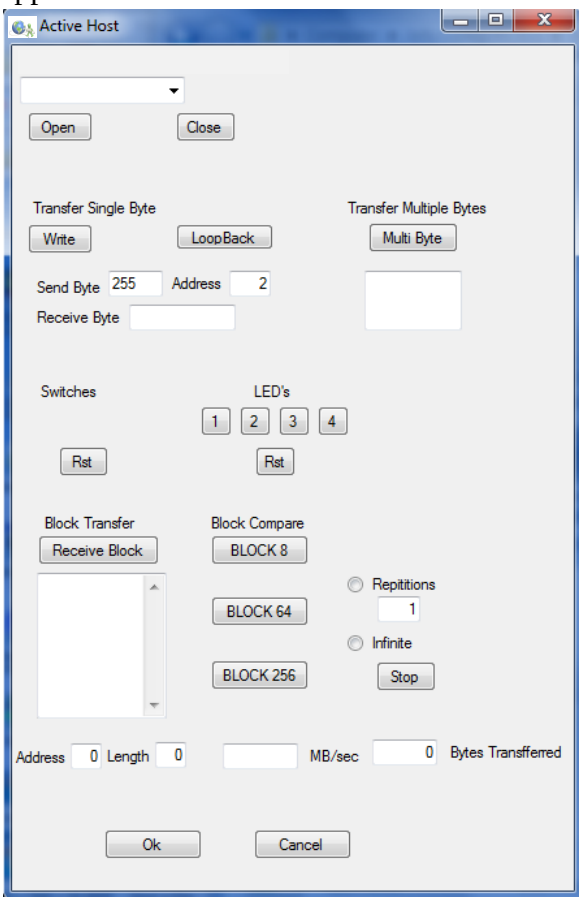


1.8.5 Testing the Project

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_Transfer_Text.exe file can now be tested using the EPT-570-AP-U2 board. To test the file, connect the EPT-570-AP-U2 to the Windows PC using Type A to Type Mini B USB cable. Make sure the driver for the board loads. If the USB driver fails to load, the Windows OS will indicate that no driver was loaded for the device. Go to the folder where the EPT_Transfer_Text.exe file resides, and double click on the file. The application should load with a Windows form.



With the application loaded, select the USB/PLD board from the dropdown combo box and click on the “Open” button. Click on one of the LED buttons in the middle of the window. The corresponding LED on the EPT-570-AP-U2 board should light up.
[Add more steps to verify the operation of the project]

2 2. EPT Drivers

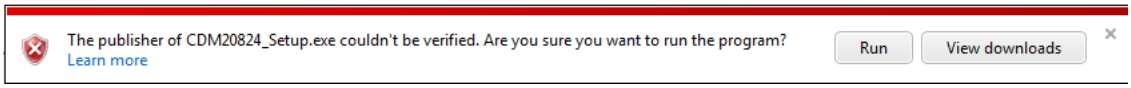
The EPT USB/PLD Development system requires drivers for any interaction between PC and the EPT-570-AP-U2. The communication between the two consists of programming the PLD and data transfer. In both cases, the USB Driver is required. This will allow Windows to recognize the USB Chip and setup a pathway for Windows to communicate with the USB hardware.

2.1 USB Driver

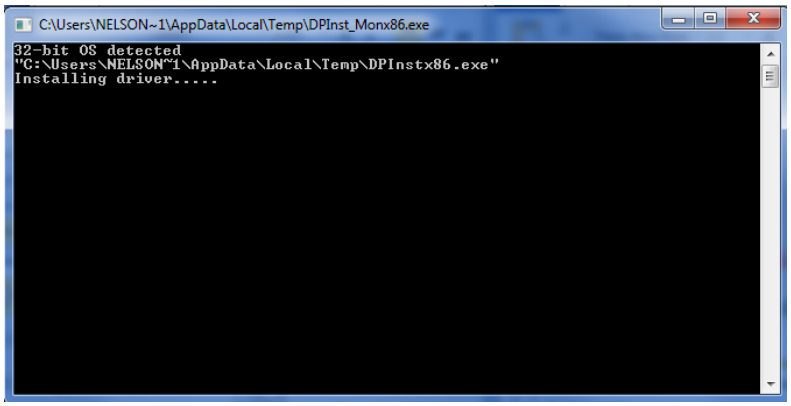
The EPT-570-AP uses an FTDI FT2232H USB to Serial chip. This chip provides the USB interface to the PC and the serial/FIFO interface to the PLD. The FT2232H uses the ftdibus.sys driver. This driver is loaded upon connection of the EPT-570-AP to the PC by the ftdibus.inf file. To install these two files onto your PC, use the FTDI Combined Driver Model CDM 2.08.xx WHQL Certified (xx signifies the latest release). The installation of the CDM 2.08.xx is easily accomplished using the executable. Just download the executable

