# BTIO50
# USB/Ethernet I/O Card
# Game Port, Joystick, Custom HID
# Discrete Signals Interface Board

## User and Interface Manual

**Part # 15B180000-1001 Rev. K**

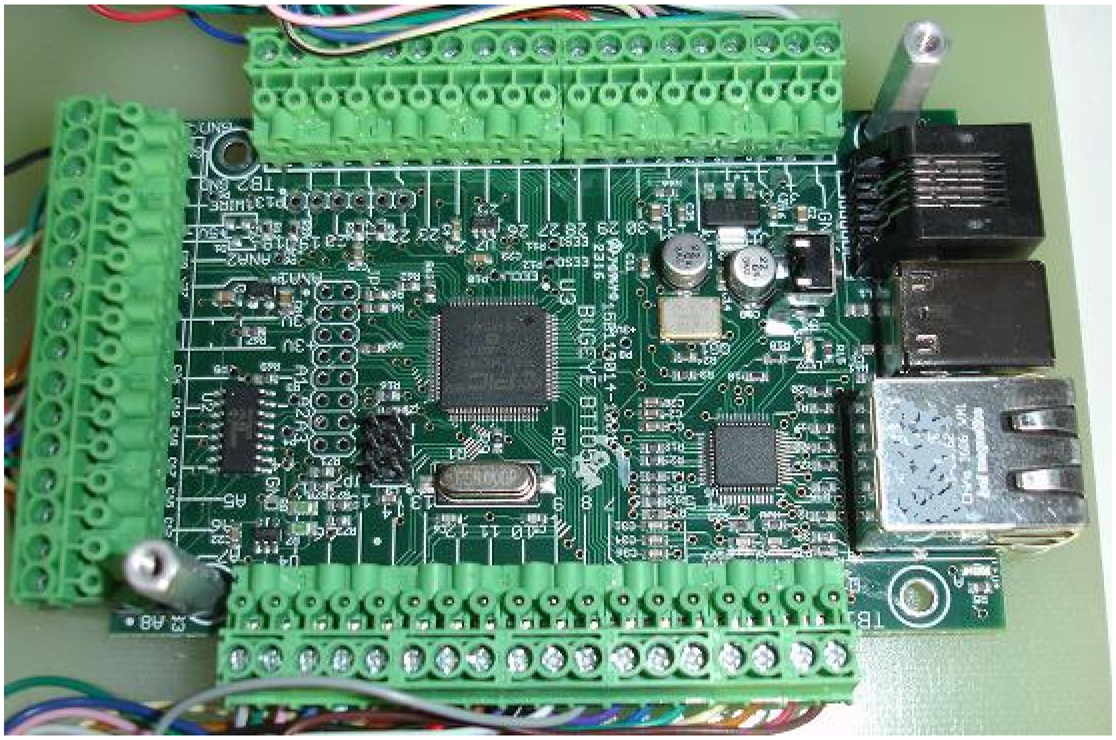
Chris Ruff

2-14-2025

## Contents

## 1.0 Overview

In this document “Device”, “the device”, “this device”, “the board”, or “this board” all refer to the BTIO50 USB/Ethernet I/O board.



This device listens to analog and digital inputs and packages the input values for transmission to the USB and/or Ethernet-attached host computer. The USB interface can be Game Port, Joystick Port, or Custom HID packets. The Ethernet interface transfers data in the Bugeye messaging format via UDP packets.

The device is powered by either the USB cable or via an external 5 volt power supply. If only Ethernet is to be used, the USB connector does not need to be connected.

Bugeye supplies Windows factory applications that allow the boards to be found, firmware to be updated, analog inputs to be calibrated, and inputs to be viewed:

1. BFinder201.exe[1]: This application runs on Windows and it allows the user to find BTIO50 units, set BTIO50 IP address or set DHCP, set the BTIO50 UDP reception port, the BTIO50 USB PID value (32767 discrete values), and Identification string.

[1] BFinder201.exe is numbered 15B180000-2001

2. BTIO50EthCalib200.exe[2]: this application runs on Windows and allows the analog inputs: Type, Minimum, Deadband Start and Stop, and Maximum values to be calibrated for each channel.
3. BTIO50EthClient108[3]: this Win32 application allows the BTIO50 inputs to be viewed through the Ethernet interface. Bugeye provides the source code for this application.
4. BTIO50USBClient200.exe[4] client: this Windows Managed C++ application allows the BTIO50 (running the Custom HID firmware application) values to be viewed through the USB Custom HID interface. The source code is available for this application.
5. BTIO50WinGameApp200.exe[5] Game/Joystick port client: this Windows Managed C++ Direct-X application allows the BTIO50 (running the GameApp or the bGameApp firmware application) inputs to be viewed. The source code is available for this application.

In the case of Game/Joystick firmware application it is also possible to use the various Game Port/ Joystick apps/ applets available in the various operating systems to monitor and provide OS-located calibration for axes.

Ethernet Communication with this device is accomplished via UDP over 100base-T wired network. The Ethernet communication is messaging-based and the Bugeye Windows application provides a working indication of this capability. The source code for this application is provided to the customer.

The Electronics/Firmware design supports Static IP and DHCP. The IP address and the DHCP capability are altered via UDP using the Bugeye windows application BFinder201.exe.

The firmware is easily updated via Ethernet using the same BFinder201.exe application.

The board provides two modes of digital input operation: “Din47” and “Scan40”. Din47 mode provides 47 low-true inputs (pulled to ground to activate) and Scan40 provides 40 (4x10) scanned keys and 35 pull-down inputs remaining. Each digital low-true input is internally pulled with a 4.7k resistor up to +3.3 Volts.

10 analog inputs are provided, 8 of which are available at the USB Game Port interface. Analog inputs 1-8 have an active voltage range of 0-3 volts. The two analog inputs 9 and 10 may be driven up to 3.3 volts. The analog inputs are attenuated with a 0.1uF cap on each input to ground.

Each analog input that is used must be calibrated to a certain type of control, either:

“end-zero” or “center-zero”.

The result of the calibration of a given analog control/input is quiet operation at “hands-off” time and as the control is moved, the previously quiet number will

[2] BTIO50EthCalib200.exe is numbered 15B180000-2002
[3] BTIO50EthClient108.exe is numbered 15B180000-2003
[4] BTIO50USBClient200.exe is numbered 15B180000-2004
[5] BTIO50WinGameApp200.exe is numbered 15B180000-2005

commence its count up or countdown in the direction the control is moved when the control is moved off of the “dead band” and into the active area of the control’s travel.

In an “end-zero” calibrated axis, the minimum reported value to the client will be zero and the largest value reported will be 32767 to Ethernet, and 65535 in Game Port direct Input counts. A dead band may be specified at the minimum counts end and up to any position above that.

In the center-zero calibration, the full minimum direction will report zero, the full maximum value reported will be 32767/65535. The “zero” value will be roughly in the center near 16382/32764, but the calibration procedure along with the calibrating person specifies by their actions whatever that value is with the mechanicals that are being calibrated to the board. The calibration procedure allows the operator to insert a dead band that is centered about the zero position. A possible center-zero calibration result:

| | Minimum | Db low | Db high | maximum |
|---|---|---|---|---|
| AD counts | 532 | 657 | 673 | 1855 |
| Eth Output value | 0 | 16300 | 16300 | 32767 |
| Direct Input Output value | 0 | 32600 | 32600 | 65535 |

When the control is in the center position (661 or so counts here), there will be no update of the reported number to the host until the axis is moved either down to 656 counts or up to 674 counts- at which point the numbers will descend or climb.

The board has a small EEPROM that stores the calibration data for the 10 analog channels and the other configurable board capabilities. The calibration points can be read out of the board and can be written in to the board. The Ethernet-based calibration application can read and write the files to clone a previously calibrated board for easy replacement.

The board supports 2 channels of two different quad encoder types: open contact (MECH) and electronic (OPTO). Electronic encoders generally are powered by 5 volts and provide clean quadrature output signals. Open contact encoders create a large amount of electronic noise when the contacts are transitioning from one state to the other. The board provides two different algorithms to handle the cheaper MECH encoders or the more expensive OPTO encoders. The inputs for encoder 1 are TB1-1 and TB1-2. The inputs for encoder 2 are TB1-3 and TB1-4. The board incorporates 4.7k pull up resistors and a simple R-C low pass filter to attenuate noise on the encoder lines.

The Rev. D (and above) board contains circuitry that provides variable backlight drive capability. The board provides both 0-5VDC and 0-28VDC pins for the control voltage. In addition, the backlights can be controlled through Ethernet control channel. The Rev D and above board can drive up to 800 mA of open drain current. The LED backlights can be powered by any power supply between and including 3.3VDC and 28VDC. The backlight can be set to a specific brightness and the value pushed into EEPROM through the Ethernet interface. In this manner, the backlights brightness can be set and no control at all would be necessary in the future. The board would always wake up and display the backlight brightness stored earlier.

The board provides two digital outputs that can be used to drive 20mA or less loads at 3.3 volts. An example wiring diagram is provided later in this document. The outputs can be exercised via the Ethernet interface.

The board provides the capability to readout the raw AD values directly through the normal analog interface values. The choice of raw versus scaled analog outputs may be chosen in the calibration application and saved to flash storage so that the board wakes up in the desired analog readout mode. See the calibration application specification later in this document.

BTIO50 Firmware-based Modes:

- Game Port Mode

Firmware: BTIO50GameApp220 Bugeye number 15B180000-3002
Overview: BTIO50 write-only to host machine via USB
Windows USB Factory App: WinGameApp105.exe
Windows Ethernet Factory App: btio50EthClient108.exe

- Joystick Mode

Firmware: BTIO50bGameApp200 Bugeye number 15B180000-3003
Overview: BTIO50 write-only to host machine via USB
Windows USB Factory App: WinGameApp105.exe
Windows Ethernet Factory App: btio50EthClient108.exe

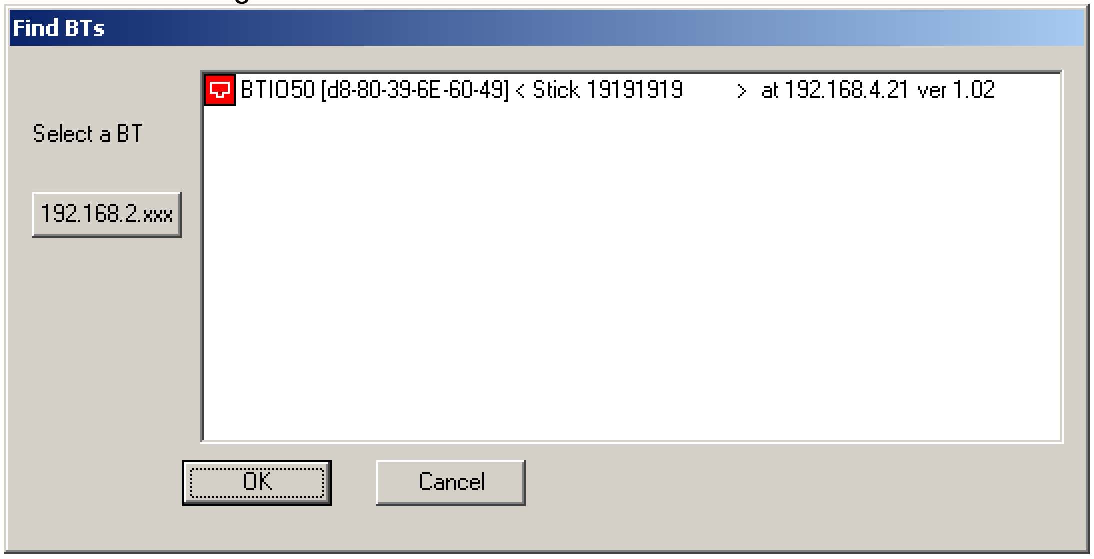
- Custom-HID Mode

Firmware: BTIO50CustomHidApp200 Bugeye number 15B180000-3001
Overview: BTIO50 read/write from/to host machine via USB
Windows USB Factory App: BTIO50UsbClient200.exe
Windows Ethernet Factory App: btio50EthClient108.exe

## 1.1 USB Game/Joystick communication

The BTIO50 is seen as a game controller by the operating system. The IDENT string (seen here in BFinder201.exe screens) is made available to the operating system.

The IDENT string in the BFinder201.exe find screen:



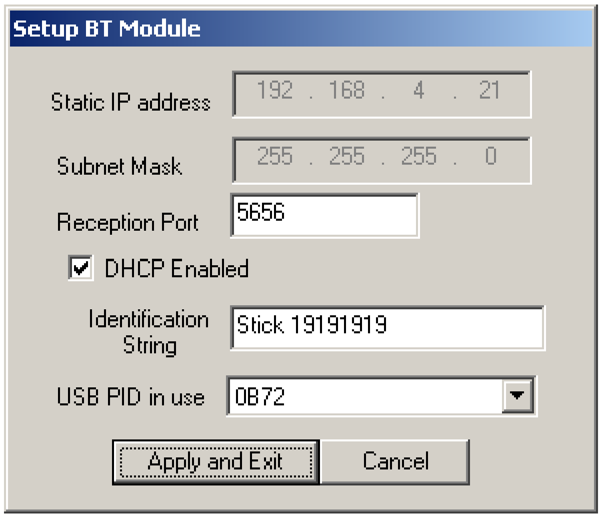
Find BTs

Select a BT

192.168.2.xxx

BTIO50 [d8-80-39-6E-60-49] < Stick 19191919 > at 192.168.4.21 ver 1.02

The IDENT string in the BFinder201.exe setup dialog



Setup BT Module

Static IP address: 192 . 168 . 4 . 21

Subnet Mask: 255 . 255 . 255 . 0

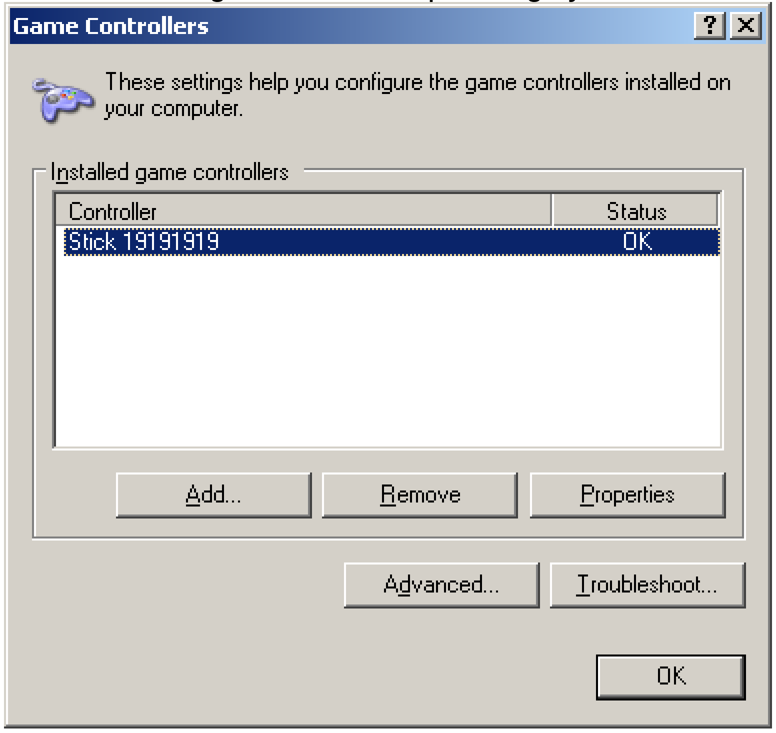
Reception Port: 5656

[✓] DHCP Enabled

Identification String: Stick 19191919

USB PID in use: 0B72

The Ident string is visible on operating system screens.



