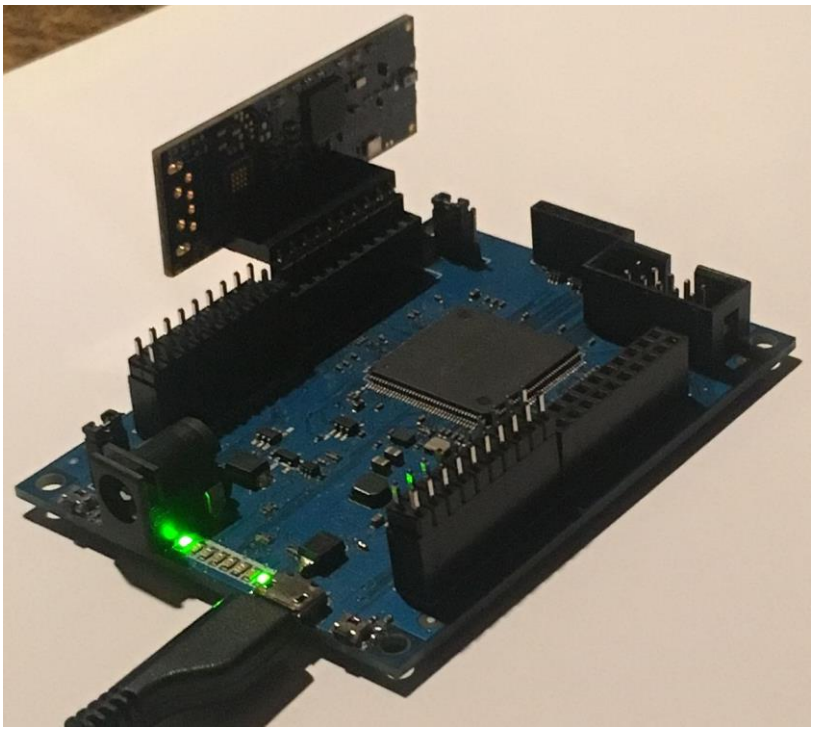
EARTH PEOPLE TECHNOLOGY

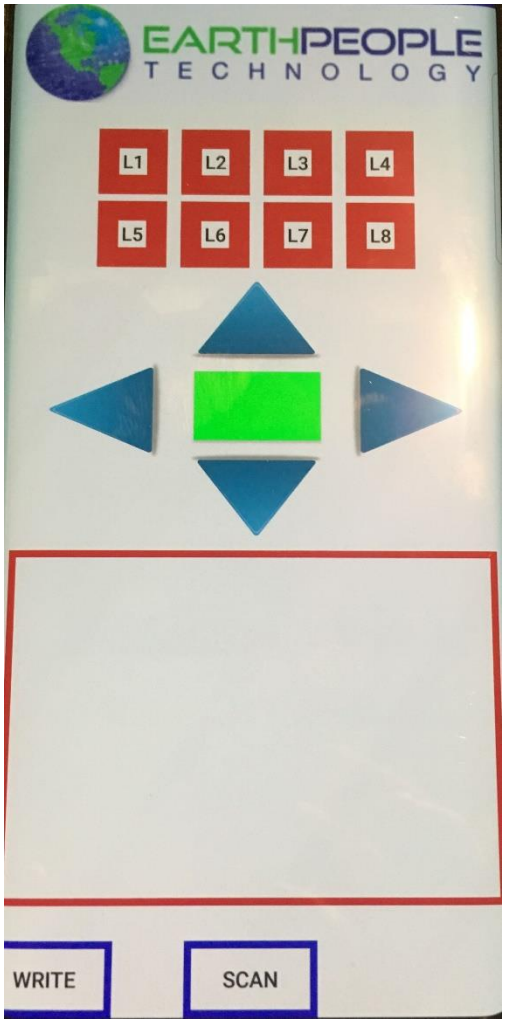
ODIN-LINK BLE
+ MAXPROLOGIC
Development System
User Manual



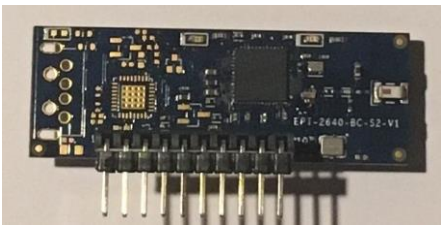
This User Manual covers the Odin-Link and MaxProLogic Development System. The Odin-Link BLE board is a complete v4.2 BLE board with integral 2.4GHz antenna and matching network. This board plugs directly into the MaxProLogic. It communicates with the FPGA over a UART serial link. The Odin-Link BLE board runs proprietary firmware along with the radio control code. The Odin-Link BLE board communicates wirelessly with a custom app on the Android phone. The MaxProLogic has a proprietary Verilog interface running in the FPGA that allows full bidirectional communication with the Android phone.

1 Getting Started with the MaxProLogic + Odin-Link BLE

The Odin-Link BLE board is the simplest BLE development kit on the market. The Android phone app is completely self contained, there is no programming or third party provider access. Just load the “JoyStick.apk” onto your Android phone.



The Odin-Link BLE board comes pre-programmed with all the profiles required for the Android phone app. No programming is needed for the board. The Odin-Link BLE Board connects directly onto J5 of the MaxProLogic Board.



EPT has created a Verilog library that runs on the MaxProLogic MAX10 FPGA. The library makes interacting with the BLE Phone App extremely easy. The Odin-Link BLE and MaxProLogic Development System comes complete with a Blinky Demo project installed. This project allows the user to explore the JoyStick app that runs on the Android phone. The user will learn what each function of the JoyStick app is capable of. Once the user has become familiar with the app, the user manual will explain how to user the Verilog library to create a custom project that can perform any function the user desires. Some example projects are phone app controlled robot, light controller, door lock and more.

1.1 The Odin-Link BLE to MaxProLogic Blinky Demo

EPT has created an Odin-Link BLE to MaxProLogic Blinky Demo that allow the BLE Phone App to turn on and off the LEDs of the MaxProLogic board. The following description explains how to set up the hardware and get the demo working.

1.1.1 Setting up the Phone App

The Phone App is a single apk file labeled, Joystick_v-1.0.apk. This is a standalone android application. The app has been tested with the following versions of Android:

- TBD

There are two easy methods to add the phone app to an Android phone.

- Download phone app to a PC. Connect a USB cable from PC to phone. Use Windows Explorer on the PC to add the phone app to the storage folder on the phone. Open the app on the phone and allow install to occur.
- Email the phone app as an attachment to your email account. Open the email service on the phone. Click on the attachment. Open the app on the phone and allow install to occur.

1.1.1.1 Load phone app from PC

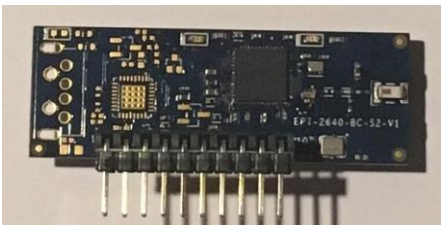
Add the Joystick_v-1.0.apk file from either the DVD or download from the earthpeopletechnology.com website to a folder on your PC. Connect a USB cable from the PC to the Android phone. Use the Windows Explorer to view the folders on the Android phone. Open the storage folder under the Android phone. Copy the Joystick_v-1.0.apk file onto the storage folder of the Android phone. When the file has completed the copying process, the Android phone will attempt the install it. A user message will appear on the Android display. This message will query the user about installing the app. Click "Yes". When the app is fully installed, it will an icon to the users apps. Click on this icon open the app.

1.1.1.2 Load phone app from email

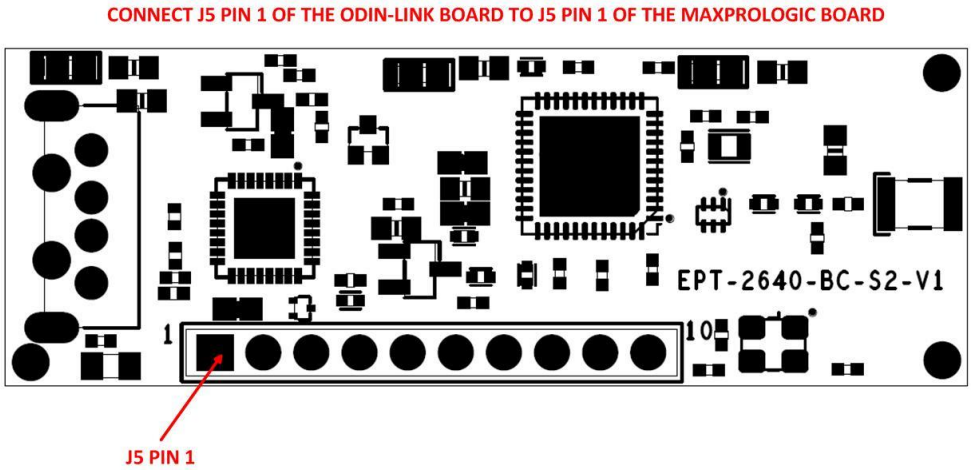
Add the Joystick_v-1.0.apk file from either the DVD or download from the earthpeopletechnology.com website to a folder on your PC. Open your email editor such as yahoo mail or school/work mail. Compose a new email. Use an email address that is linked to your Android phone such as a gmail account. Use the Attach feature of the email editor and locate the Joystick_v-1.0.apk file. Attach the apk to the email. Hit the Send button. Go to your Android phone. Open up the email editor of the account which you sent the email to. Locate the email. Click on it, then click on the attached apk file. When the file has completed the copying process, the Android phone will attempt the install it. A user message will appear on the Android display. This message will query the user about installing the app. Click "Yes". When the app is fully installed, it will an icon to the users apps. Click on this icon open the app.

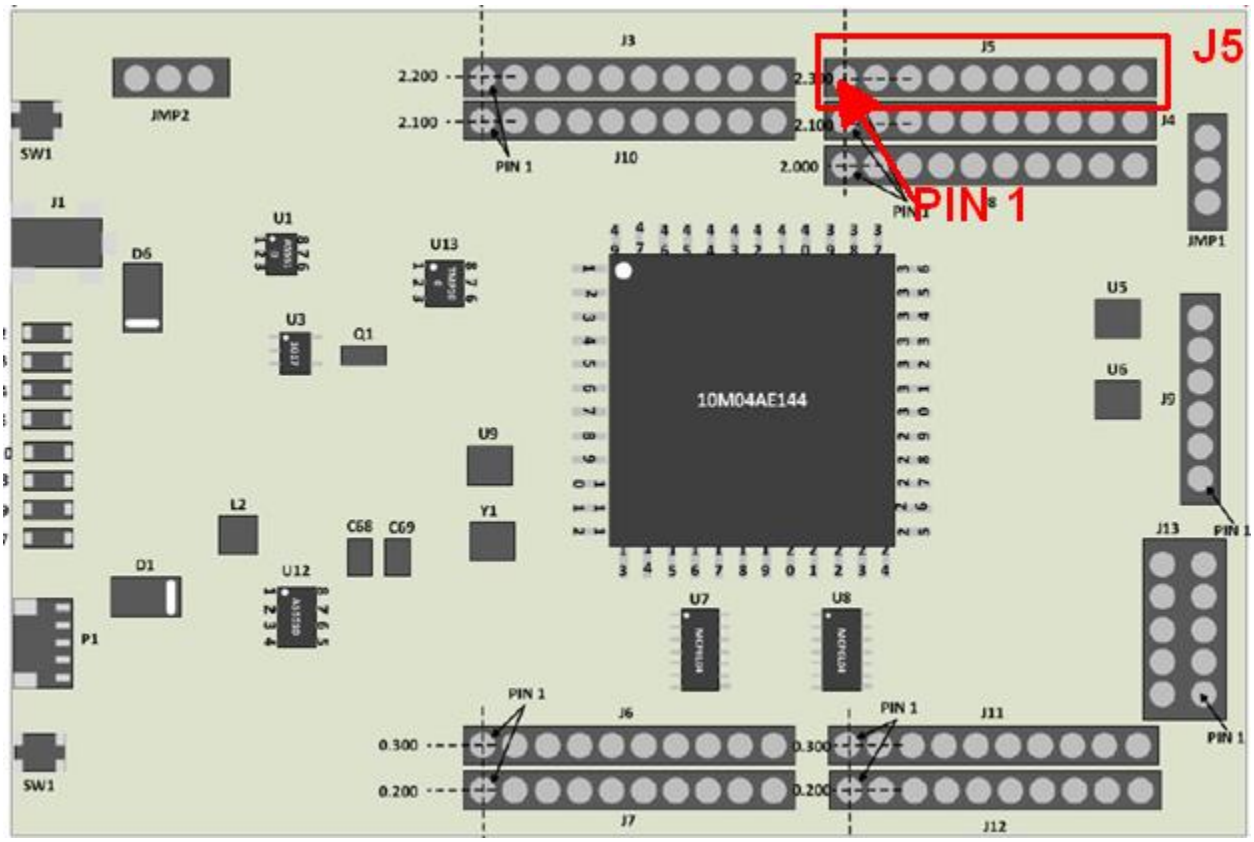
1.1.2 Connecting the Odin-Link BLE to the MaxProLogic

The Odin-Link BLE Board connects directly onto J5 of the MaxProLogic Board. The Odin-Link BLE Board is pre-programmed with the radio firmware, interface firmware and the BLE profiles that link to the Joystick_v-1.0.apk. Pin 1 of the BLE Board is located on the lower left of the component side of the PCBA.

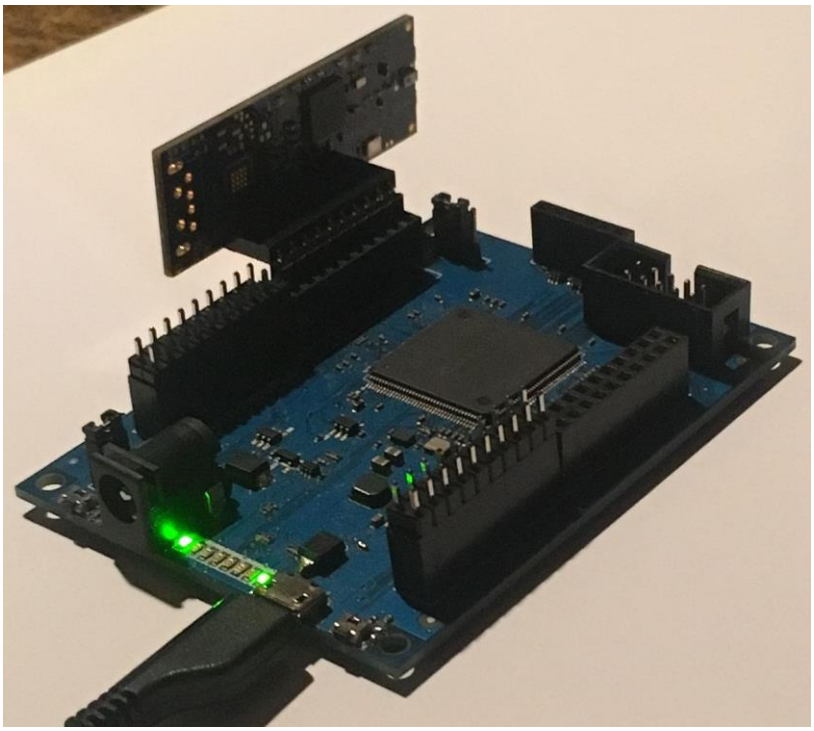


Pin 1 of the MaxProLogic J5 connector is located on the upper left side of the component side of the PCBA.





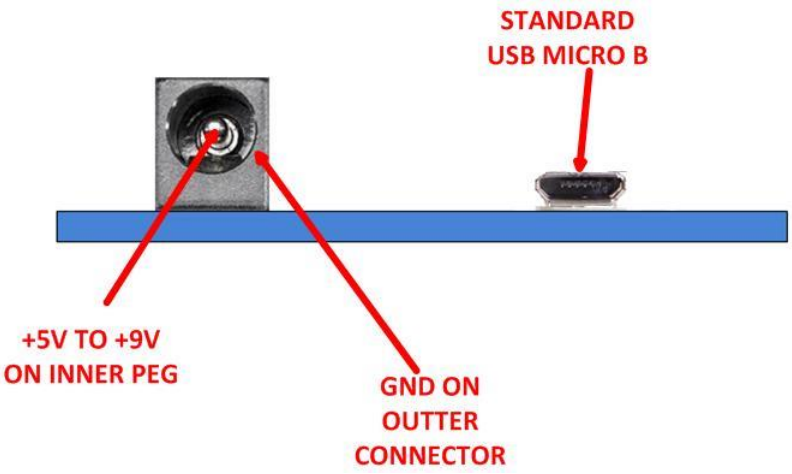
Once the Odin-Link Board is installed in the MaxProLogic, connect power to the system.



1.1.3 Powering the MaxProLogic

You can run the MaxProLogic from a laptop with 2.5W of power. Or you can run it from the +5V @ 2A wall USB chargers for 10W of power. The barrel connector can handle up to +9V @ 3 A for 27W of power.

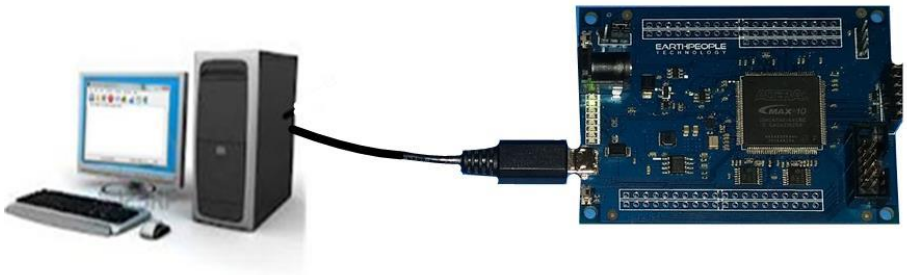
- Standard USB cable from Laptop/PC.
- +5 VDC wall charger (phone charger) through USB cable.
- +5.5 to +9 VDC supplied through the DC power jack.
- +5.5 to +9 VDC supplied through I/O connector pin.