When the download is complete, click Run on the banner to start the install of the driver files.

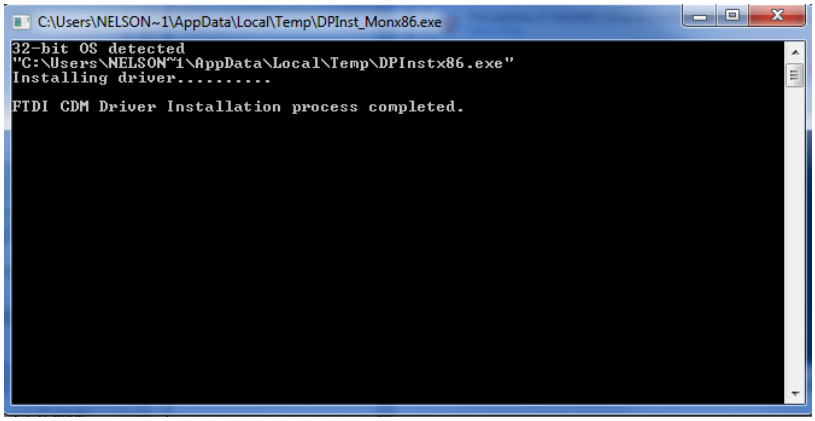
Click on the Run button.



Click on the Run button again. The Console will come up and indicate the installation and completion of the drivers.



The driver files will automatically install in the System Registery.



When this is complete, the drivers are installed and the EPT-570-AP can be connected to the PC and used.

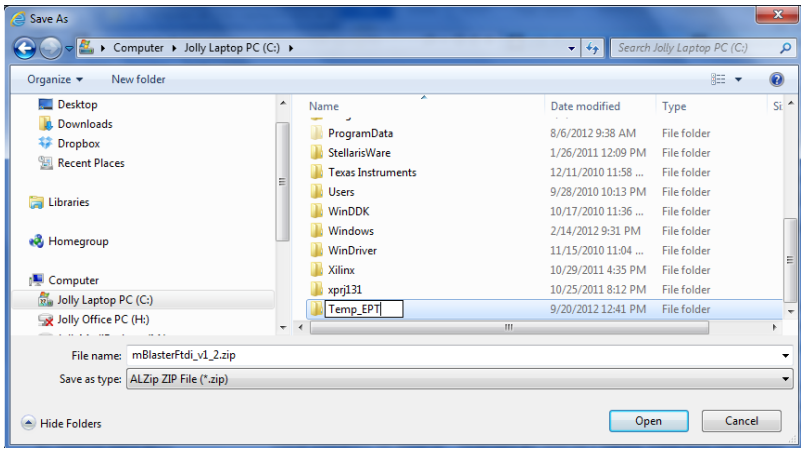
2.2 JTAG DLL Insert to Quartus II

The JTAG DLL Insert to Quartus II allows the Programmer Tool under Quartus to recognize the EPT-570-AP. The EPT-570-AP can then be selected and perform programming of the PLD. The file, jtag_hw_ mbftdi_blaster.dll must be placed into the folder that hosts the jtag_server for Quartus. This dll is available for Windows XP 32-bit, Windows 7 32-bit and Windows 7 64-bit.

Close out the Quartus II application. Download the file from the EPT website.

Open file and click on the Save As button.