Game Controllers

These settings help you configure the game controllers installed on your computer.

Installed game controllers

| Controller | Status |
|---|---|
| Stick 19191919 | OK |

In the section on using the BFinder201.exe application, the methods necessary to change the IDENT string are discussed.

This USB 2.0 device supports one VID: 3475 and 32767 PIDs.

The lower (0-16383) PIDs send out shorts to the OS for analog values, and the top (16384-32767) PIDs send out unsigned bytes for the analog values.

Certain SIM programs read the raw data from the HID stream instead of using direct-X Input to acquire the board's data, so the two different variable types are available to make the board more compatible with more simulation software.

When Direct Input is used, it makes no difference what the PID is set to (if we ignore the resolution difference between 8 and 16 bits analog fields).

Up to 32767 boards can co-exist in the simulation system on one computer. Each board draws 120-200mA from the host machine. The 47 digital inputs are packaged along with the 10 analog inputs and made available to the host computer at 100 HZ.

The following table lists the 8 shorts version of the assignment locations in the 31 byte packet transmitted to the host computer.  The analog words are built from two bytes as shown. The source code later in the document also shows the shift/or operation to build a WORD from the BYTEs.

When the encoder #1 is turned on, analog channel #8 will report the encoder value instead of analog information

| Ana WORD | Byte Offset | description |
|---|---|---|
| 0 | 0(hi) 1 (lo) | Analog channel 1 |
| 1 | 2(hi) 3(lo) | Analog channel 2 |
| 2 | 4 | Analog channel 3 |
| 3 | 6 | Analog channel 4 |
| 4 | 8 | Analog channel 5 |
| 5 | 10 | Analog channel 6 |
| 6 | 12 | Analog channel 7 |
| 7 | 14 | Analog channel 8/ encoder 1 ABS |
| Dig BYTE | offset | Description |
| 1 | 16 | Inputs I1-I8 |
| 2 | 17 | Inputs I9-I16 |
| 3 | 18 | Inputs I17-I24 |
| 4 | 19 | Inputs I25-I32 |
| 5 | 20 | Inputs I33-I40 |
| 6 | 21 | Inputs I41-I47 |
| Scan BYTE | offset | Description |
| 1 | 22 | Scanned Key 1-8 |
| 2 | 23 | Scanned Key 9-16 |
| 3 | 24 | Scanned Key 17-24 |
| 4 | 25 | Scanned Key 25-32 |
| 5 | 26 | Scanned Key 33-40 |
| Encoder Bytes | Offset | Description |
| 1 | 27 | Encoder 1 Value absolute |
| 2 | 28 | Encoder 2 Value absolute |
| 3 | 29 | Encoder 1 Value relative |
| 4 | 30 | Encoder 2 Value relative |

The following table lists the 8 analog BYTES version of the assignment locations in the 23 byte packet transmitted to the host computer.

| Ana UBYTE | Byte Offset | Description |
|---|---|---|
| 0 | 0 | Analog channel 1 |
| 1 | 1 | Analog channel 2 |

| | | |
|---|---|---|
| 2 | 2 | Analog channel 3 |
| 3 | 3 | Analog channel 4 |
| 4 | 4 | Analog channel 5 |
| 5 | 5 | Analog channel 6 |
| 6 | 6 | Analog channel 7 |
| 7 | 7 | Analog channel 8/ encoder 1 ABS |
| Dig BYTE | offset | Description |
| 1 | 8 | Inputs I1-I8 |
| 2 | 9 | Inputs I9-I16 |
| 3 | 10 | Inputs I17-I24 |
| 4 | 11 | Inputs I25-I32 |
| 5 | 12 | Inputs I33-I40 |
| 6 | 13 | Inputs I41-I47 |
| Scan BYTEs | offset | Description |
| 1 | 14 | Scanned Key 1-8 |
| 2 | 15 | Scanned Key 9-16 |
| 3 | 16 | Scanned Key 17-24 |
| 4 | 17 | Scanned Key 25-32 |
| 5 | 18 | Scanned Key 33-40 |
| Encoder Bytes | Offset | Description |
| 1 | 19 | Encoder 1 Value absolute |
| 2 | 20 | Encoder 2 Value absolute |
| 3 | 21 | Encoder 1 Value relative |
| 4 | 22 | Encoder 2 Value relative |

When the board is in Din47 mode the Scan bytes 1-5 are undetermined. When the board is in Scan40 mode, scan bytes 1-5 contain the result of the ongoing keyboard scan and the upper nibble of dig byte 1 is used for the scan driver. The dig byte 2 and 2 bits of dig byte 3 are also undetermined in Scan40 mode.

When a scanned key is pressed, a given bit will go high that represents the pressed key in the scanned array. See Scan Line Reporting below for more detail on scanned keys and their reporting to the host. The pin-out section below will cover the physical placement of I/O connections for Inputs.

The two encoder channels are reported on in Bytes 37-40. The absolute bytes report 0-0xff. When the encoder is turned beyond 0xff, the value folds over back to 0. When at zero and the encoder is turned lower, the value pops back up to 0xff.
The encoder relative bytes sit at 127. When the encoder is moved in a (-) direction the number will change to some value below 127. When the encoder is

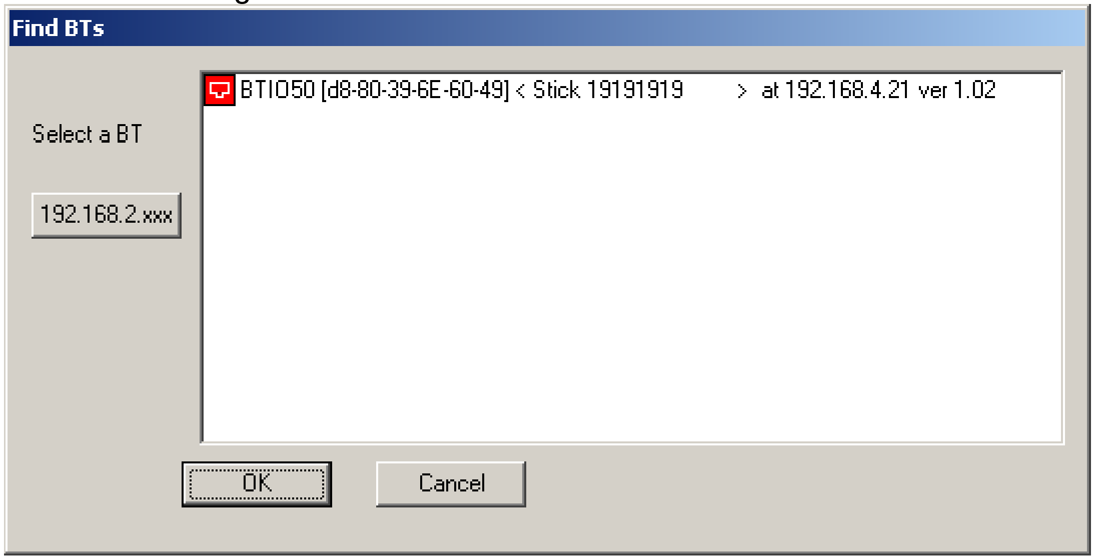
moved in the (+) direction the number will go to some value above 127. The size of this number is a function of the number of clicks that have transpired since the last report to the host.

## 1.2 USB Custom HID communication

This USB 2.0 device supports one VID: 3475 and 32767 PIDs: 0-32767, allowing up to 32767 boards at once to be attached to the host computer. Each board draws 120-200mA from the host machine.

The IDENT string (seen here in BFinder201.exe screens) is made available to the operating system. The string can be seen in the HID device listings in the Devices listing in Windows.

The IDENT string in the BFinder201.exe find screen:



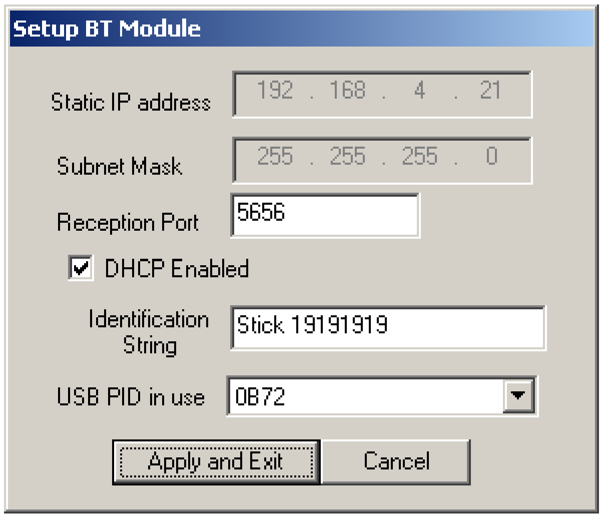
Find BTs

Select a BT

192.168.2.xxx

BTIO50 [d8-80-39-6E-60-49] < Stick 19191919 > at 192.168.4.21 ver 1.02

The IDENT string in the BFinder201.exe setup dialog



Setup BT Module

Static IP address: 192 . 168 . 4 . 21

Subnet Mask: 255 . 255 . 255 . 0

Reception Port: 5656

[✓] DHCP Enabled

Identification String: Stick 19191919

USB PID in use: 0B72

In the section on using the BFinder201.exe application, the methods necessary to change the IDENT string are discussed.

The 47 ground-based digital inputs are packaged along with the 10 analog inputs and made available to the host computer at (at least) 100hz throughput. The following table lists the assignment locations in the 64 byte packet transmitted to the host computer.  The analog words are built from two bytes as shown. The source code later in the document also shows the shift/or operation to build a WORD from the BYTEs.

USB Data Board->Client

| Ana WORD | Byte Offset | description |
|---|---|---|
| 0 | 4(hi) 5 (lo) | Analog channel 1 |
| 1 | 6(hi) 7(lo) | Analog channel 2 |
| 2 | 8 | Analog channel 3 |
| 3 | 10 | Analog channel 4 |
| 4 | 12 | Analog channel 5 |
| 5 | 14 | Analog channel 6 |
| 6 | 16 | Analog channel 7 |
| 7 | 18 | Analog channel 8 |
| 8 | 20 | Analog channel 9 |
| 9 | 22 | Analog channel 10 |
| Dig BYTE | offset | Description |
| 1 | 24 | Inputs I1-I8 |
| 2 | 25 | Inputs I9-I16 |
| 3 | 26 | Inputs I17-I24 |
| 4 | 27 | Inputs I25-I32 |
| 5 | 28 | Inputs I33-I40 |
| 6 | 29 | Inputs I41-I47 |
| 7 | 30 | Scanned Key 1-8 |
| 8 | 31 | Scanned Key 9-16 |
| 9 | 32 | Scanned Key 17-24 |
| 10 | 33 | Scanned Key 25-32 |
| 11 | 34 | Scanned Key 33-40 |
| 12 | 35 | Future use |
| 13 | 36 | Future Use |
| 14 | 37 | Encoder 1 Value absolute |
| 15 | 38 | Encoder 2 Value absolute |
| 16 | 39 | Encoder 1 Value relative |
| 17 | 40 | Encoder 2 Value relative |

When the board is in Din47 mode the Dig bytes 7-11 are undetermined. When the board is in Scan40 mode, Dig BYTE 2 is undetermined and bytes 7-11

contain the result of the ongoing keyboard scan. A given bit will go high that represents a key being pressed in the scanned array. See Scan Line Reporting below for more detail on scanned keys and their reporting to the host. The pin-out section below will cover the physical placement of I/O connections for Inputs. Analog values will report between 0 and 32767, MIN to MAX for movement of physical devices.
The two encoder channels are reported on in Bytes 37-40. The absolute bytes report 0-0xff. When the encoder is turned beyond 0xff, the value folds over back to 0. When at zero and the encoder is turned lower, the value pops back up to 0xff.
The encoder relative bytes sit at 127. When the encoder is moved in a (-) direction the number will change to some value below 127. When the encoder id mover in the (+) direction the number will go to some value above 127. The size of this number is a function of the number of clicks that have transpired since the last report to the host.

USB Data Client->Board

| Name | Byte Offset | Description |
|---|---|---|
| Not used | 0 | Report ID |
| BL Mode | 1 | Blight control source specified |
| BL Value | 2 | Blight dim value 0-39 |

The backlight mode byte is one of the following values:

```
#define DIM_MODE_ELECTRIC 0
#define DIM_MODE_ETH_USB 1
```

When the client sets the BL Mode byte to DIM_MODE_ELECTRIC the dimmer circuitry is controlled by the DC lines- either 0-5 or 0-28.  When the byte is set to DIM_MODE_ETH_USB the backlights are controlled by the byte BL Value.  The value 0 turns off the backlights and values between and including 1-39 drive the backlights from dim through fully bright.