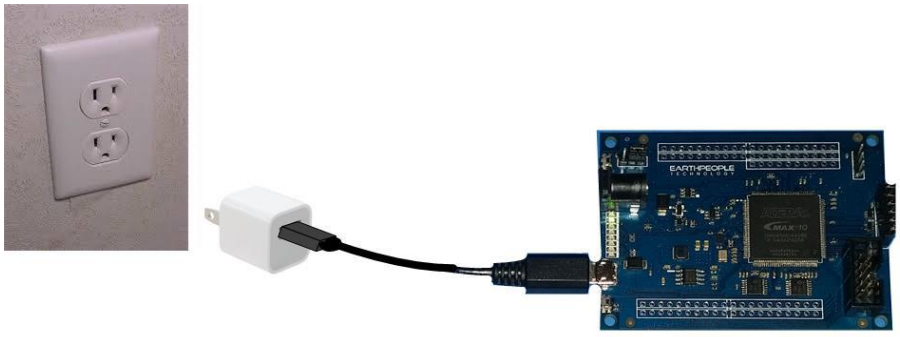
The barrel connector is the typical size used on many popular DIY boards such as the Arduino series. It has the following mechanical specs:

- 2.0mm Inner Diameter
- 5.5mm Outer Diameter

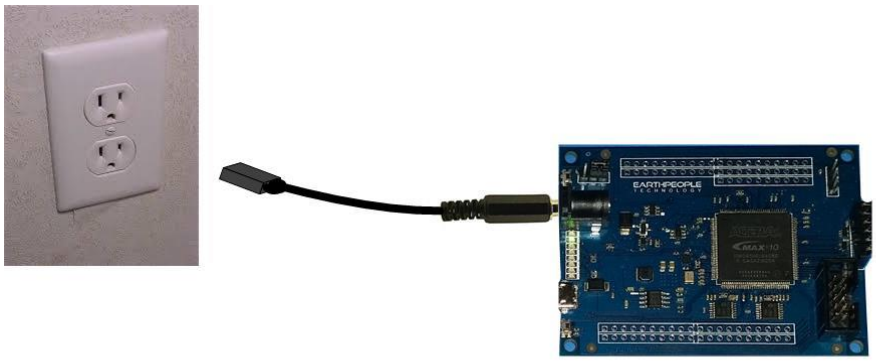
The barrel connector does not include a diode protection to prevent reverse polarity connection. So, care must be exercised when connecting up your cable to the barrel connector. Please ensure the correct polarity connections are made before connecting to the MaxProLogic. Also, there is no discrete protection to the power input. The power supply does include a high current protection circuit. The current limit is around 4.7Amps. But, the MaxProLogic is only designed to handle 2Amps of current. So, damage may occur to the MaxProLogic if the user does not exercise care in design and use of the Inputs/Outputs.



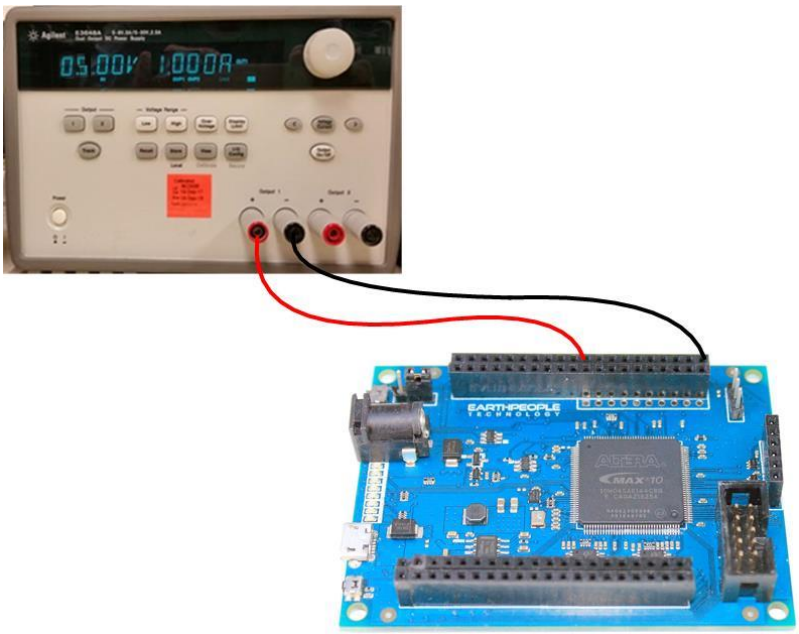
Power the MaxProLogic directly from the PC. +5V@0.5A



Power the MaxProLogic directly from the wall charger. +5V@2A



Power the MaxProLogic from the Barrel Connector. +9V@3A



Power the MaxProLogic directly from a power supply. +5V@2A

With the MaxProLogic power up, it applies power to the Odin-Link BLE Board.

1.1.4 Adding the EPT-BLE-Demo Project to the MaxProLogic

The Odin-Link BLE and MaxProLogic Development Kit comes complete with the Blinky Demo project installed in the FPGAs internal Flash. You don't need to load any code onto the MaxProLogic. The Blinky Demo works straight out of the box. If the Blinky Demo is in the MaxProLogic you can skip section 1.1.5.

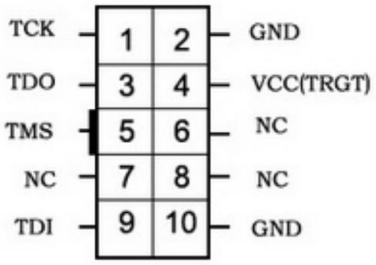
If for any reason the Blinky Demo code needs to be loaded onto the MaxProLogic, use the following steps.

1.1.5 Loading the Blinky Demo Project onto the MaxProLogic

The EPT_10M04_AF_BLE_Driver Project must be programmed into the MaxProLogic's Internal Flash. Programming the Flash is accomplished by connecting a programmer to the JTAG interface.

The MaxProLogic has a 5x2 header for use in programming the MAX10 FPGA via JTAG. The connector is located in the bottom right corner of the MaxProLogic. It is shrouded and keyed to allow easier insertion.

This connector uses the standard Altera Blaster connector pinout.

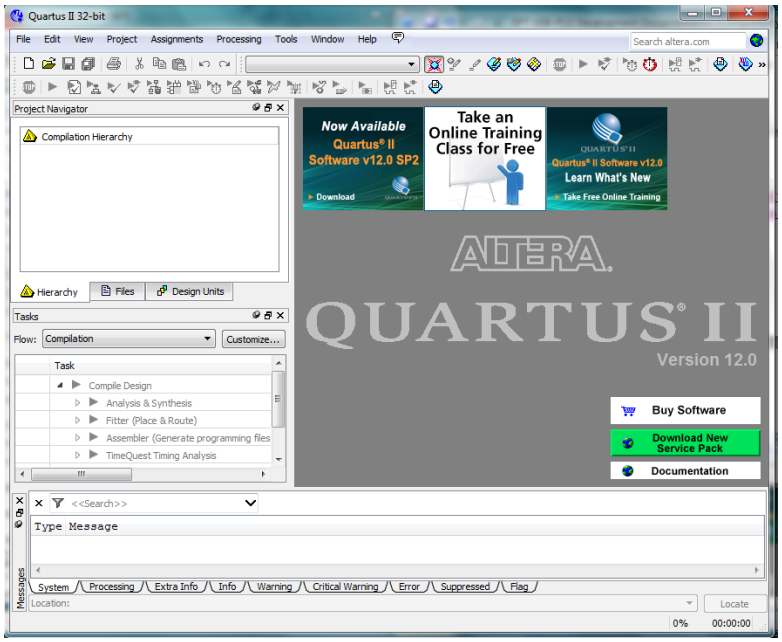


The VCC(TRGT) is set to +3.3V on the MaxProLogic. There are no jumper settings to make in order to program the MAX10 FPGA. Just connect a compatible Blaster to the connector and the PC, then use the Quartus software to program the FPGA.

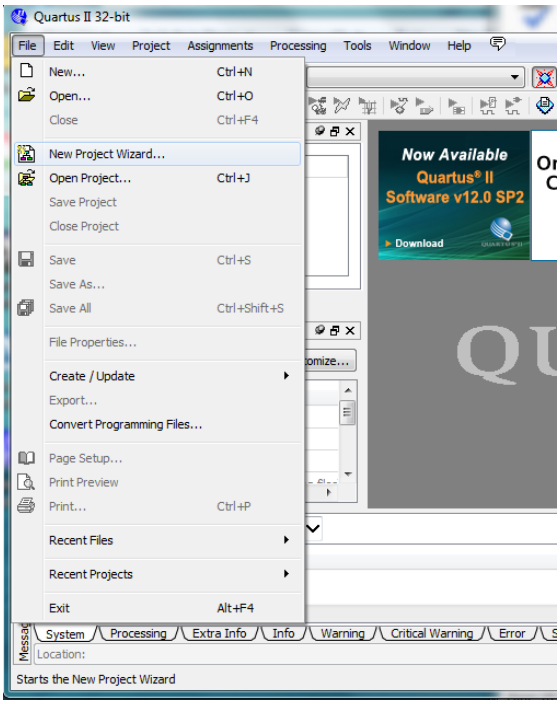
To program the MAX10 Configuration Flash, connect the Blaster and ensure the JTAG Driver is loaded for Quartus II.

Open Quartus II by clicking on the icon 

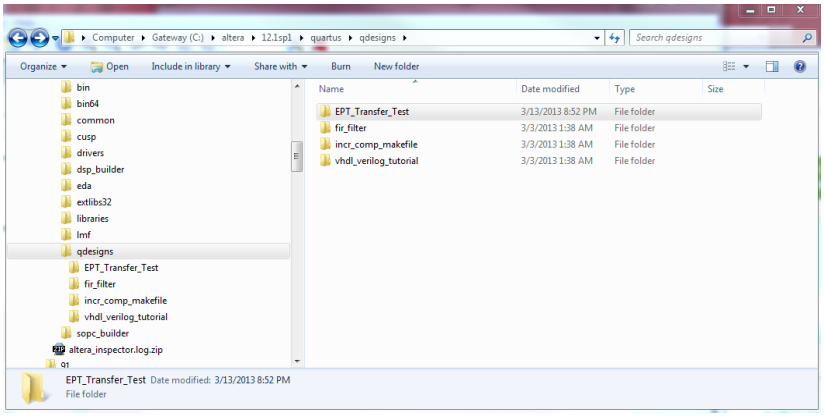
.



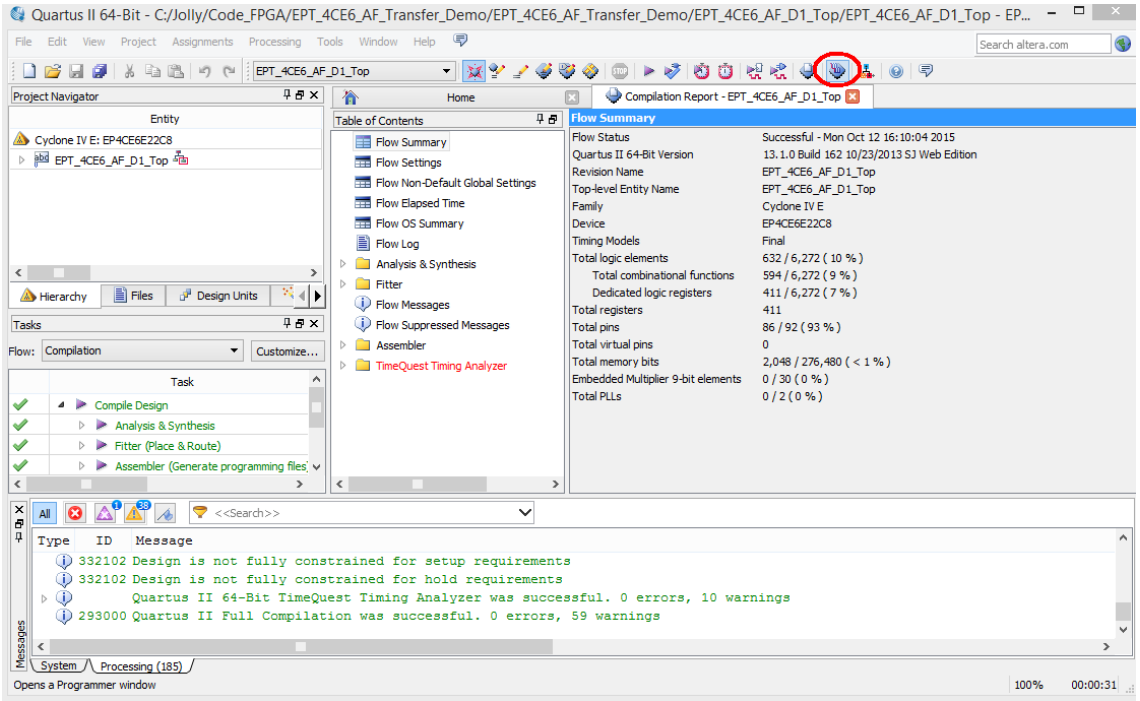
Wait for Quartus II to open, under Quartus, Select File->Open Project.



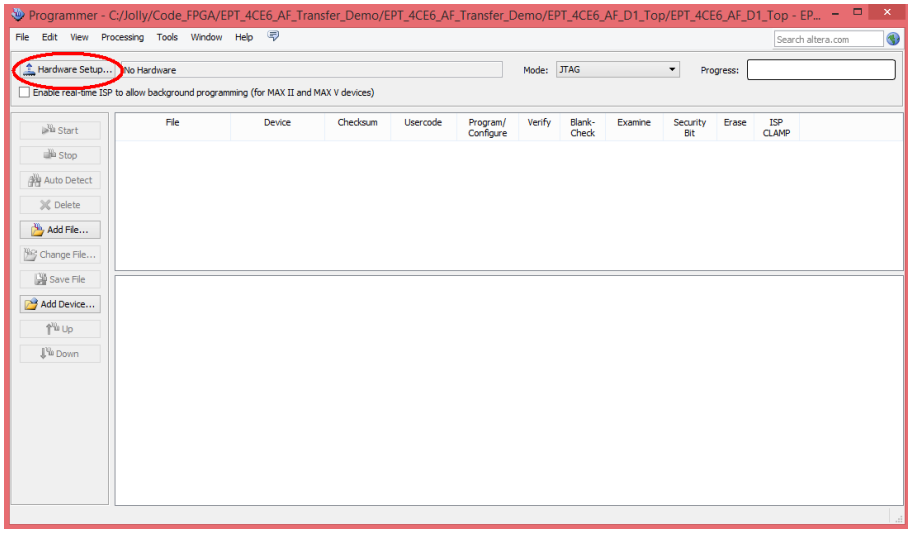
At the Windows Explorer, locate the EPT_10M04_AF_BLE_Driver project on the DVD. Open up the EPT_10M04_AF_BLE_Driver project.



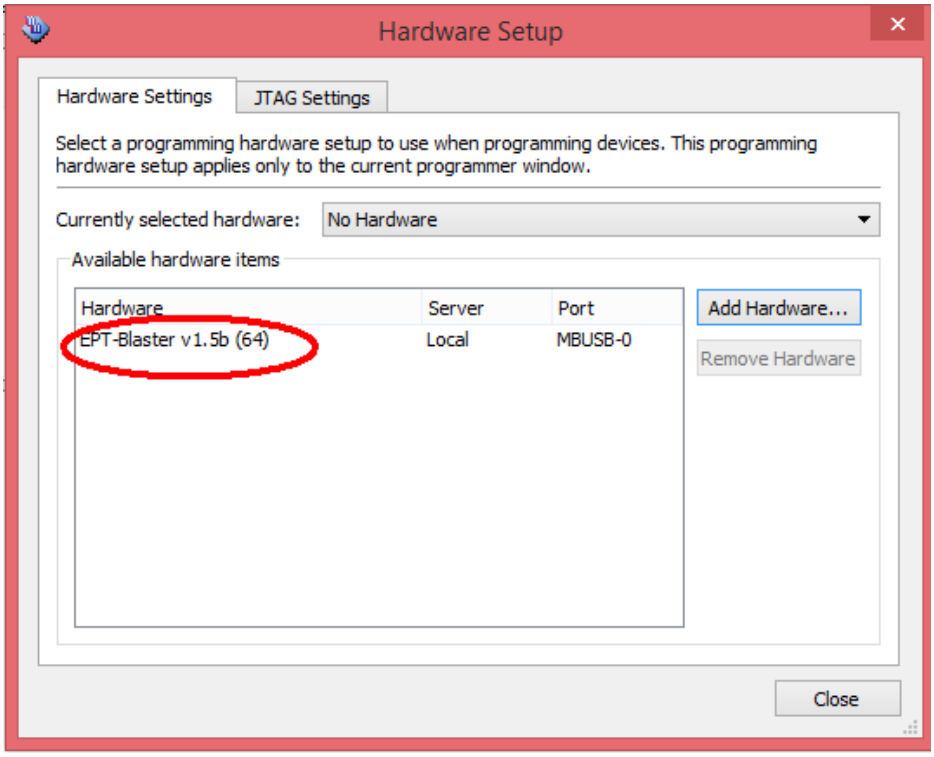
Click on the Programmer button.