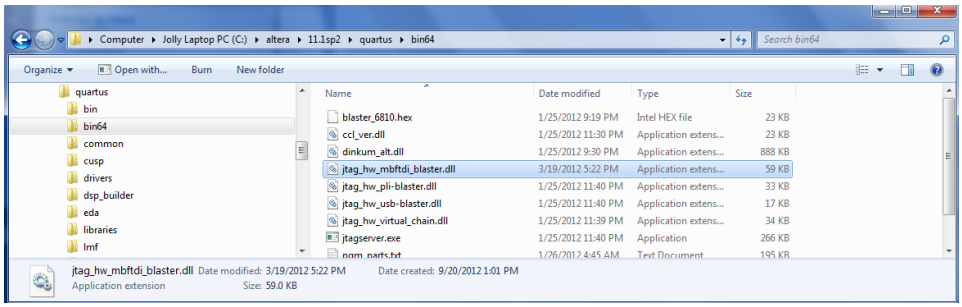
When the Save Dialog box opens, click on “New Folder” and enter Temp_EPT” as the folder name. Click Enter.



Unpack the zip file using an unzip tool.
If your system is Windows 64 bit follow these directions:

Windows 64 bit

1. Open the C:\Temp_EPT\Win_Quartus_Programmer_v10b\win_quartus_prog_v10b\x64 folder.
2. Select the file “jtag_hw_mbftdi_blaster.dll” and copy it.
3. Browse over to C:altera\11.1sp2\quartus\bin64.
4. Right click in the folder and select Paste
5. Click Ok.
6. Open the Quartus II application.



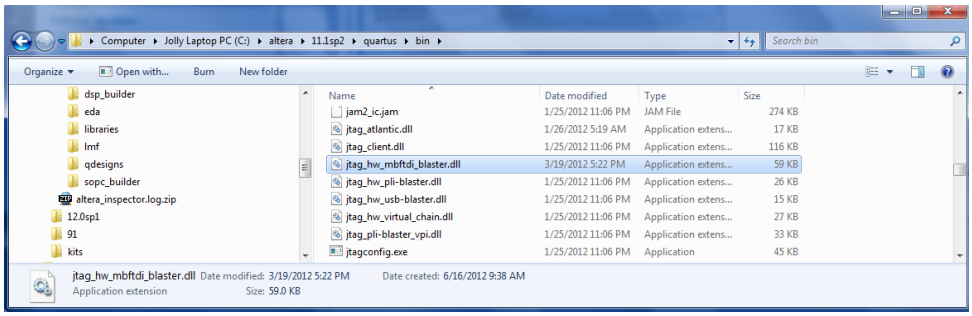
The DLL is installed and the JTAG server should recognize it. Go to the section “Programming the PLD” of this manual for testing of the programming. If the driver is not found in the Programmer Tool->Hardware Setup box, see the JTAG DLL Insert to Quartus II Troubleshooting Guide.

If your system is Windows 32 bit follow these directions:

Windows 32 bit

1. Open the C:\Temp_EPT\Win_Quartus_Programmer_v10b\win_quartus_prog_v10b\win32 folder.

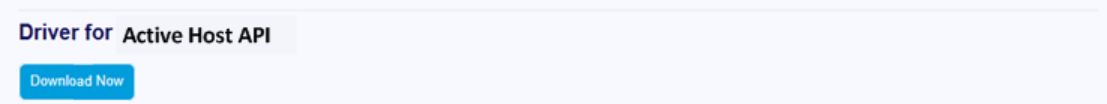
2. Select the file “jtag_hw_mbftdi_blaster.dll” and copy it
3. Browse over to C:altera\11.1sp2\quartus\bin.
4. Right click in the folder and select Paste
5. Click Ok.
6. Open the Quartus II application.



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

2.3 Active Host Application DLL

To use the Active Host Application Software, the Active Host DLL must be included in the Microsoft Visual project. The Active Host Application Software will allow the user to create a custom application on the PC to perform Triggers and Data Transfer to/from the EPT-570-AP. The methods and parameters of the Active Host DLL are explained in the Active Host Application section. Download this file from the EPT website.



Save the dll in the top level folder of the user project under the Microsoft Visual project. See the Active Host Application for instructions on how to add the dll to the Microsoft Visual project.