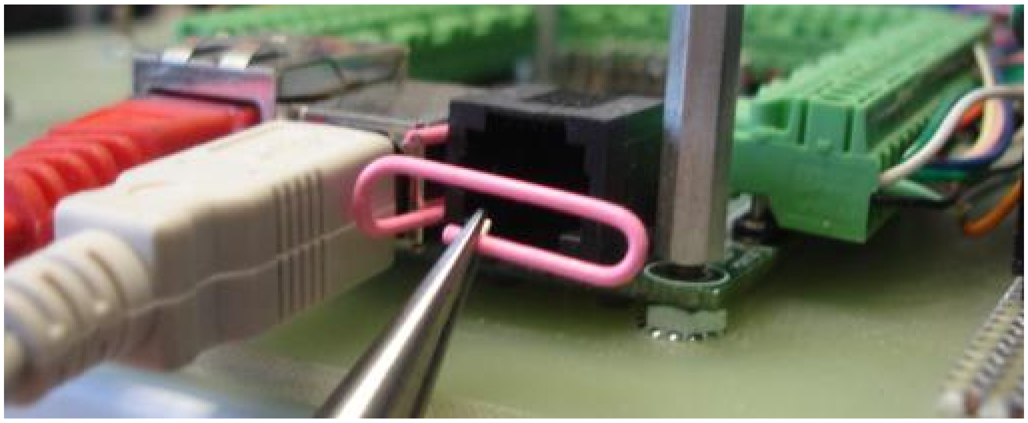
This control method provides only run-time control of the backlights. No changes here are pushed into the EEPROM.

## 1.3 Ethernet Communications

This device operates as a master-slave server for UDP messaging. The Ethernet interface is used for both input reporting and board setup/ calibration/ configuration. The USB PID is specified, the analog inputs are setup initially, the board configuration is uploaded and downloaded, the firmware is updated, and various other operations are all performed through the Ethernet interface. The basic protocol will be discussed first, an overview of the messages will be discussed, and then detail will be added for some of the messages that are used by the customer.
The Ethernet hardware supports 10/100Mbps speed. The cable interface is CAT-5 RJ-45 and the board supports Auto-MDIX. Any cable type can be used to connect to the host device, hub, switch, or computer directly.

## 1.3 IP Address and ‘The Button’

The button on the board is used to setup either a known hard coded IP address or place the board into DHCP mode. The button is behind the RJ12 and the USB connectors and can be pressed by inserting a paper clip or a small knife blade between the connectors.



The blue LED is adjacent to the Ethernet connector and its glow is viewed at the outside of the Ethernet connector. The orange LED is adjacent to the Ethernet connector and its glow is viewed at the outside of the Ethernet connector. The blue LED is adjacent to the RJ-12 connector and its glow can be seen to the outside of the RJ-12 connector.

a. To set the IP address to **192.168.2.242**:
    - Power down the board
    - Press and hold the button
    - Power up the board
    - Release the button after a couple of seconds.
    - The orange LED will blink and the board will be sitting in the boot monitor at IP address **192.168.2.242**
    - Cycle the power and the board will execute the firmware application and can be found at address **192.168.2.242**

b. To place the board into DHCP mode
    - Power up the board and wait at least 10 seconds or until the orange LED stops blinking and the Blue and Orange LEDs are extinguished.
    - Press and hold the button for approximately ten seconds or until the blue LED comes on solid
    - Release the button and repower the BTIO50. The unit will reboot in DHCP mode and retrieve its IP address from your network. You can then “find” the board with BFinder201.exe.

BTIO50 supports and responds to PING in both the boot monitor (orange LED flashing) and the application. Under normal usage the BTIO50 will not be sitting in the boot monitor at any time.

If BTIO50 is setup in DHCP mode and the network does not respond to DHCP requests, the Blue LED will flash rapidly at 4 times a second after power is applied for approx. 12 seconds and then go dark. When the DHCP server becomes visible BTIO50 will obtain an address normally.

## 1.4 Bugeye Protocol Ver 6

A Bugeye Protocol Version 6 UDP host->device message will look like this:

<STX><BAR><PROTO><PORT><FUNC><LENG><payload><CKSUM><BAR><ETX>

Where

BAR = 0x7c (1 byte)
PROTO=0x06 (1 byte)
PORT = (4 byte) the host UDP target port. This is the port that the client application is specifying to the BTIO50 that it is receiving packets on. Every command sent to the BTIO50 is able to focus the board's response traffic to a different port if desired, or change the send-to port in mid-stream.
FUNC = (4 bytes) enumerated values listed below
LENG = length of data payload (4 bytes)
payload = data payload; encoded ASCII 7-bit hex. Do not include STX, ETX or the checksum. Do not include NULLS- they may 'disappear' due to host computer OS driver activity.
CKSUM = (4 bytes) protects all bytes up to itself.
The checksum is hexadecimal encoded unsigned 4 bytes of the sum after all bytes starting at STX and ending at the end of the payload are added up.
STX = 2 (1 byte)
ETX = 3 (1 byte)

If any Command is received that does not have this format, it is not a Bugeye ver_6 packet and will be ignored. .

```
//
// from the parsing code:
//
#define MAX_BUG_6_LEN 500
#define BUG_PROTOCOL_VER6 6
//
// messages
//
enum BTIO_MESSAGES
{
  MSG_ACK,
  MSG_NAK,
  MSG_GET_VER,
  MSG_RETURN_VER,
  MSG_GET_STATUS,
  MSG_RETURN_STATUS,
  MSG_GET_ALL,
  MSG_RETURN_ALL,
  MSG_SET_CALIB,
  MSG_GET_CALIB,
  MSG_RETURN_CALIB,
  MSG_SET_MODE,
  MSG_GET_AVAL,
  MSG_RETURN_AVAL,
  MSG_SET_BL_MODE,
  MSG_GET_BL_MODE,
  MSG_RETURN_BL_MODE,
  MSG_SET_BL_VAL
  MSG_SET_SCALING_TYPE,
  MSG_SET_IO_STATE,};
```

The following code builds a bugeye packet:

```
// *********************************************************************************
// *********************************************************************************
// *********************************************************************************
int BuildBugeyePacket(char *outbuf, short func, char *data, int length)
// *********************************************************************************
{
  int cksum=0;
  int i;
  if (length > MAX_BUG_6_LEN)
  {
    //zprintf("\r\nMessage Too Long");
    return 0;
  }
  //
//strcpy(build,"                ");
        sprintf(outbuf,"%c|%c%04X%04X%04X",
    STX,
    BUG_PROTO_6,
    nListenPort,  // the port I am listening on
    func,
    length);
  // place payload
  if (length)
  {
    for(i=0;i<length;i++)
    {
      outbuf[i+15]=data[i];
    }
  }
  //
  // now build has all the way including payload
  //
  // calculate the checksum
  for(i =0;i<length+15;i++)
  {
    cksum += outbuf[i];
  }
  //
  sprintf(&outbuf[length+15],"%04X|%c",cksum,ETX);
  return i;
}// ********************************************************************************
// parse a Bugeye Proto Ver 6 message
// ********************************************************************************
//
char payload[MAX_BUG_6_LEN+2];
char build[500];
// ********************************************************************************
int ParseBugeyePacket(char *pData)
{
  int i,func, len,cksum,port;
  int dcksum=0;
  //
  if (!((pData[0] == STX) && (pData[1] == '|')&& (pData[2] == BUG_PROTO_6)))
  {
    // this is not bugeye protocol!
    return 1;
  }
  // step over STX,BAR, PROTO
  sscanf(&pData[3],"%04X%04X%04X",
    &port,
    &func,
    &len);
  //
  // port is the port that the sender is listening on
  //
  // index to the end of the message-just after the checksum
  // 3 == skip over STX,BAR,PROTO
  // len was calculated above and is the length of the payload in bytes
  // + 13 is skip over PORT + FUNC + LENG  point to first byte of payload
  // +len the length of the payload
  // + 4 is the length of the CKSUM
  // -1 point at closing BAR;
  // look for the closing BAR and ETX
```

```
if (!((pData[i] == '|') && (pData[i+1] == ETX) ))
{
   // this is not bugeye protocol!
   return 2;
}
//
// index the checksum field
sscanf(&pData[3+13+len],"%04X",&cksum);

//
// add the first two items to the local checksum
dcksum = STX + '|' + BUG_PROTO_6;
//
// add in the FUNC, LEN, etc.
for(i=3;i<(3+12);i++)
{
   dcksum += pData[i];
}
// is there a payload?
if (len)
{
   // there is a payload
   if (len > MAX_BUG_6_LEN)
   {
      // something is wrong, there are no payloads
      // this long in bugeye protocol!
      return 3;
   }
   //
   // get a copy of the payload for the switch below
   memcpy(payload,&pData[3+12],len);
   // add the payload to the checksum
   for(i=0;i<len;i++)
   {
      dcksum += payload[i];
   }
}  // if len
//
// check the checksum for data corruption
if (dcksum != cksum)
{
   // checksum fails
   return 5;
}
//
```

```
    // at this point we have enough data to pull a message
    //
    //
    switch(func)
    {
    case MSG_GET_VER:
    {
      char buf[10];
      sprintf(buf,"%08lX",VERSION);
      BuildBugeyePacket(build, MSG_RETURN_VER,buf,strlen(buf));
      SendUDP(dwLastIP,port,build,strlen(build));
      break;
    }
    //case …
    //    break;
      default:
      BuildBugeyePacket(build, MSG_NAK,””,0);
      SendUDP(dwLastIP,port,build,strlen(build));
    } // sw
    // no error
    return 0;
}
```

## 1.5 BTIO50 Specific Payload Data Detail

When the device receives the command:

MSG_GET_VER

This message returns the firmware version to the calling client:

The return payload is built:

The payload string is an 8 byte hex-encoded UINT32 that represents the version of the firmware (100 == 1.00, 101 == 1.01, etc.).

```
char payload[];
sprint(payload,"%08LX%s",VERSION);
```

Payload is then encased in a VER6 UDP message MSG_RETURN_VER and sent back to the client app

MSG_GET_STATUS

This message returns the IdentString (initially specified during calibration of the board) and the WORD status to the calling client. Presently the status of the built in test is reported in the status WORD.

The return payload is built:

```
#define STAT_BIT_BIT_TEST_OK        0x01    // HIGH when BIT passed

WORD StatusWord |= STAT_BIT_BIT_TEST_OK;
```

The payload string is a 22 byte string and two hex-encoded bytes that represent a WORD (UINT16) with the above flag(s) encoded.

```
char payload[25];
sprint(payload,"%02X%s",statusWord,IdentString);
```

Payload is then encased in a VER6 UDP message MSG_RETURN_STATUS and sent back

MSG_GET_ALL

This message is used to query the BTIO50 digital and analog inputs.

The payload string is built:

```
char payload[80];
sprint(payload,"%04X%04X%04X%04X%04X%04X%04X%04X%04X%04X%02
X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X
%02X%02X%02X",
  analog1,
  analog2,
  analog3,
  analog4,
  analog5,
  analog6,
  analog7,
  analog8,
  analog9,
  analog10,
  inputs1-8,
  inputs9-16,
  inputs17-24,
  inputs25-32,
  inputs33-40,
  inputs 41-47,
  scannedKey1-8,
  scannedKey9-16,
  scannedKey17-24,
  scannedKey25-32,
  scannedKey33-40,
  future Use,
  future Use,
  encoder1abs,
  encoder2abs,
  encoder1rel,
  encoder2rel);
```

Payload is then encased in a PROTO_6 UDP message MSG_RETURN_ALL and sent back to client

MSG_SET_BL_MODE

This message specifies whether the board controls backlights from the DC signals or from the Ethernet or USB interfaces.  There are two possible values to set the backlight control to:

```
#define DIM_MODE_ELECTRIC 0
#define DIM_MODE_ETH 1
```

The payload is built:

```
WORD blightMode = DIM_MODE_ETH;
char payload[20];
sprint(payload,”%04X”,blightMode);
```

the payload is then encased in a VER6 UDP message MSG_SET_BL_MODE and sent to the board.  The board will respond with an ACK message

MSG_GET_BL_MODE

This message returns the backlight mode WORD to the calling client.

The return payload is built:

The payload string is a single WORD, the backlight mode WORD.

```
char payload[20];
sprint(payload,”%04X”,blightMode);
```

Payload is then encased in a VER6 UDP message MSG_RETURN_BL_MODE and sent back

MSG_SET_BL_VAL

This message specifies the backlight brilliance.  The range of values to set the backlight control to is 0-100.  0 turns off the backlights and 1 is the dimmest, 100 is the brightest.

The payload is built:

```
WORD blightVal = 22;
char payload[20];
sprint(payload,”%04X”,blightVal);
```

the payload is then encased in a VER6 UDP message MSG_SET_BL_VAL and sent to the board.  The board will respond with an ACK message

MSG_SET_CALIB, MSG_GET_CALIB, MSG_SET_MODE, and other commands not listed here- these are used for calibration and setup of the board and the analog inputs.