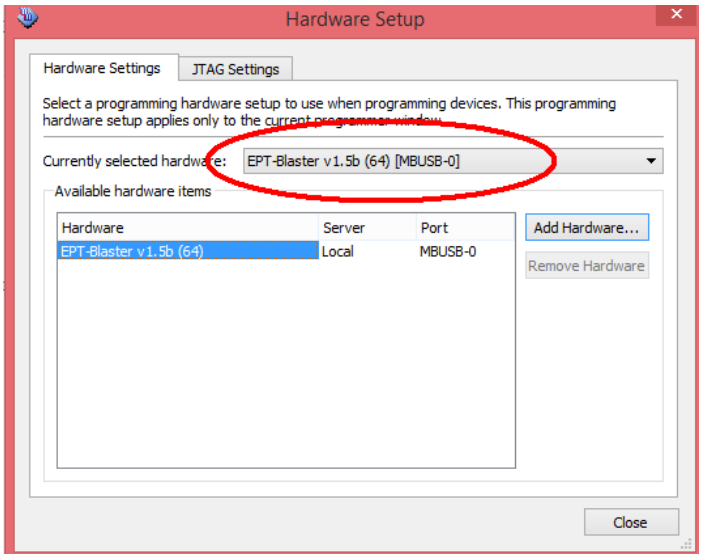
The Programmer Window will open up with the programming file selected. Click on the Hardware Setup button in the upper left corner.



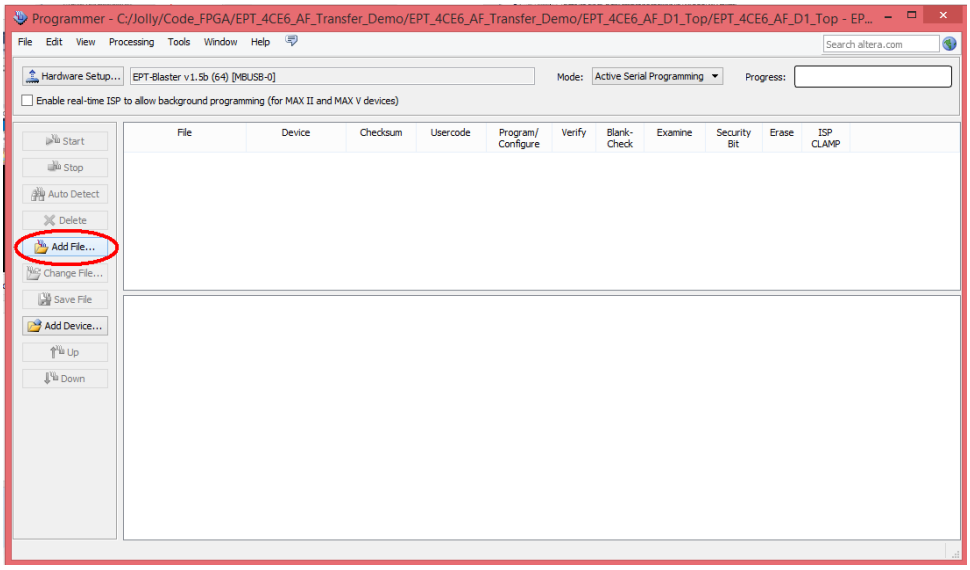
The Hardware Setup Window will open. In the “Available hardware items”, double click on “EPT-Blaster v1.6b”.

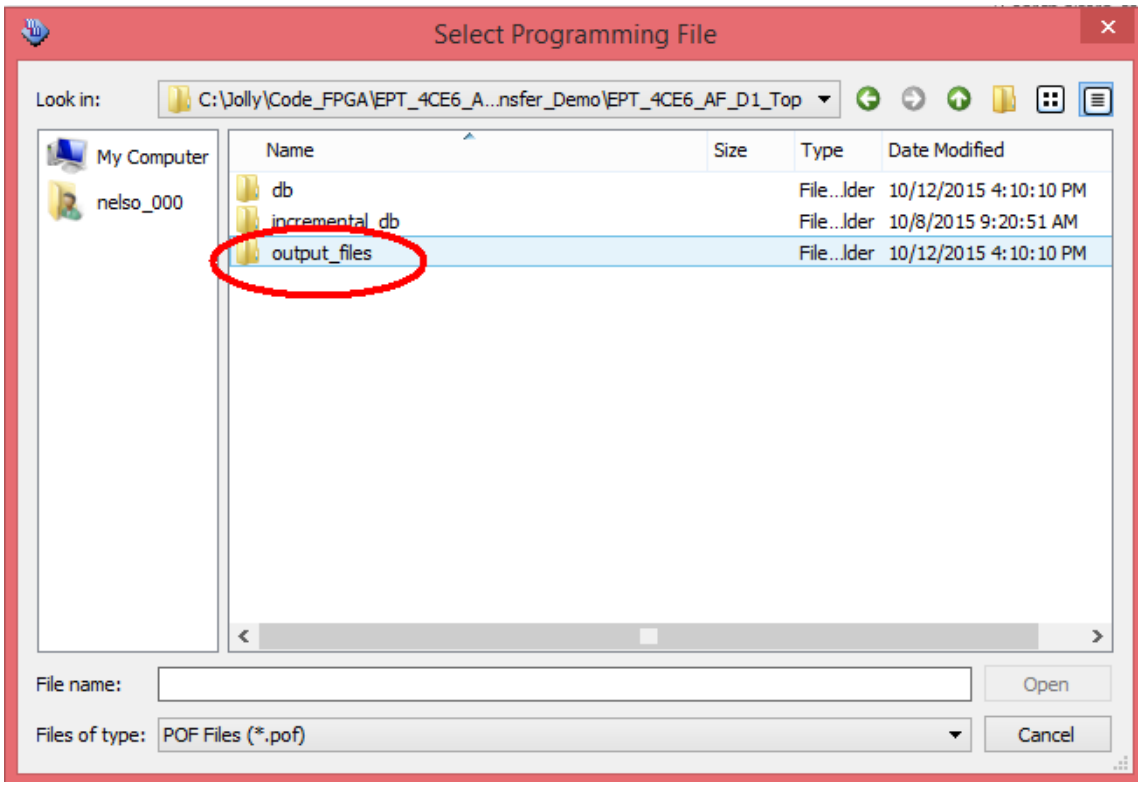


If you successfully double clicked, the “Currently selected hardware:” dropdown box will show the “EPT-Blaster v1.6b”.

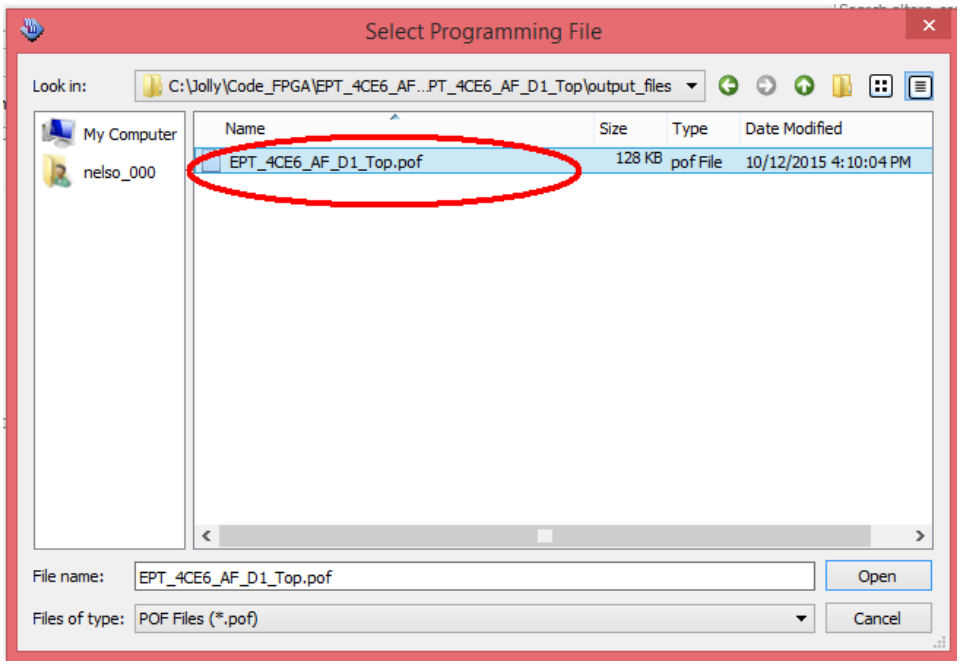


Click on the “Add File” button

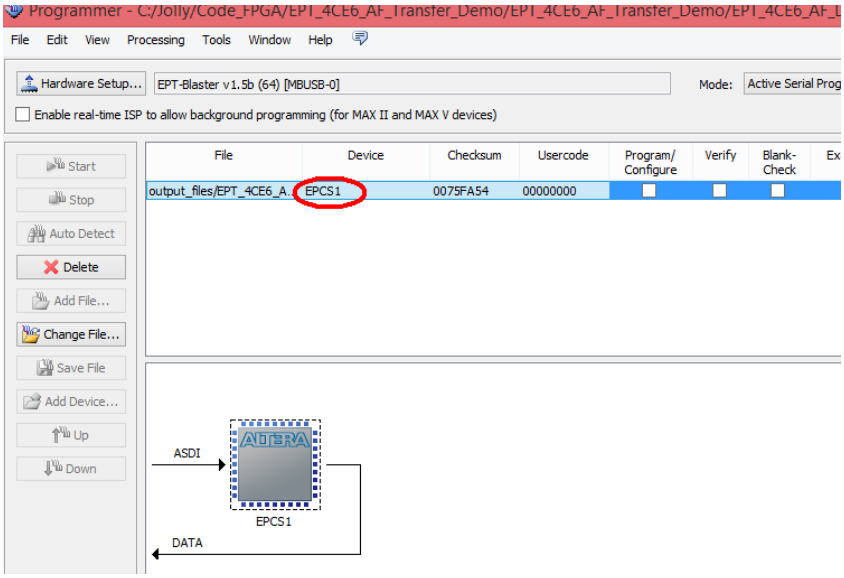




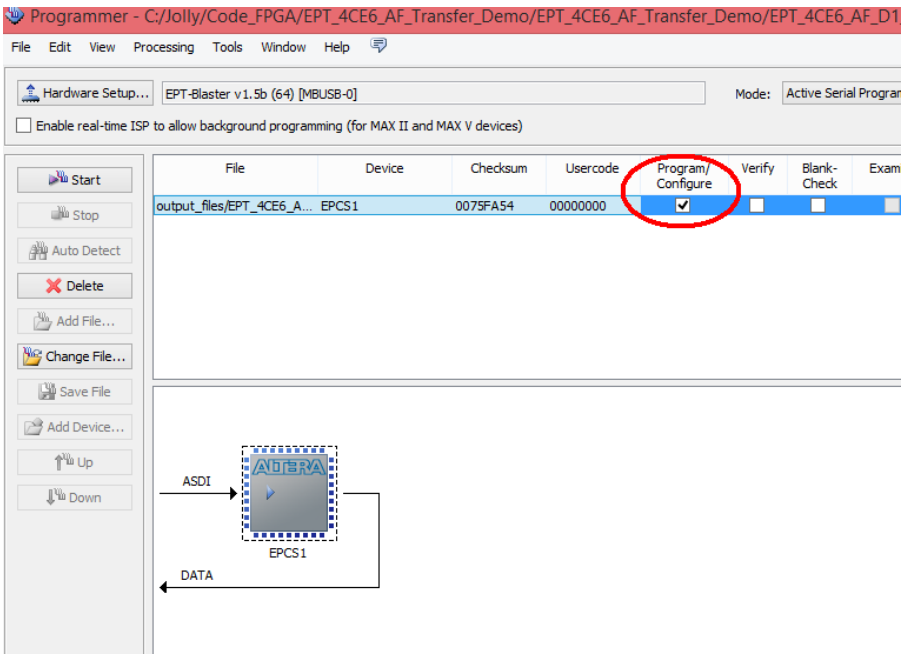
At the Browse window, double click on the output files folder.



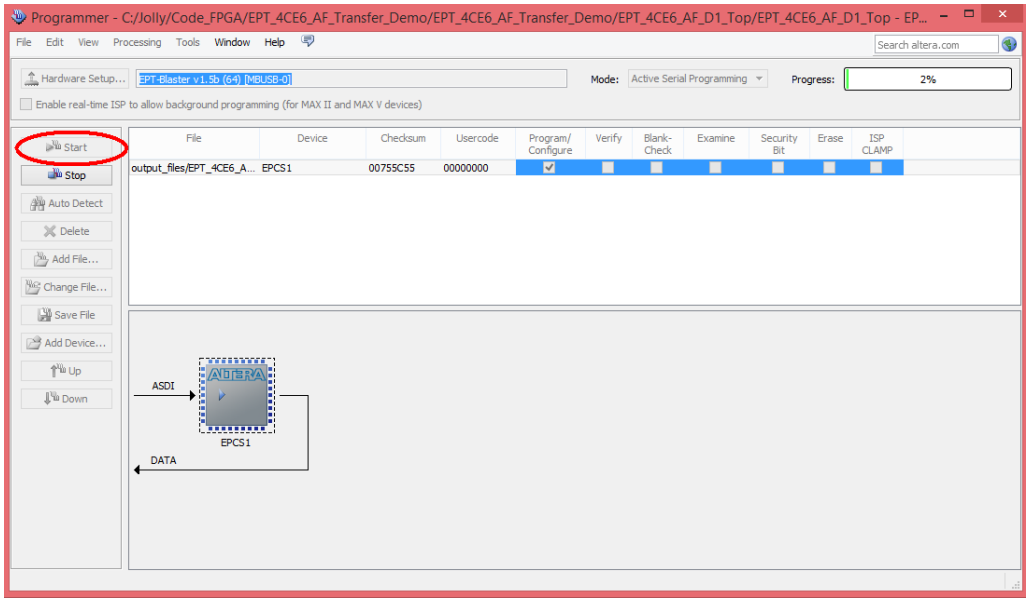
Double click on the “EPT_10M04_AF_Top.pof” file. Click the Open button in the lower right corner.



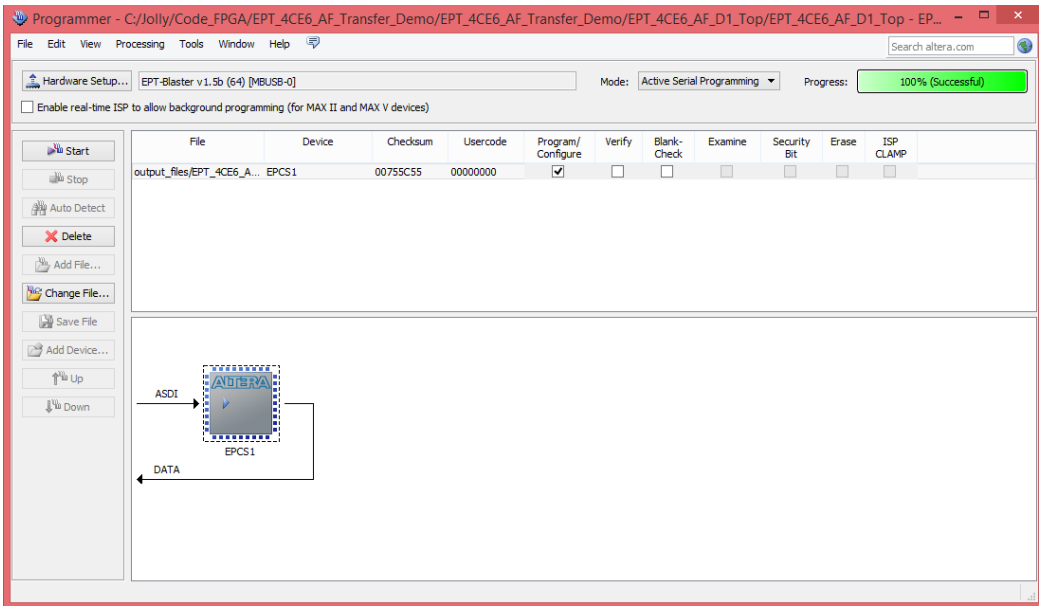
Next, selet the checkbox under the “Program/Configure” of the Programmer Tool.