[Insert picture of Active Host Application DLL in Microsoft Visual project project folder]

3 3. The Development Process

There is no standard for developing embedded electronics. The best method is the one that works for the user. These methods can range from a top down approach where the design is written down first and all code is written, then compile, execute and test. Or a bottom up approach can be pursued where a small piece of the project is assembled and verified (i.e. I2C communication to a sensor). Then the next piece is assembled and verified (i.e. collect sensor data in a storage buffer) and connected to the first. And so

on, until the whole design is complete. Or any infinite combination of these two extremes.

3.1 Designing a Simple Data Collection Sampler

3.1.1 The User Microcontroller Board (Objective Device)

Using the features and capabilities of the development system associated with the objective device circuit board (referenced in figure 1 as "Microcontroller"), the user will develop the source code and download the resulting binary code to the memory of the objective device. This code will control the various functions of the objective device and produce data which can check the health and operation of the device. The user should include recording capability of appropriate signals and data to permit monitoring of the objective device operation through the PLD program.

3.1.2 Designing a Simple Data Collection Sampler

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

3.1.4 Coding the Arduino Data Sampler

3.1.5 Building Arduino Project

3.1.6 Programming the Arduino

3.1.7 PLD Active Transfer Coding and Initiation

In order to control and debug the objective device operation it is necessary to devise a test function to be loaded and executed in the PLD of the EPT-570-AP board. This test function will transfer data and control signals to and from the objective device to execute the user's code in the objective device and permit the results of its operation to be displayed on the Hyper Serial Port GUI.

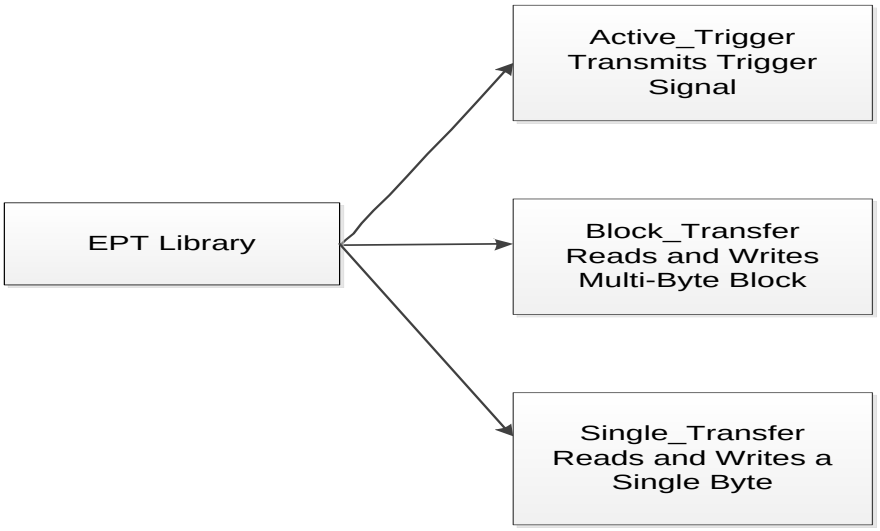
The test function is coded using the Altera Quartus II system software. Development of the Quartus II program to be downloaded to the PLD involves four steps, briefly described as follows:

3.1.8 PLD: Define the User Project.

This step defines the user's test program and includes modules from the EPT Active Transfer Library. It contains a set of files with a ".xco" name extension which select particular operations to perform (e.g., byte transfer, block transfer, trigger). These files connect to elements of the "EPT-Active-Transfer Library.xco" file. All of these files are available on the Earth People Technology website. The primary file defining the user's PLD configuration is named "Top-Level.v" which defines PLD pin distribution and logic functions. Clearly, logic and instructions laying out the PLD code

must be compatible with the code and pin layout of the objective device. PLD pin usage is defined in the pin planner file "Top-Level.pin". Details of how to generate Top-Level.v files and Top-Level.pin files are given in the Quartus II software usage description. EPT Active Transfer library modules and their functions are listed in Figure 5.

Figure 5. EPT Library Modules



3.1.9 PLD: Compile/Synthesize

The Quartus II window contains a command to select the compile/ synthesize operation. The result of this step is either a file containing the PLD code with ".pof" name extension or errors are identified which the user should correct and then recompile. Note that in addition to fatal errors, the compile process can produce "warnings" which do not necessarily prevent execution of the code but which should be corrected eventually. The Notepad++ text editor is a valuable tool in editing and correcting the Quartus II source code.