## OUTPUT PINS PROGRAMMING DETAILS

The output pins JP1-1 and JP1-2 are changed from inputs to outputs when commanded by the Ethernet client.

```
#define DIG51         1  // this controls JP1 pin 1
#define DIG52         2  // this controls JP1 pin 2
```

The payload is built:

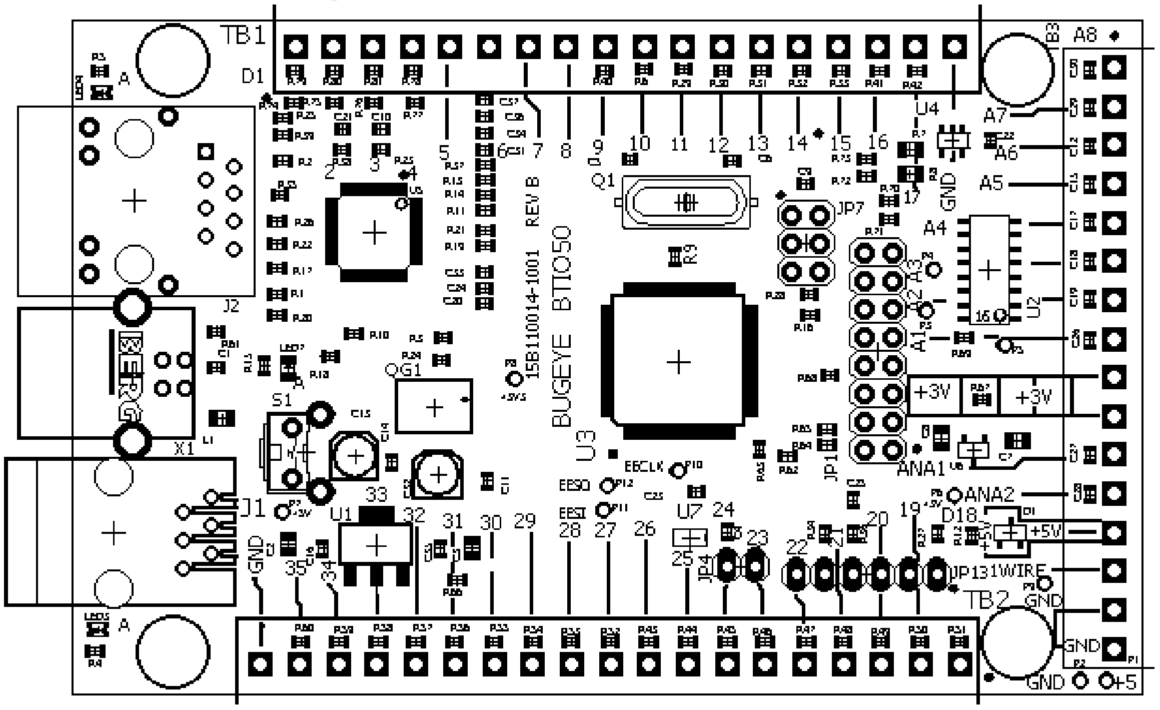
```
BYTE Outs=3;
char payload[20];

// turn on bit DIG51, JP1 pin 1
Outs &= ~DIG51;

// turn on bit DIG52, JP1 pin 2
Outs &= ~DIG52;

sprint(payload,”%02X”,Outs);
```

the payload is then encased in a VER6 UDP message MSG_SET_IO_STATE and sent to the board. The board will respond with an ACK message

When a bit is set to 0 the associated pin is set to output mode and the pin is pulled low. Any pin addressed sets both pins to outputs. Once pins are set to output mode they stay outputs until power is cycled.

# 2.0 Hardware Information

## 2.1 Device pins



| | Terminal Block TB1 |
|---|---|
| Tb1-D1 | Din47:input 1 Encoder 1 A |
| Tb1-2 | Din47:input 2 Encoder 1 B |
| Tb1-3 | Din47:input 3 Encoder 2 A |
| Tb1-4 | Din47:input 4 Encoder 2 B |
| Tb1-5 | Din47:input 5 Scan40:output D1 |
| Tb1-6 | Din47:input 6 Scan40:output D2 |
| Tb1-7 | Din47:input 7 Scan40:output D3 |
| Tb1-8 | Din47:input 8 Scan40:output D4 |
| Tb1-9 | Din47:input 9 Scan40:input S1 |
| Tb1-10 | Din47:input 10 Scan40:input S2 |
| Tb1-11 | Din47:input 11 Scan40:input S3 |
| Tb1-12 | Din47:input 12 Scan40:input S4 |
| Tb1-13 | Din47:input 13 Scan40:input S5 |
| Tb1-14 | Din47:input 14 Scan40:input S6 |
| Tb1-15 | Din47:input 15 Scan40:input S7 |
| Tb1-16 | Din47:input 16 Scan40:input S8 |
| Tb1-17 | Din47:Input 17 Scan40:input S9 |
| Tb1-GND | Common Ground |
| | |

| | |
|---|---|
| | Terminal Block TB2 |
| Tb2-D18 | Din47:Input 18    Scan40:input S10 |
| Tb2-19 | Input 19 |
| Tb2-20 | Input 20 |
| Tb2-21 | Input 21 |
| Tb2-22 | Input 22 |
| Tb2-23 | Input 23 |
| Tb2-24 | Input 24 |
| Tb2-25 | Input 25 |
| Tb2-26 | Input 26 |
| Tb2-27 | Input 27 |
| Tb2-28 | Input 28 |
| Tb2-29 | Input 29 |
| Tb2-30 | Input 30 |
| Tb2-31 | Input 31 |
| Tb2-32 | Input 32 |
| Tb2-33 | Input 33 |
| Tb2-34 | Input 34 |
| Tb2-35 | Input 35 |
| Tb2-GND | Common Ground |
| | |
| | Terminal Block TB3 |
| Tb3-GND | Common Ground |
| Tb3-GND | Common Ground |
| Tb3-+5V | 5 Volts input input (when USB is not used) or output for OPTO encoder(s) |
| TB3-+5V | 5 volts input (when USB is not used) or output for OPTO encoder(s) |
| Tb3-ANA2 | 3.3 volt tolerant analog input |
| Tb3-ANA1 | 3.3 volt tolerant analog input |
| Tb3- +3V | 3 volts output to top of potentiometers 250mA max load |
| Tb3- +3V | 3 volts output to top of potentiometers 250 mA max load |
| Tb3-A1 | 3 volt tolerant analog input |
| Tb3-A2 | 3 volt tolerant analog input |
| Tb3-A3 | 3 volt tolerant analog input |
| Tb3-A4 | 3 volt tolerant analog input |
| Tb3-A5 | 3 volt tolerant analog input |
| Tb3-A6 | 3 volt tolerant analog input |
| Tb3-A7 | 3 volt tolerant analog input |
| Tb3-A8 | 3 volt tolerant analog input |
| | |
| | 16 pin header- 0.1 centers, dual row |
| JP1-1 | Switched Output 1    DIG51 |
| JP1-2 | Switched Output 2    DIGR |

| | |
|---|---|
| JP1-3 | Input 38 |
| JP1-4 | Input 39 |
| JP1-5 | Input 40 |
| JP1-6 | Input 41 |
| JP1-7 | Input 42 |
| JP1-8 | Input 43 |
| JP1-9 | Not used |
| JP1-10 | Input 45 |
| JP1-11 | Not used |
| JP1-12 | Not used |
| JP1-13 | Common Ground |
| JP1-14 | Dimmable LED output |
| JP1-15 | Common Ground |
| JP1-16 | +3V3 output |
| | |
| | MOD-6 Jack |
| J1-1 | Common Ground |
| J1-2 | +5V power in or out |
| J1-3 | ANA2 |
| J1-4 | ANA3 |
| J1-5 | ANA4 |
| J1-6 | +3 volts out to potentiometers 250mA max load |
| | |
| | |

The On-board 3 volt voltage regulator that drives the potentiometer top-side connections is a 250mA part. Considering that a 1K pot will draw .003 amps at 3 volts, this regulator could drive 83 1K pots- so the current limit should not be a problem, even if a 500 ohm pot (6 mA draw) is used for some of the analog inputs.

JP1 9,11,12 have been assigned to the backlight hardware now.

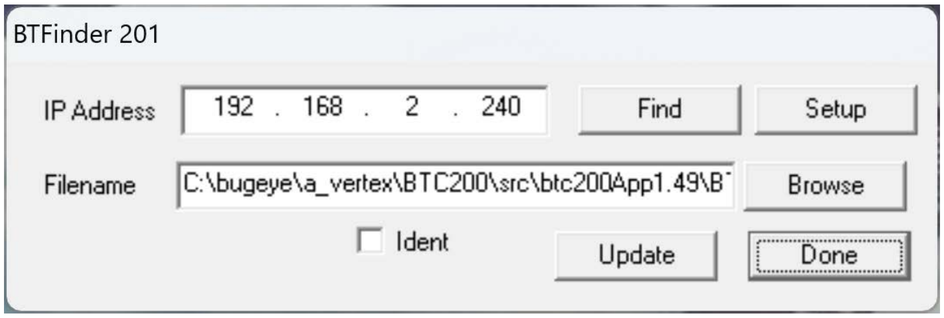
# 3.0 Support Applications

### 3.1 Change USB/Networking Parameters, Update firmware with BFinder201.exe

This windows application performs the following tasks:

- Locate BTIO50 boards on the local network
- Update firmware in a given BTIO50 board
- Specify a static IP or DHCP IP address assignment
- Specify which PID will be used for the USB HID assignment
- Set the identification string that shows up in the FIND dialog and in the various USB applications and applets

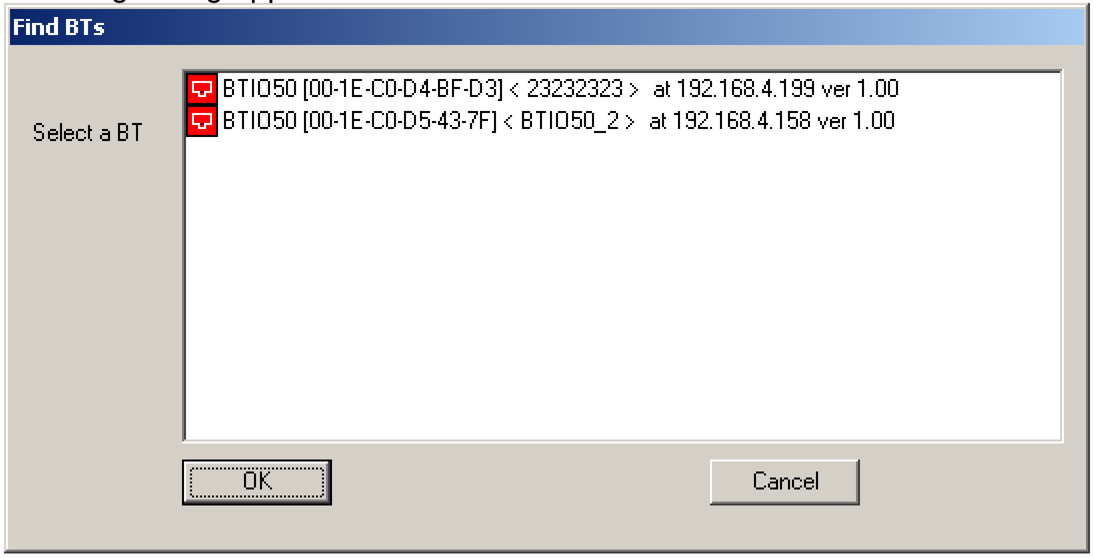
BFinder201.exe stores last used information on the hard drive and will invoke UAC complaints on certain operating systems.



IP Address: 192 . 168 . 2 . 240

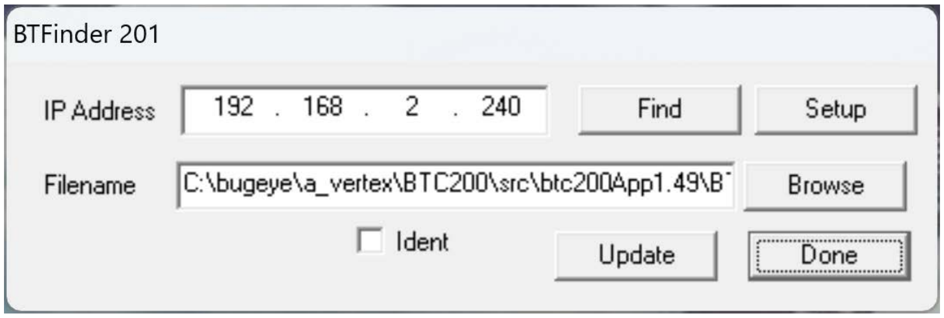
Filename: C:\bugeye\a_vertex\BTC200\src\btc200App1.49\B

Execute the application BFinder201.exe, then click the “Find” button and the following dialog appears:



BTIO50 [00-1E-C0-D4-BF-D3] < 23232323 > at 192.168.4.199 ver 1.00
BTIO50 [00-1E-C0-D5-43-7F] < BTIO50_2 > at 192.168.4.158 ver 1.00

Click on one and then OK, the IP address will show up in the main dialog IP Address field.



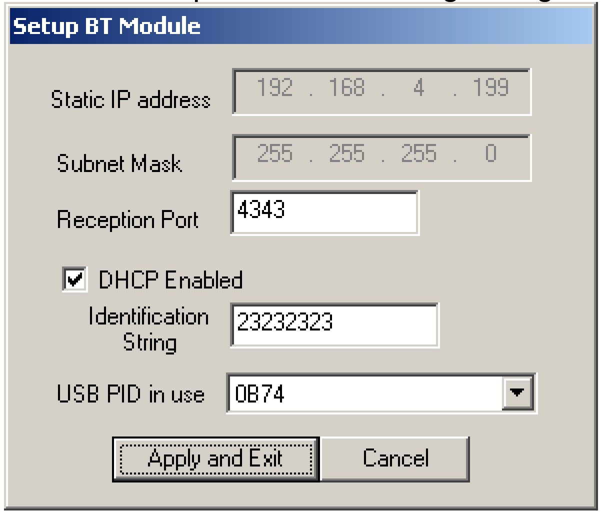
BTFinder 201

IP Address: 192 . 168 . 2 . 240

Filename: C:\bugeye\a_vertex\BTC200\src\btc200App1.49\B

☐ Ident

Click on the Ident checkbox, The Blue LED on the board will flash for a visual identification.
Click on Setup and the following dialog will appear:



Setup BT Module

Static IP address: 192 . 168 . 4 . 199

Subnet Mask: 255 . 255 . 255 . 0

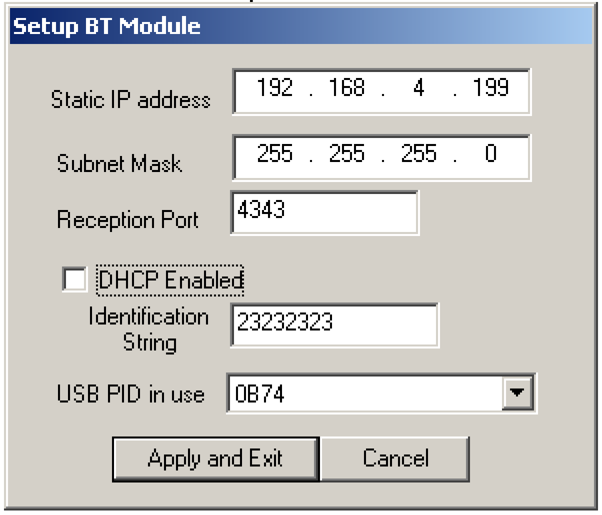
Reception Port: 4343

☑ DHCP Enabled

Identification String: 23232323

USB PID in use: 0B74

In this dialog the IP address can be selected by DHCP if it is checked and the static IP can be specified if not