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



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



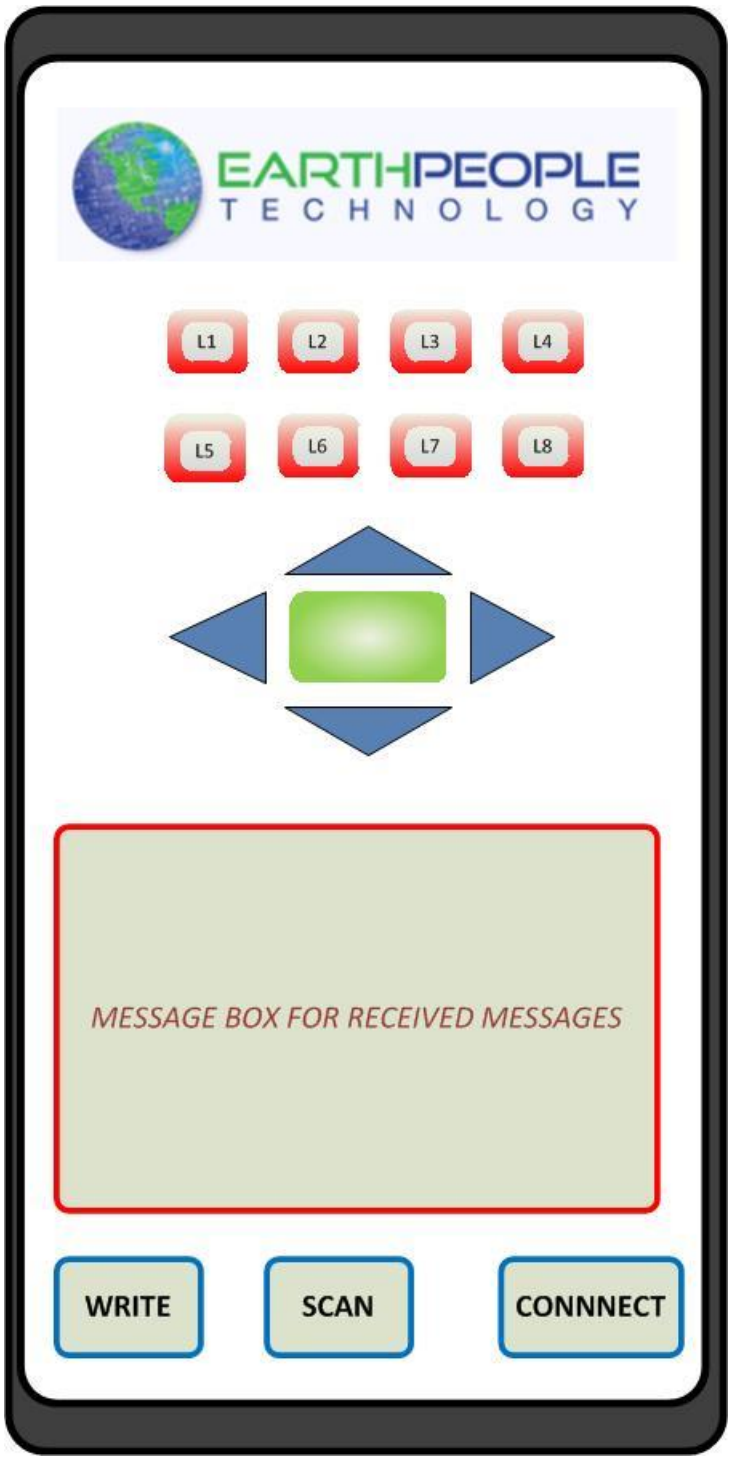
At this point, the MAXPROLOGIC is programmed and ready for use.

1.1.6 Using the Phone App

Ensure the Joystick_v-1.0.apk is installed on your Android phone. Ensure the Odin-Link BLE Board is connected to the MaxProLogic J5 with both Pin 1's aligned between the boards. Ensure the power is connected to the MaxProLogic. And, ensure the MAX10 is programmed with the EPT_10M04_AF_BLE_Driver project.

Now, you are ready to use the app and observe the functioning of the project. Go to your Android apps and locate the JoyStick app. Click on the icon.

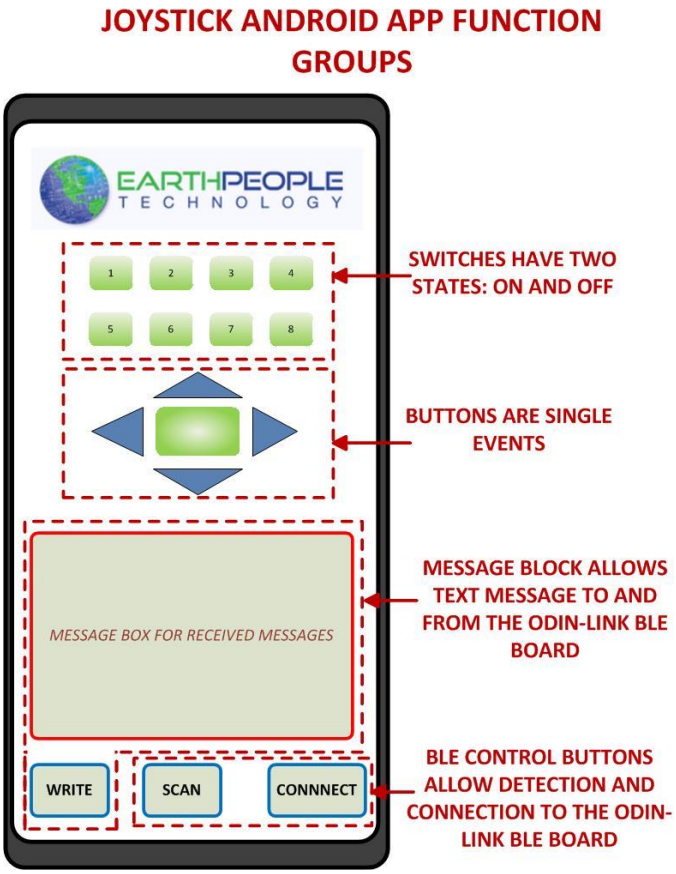
The Joystick app will open and display on the screen.



Take note that there are four groups of functions on the JoyStick Android app:

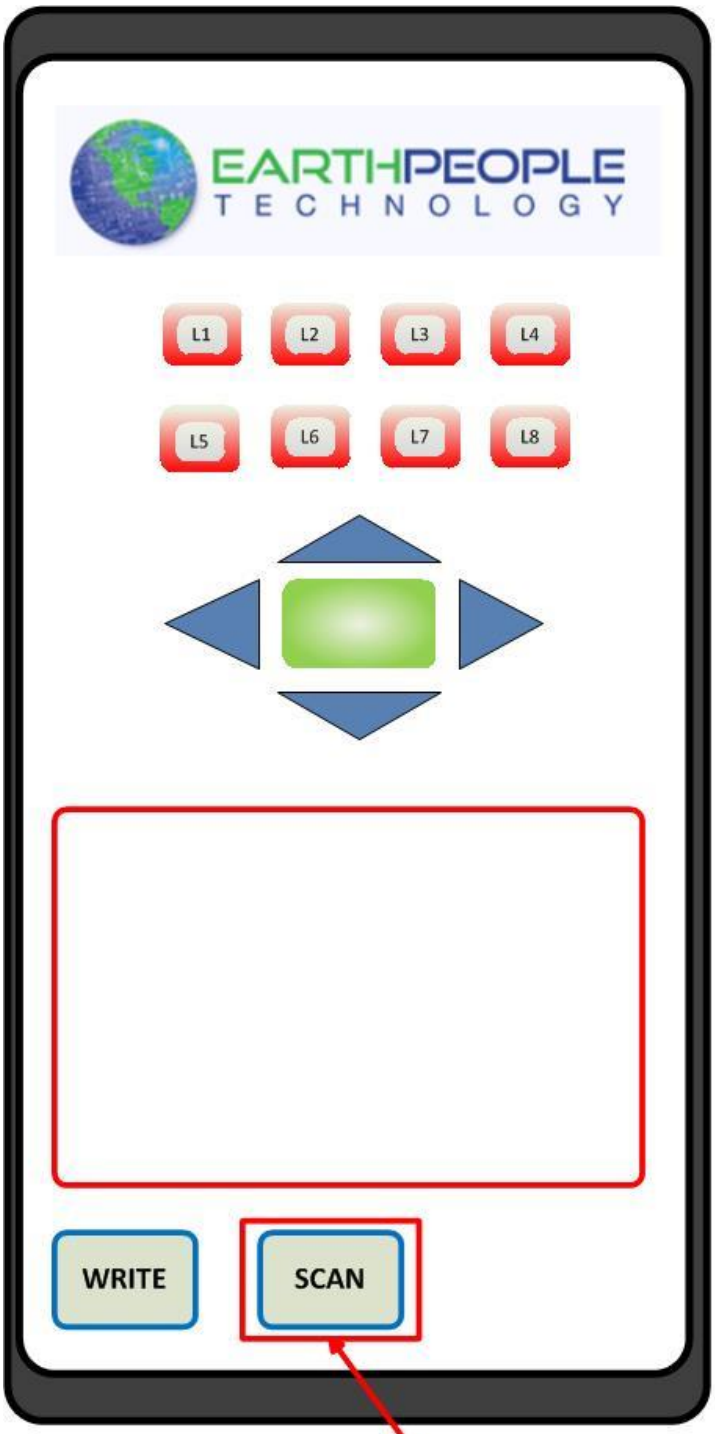
- Switches have two states: on and off

- Buttons are single events
- Message block allows text message to and from the Odin-Link Ble board
- BLE Control buttons allow detection and connection to the Odin-Link BLE board



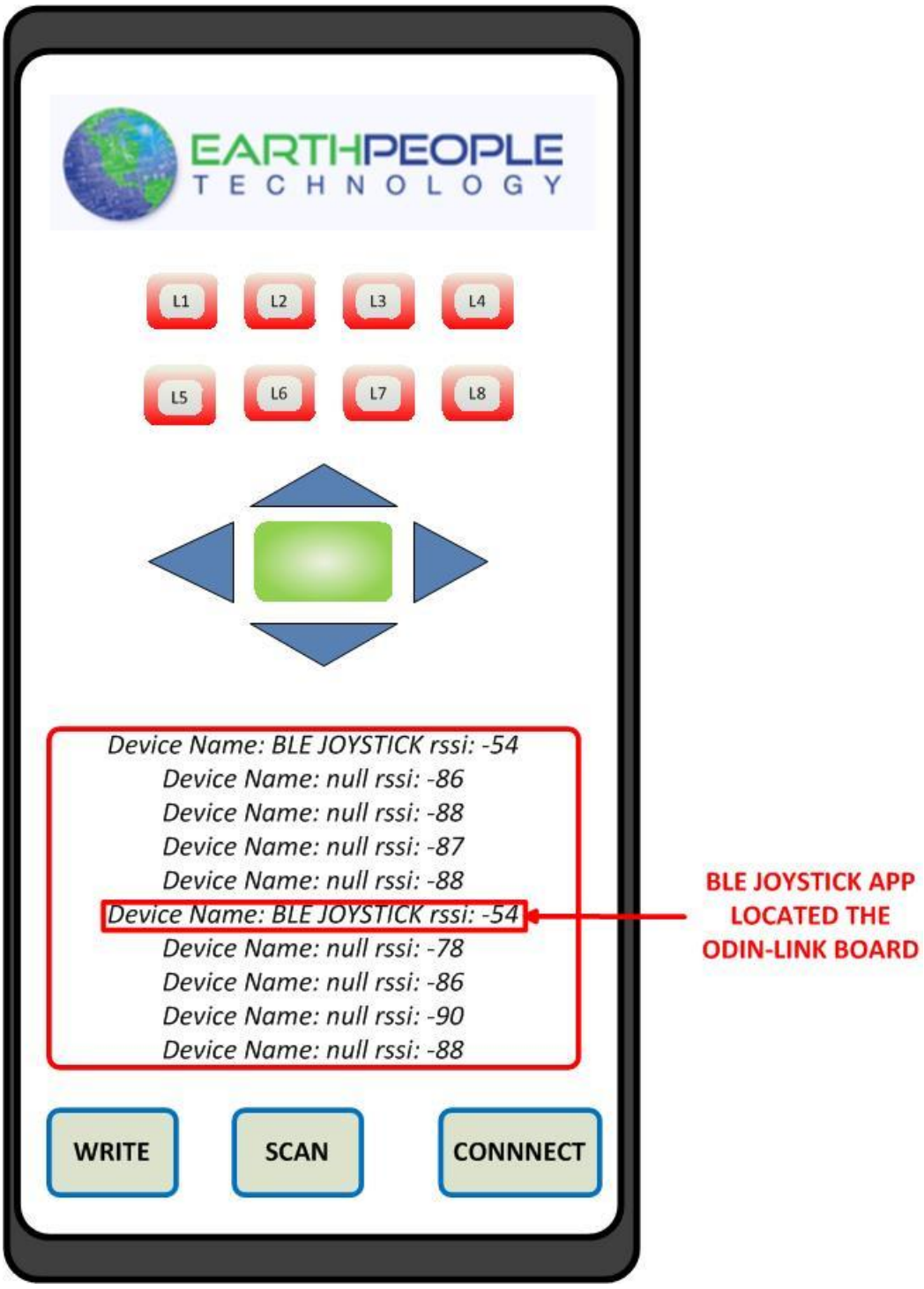
1.1.6.1 Connecting the JoyStick app to the Odin-Link BLE Board

Ensure the MAX10 is programmed with the EPT_10M04_AF_BLE_Driver project and the power is applied to the MaxProLogic. When the power is applied to the MaxProLogic, all LEDs will blink twice. Next, you will pair the Android phone with the Odin-Link BLE Board. This is accomplished by pressing the “Scan” button first.


EARTHPEOPLE
TECHNOLOGY
L1
L2
L3
L4
L5
L6
L7
L8
WRITE
SCAN
PRESS "SCAN" BUTTON TO DETERMINE NEARBY DEVICES IN ADVERTISING MODE

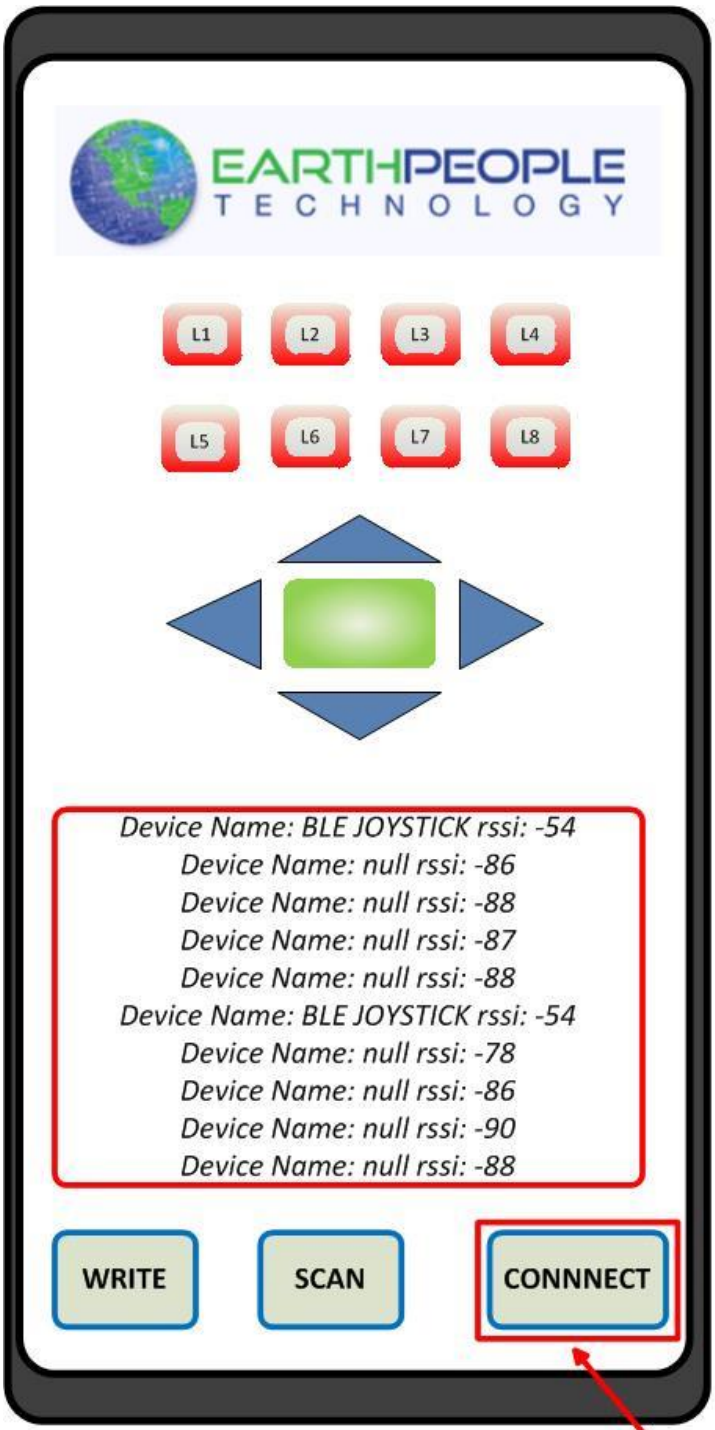
ADD PICTURE OF SCAN SCREEN WITH LOCATED DEVICES

Examine the text screen and verify that the app has located the Odin-Link BLE Board.



ADD PICTURE OF SCAN SCREEN WITH ODIN-LINK BOARD DISCOVERED

Press the “Connect” button. If the Android phone is within the BLE Advertising range of the Odin-Link, it will automatically connect.