3.1.10 PLD: Program the PLD

The final step is downloading the ".pof" file to the PLD. This is initiated in the Quartus II software by selecting the "Program" command in the Quartus II window. Note that the software driver described in section 2.2 must be loaded in Quartus II, which will ensure that the program code is downloaded to the PLD via the JTAG/Side A I/O bus.

3.1.11 PC Coding the Active Host

After the program is downloaded to the PLD, the software becomes ready and starts the execution process.

3.1.12 Compiling the Active Host Application

3.1.13 Connecting the Project Together

3.1.14 Testing the Project

3.1.15 Hyper Serial Port (HSP) Application

Communication with the PLD code from the PC is effected through the Hyper Serial Port which transmits to and from the PLD via the FT245/Side B channel. Note that the Hyper Serial Port does not download any information for storage in the PLD memory, but merely sends and receives signals to and from the active library modules in the PLD. The EPT USB/PLD system permits up to eight separate trigger, byte transfer and block transfer modules to be loaded in the PLD via the Quartus II software system.

HSP contains an embedded scripting evaluator which enables Python scripts to be used to generate commands to read and write to the PLD trigger and active transfer modules in the PLD. Details of the Python language are given at http://www.python.org/. IronPython is an implementation of the Python language for operation within the Microsoft .NET environment. Details of the IronPython implementation are given at http://www.python.org/. Notepad++ is a useful text editor for generating and editing the Python scripts.

HSP also provides a visual control and interface window with a variety of control options available. Details are provided at the HSP website, http://earthpeopletechnology.com/ where the software may be downloaded to a user's PC.

4 4. Summary of Hyper Serial Port (HSP) Capabilities

HyperSerialPort is a serial terminal emulator which has built in tools to assist developers of embedded electronics to quickly diagnose problems associated with the design and programming of microcontrollers. It is built on the Microsoft .NET Framework 4.0 and designed for Windows 7, 32-bit. It has been tested also with Windows XP. Various embedded electronics evaluation tools are included in HSP. These are briefly described in the following sub-sections.

4.1 Embedded Scripting Evaluator

The Embedded Scripting Evaluator allows developers to run Python scripts within the HSP environment. This will allow users to develop real time feedback loops to mimic processor-to-device communications. The user could write a script to emulate an I2C device and use an embedded processor to communicate with the I2C device and

debug communication errors. Caution must be used when using this mode. HSP does not put any restrictions on using Python scripts and thus user scripts could be generated which could damage the OS. Care must used when writing Python scripts in this mode.

4.2 Send Character Timer

The Send Character Timer Function allows the user to send a repeating character or string at a timed sequence over the serial port. This function utilizes the high resolution property of Windows 7 OS. This will allow the timer to reach up to 700 microseconds between characters sent. Some caution must be exercised with timed sequences in the Windows 7 OS. This operating system is a non-deterministic operating system. This means it cannot guarantee the timer will expire at 700 microseconds for each sequence.

4.3 External Trigger

The External Trigger function will pause the incoming characters on a serial port until a selected character or string appears on the selected trigger serial stream. This allows the user to capture an event that may be difficult to see when a serial stream is continuously updating the serial terminal window.

5 5. Example Application Arduino Motor Controller with PID Closed Loop Control Using the EPT USB/PLD Development System

The Arduino Open Source Prototyping Platform is a readily available family of circuit boards (e.g., available from the Radio Shack Corporation). The Arduino UNO R3 board will be used here to illustrate an example application of the EPT USB/PLD Development System. This board contains a USB connector, an 8-bit microcomputer IC (also known as a microcontroller), and some memory plus several connector headers.

The Arduino system includes PC host software to generate and download instructions to the microcontroller. The user is referred to that source for information for programming the Arduino board. Following is an example Quartus II program for communicating with the Arduino board using the EPT USB/PLD system.

APPENDIX I

List of Abbreviations and Acronyms

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

APPENDIX II

Details of the Altera EPM570 PLD