Setup BT Module

Static IP address: 192 . 168 . 4 . 199

Subnet Mask: 255 . 255 . 255 . 0

Reception Port: 4343

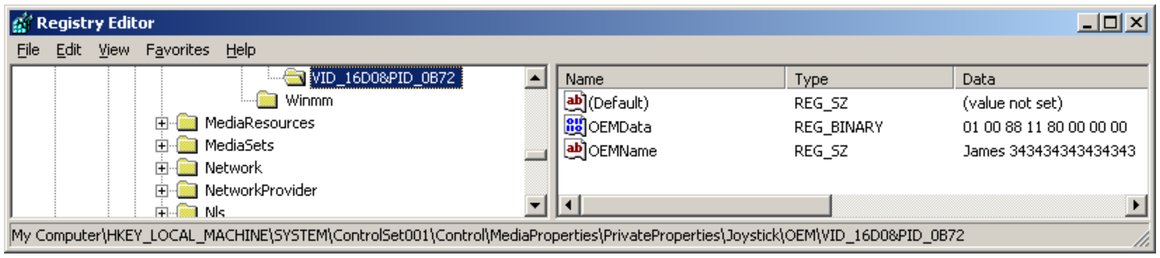
☐ DHCP Enabled

Identification String: 23232323

USB PID in use: 0B74

Here you can specify the IDENT string, the BTIO50 board UDP listener port, and the USB PID.
When the IP address is changed, or the DHCP assignment is altered, or the reception port is changed, or the PID is changed, or the IDENT string is changed. - then if the button “Apply and Exit” is clicked, the board will be updated and will reboot- then it will use the new settings that were entered on this screen.

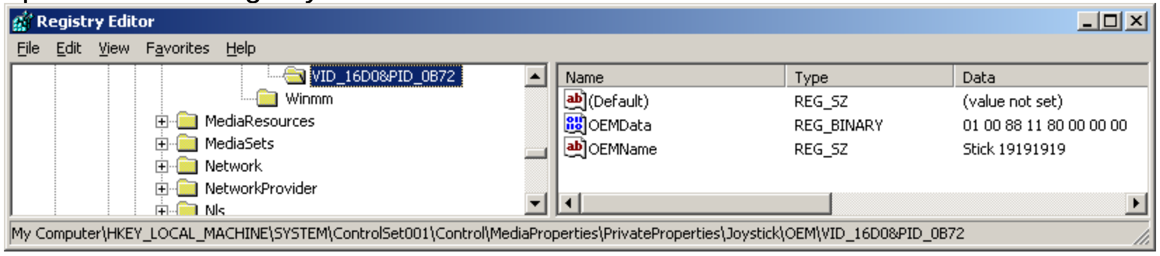
When changing the IDENT Identification String, the operating system may ignore your change. The way to handle this problem on Windows is to open the registry edit (Regedit) program, search for the old name and delete the OEMName line item containing that previous name:



| Name | Type | Data |
|---|---|---|
| (Default) | REG_SZ | (value not set) |
| OEMData | REG_BINARY | 01 00 88 11 80 00 00 00 |
| OEMName | REG_SZ | James 343434343434343 |

My Computer\HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Control\MediaProperties\PrivateProperties\Joystick\OEM\VID_16D0&PID_0B72

In this case the “James 343434…” line should be deleted. Unplug, and then plug in the BTIO50 you have changed the name in. The IDENT name will then show up on applications, applets, etc.
After the name is requested by some app or applet, the OEMName will be updated in the registry.

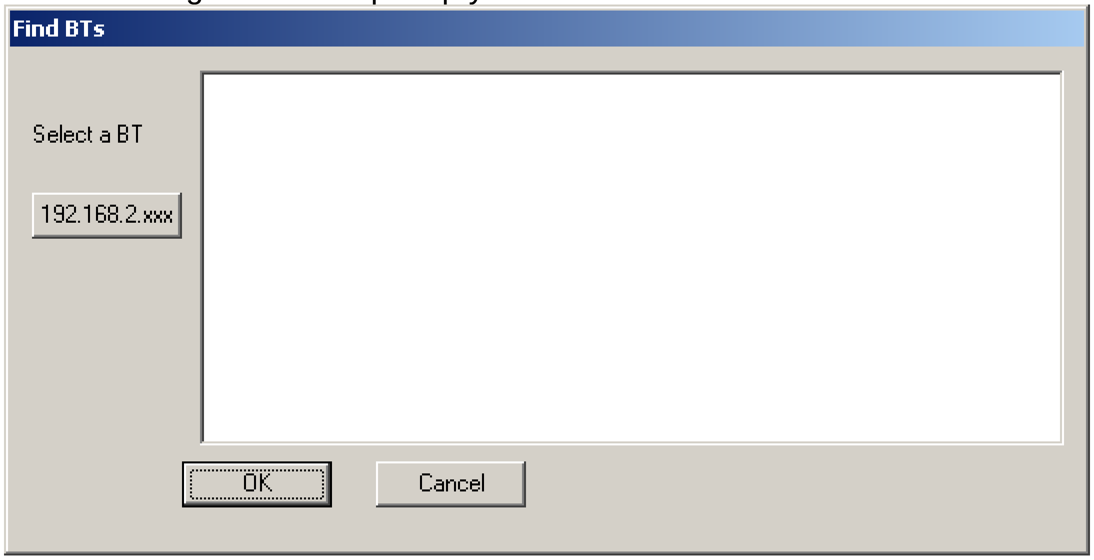


| Name | Type | Data |
|---|---|---|
| (Default) | REG_SZ | (value not set) |
| OEMData | REG_BINARY | 01 00 88 11 80 00 00 00 |
| OEMName | REG_SZ | Stick 19191919 |

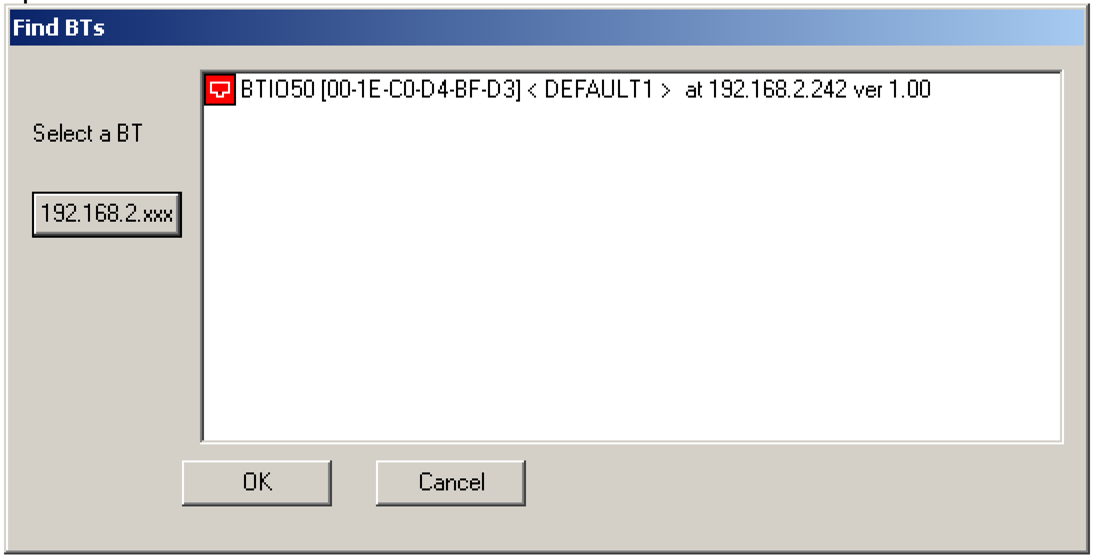
My Computer\HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Control\MediaProperties\PrivateProperties\Joystick\OEM\VID_16D0&PID_0B72

## More than one Ethernet Connection in a computer

In the case that the machine running BFinder201.exe has more than one network (a wired port + WiFi, for instance) interface that are on different networks (192.168.2.12 for wired and 10.2.10.24 WiFi, for example) BFinder201.exe doesn't broadcast on all interfaces and a board may not be found. In that case, the find dialog will come up empty:



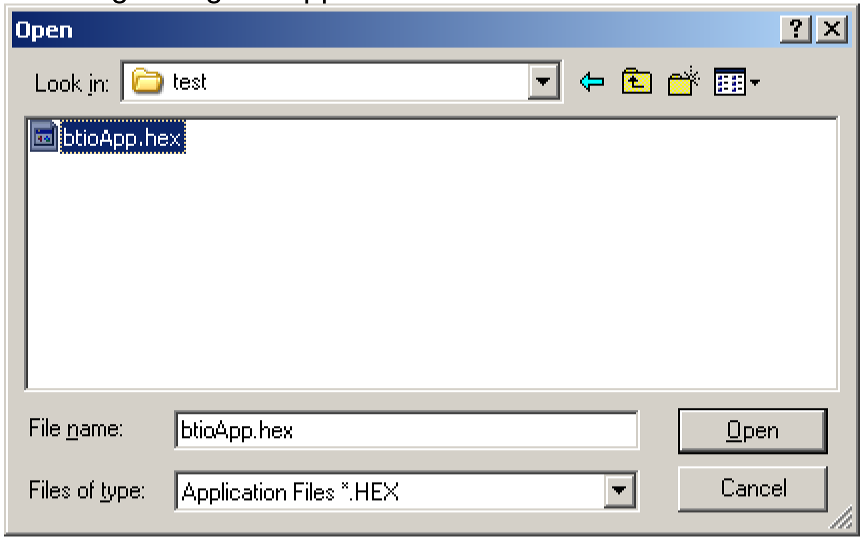
***Specifically for the case*** that the board button has been pressed when powered up, and the wired network adapter on the host computer has been set to 192.168.2.xx (e.g. 192.168.2.12), Pressing the button labeled "192.168.2.xxx" will multicast to the wired network adapter and will find the board. Wait a few seconds for the "192.168.2.xxx" button to become enabled after the find operation fails.



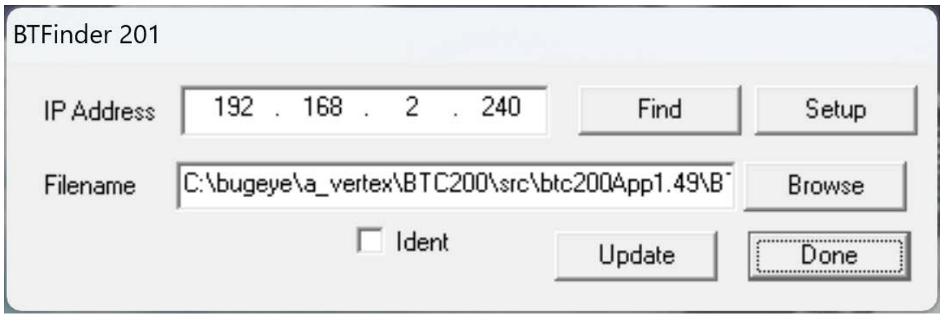
BTIO50 [00-1E-C0-D4-BF-D3] < DEFAULT1 > at 192.168.2.242 ver 1.00

## 3.2 Firmware update procedure

The BTIO50 board firmware can be updated by clicking the “Browse” button- the following dialog will appear:



Find the firmware file that was sent to you by Bugeye and click “Open”.



Click “Update” to load the board with the new firmware. After the firmware has been updated in the board the board will reboot and begin running the new firmware.

** If Updating the BTIO50 Firmware Fails:

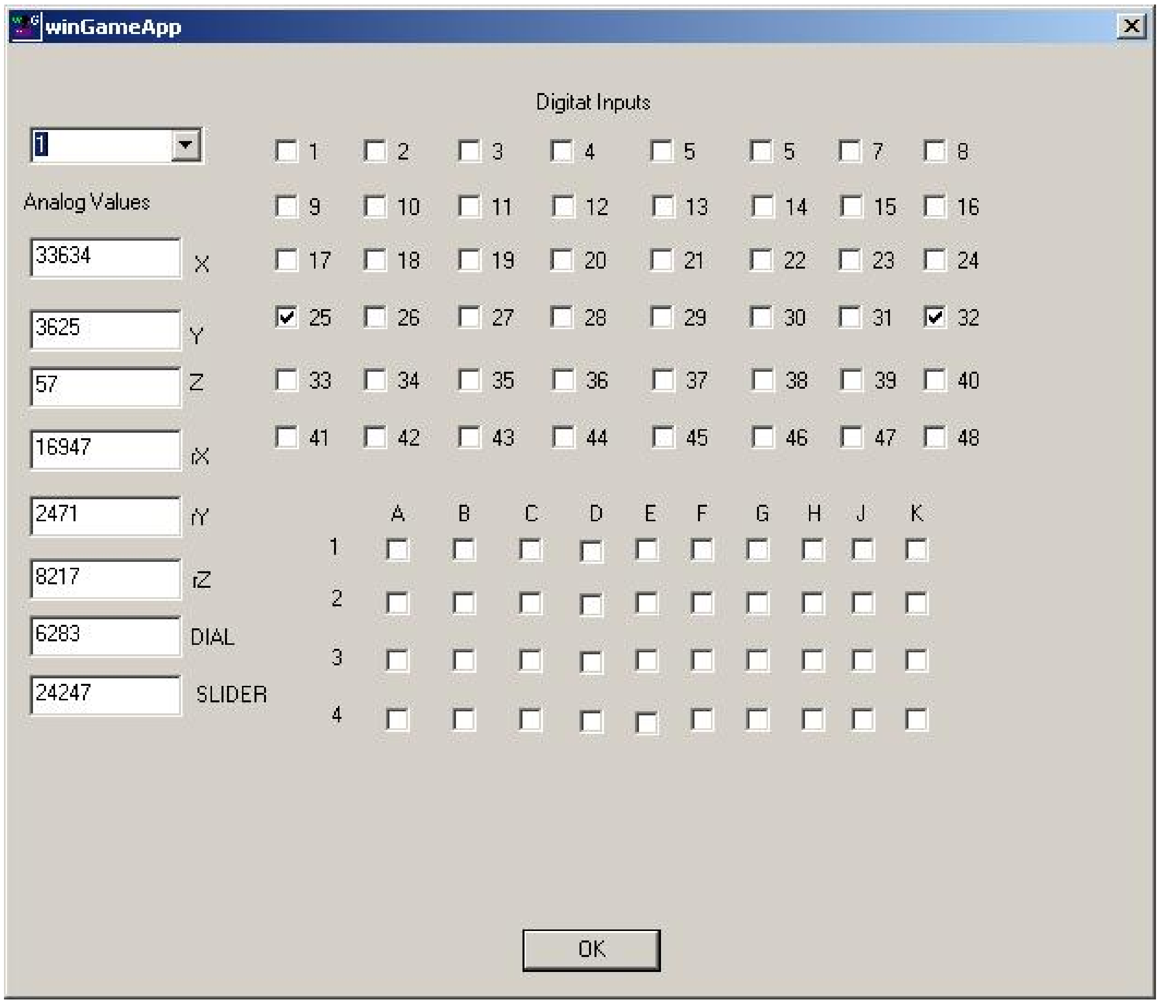
If the update fails for some reason- the following procedure will get you back operating.