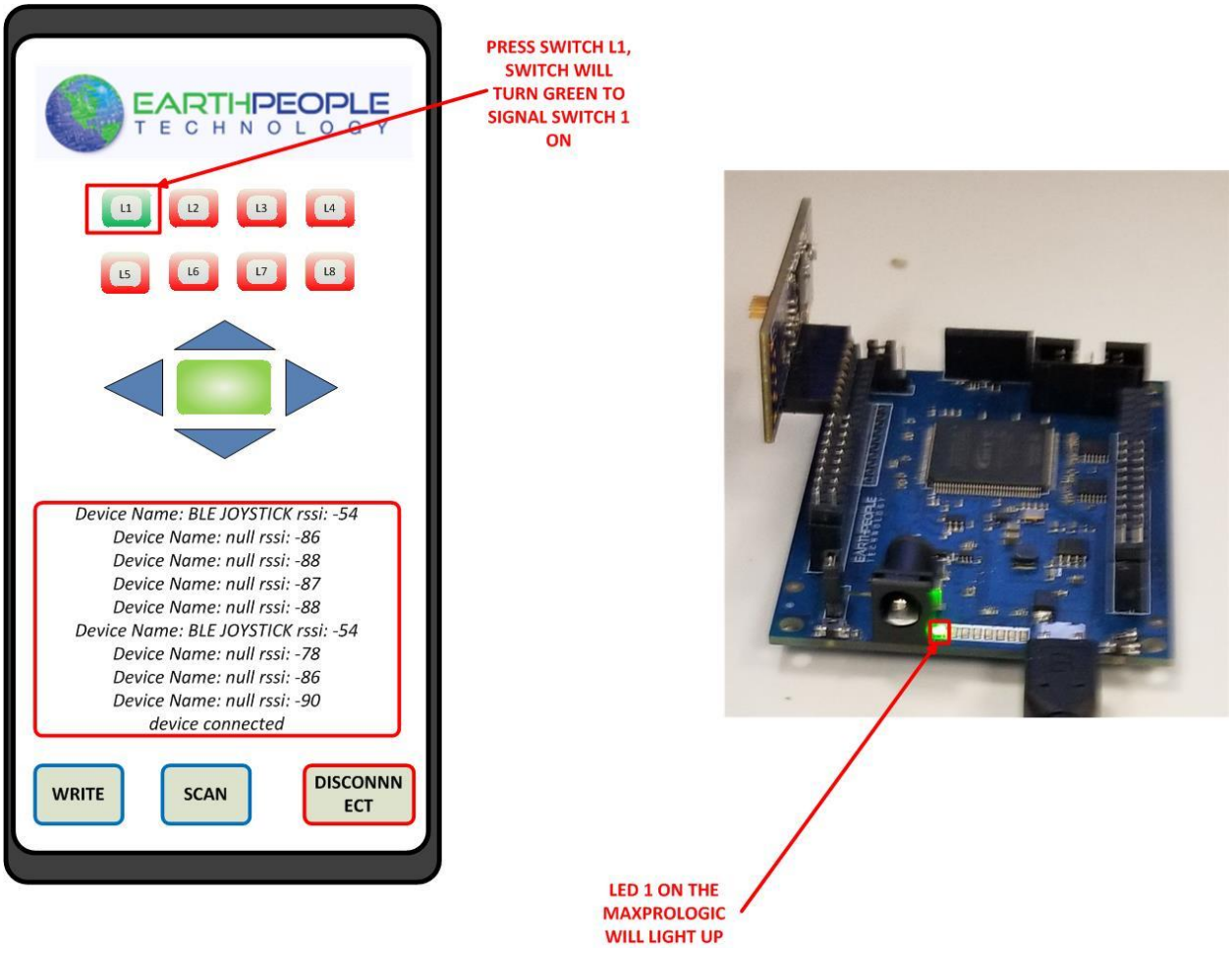
PRESS THE "CONNECT" BUTTON TO CONNECT TO THE ODIN-LINK BLE BOARD

ADD PICTURE OF SCAN SCREEN WITH MESSAGE “DEVICE CONNECTED”

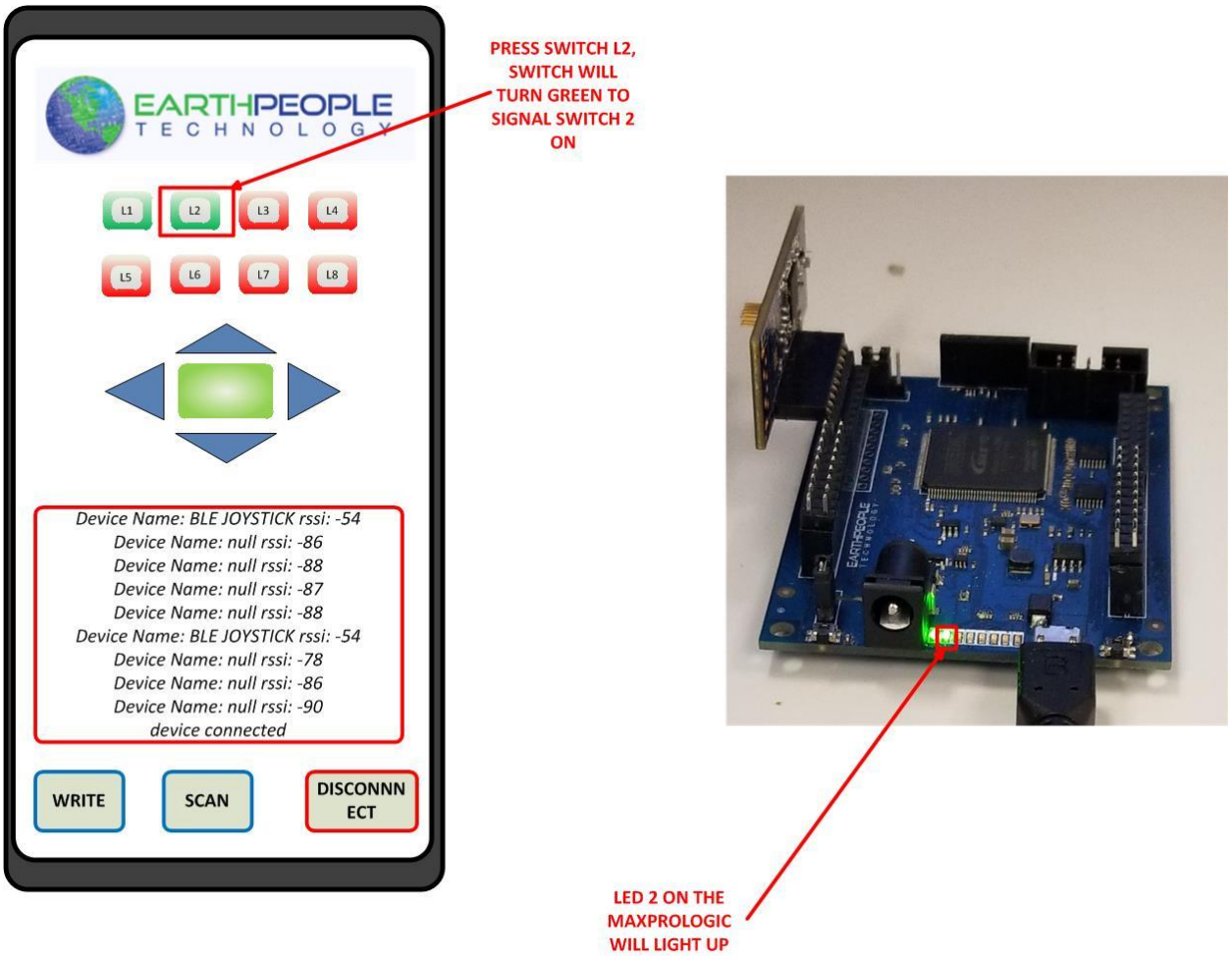
At this point the Android phone JoyStick app is connected with the Odin-Link BLE Board. Next, we will cover the operation of the functions

1.1.6.2 Using the Switch Functions

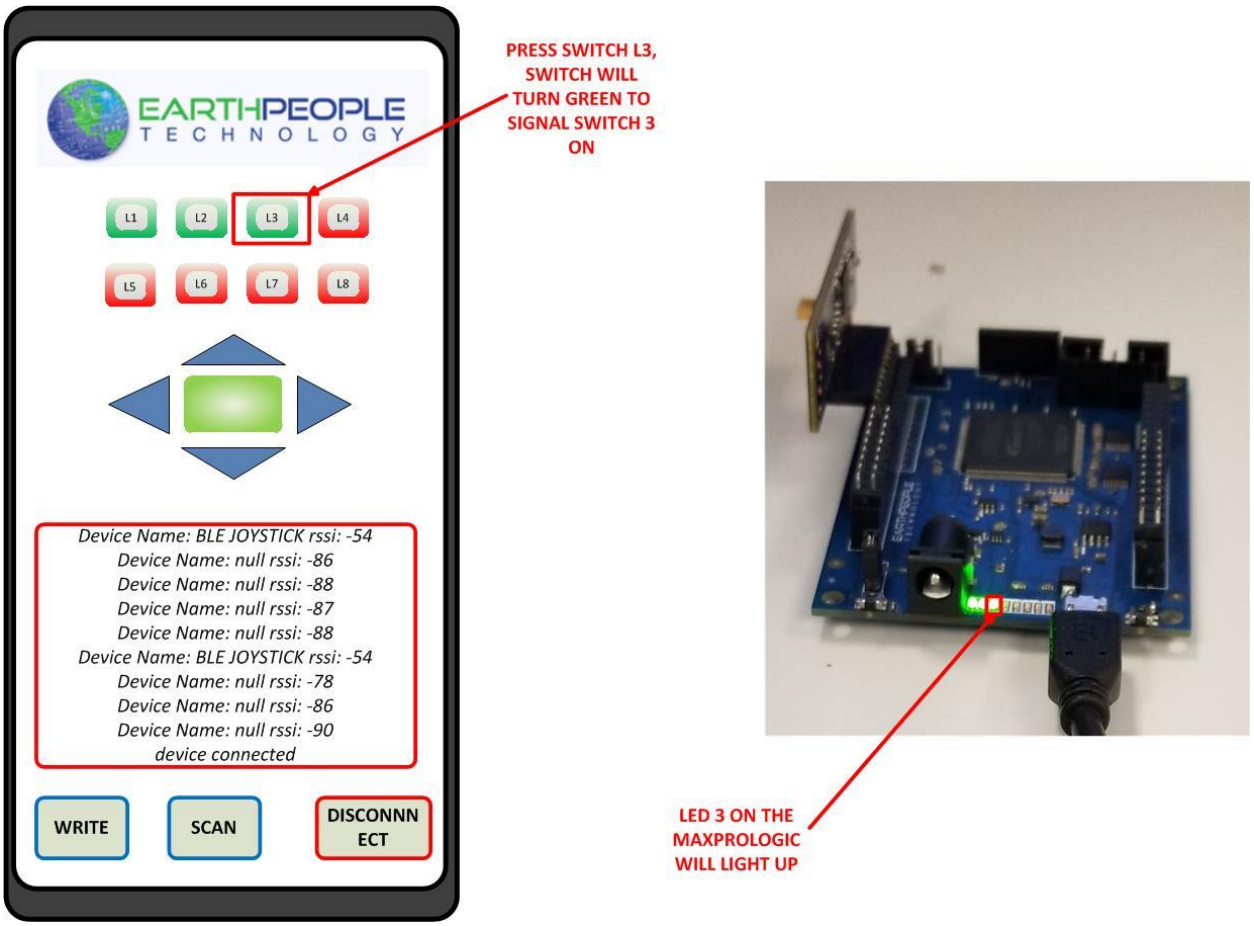
The BLE JoyStick app is easy and intuitive to use. The switches are dual function use device. They operate similar to a light switch. Press once to turn the switch on, press again to turn the switch off. When a switch is turned on, it changes color to green in the JoyStick app. When the switch is turned off, it changes color to red in the JoyStick app. For the Switch Functions, press the L1 button on the app to light the LED 1 on the MaxProLogic Board.



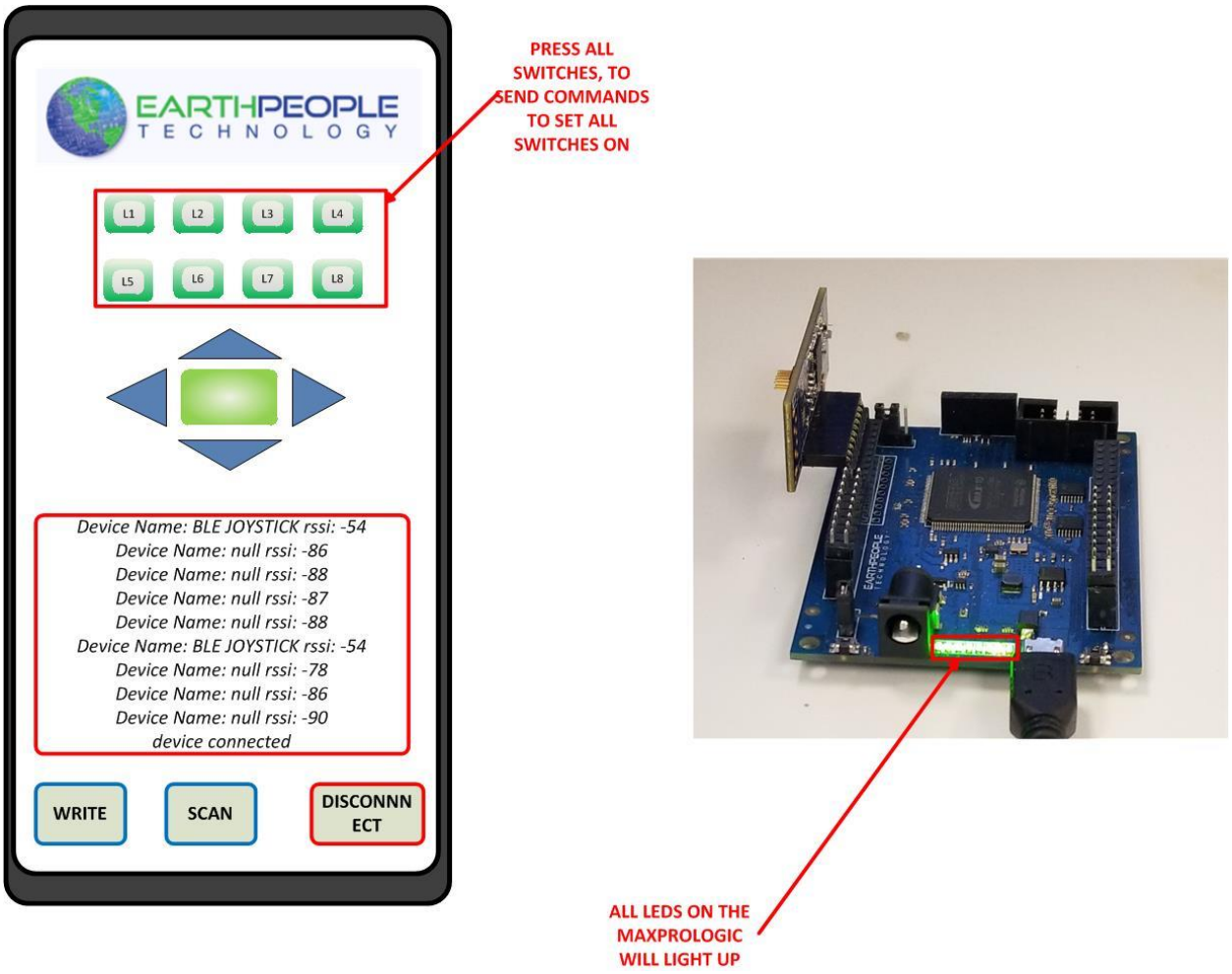
Press the L2 button on the app to light the LED 2. The previous LEDs retain their state during the current switch update.



Press L3 to light the LED 3 on the MaxProLogic board.