- Power down the BTIO50
- Direct connect the BTIO50 to a 192.168.2.xxx network. (you can set your computer to static IP 192.168.2.20, MASK 255.255.255.0 and directly connect the BTIO50 to your computer with a cable)

- Push the BTIO50 button (see above image) and hold
- Power up the BTIO50 (e.g. plug in the USB cable)
- Release the button after 3 seconds
- Cycle the BTIO50 power again
- On BFinder201.exe application, enter 192.168.2.242 into the “IP Address” field on the main screen and click “Update”. The firmware will load into the BTIO50.

### 3.3 View USB GamePort Data with winGameApp.exe

This windows app displays up to 8 BTIO50 boards- analog and digital values.


winGameApp

1

Analog Values

| | |
|---|---|
| 33634 | X |
| 3625 | Y |
| 57 | Z |
| 16947 | rX |
| 2471 | rY |
| 8217 | rZ |
| 6283 | DIAL |
| 24247 | SLIDER |

Digitat Inputs

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| ☐ 1 | ☐ 2 | ☐ 3 | ☐ 4 | ☐ 5 | ☐ 5 | ☐ 7 | ☐ 8 |
| ☐ 9 | ☐ 10 | ☐ 11 | ☐ 12 | ☐ 13 | ☐ 14 | ☐ 15 | ☐ 16 |
| ☐ 17 | ☐ 18 | ☐ 19 | ☐ 20 | ☐ 21 | ☐ 22 | ☐ 23 | ☐ 24 |
| ✓ 25 | ☐ 26 | ☐ 27 | ☐ 28 | ☐ 29 | ☐ 30 | ☐ 31 | ✓ 32 |
| ☐ 33 | ☐ 34 | ☐ 35 | ☐ 36 | ☐ 37 | ☐ 38 | ☐ 39 | ☐ 40 |
| ☐ 41 | ☐ 42 | ☐ 43 | ☐ 44 | ☐ 45 | ☐ 46 | ☐ 47 | ☐ 48 |

| | A | B | C | D | E | F | G | H | J | K |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ |
| 2 | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ |
| 3 | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ |
| 4 | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ |

This is Win32 C++ application that can be built with Visual Studio 6.0 and greater. This application:

- Can display from 1 to 8 BTIO50 boards
- Displays the Direct Input data transmitted from the BTIO50 board

The Windows Direct X SDK must be present in the development machine to build this application.

Here is a bit of code from the application that shows the communication core:

```
// **||****************************************************************
void CWinGameAppDlg::Populate(DIJOYSTATE2 *js)
// **||****************************************************************
```

```
{
  // Populate the fields on the display
  SetDlgItemInt(IDC_ANA1,js->lX);
  SetDlgItemInt(IDC_ANA2,js->lY);
  SetDlgItemInt(IDC_ANA3,js->lZ);
  SetDlgItemInt(IDC_ANA4,js->lRx);
  SetDlgItemInt(IDC_ANA5,js->lRy);
  SetDlgItemInt(IDC_ANA6,js->lRz);
  SetDlgItemInt(IDC_ANA7,js->rglSlider[0]);
  SetDlgItemInt(IDC_ANA8,js->rglSlider[1]);

  CButton *pbut;

  pbut = (CButton *)GetDlgItem(IDC_CK1  ); pbut->SetCheck( (js->rgbButtons[
0]>1 ? 1 : 0));
  pbut = (CButton *)GetDlgItem(IDC_CK2  ); pbut->SetCheck( (js->rgbButtons[
1]>1 ? 1 : 0));
  pbut = (CButton *)GetDlgItem(IDC_CK3  ); pbut->SetCheck( (js->rgbButtons[
2]>1 ? 1 : 0));
  pbut = (CButton *)GetDlgItem(IDC_CK4  ); pbut->SetCheck( (js->rgbButtons[
3]>1 ? 1 : 0));
  pbut = (CButton *)GetDlgItem(IDC_CK5  ); pbut->SetCheck( (js->rgbButtons[
4]>1 ? 1 : 0));
  pbut = (CButton *)GetDlgItem(IDC_CK6  ); pbut->SetCheck( (js->rgbButtons[
5]>1 ? 1 : 0));
  pbut = (CButton *)GetDlgItem(IDC_CK7  ); pbut->SetCheck( (js->rgbButtons[
6]>1 ? 1 : 0));
  pbut = (CButton *)GetDlgItem(IDC_CK8  ); pbut->SetCheck( (js->rgbButtons[
7]>1 ? 1 : 0));

  pbut = (CButton *)GetDlgItem(IDC_CK9  ); pbut->SetCheck( (js->rgbButtons[
8]>1 ? 1 : 0));
  pbut = (CButton *)GetDlgItem(IDC_CK10 ); pbut->SetCheck( (js->rgbButtons[
9]>1 ? 1 : 0));
  pbut = (CButton *)GetDlgItem(IDC_CK11 ); pbut->SetCheck( (js-
>rgbButtons[10]>1 ? 1 : 0));
  pbut = (CButton *)GetDlgItem(IDC_CK12 ); pbut->SetCheck( (js-
>rgbButtons[11]>1 ? 1 : 0));
  pbut = (CButton *)GetDlgItem(IDC_CK13 ); pbut->SetCheck( (js-
>rgbButtons[12]>1 ? 1 : 0));
  pbut = (CButton *)GetDlgItem(IDC_CK14 ); pbut->SetCheck( (js-
>rgbButtons[13]>1 ? 1 : 0));
  pbut = (CButton *)GetDlgItem(IDC_CK15 ); pbut->SetCheck( (js-
>rgbButtons[14]>1 ? 1 : 0));
```