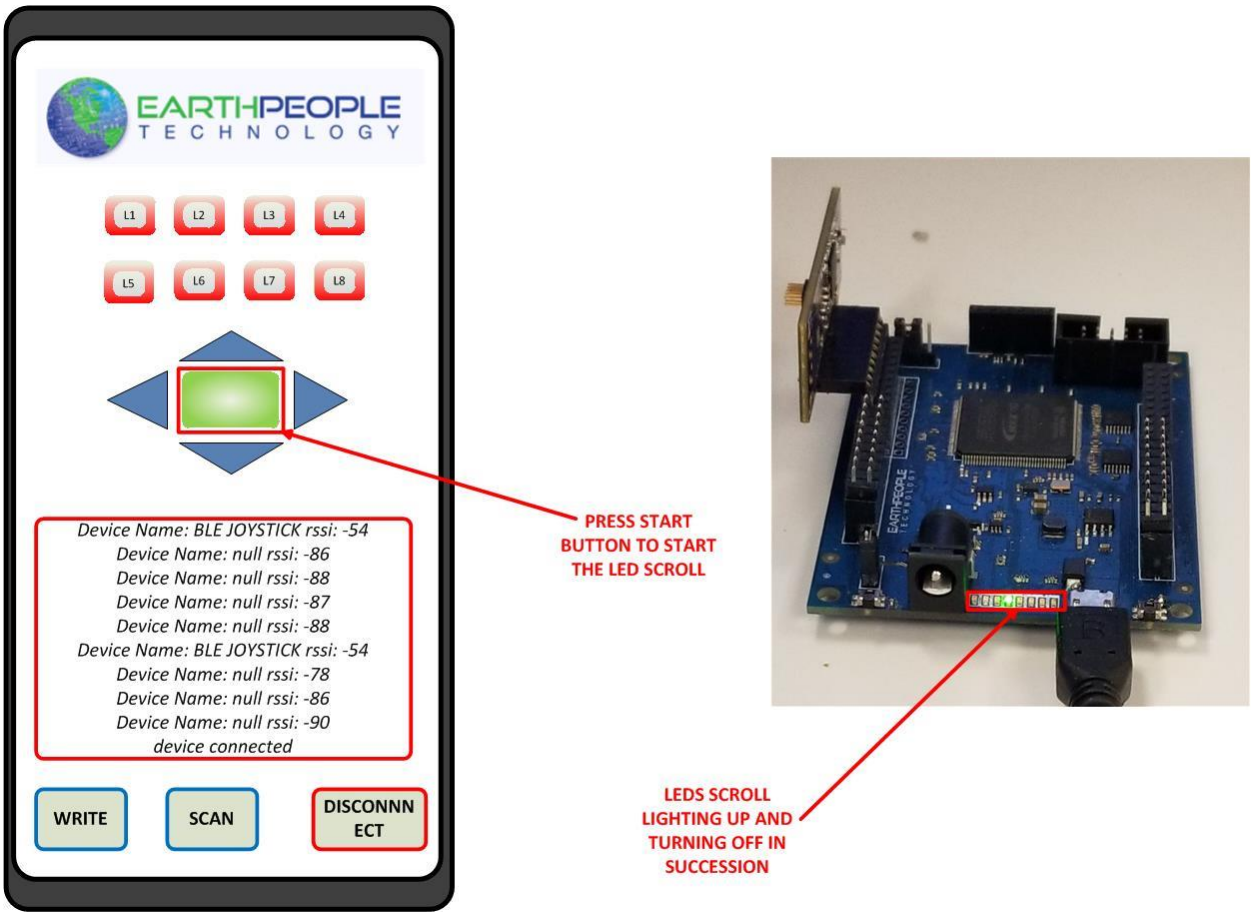


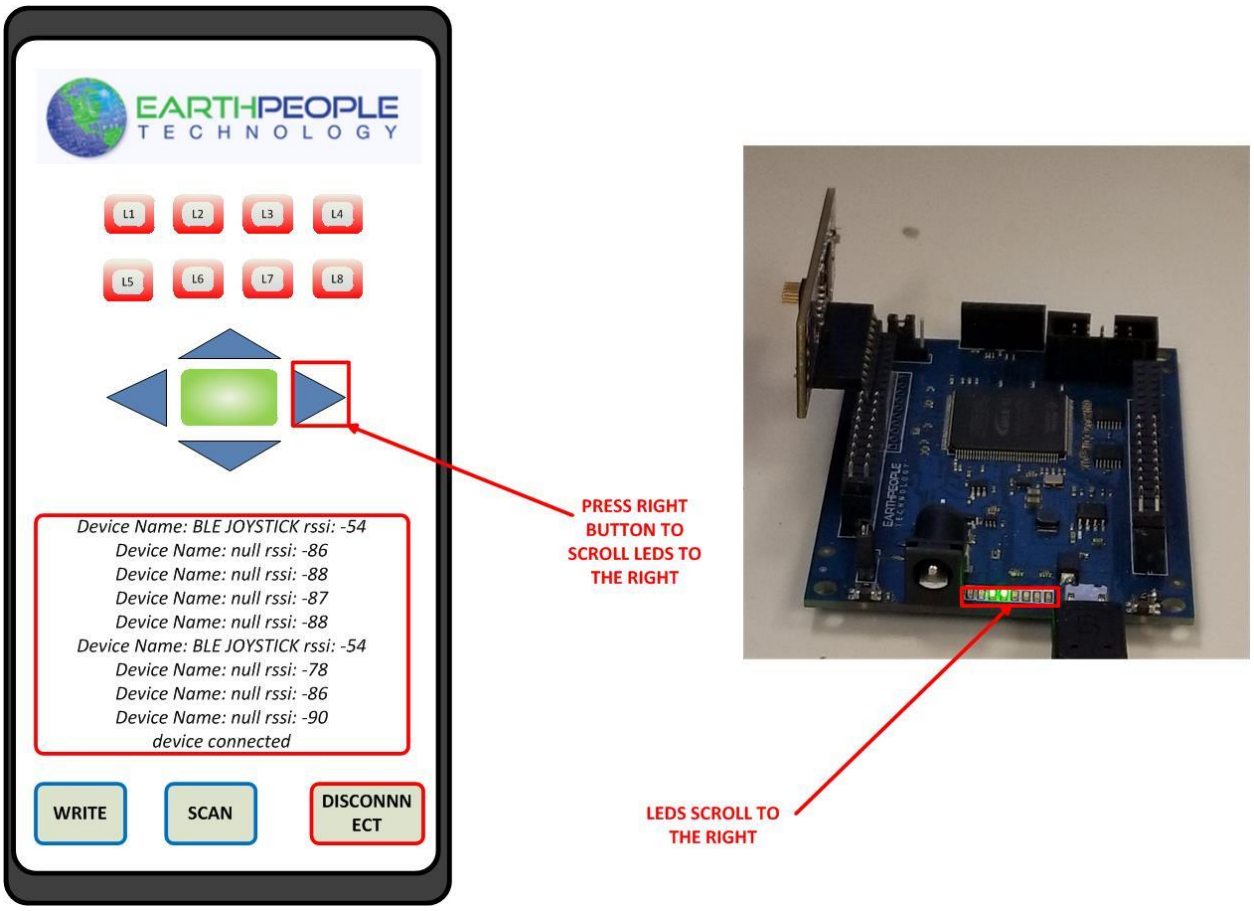
Press the remaining L4 through L8 buttons to light all LEDs.

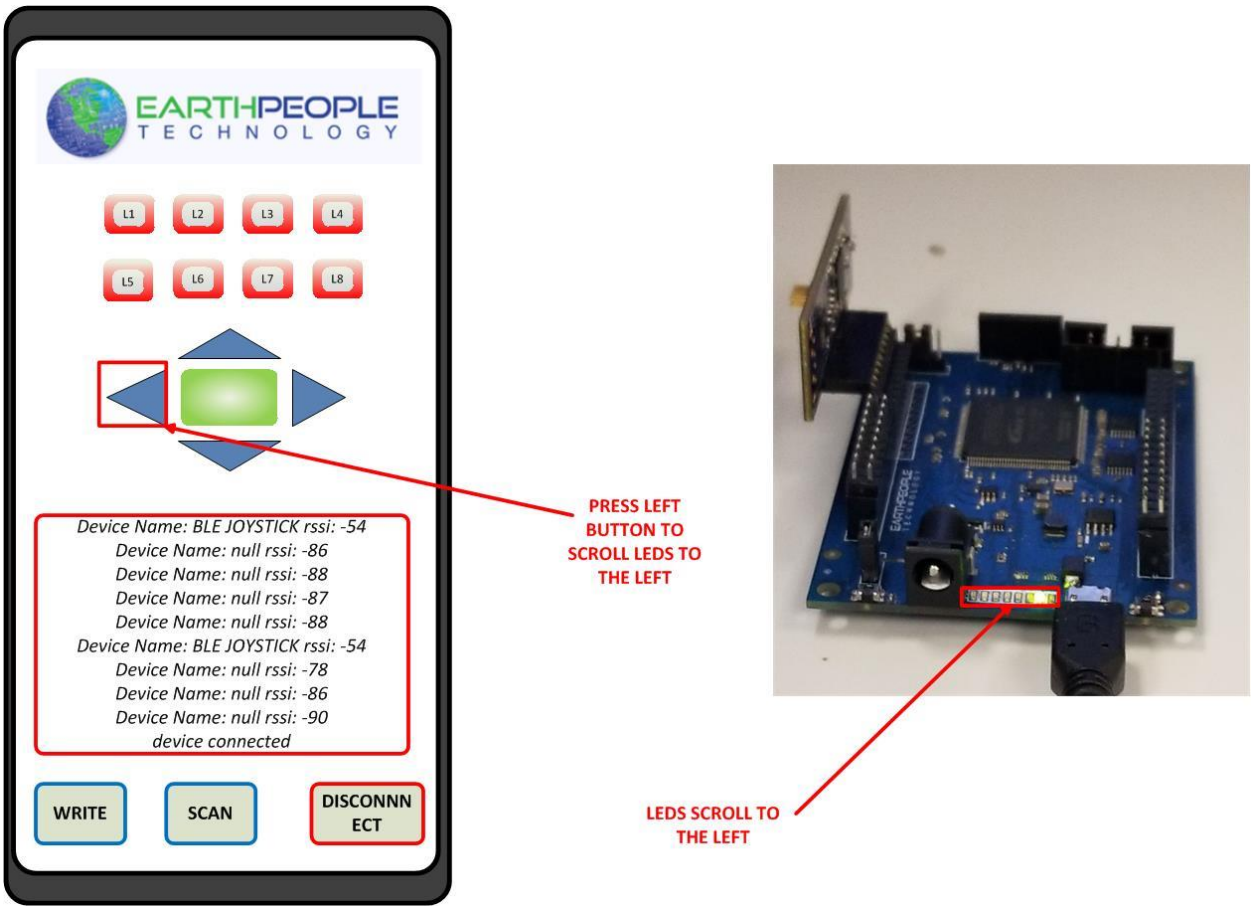


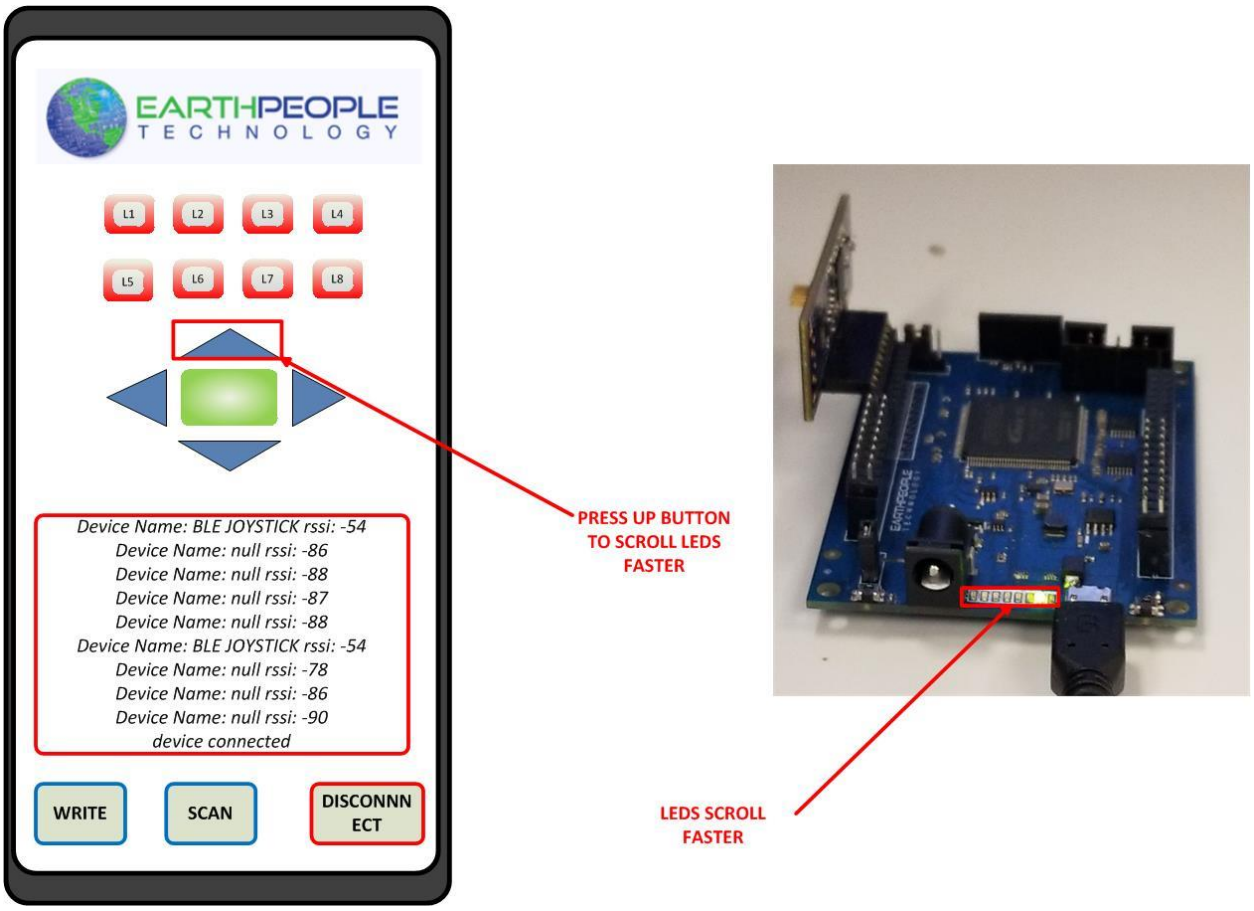
1.1.6.3 Using the Button Functions

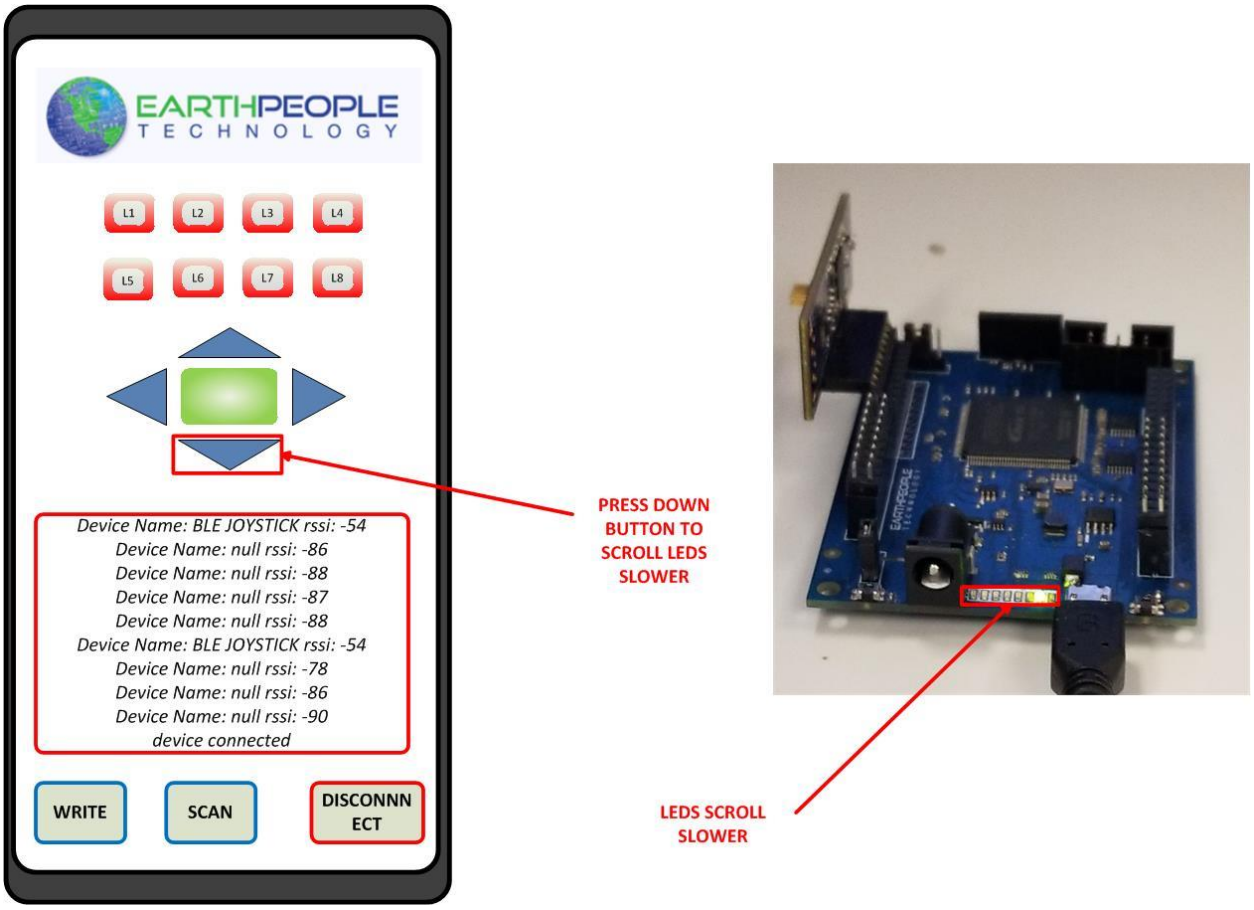
The Button Functions operate as momentary switches. Press to send a command to the Odin-Link BLE Board to indicate a button was pressed. The BLE Driver Demo program has code that will perform a Blinky LED light show based on the button pressed.


EARTHPEOPLE
TECHNOLOGY
L1
L2
L3
L4
L5
L6
L7
L8
Device Name: BLE JOYSTICK rssi: -54
Device Name: null rssi: -86
Device Name: null rssi: -88
Device Name: null rssi: -87
Device Name: null rssi: -88
Device Name: BLE JOYSTICK rssi: -54
Device Name: null rssi: -78
Device Name: null rssi: -86
Device Name: null rssi: -90
device connected
WRITE
SCAN
DISCONNN ECT
PRESS START BUTTON TO START THE LED SCROLL
LEDS SCROLL LIGHTING UP AND TURNING OFF IN SUCCESSION


EARTHPEOPLE
TECHNOLOGY
L1
L2
L3
L4
L5
L6
L7
L8
Device Name: BLE JOYSTICK rssi: -54
Device Name: null rssi: -86
Device Name: null rssi: -88
Device Name: null rssi: -87
Device Name: null rssi: -88
Device Name: BLE JOYSTICK rssi: -54
Device Name: null rssi: -78
Device Name: null rssi: -86
Device Name: null rssi: -90
device connected
WRITE
SCAN
DISCONNN ECT
PRESS RIGHT BUTTON TO SCROLL LEDS TO THE RIGHT
LEDS SCROLL TO THE RIGHT


EARTHPEOPLE
TECHNOLOGY
L1
L2
L3
L4
L5
L6
L7
L8
Device Name: BLE JOYSTICK rssi: -54
Device Name: null rssi: -86
Device Name: null rssi: -88
Device Name: null rssi: -87
Device Name: null rssi: -88
Device Name: BLE JOYSTICK rssi: -54
Device Name: null rssi: -78
Device Name: null rssi: -86
Device Name: null rssi: -90
device connected
WRITE
SCAN
DISCONNN ECT
PRESS LEFT BUTTON TO SCROLL LEDS TO THE LEFT
LEDS SCROLL TO THE LEFT

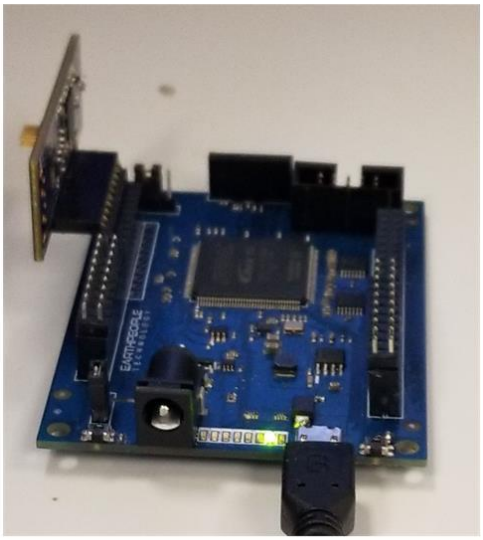

EARTHPEOPLE
TECHNOLOGY
L1
L2
L3
L4
L5
L6
L7
L8
Device Name: BLE JOYSTICK rssi: -54
Device Name: null rssi: -86
Device Name: null rssi: -88
Device Name: null rssi: -87
Device Name: null rssi: -88
Device Name: BLE JOYSTICK rssi: -54
Device Name: null rssi: -78
Device Name: null rssi: -86
Device Name: null rssi: -90
device connected
WRITE
SCAN
DISCONNN ECT
PRESS UP BUTTON TO SCROLL LEDS FASTER
LEDS SCROLL FASTER

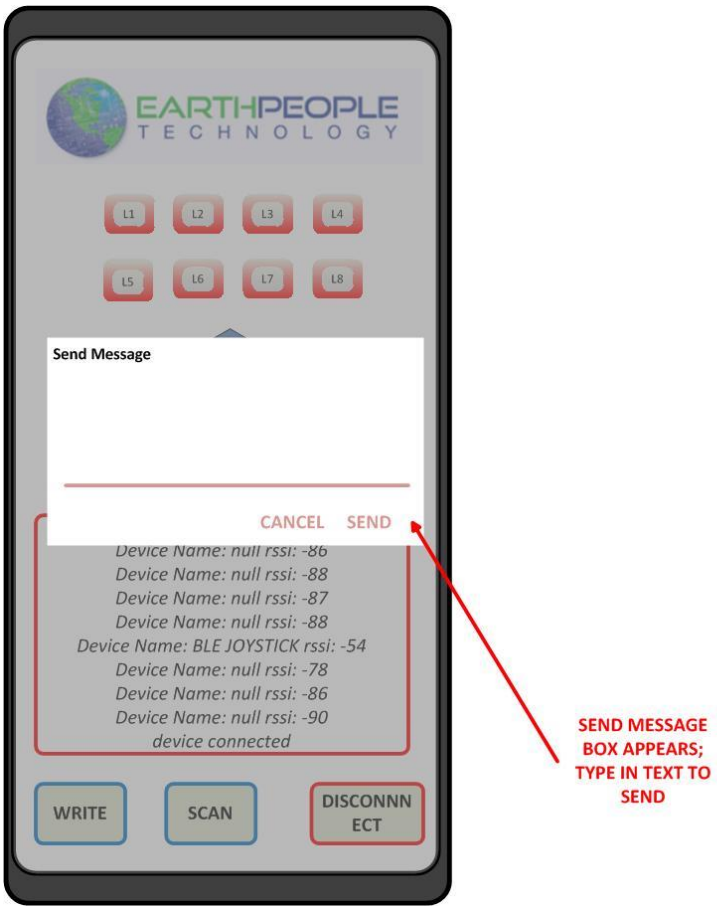


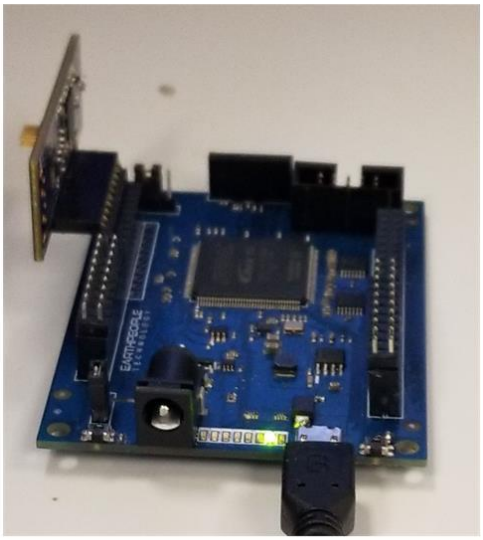
1.1.6.4 Using the Text Write/Receive Functions

The JoyStick app allows the user to send and receive text. The Text window will display any text received from the Odin-Link BLE Board. To write, use the “WRITE” button.


EARTHPEOPLE
TECHNOLOGY
L1
L2
L3
L4
L5
L6
L7
L8
Device Name: BLE JOYSTICK rssi: -54
Device Name: null rssi: -86
Device Name: null rssi: -88
Device Name: null rssi: -87
Device Name: null rssi: -88
Device Name: BLE JOYSTICK rssi: -54
Device Name: null rssi: -78
Device Name: null rssi: -86
Device Name: null rssi: -90
device connected
WRITE
SCAN
DISCONNN ECT
PRESS WRITE BUTTON TO WRITE TEXT

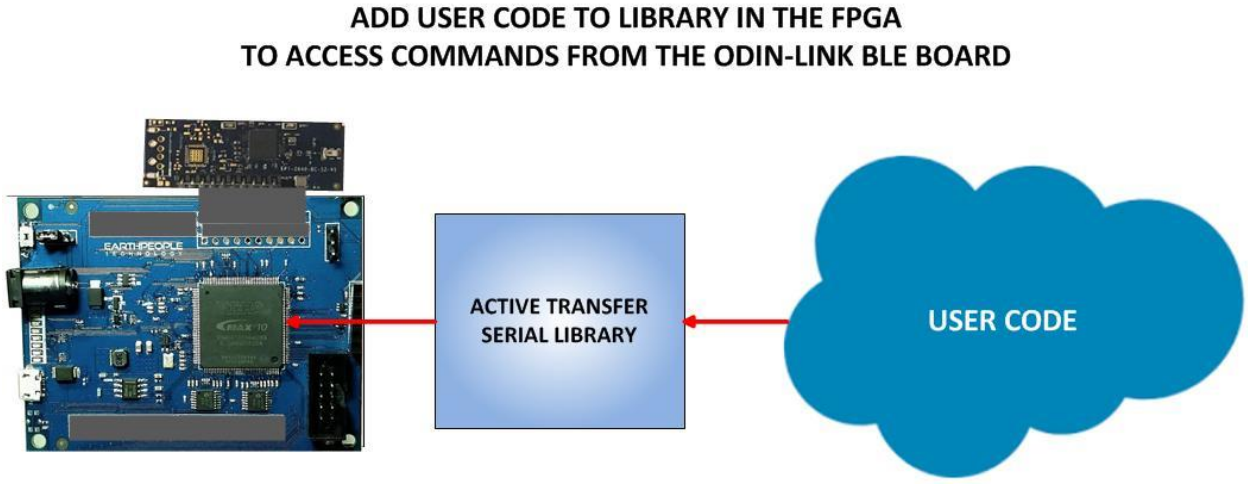






2 The Active Transfer Serial Verilog Library

The EPT_10M04_AF_BLE_Driver is a project that is designed to demonstrate the functionality of the Odin-Link BLE Board and the FPGA operation. There is a Verilog library called the Active_Transfer_serial. This library performs all of the serial decoding, storage, command decipher, and user interface control needed to receive commands from the Odin-Link BLE Board.



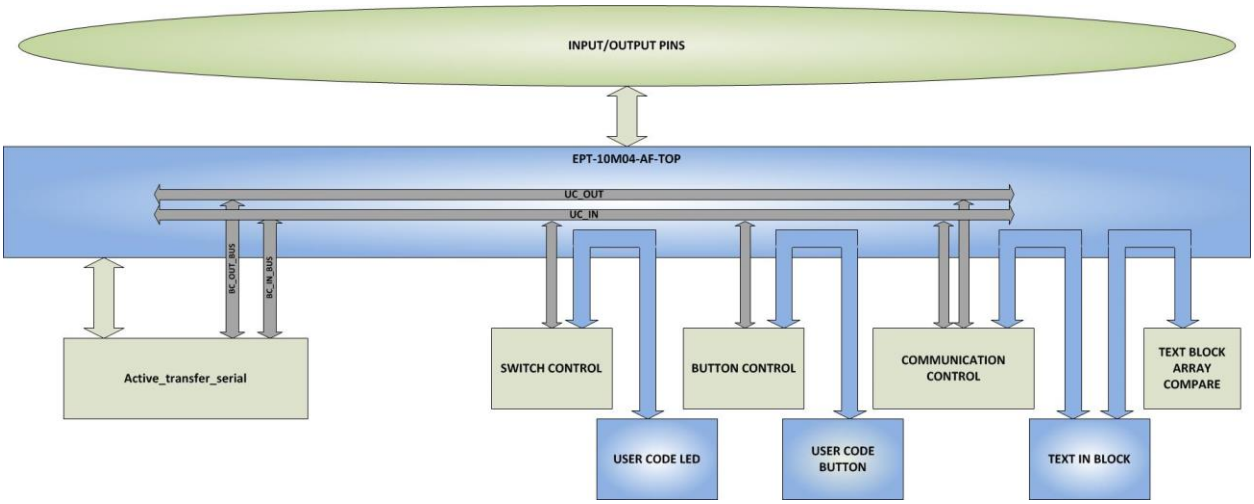
The user will include the Active_Transfer_Serial library in the project along with the user code. Compile and Synthesize the project using the Quartus Software Tool to produce a program file for download into the FPGA.

2.1 User Interface

The Active_Serial_Transfer library has a relatively easy interface for user code. The interface is designed for flexibility and ease of use. It is written in Verilog and uses the 50MHz oscillator. It consists of four modules that access the user code.

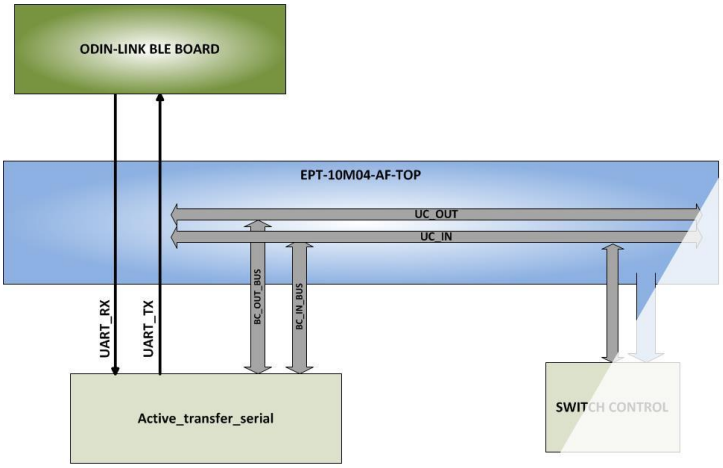
- Active_Transfer_Serial.v
- Switch_Command.v
- Button_Command.v
- Communication_Command.v
- Text_Block_Array_Compare.v

The Android app will send commands from the phone and these commands get converted into signals in the Verilog code interface. The user code will create a leaf instantiation to access the signals of each of these modules. The Block Diagram below shows how the library is connected to the user project. The beige blocks comprise the Active_Serial_Transfer library. The Blue and Green blocks show the user code. The following sections explain how to connect and use the Active_Serial_Transfer library with the user code.



2.1.1 Active_Serial_Transfer

The Active_Serial_Transfer.v block connects the Odin-Link BLE board to the library. It provides the path from Command decode to the control blocks. The connection is provided by the UC bus. This bus consists of two busses the UC_OUT and UC_IN. The UC must must be declared in the user code and requires a Wired-OR gate for the UC_IN bus.



The Verilog code interface must be added to the user project as follows:

```
//------------------------------------------------
// Active Transfer Serial Library Instantiation
//------------------------------------------------

active_transfer_serial        ACTIVE_TRANSFER_SERIAL_INST
(
  .CLK_50MHZ                 (CLK_50MHZ),
  .RST_N                     (RST_N),

  .UART_OUT                  (XIO4_6), //To external UART
  .UART_IN                   (XIO4_7), //From external UART

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


 );

//------------------------------------------------
```

You must add the

- Clock – CLK_50MHZ
- Reset – RST_N
- UART TX – XIO4_6 (This connects with the XIO4 connector pin 6)
- UART RX – XOI4_7 (This connects with the XIO4 connector pin 7)
- UC in bus – UC_IN
- UC out bus – UC_OUT

See the exact source code for definitions of the clock, reset, UART in and out.

Next, we need to add a Wired-OR gate for the UC_IN bus. This is required as each command module (such as Switch_Command.v, Button_Command.v) has active input to the Serial_Transfer_Library.v block. The Wired-OR gate on the UC_IN bus acts as a multiplexor to allow each module to communicate with the library but without the control statements needed for a multiplexor.

```
    //------------------------------------------------
    // Instantiate the EPT Active Modules
    //------------------------------------------------
wire [23*3-1:0]  uc_in_m;
eptWireOR # (.N(3)) wireOR (UC_IN, uc_in_m);

    switch_control              LED_CONTROL_INST
    (
     .UC_CLK                    (CLK_50MHZ),
     .UC_RESET                  (RST_N),
     .UC_IN                     (uc_in_m[ 0*23 +: 23 ]),
     .UC_OUT                    (UC_OUT),

     .START_TRANSFER            (),
     .COMMAND_RECIEVED          (led_command),

     .TRANSFER_BUSY             (),

     .SWITCH_ON                 (leds_on),
     .SWITCH_OFF                (leds_off)

    );
```

The red box highlights the Wired-OR gate for the UC_IN bus. The UC_IN bus is declared as a 23 bit bus in the “Internal Signals and Registers Declarations” section.

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

    //Finite State Machine control registers
    reg     [6:0]               state, next;

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


    //UC Buses
    wire   [22:0]               UC_OUT;
    wire   [21:0]               UC_IN;

    //LEDs
    wire   [7:0]                leds;
    wire   [7:0]                leds_on;
    wire   [7:0]                leds_off;
```

The register uc_in_m must declared as a bus that is 23*N-1:0 bits wide. Where “N” is the number of attached modules. If you wished to just attach the Switch_Command module to handle the switch commands from the JoyStick Android app, then set N=1. If you would like Switch_Command and Button_Command modules attached, then set N=2. When using “uc_in_m” to connect the UC_IN bus to a module, you must add the section of the multiplexed bus to the module. The example:

```
    //------------------------------------------------
    // Instantiate the EPT Active Modules
    //------------------------------------------------
wire [23*3-1:0]  uc_in_m;
eptWireOR # (.N(3)) wireOR (UC_IN, uc_in_m);

    switch_control              LED_CONTROL_INST
    (
     .UC_CLK                    (CLK_50MHZ),
     .UC_RESET                  (RST_N),
     .UC_IN                     (uc_in_m[ 0*23 +: 23 ]),
     .UC_OUT                    (UC_OUT),

     .START_TRANSFER            (),
     .COMMAND_RECIEVED          (led_command),

     .TRANSFER_BUSY             (),

     .SWITCH_ON                 (leds_on),
     .SWITCH_OFF                (leds_off)

    );
```

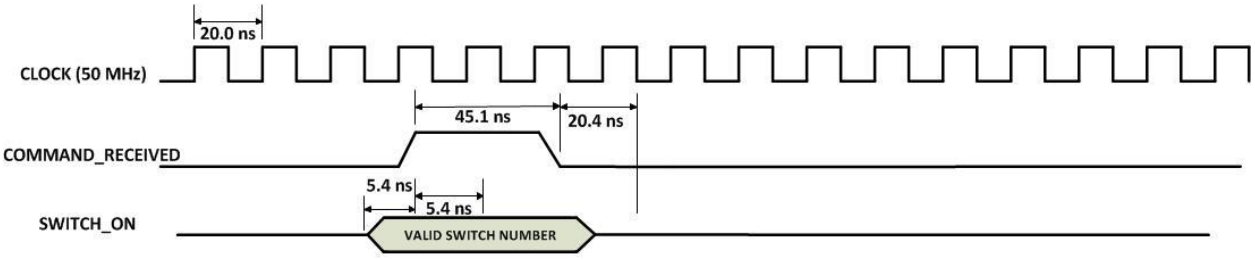
Add “uc_in_m[N*23 +: 23]” to the UC_IN bus instantiation. Where N = 0, 1, 2 etc. The order in which the modules are added is irrelevant. The Button_Command could be added before Switch_Command or Communication_Command before both of the modules. The eptWireOR module will multiplex the UC_IN bus accurately. Ensure that each module has a unique “N” number in the “uc_in_m[] instantiation.

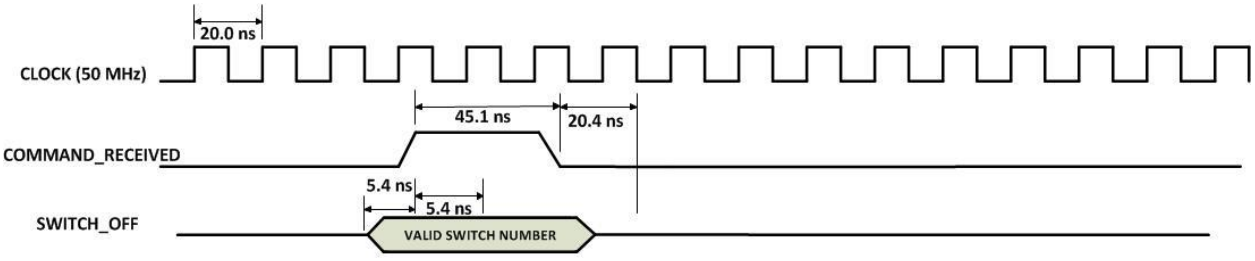
2.1.2 Switch_Command

The Switch Command from the JoyStick Android app has eight separate callable switches. These switch commands are either on or off. This is similar to the light switch in most houses. Flip the switch to the on position to apply power to the light. Flip it to the off position to turn the power to the light off. The Switch Command Verilog module interface connects easily to the user code. Add the uc_in_m[] bus and UC_OUT bus declarations in the leaf instantiation to connect the module to the Active_Serial_Transfer Library.