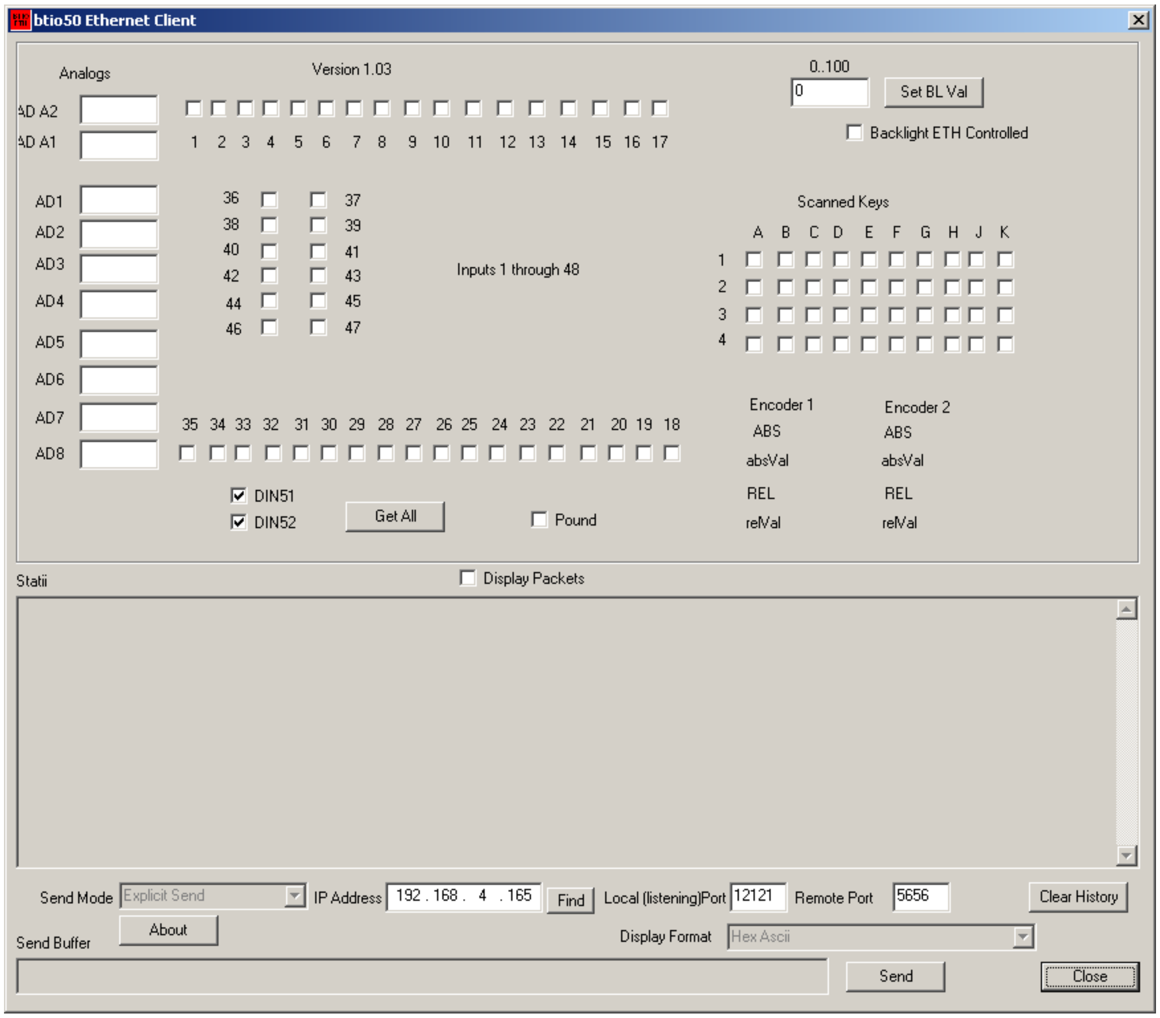
```
    pbut = (CButton *)GetDlgItem(IDC_CK16 ); pbut->SetCheck( (js-
>rgbButtons[15]>1 ? 1 : 0));

    pbut = (CButton *)GetDlgItem(IDC_CK17 ); pbut->SetCheck( (js-
>rgbButtons[16]>1 ? 1 : 0));
    pbut = (CButton *)GetDlgItem(IDC_CK18 ); pbut->SetCheck( (js-
>rgbButtons[17]>1 ? 1 : 0));
    pbut = (CButton *)GetDlgItem(IDC_CK19 ); pbut->SetCheck( (js-
>rgbButtons[18]>1 ? 1 : 0));
    pbut = (CButton *)GetDlgItem(IDC_CK20 ); pbut->SetCheck( (js-
>rgbButtons[19]>1 ? 1 : 0));
    pbut = (CButton *)GetDlgItem(IDC_CK21 ); pbut->SetCheck( (js-
>rgbButtons[20]>1 ? 1 : 0));
    pbut = (CButton *)GetDlgItem(IDC_CK22 ); pbut->SetCheck( (js-
>rgbButtons[21]>1 ? 1 : 0));
    pbut = (CButton *)GetDlgItem(IDC_CK23 ); pbut->SetCheck( (js-
>rgbButtons[22]>1 ? 1 : 0));
    pbut = (CButton *)GetDlgItem(IDC_CK24 ); pbut->SetCheck( (js-
>rgbButtons[23]>1 ? 1 : 0));

    pbut = (CButton *)GetDlgItem(IDC_CK25 ); pbut->SetCheck( (js-
>rgbButtons[24]>1 ? 1 : 0));
    pbut = (CButton *)GetDlgItem(IDC_CK26 ); pbut->SetCheck( (js-
>rgbButtons[25]>1 ? 1 : 0));
    pbut = (CButton *)GetDlgItem(IDC_CK27 ); pbut->SetCheck( (js-
>rgbButtons[26]>1 ? 1 : 0));
    pbut = (CButton *)GetDlgItem(IDC_CK28 ); pbut->SetCheck( (js-
>rgbButtons[27]>1 ? 1 : 0));
    pbut = (CButton *)GetDlgItem(IDC_CK29 ); pbut->SetCheck( (js-
>rgbButtons[28]>1 ? 1 : 0));
    pbut = (CButton *)GetDlgItem(IDC_CK30 ); pbut->SetCheck( (js-
>rgbButtons[29]>1 ? 1 : 0));
    pbut = (CButton *)GetDlgItem(IDC_CK31 ); pbut->SetCheck( (js-
>rgbButtons[30]>1 ? 1 : 0));
    pbut = (CButton *)GetDlgItem(IDC_CK32 ); pbut->SetCheck( (js-
>rgbButtons[31]>1 ? 1 : 0));

    pbut = (CButton *)GetDlgItem(IDC_CK33 ); pbut->SetCheck( (js-
>rgbButtons[32]>1 ? 1 : 0));
    pbut = (CButton *)GetDlgItem(IDC_CK34 ); pbut->SetCheck( (js-
>rgbButtons[33]>1 ? 1 : 0));
    pbut = (CButton *)GetDlgItem(IDC_CK35 ); pbut->SetCheck( (js-
>rgbButtons[34]>1 ? 1 : 0));
    pbut = (CButton *)GetDlgItem(IDC_CK36 ); pbut->SetCheck( (js-
>rgbButtons[35]>1 ? 1 : 0));
```

```
   pbut = (CButton *)GetDlgItem(IDC_CK37 ); pbut->SetCheck( (js-
>rgbButtons[36]>1 ? 1 : 0));
   pbut = (CButton *)GetDlgItem(IDC_CK38 ); pbut->SetCheck( (js-
>rgbButtons[37]>1 ? 1 : 0));
   pbut = (CButton *)GetDlgItem(IDC_CK39 ); pbut->SetCheck( (js-
>rgbButtons[38]>1 ? 1 : 0));
   pbut = (CButton *)GetDlgItem(IDC_CK40 ); pbut->SetCheck( (js-
>rgbButtons[39]>1 ? 1 : 0));

   pbut = (CButton *)GetDlgItem(IDC_CK41 ); pbut->SetCheck( (js-
>rgbButtons[40]>1 ? 1 : 0));
   pbut = (CButton *)GetDlgItem(IDC_CK42 ); pbut->SetCheck( (js-
>rgbButtons[41]>1 ? 1 : 0));
   pbut = (CButton *)GetDlgItem(IDC_CK43 ); pbut->SetCheck( (js-
>rgbButtons[42]>1 ? 1 : 0));
   pbut = (CButton *)GetDlgItem(IDC_CK44 ); pbut->SetCheck( (js-
>rgbButtons[43]>1 ? 1 : 0));
   pbut = (CButton *)GetDlgItem(IDC_CK45 ); pbut->SetCheck( (js-
>rgbButtons[44]>1 ? 1 : 0));
   pbut = (CButton *)GetDlgItem(IDC_CK46 ); pbut->SetCheck( (js-
>rgbButtons[45]>1 ? 1 : 0));
   pbut = (CButton *)GetDlgItem(IDC_CK47 ); pbut->SetCheck( (js-
>rgbButtons[46]>1 ? 1 : 0));
   pbut = (CButton *)GetDlgItem(IDC_CK48 ); pbut->SetCheck( (js-
>rgbButtons[47]>1 ? 1 : 0));
   //
   // scans
   //
   pbut = (CButton *)GetDlgItem(IDC_CK_A1 );  pbut->SetCheck( (js-
>rgbButtons[48]>1 ? 1 : 0));
   pbut = (CButton *)GetDlgItem(IDC_CK_A2 );  pbut->SetCheck( (js-
>rgbButtons[49]>1 ? 1 : 0));
   pbut = (CButton *)GetDlgItem(IDC_CK_A3 );  pbut->SetCheck( (js-
>rgbButtons[50]>1 ? 1 : 0));
   pbut = (CButton *)GetDlgItem(IDC_CK_A4 );  pbut->SetCheck( (js-
>rgbButtons[51]>1 ? 1 : 0));
   pbut = (CButton *)GetDlgItem(IDC_CK_B1  );  pbut->SetCheck( (js-
>rgbButtons[52]>1 ? 1 : 0));
   pbut = (CButton *)GetDlgItem(IDC_CK_B2  );  pbut->SetCheck( (js-
>rgbButtons[53]>1 ? 1 : 0));
   pbut = (CButton *)GetDlgItem(IDC_CK_B3  );  pbut->SetCheck( (js-
>rgbButtons[54]>1 ? 1 : 0));
   pbut = (CButton *)GetDlgItem(IDC_CK_B4  );  pbut->SetCheck( (js-
>rgbButtons[55]>1 ? 1 : 0));
```

```
  pbut = (CButton *)GetDlgItem(IDC_CK_C1 ); pbut->SetCheck( (js-
>rgbButtons[56]>1 ? 1 : 0));
  pbut = (CButton *)GetDlgItem(IDC_CK_C2 ); pbut->SetCheck( (js-
>rgbButtons[57]>1 ? 1 : 0));
  pbut = (CButton *)GetDlgItem(IDC_CK_C3 ); pbut->SetCheck( (js-
>rgbButtons[58]>1 ? 1 : 0));
  pbut = (CButton *)GetDlgItem(IDC_CK_C4 ); pbut->SetCheck( (js-
>rgbButtons[59]>1 ? 1 : 0));
  pbut = (CButton *)GetDlgItem(IDC_CK_D1 ); pbut->SetCheck( (js-
>rgbButtons[60]>1 ? 1 : 0));
  pbut = (CButton *)GetDlgItem(IDC_CK_D2 ); pbut->SetCheck( (js-
>rgbButtons[61]>1 ? 1 : 0));
  pbut = (CButton *)GetDlgItem(IDC_CK_D3 ); pbut->SetCheck( (js-
>rgbButtons[62]>1 ? 1 : 0));
  pbut = (CButton *)GetDlgItem(IDC_CK_D4 ); pbut->SetCheck( (js-
>rgbButtons[63]>1 ? 1 : 0));

  pbut = (CButton *)GetDlgItem(IDC_CK_E1 ); pbut->SetCheck( (js-
>rgbButtons[64]>1 ? 1 : 0));
  pbut = (CButton *)GetDlgItem(IDC_CK_E2 ); pbut->SetCheck( (js-
>rgbButtons[65]>1 ? 1 : 0));
  pbut = (CButton *)GetDlgItem(IDC_CK_E3 ); pbut->SetCheck( (js-
>rgbButtons[66]>1 ? 1 : 0));
  pbut = (CButton *)GetDlgItem(IDC_CK_E4 ); pbut->SetCheck( (js-
>rgbButtons[67]>1 ? 1 : 0));
  pbut = (CButton *)GetDlgItem(IDC_CK_F1 ); pbut->SetCheck( (js-
>rgbButtons[68]>1 ? 1 : 0));
  pbut = (CButton *)GetDlgItem(IDC_CK_F2 ); pbut->SetCheck( (js-
>rgbButtons[69]>1 ? 1 : 0));
  pbut = (CButton *)GetDlgItem(IDC_CK_F3 ); pbut->SetCheck( (js-
>rgbButtons[70]>1 ? 1 : 0));
  pbut = (CButton *)GetDlgItem(IDC_CK_F4 ); pbut->SetCheck( (js-
>rgbButtons[71]>1 ? 1 : 0));

  pbut = (CButton *)GetDlgItem(IDC_CK_G1 ); pbut->SetCheck( (js-
>rgbButtons[72]>1 ? 1 : 0));
  pbut = (CButton *)GetDlgItem(IDC_CK_G2 ); pbut->SetCheck( (js-
>rgbButtons[73]>1 ? 1 : 0));
  pbut = (CButton *)GetDlgItem(IDC_CK_G3 ); pbut->SetCheck( (js-
>rgbButtons[74]>1 ? 1 : 0));
  pbut = (CButton *)GetDlgItem(IDC_CK_G4 ); pbut->SetCheck( (js-
>rgbButtons[75]>1 ? 1 : 0));
  pbut = (CButton *)GetDlgItem(IDC_CK_H1 ); pbut->SetCheck( (js-
>rgbButtons[76]>1 ? 1 : 0));
```

```
   pbut = (CButton *)GetDlgItem(IDC_CK_H2 ); pbut->SetCheck( (js-
>rgbButtons[77]>1 ? 1 : 0));
   pbut = (CButton *)GetDlgItem(IDC_CK_H3 ); pbut->SetCheck( (js-
>rgbButtons[78]>1 ? 1 : 0));
   pbut = (CButton *)GetDlgItem(IDC_CK_H4 ); pbut->SetCheck( (js-
>rgbButtons[79]>1 ? 1 : 0));

   pbut = (CButton *)GetDlgItem(IDC_CK_J1 ); pbut->SetCheck( (js-
>rgbButtons[80]>1 ? 1 : 0));
   pbut = (CButton *)GetDlgItem(IDC_CK_J2 ); pbut->SetCheck( (js-
>rgbButtons[81]>1 ? 1 : 0));
   pbut = (CButton *)GetDlgItem(IDC_CK_J3 ); pbut->SetCheck( (js-
>rgbButtons[82]>1 ? 1 : 0));
   pbut = (CButton *)GetDlgItem(IDC_CK_J4 ); pbut->SetCheck( (js-
>rgbButtons[83]>1 ? 1 : 0));
   pbut = (CButton *)GetDlgItem(IDC_CK_K1 ); pbut->SetCheck( (js-
>rgbButtons[84]>1 ? 1 : 0));
   pbut = (CButton *)GetDlgItem(IDC_CK_K2 ); pbut->SetCheck( (js-
>rgbButtons[85]>1 ? 1 : 0));
   pbut = (CButton *)GetDlgItem(IDC_CK_K3 ); pbut->SetCheck( (js-
>rgbButtons[86]>1 ? 1 : 0));
   pbut = (CButton *)GetDlgItem(IDC_CK_K4 ); pbut->SetCheck( (js-
>rgbButtons[87]>1 ? 1 : 0));
}

// **||*****************************************************************
void CWinGameAppDlg::OnTimer(UINT nIDEvent)
// **||*****************************************************************
{
   DIJOYSTATE2 js;
   joysticks[m_iPresJoy]->poll(&js);

   Populate(&js);

         CDialog::OnTimer(nIDEvent);
}
```

## 3.3 View Ethernet channel digital and analog inputs with windows application: btio50ETHclient.exe



Version 1.03

0..100: 0

[✓] DIN51

[✓] DIN52

Send Mode: Explicit Send

IP Address: 192 . 168 . 4 . 165

Local (listening)Port: 12121

Remote Port: 5656

Display Format: Hex Ascii

This is an unmanaged C++ MFC application that can be built with any Microsoft Visual Studio starting with VC++ 6.0. The features of this app are:

- Finding the BTIO50 on the network
- Sending one MSG_GET_ALL at a time along with the ability to see the messages printed
- “pounding” messages – illustrating the method of polling the BTIO50 at the Ethernet port.

The fields DOUT51 and DOUT52 are used to change pins JP1-pin1 and JP1-pin2 (respectively) into outputs and the level of the pin(s) matches the state of the check boxes. If a box is unchecked, the pin is pulled low, and vice-versa.

Board art revision Rev. D and above:

The checkbox “Backlight ETH Controlled” is unchecked when any BTIO50 backlight load is attached (see schematic later in this document). If the backlight brightness level is to be controlled via Ethernet messaging, click the checkbox and enter a value 0..100 into the field.

Here is a bit of the code showing the parsing of the incoming message and breaking out of the parameters.

```
// ******************************************************************************
int ParseBugeyePacket(char *pData)
{
  int i,func, len, cksum,port;
  int dck=0;


  if (!((pData[0] == STX) && (pData[1] == '|')&& (pData[2] == BUG_PROTO_6)))
  {
    // this is not bugeye protocol!
    return 1;
  }
  // step over STX,BAR
  sscanf(&pData[3],"%04X%04X%04X",
    &port,
    &func,
    &len);
  //
  // port is the port that the sender is listening on
  //
  i = 3 + 13 + len + 4 -1;
  if (!((pData[i] == '|') && (pData[i+1] == ETX) ))
  {
    // this is not bugeye protocol!
    return 2;
  }
  //
  // this is a good command
  sscanf(&pData[3+13+len],"%04X",&cksum);
  // is there any data in this payload?
  //
  dck = STX + '|' + BUG_PROTO_6;
  //
  for(i=3;i<(3+12);i++)
  {
    dck += pData[i];
  }
  if (len)
  {
    // there is data
    if (len > MAX_BUG_6_LEN)
    {
      // something is wrong, there are no data messages this long in bugeye
protocol!
      return 3;
    }
    //
    memcpy(payload,&pData[3+12],len);
    for(i=0;i<len;i++)
    {
      dck += payload[i];
    }
  }  // if len
  //
  // check the checksum for data corruption
  if (dck != cksum)
  {
    // checksum fails
    return 5;
  }
  //
  // at this point we have enough data to pull a message
  //
  //
  switch(func)
  {
  case MSG_GET_ALL:
    {

sscanf(payload,"%04X%04X%04X%04X%04X%04X%04X%04X%04X%04X%0
2X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X
%02X%02X%02X",
      &analogs[0],
      &analogs[1],
      &analogs[2],
      &analogs[3],
      &analogs[4],
      &analogs[5],
      &analogs[6],
      &analogs[7],
      &analogs[8],
      &analogs[9],
      //
      &inputs[0],
      &inputs[1],
      &inputs[2],
      &inputs[3],
      &inputs[4],
      &inputs[5],
      //
      &scans[0],
      &scans[1],
      &scans[2],
      &scans[3],
      &scans[4],
      //
      &future[0],
      &future[1],
      //
      &bEnc1Abs,
      &bEnc2Abs,
      &bEnc1Rel,
      &bEnc2Rel);

    break;
    }
  default:
    ;
  } // sw
  return 0;
}
```