```
switch_control              LED_CONTROL_INST
(
 .UC_CLK                    (CLK_50MHZ),
 .UC_RESET                  (RST_N),
 .UC_IN                     (uc_in_m[ 0*23 +: 23 ]),
 .UC_OUT                    (UC_OUT),

 .START_TRANSFER            (),
 .COMMAND_RECIEVED          (led_command),

 .TRANSFER_BUSY             (),

 .SWITCH_ON                 (leds_on),
 .SWITCH_OFF                (leds_off)

);
```

The user code must send the clock and reset to the module. The “COMMAND_RECEIVED” leaf instantiation signal is used to communicate to the user code when command has been received from the JoyStick Android app. The actual command will be either in the “SWITCH_ON” or “SWITCH_OFF” leaf register. These two leaf registers are eight bits and correspond to the same eight switches.





The “COMMAND_RECEIVED” signal will be used in the user code to start a process to read the “SWITCH_ON” or “SWITCH_OFF” registers to determine what function occurred and what process to start because of it.

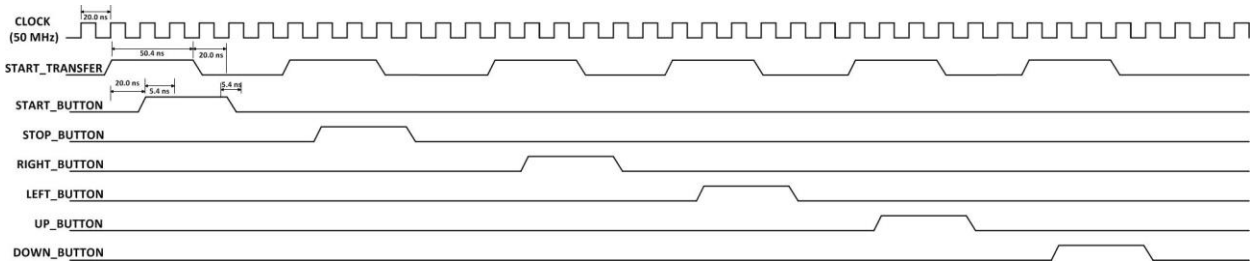
```
if(led_command)
begin
    if(leds_on != 0)
    begin
        control_register[7:4] <= STATIC_VALUE;
        led_device_transfer_byte <= (led_device_transfer_byte | leds_on);
    end
    else if(leds_off != 0)
    begin
        control_register[7:4] <= STATIC_VALUE;
        led_device_transfer_byte <= (led_device_transfer_byte & ~leds_off);
    end
    trigger_in_byte[0] <= 1'b1;
end
else if(trigger_in_byte[0])
    trigger_in_byte[0] <= 1'b0;
```

In this block of code, an if statement is used to determine when the “COMMAND_RECEIVED” signal is asserted. The nested if-else if statements are used to determine which function was commanded. The subsequent statements are used to grab which LEDS to turn on of off.

2.1.3 Button_Command

The Button Command Verilog module interface connects easily to the user code. Add the uc_in_m[] bus and UC_OUT bus declarations in the leaf instantiation to connect the module to the Active_Serial_Transfer Library.

```
button_control              BUTTON_CONTROL_INST
(
 .UC_CLK                    (CLK_50MHZ),
 .UC_RESET                  (RST_N),
 .UC_IN                     (uc_in_m[ 1*23 +: 23 ]),
 .UC_OUT                    (UC_OUT),

 .START_TRANSFER            (),
 .BUTTON_RECIEVED           (button_command),

 .TRANSFER_BUSY             (),

 .UP_BUTTON                 (up_button_pressed),
 .DOWN_BUTTON               (down_button_pressed),
 .RIGHT_BUTTON              (right_button_pressed),
 .LEFT_BUTTON               (left_button_pressed),
 .START_BUTTON              (start_button_pressed),
 .STOP_BUTTON               (stop_button_pressed)
);
```



When a command has been received and correctly parsed by the MaxProLogic Demo Code, the "button_control" module will assert the "button_command" signal.

```
button_control              BUTTON_CONTROL_INST
(
 .UC_CLK                    (CLK_50MHZ),
 .UC_RESET                  (RST_N),
 .UC_IN                     (uc_in_m[ 1*23 +: 23 ]),
 .UC_OUT                    (UC_OUT),

 .START_TRANSFER            (),
 .BUTTON_RECIEVED           (button_command),

 .TRANSFER_BUSY             (),

 .UP_BUTTON                 (up_button_pressed),
 .DOWN_BUTTON               (down_button_pressed),
 .RIGHT_BUTTON              (right_button_pressed),
 .LEFT_BUTTON               (left_button_pressed),
 .START_BUTTON              (start_button_pressed),
 .STOP_BUTTON               (stop_button_pressed)
);
```

Along with the “button_command” assertion, one of the button signals will assert.

```
button_control              BUTTON_CONTROL_INST
(
 .UC_CLK                    (CLK_50MHZ),
 .UC_RESET                  (RST_N),
 .UC_IN                     (uc_in_m[ 1*23 +: 23 ]),
 .UC_OUT                    (UC_OUT),

 .START_TRANSFER            (),
 .BUTTON_RECIEVED           (button_command),

 .TRANSFER_BUSY             (),

 .UP_BUTTON                 (up_button_pressed),
 .DOWN_BUTTON               (down_button_pressed),
 .RIGHT_BUTTON              (right_button_pressed),
 .LEFT_BUTTON               (left_button_pressed),
 .START_BUTTON              (start_button_pressed),
 .STOP_BUTTON               (stop_button_pressed)
);
```

The “xx_button_pressed” signal will correspond to the button that was pressed on the JoyStick app.

The user code will use the combination of “button_command” and “xx_button_pressed” signal assertions to perform some function.

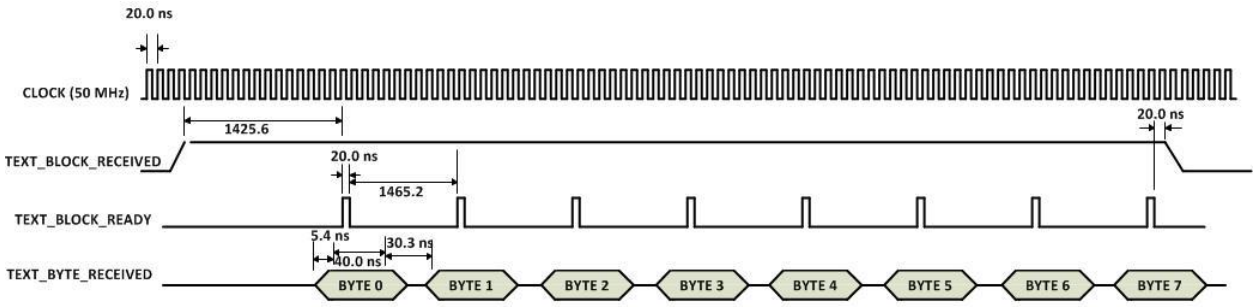
```
if(button_command)
begin
    if(start_button_pressed)
    begin
        control_register[7:4] <= SHIFT_RIGHT;
        led_device_transfer_byte <= 8'h1;
        trigger_in_byte[1] <= 1'b1;
        buttons_active_flag <= 1'b1;
    end
    else if(stop_button_pressed)
    begin
        control_register[7:4] <= STATIC_VALUE;
        led_device_transfer_byte <= 0;
        buttons_active_flag <= 1'b0;
    end

    if(buttons_active_flag)
    begin
        if(right_button_pressed)
        begin
            control_register[7:4] <= SHIFT_RIGHT;
            led_device_transfer_byte <= 8'h1;
```

In the above code snippet, a nested if-else if block of code uses the “button_command” as a method to start the process of determining what function to perform. The nested if-else if statements compare the incoming “xx_button_pressed” signal to determine which function to update the LED_Blinky light show. If the “start_button_pressed” is asserted, the “SHIFT_RIGHT” parameter is added to the “control_register”. The “control_register” is used to start the LED_Blinky light show, select the operation to perform (such as strobe left, strobe right, blink faster, blink slower).

2.1.4 Communication_Command

The Communication Command Verilog module interface connects easily to the user code. Add the uc_in_m[] bus and UC_OUT bus declarations in the leaf instantiation to connect the module to the Active_Serial_Transfer Library.

```
communication_control      COMMUNICATION_CONTROL_INST
(
 .UC_CLK                   (CLK_50MHZ),
 .UC_RESET                 (RST_N),
 .UC_IN                    (uc_in_m[ 2*23 +: 23 ]),
 .UC_OUT                   (UC_OUT),

 .START_TRANSFER           (),
 .TRANSFER_RECIEVED        (text_block_received),

 .TRANSFER_BUSY            (),
 .TRANSFER_READY           (text_block_ready),

 .EPT_LENGTH_OUT           (),
 .UC_LENGTH_IN             (text_block_length),

 .TRANSFER_TO_HOST         (),
 .TRANSFER_TO_DEVICE       (text_byte_received)
);
```



The Communication Control block is used to send a block of bytes from the JoyStick App to the MaxProLogic FPGA. The user will use the “Write” button on the app to initiate a communication text transmit. The app transmits all bytes to the Odin-Link board. The communication bytes are received by the Active_Transfer_Serial library and converted to a command block. The “communication_control” block will then provide the interface to the user code.

When the “text_block_received” signal asserts, there is a block of bytes ready to be transmitted into the user code. The “text_block_length” is used to inform the user code of the exact number of bytes to be transmitted into the user code. Use the length to as a maximum number to count in the bytes from the “communication_control” block. The user code must wait until the “text_block_ready” signal asserts to read a byte. The “text_byte_received” is the actual byte to read into user code. The “text_block_ready” is a short duration assert signal, only one clock pulse wide. The “text_block_ready” signal will only assert once during the three clock cycles that the byte to read is valid. The user code must be able to trigger on positive edge of the “text_block_ready” signal and immediately read the byte into local memory. Once all bytes in the “text_block_length” are transmitted from the Active_Transfer_library, there will be no more “text_block_ready” signal assertions and the “text_block_received” signal will de-assert.

Next, the user code must properly store the text bytes into memory and perform some function using this communication.

2.1.5 Text_Block_Array_Compare

The “communication_control” block is used primarily for users to set up their own custom commands in the MaxProLogic FPGA. The most efficient method to do this is read all bytes from the “communication_control” block into an array in the user code. Then, use the “text_block_array_compare” to determine which command to operate on.

```
//-----------------------------------------------
// Text Block Command Selection Instantiation
//-----------------------------------------------

text_block_array_compare    TEXT_BLOCK_ARRAY_COMPARE_INST
(
  .CLK                              (CLK_50MHZ),
  .RST_N                            (RST_N),

 .TEXT_BLOCK_LENGTH                 (text_block_length),
 .MEMORY_ARRAY_0                    (text_in_array[0]),
 .MEMORY_ARRAY_1                    (text_in_array[1]),
 .MEMORY_ARRAY_2                    (text_in_array[2]),
 .MEMORY_ARRAY_3                    (text_in_array[3]),
 .MEMORY_ARRAY_4                    (text_in_array[4]),
 .MEMORY_ARRAY_5                    (text_in_array[5]),
 .MEMORY_ARRAY_6                    (text_in_array[6]),
 .MEMORY_ARRAY_7                    (text_in_array[7]),
 .MEMORY_ARRAY_8                    (text_in_array[8]),
 .MEMORY_ARRAY_9                    (text_in_array[9]),
 .MEMORY_ARRAY_10                   (text_in_array[10]),
 .MEMORY_ARRAY_11                   (text_in_array[11]),
 .MEMORY_ARRAY_12                   (text_in_array[12]),
 .MEMORY_ARRAY_13                   (text_in_array[13]),
 .MEMORY_ARRAY_14                   (text_in_array[14]),
 .MEMORY_ARRAY_15                   (text_in_array[15]),
 .MEMORY_ARRAY_16                   (text_in_array[16]),
 .MEMORY_ARRAY_17                   (text_in_array[17]),
 .MEMORY_ARRAY_18                   (text_in_array[18]),
 .MEMORY_ARRAY_19                   (text_in_array[19]),
 .MEMORY_ARRAY_20                   (text_in_array[20]),
 .MEMORY_ARRAY_21                   (text_in_array[21]),
 .MEMORY_ARRAY_22                   (text_in_array[22]),
 .MEMORY_ARRAY_23                   (text_in_array[23]),
 .MEMORY_ARRAY_24                   (text_in_array[24]),
 .MEMORY_ARRAY_25                   (text_in_array[25]),
 .MEMORY_ARRAY_26                   (text_in_array[26]),
 .MEMORY_ARRAY_27                   (text_in_array[27]),
 .MEMORY_ARRAY_28                   (text_in_array[28]),
 .MEMORY_ARRAY_29                   (text_in_array[29]),
 .MEMORY_ARRAY_30                   (text_in_array[30]),
 .MEMORY_ARRAY_31                   (text_in_array[31]),
 .MEMORY_ARRAY_32                   (text_in_array[32]),
 .COMMAND_SELECTION                 (command_selection)
```

The “text_block_array_compare” block will accept the text block from the “communication_control” block from lowest byte

- MEMORY_ARRAY_0 == text_in_array[0]

To

- MEMORY_ARRAY_32 == text_in_array[32]

Because of the limitations of Verilog, the text array that was captured during the communication receive has to pass each byte into the leaf module individually. So, the “MEMORY_ARRAY_xx” registers in the “text_block_array_compare” block store each byte. The “TEXT_BLOCK_LENGTH” is required to match the number of bytes that will be compared. As you can see, the text block command is limited to 33 bytes in length. Once all bytes are added to the leaf module interface, the result of the compare is immediate. The “COMMAND_SELECTION” will return a byte that correlates with a pre-determined user command.

The “text_block_array_compare” block requires the user to add a custom command text array in which to compare. This text array custom command is stored in the file “user_define.v”. The user will open the file and add the ASCII equivalent Hex byte for each character that will be sent from the JoyStick app. Order of the bytes is very important. The text message will start with at the lowest number byte, 0.

```
//Custom Text Array; Add each byte to an individual define
   //Set Temperature Data Interval to 5 sec
`define  TEXT_BLOCK_5_BYTE_22 = 8'h00;//Communication End of Block
`define  TEXT_BLOCK_5_BYTE_21 = 8'h0a;//Communication End of Block
`define  TEXT_BLOCK_5_BYTE_20 = 8'h0d;//Communication End of Block
`define  TEXT_BLOCK_5_BYTE_19 = 8'h20;//Space
`define  TEXT_BLOCK_5_BYTE_18 = 8'h35;//5
`define  TEXT_BLOCK_5_BYTE_15 = 8'h20;//Space
`define  TEXT_BLOCK_5_BYTE_14 = 8'h74;//t
`define  TEXT_BLOCK_5_BYTE_13 = 8'h6e;//n
`define  TEXT_BLOCK_5_BYTE_12 = 8'h49;//I
`define  TEXT_BLOCK_5_BYTE_11 = 8'h20;//Space
`define  TEXT_BLOCK_5_BYTE_10 = 8'h65;//e
`define  TEXT_BLOCK_5_BYTE_9 = 8'h72;//r
`define  TEXT_BLOCK_5_BYTE_8 = 8'h75;//u
`define  TEXT_BLOCK_5_BYTE_7 = 8'h74;//t
`define  TEXT_BLOCK_5_BYTE_6 = 8'h61;//a
`define  TEXT_BLOCK_5_BYTE_5 = 8'h72;//r
`define  TEXT_BLOCK_5_BYTE_4 = 8'h65;//e
`define  TEXT_BLOCK_5_BYTE_3 = 8'h70;//p
`define  TEXT_BLOCK_5_BYTE_2 = 8'h6d;//m
`define  TEXT_BLOCK_5_BYTE_1 = 8'h65;//e
`define  TEXT_BLOCK_5_BYTE_0 = 8'h54;//T
```

In the code snippet above from the BLE Driver Demo program, the full text message is:

- Temperature Int 5

In hex, it shows up as above. Please note there are three bytes at the end of the text block labeled:

- “Communication End of Block”

These bytes are added to the users input by the JoyStick app. The user must include these bytes in the “user_define.v” file to signify the end of the message. Next, the user must define a message command byte.

```
//Custom Text Block Command Array defines
`define USER_COMMAND_1                  8'ha1//START_TEMPERATURE
`define USER_COMMAND_2                  8'ha2//STOP_TEMPERATURE
`define USER_COMMAND_3                  8'ha3//CHANGE_TEMPERATURE_INT_0_1
`define USER_COMMAND_4                  8'ha4//CHANGE_TEMPERATURE_INT_1
`define USER_COMMAND_5                  8'ha5//CHANGE_TEMPERATURE_INT_5
```

In this code snippet, if “TEXT_BLOCK_5_BYTE_xx” array is equal to the “MEMORY_ARRAY_xx” then, “USER_COMMAND_5” byte will be transmitted through the “COMMAND_SELECTION” to the user code. You can see from the code snippet that “USER_COMMAND_5” has a unique byte: 8’ha5. Also added is a comment about what this command is assigned to:

- CHANGE_TEMPERATURE_INT_5

Now the user code can operate using the “COMMAND_SELECTION” to perform some user defined function.