## 3.4 Scan line reporting

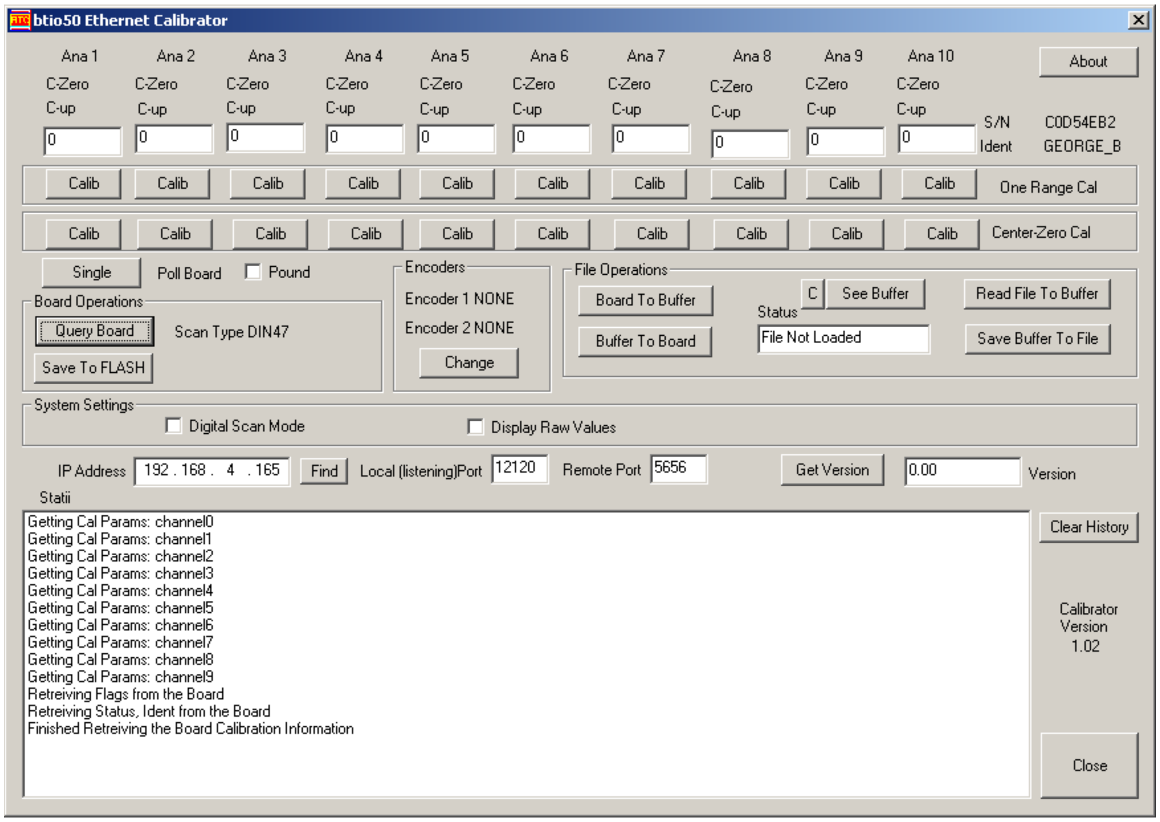
Scanned Keys hardware lines to digital scan byte and specific bit assignment

| Driving pin | Scanned pin | Title on App | Scan byte | Bit |
|---|---|---|---|---|
| Pin TB1-5 | Pin TB1-9 | A1 | 0 | 1 |
| Pin TB1-5 | Pin TB1-10 | B1 | 0 | 2 |
| Pin TB1-5 | Pin TB1-11 | C1 | 0 | 3 |
| Pin TB1-5 | Pin TB1-12 | D1 | 0 | 4 |
| Pin TB1-5 | Pin TB1-13 | E1 | 0 | 5 |
| Pin TB1-5 | Pin TB1-14 | F1 | 0 | 6 |
| Pin TB1-5 | Pin TB1-15 | G1 | 0 | 7 |
| Pin TB1-5 | Pin TB1-16 | H1 | 0 | 8 |
| Pin TB1-5 | Pin TB1-17 | J1 | 1 | 1 |
| Pin TB1-5 | Pin TB2-1 (D18) | K1 | 1 | 2 |
| | | | | |
| Pin TB1-6 | Pin TB1-9 | A2 | 1 | 3 |
| Pin TB1-6 | Pin TB1-10 | B2 | 1 | 4 |
| Pin TB1-6 | Pin TB1-11 | C2 | 1 | 5 |
| Pin TB1-6 | Pin TB1-12 | D2 | 1 | 6 |
| Pin TB1-6 | Pin TB1-13 | E2 | 1 | 7 |
| Pin TB1-6 | Pin TB1-14 | F2 | 1 | 8 |
| Pin TB1-6 | Pin TB1-15 | G2 | 2 | 1 |
| Pin TB1-6 | Pin TB1-16 | H2 | 2 | 2 |
| Pin TB1-6 | Pin TB1-17 | J2 | 2 | 3 |
| Pin TB1-6 | Pin TB2-1 (D18) | K2 | 2 | 4 |
| | | | | |
| Pin TB1-7 | Pin TB1-9 | A3 | 2 | 5 |
| Pin TB1-7 | Pin TB1-10 | B3 | 2 | 6 |
| Pin TB1-7 | Pin TB1-11 | C3 | 2 | 7 |
| Pin TB1-7 | Pin TB1-12 | D3 | 2 | 8 |
| Pin TB1-7 | Pin TB1-13 | E3 | 3 | 1 |
| Pin TB1-7 | Pin TB1-14 | F3 | 3 | 2 |
| Pin TB1-7 | Pin TB1-15 | G3 | 3 | 3 |
| Pin TB1-7 | Pin TB1-16 | H3 | 3 | 4 |
| Pin TB1-7 | Pin TB1-17 | J3 | 3 | 5 |
| Pin TB1-7 | Pin TB2-1 (D18) | K3 | 3 | 6 |
| | | | | |
| Pin TB1-8 | Pin TB1-9 | A4 | 3 | 7 |
| Pin TB1-8 | Pin TB1-10 | B4 | 3 | 8 |
| Pin TB1-8 | Pin TB1-11 | C4 | 4 | 1 |
| Pin TB1-8 | Pin TB1-12 | D4 | 4 | 2 |

| | | | | |
|---|---|---|---|---|
| Pin TB1-8 | Pin TB1-13 | E4 | 4 | 3 |
| Pin TB1-8 | Pin TB1-14 | F4 | 4 | 4 |
| Pin TB1-8 | Pin TB1-15 | G4 | 4 | 5 |
| Pin TB1-8 | Pin TB1-16 | H4 | 4 | 6 |
| Pin TB1-8 | Pin TB1-17 | J4 | 4 | 7 |
| Pin TB1-8 | Pin TB2-1 (D18) | K4 | 4 | 8 |

## 3.5 BTIO50 Calibration Application

The BTIO50 board is setup and calibrated through Ethernet by the application btio50EthCalib.exe. Any changes made in this application are initially located in the board's RAM and for the changes made to be permanently applied to the BTIO50, the "Save To FLASH" button must be pressed at the end of the setup/calibration session. The calibration constants may be stored to a file and the same file can be read in to clone another BTIO50 board.



| | Ana 1 | Ana 2 | Ana 3 | Ana 4 | Ana 5 | Ana 6 | Ana 7 | Ana 8 | Ana 9 | Ana 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| C-Zero / C-up | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

S/N C0D54EB2
Ident GEORGE_B
Scan Type DIN47
Encoder 1 NONE
Encoder 2 NONE
Status: File Not Loaded
IP Address: 192 . 168 . 4 . 165
Local (listening)Port: 12120
Remote Port: 5656
Version: 0.00
Calibrator Version 1.02

```
Getting Cal Params: channel0
Getting Cal Params: channel1
Getting Cal Params: channel2
Getting Cal Params: channel3
Getting Cal Params: channel4
Getting Cal Params: channel5
Getting Cal Params: channel6
Getting Cal Params: channel7
Getting Cal Params: channel8
Getting Cal Params: channel9
Retreiving Flags from the Board
Retreiving Status, Ident from the Board
Finished Retreiving the Board Calibration Information
```

To use the calibration application

- Execute btio50EthCalib.exe
- Click "Find", select the board from the list that you are calibrating. If the list is empty, use application "BFinder201.exe" to discover the board's IP address. If necessary, follow the above section on using the button on the board to reset the IP address so the board can be found.
- Enter the remote port that the BTIO50 board is listening on. This port is originally set and read by the BFinder201.exe program.
- Click button "Query Board" to gather all of the calibration constants of the target BTIO50 board into this application's memory. If this fails, go back to step 2 and make sure the board can be found and the remote reception port is correct

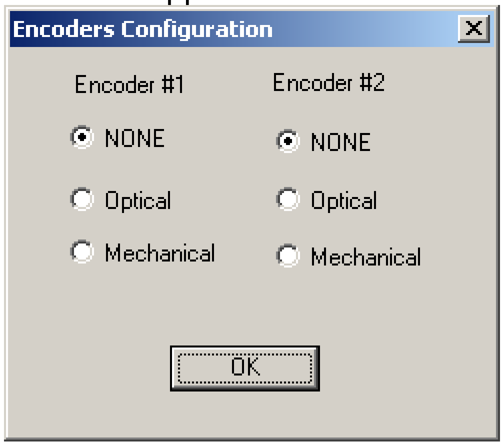
- Now the following operations are possible:
    1. Setup analog channels
    2. Specify din47 or scan40 digital inputs mode
    3. Specify if any quad encoders are installed, and which type of encoder
    4. Save changed values to the board's FLASH memory
    5. Save the app's buffered calibration values into a file
    6. Load a set of calibration values from a file into the app's buffer
    7. Copy the app's buffer to the attached board

### 3.5.1 Setup the board's digital input mode

- Click "Digital Scan Mode" to specify scan40 mode
- Clear check from "Digital Scan Mode" to specify din47 mode

### 3.5.2 Specify one or more connected quad encoder

- Click "Change" in the "Encoders" panel, the following dialog appears:



Encoders Configuration

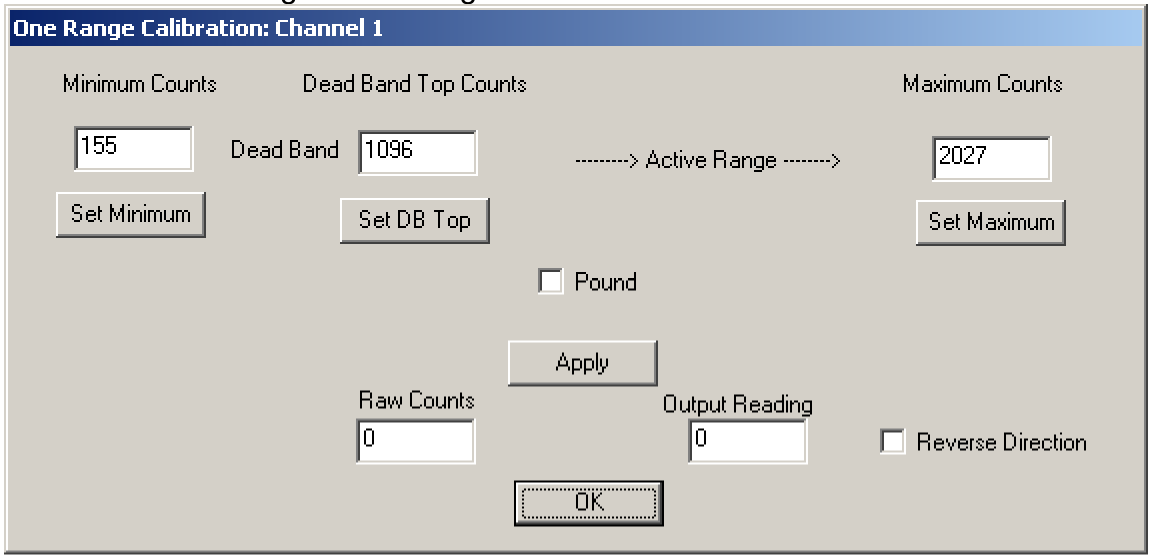
Encoder #1: [✓ NONE] Optical Mechanical

Encoder #2: [✓ NONE] Optical Mechanical

- Click the encoder type(s) that is(are) connected to the board
- Click OK

### 3.5.3 Perform One Range Analog Input calibration

- Start on the main screen. Click “pound” and move the control you want to calibrate- find the number that is changing- this is your channel
- Click the proper channel “One Range Cal” group “Calib” button to get this dialog:

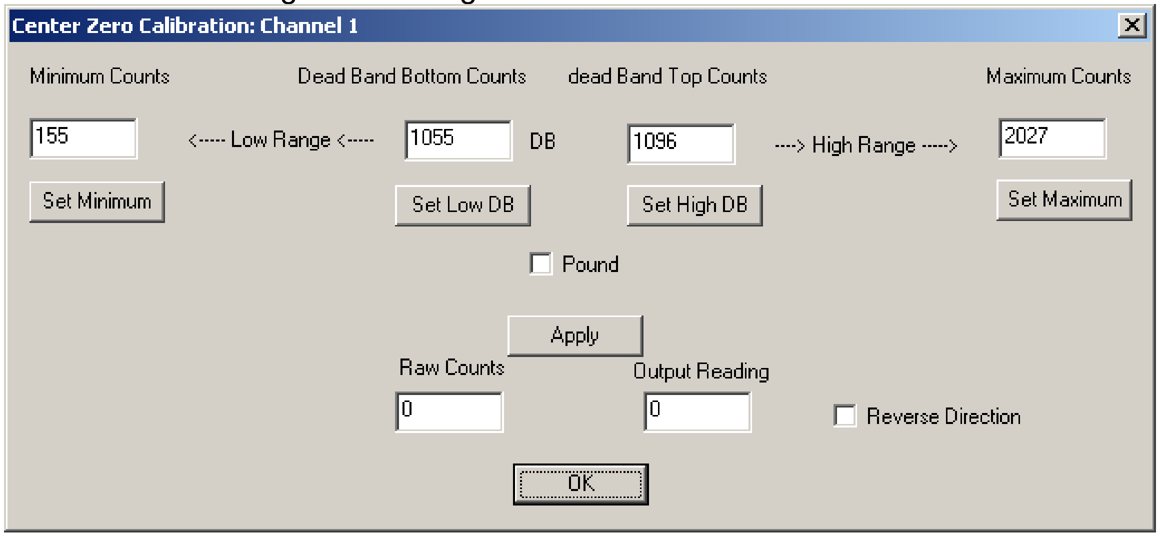

One Range Calibration: Channel 1

Minimum Counts: 155
Dead Band Top Counts: 1096
Maximum Counts: 2027
Raw Counts: 0
Output Reading: 0
Pound: ☐
Reverse Direction: ☐

- Click “pound” to get the raw count stream. These numbers show up in the “Raw Counts” window
- Move the control to minimum position and click “Set Minimum”
- Move the control to the maximum position and click “Set Maximum”
- If you wish to have a quiet zone or dead band at the minimum end, move the control to the spot just above minimum where the numbers should begin to climb, then click “Set DB Top”
- Click “apply” to send the three values to the board
- Click “pound” to start up the communication with the board
- Move the control through its travel and check the “Output Reading”
- If you want to slightly tweak any of the numbers, unclick pound and enter any value into “Minimum Counts” or “Set DB Top” or “Set Maximum” then click “Apply”
- If the readout needs to be in the opposite direction, click “Reverse Direction” and then click “Apply”

## 3.5.4 Perform Center-Zero Analog Input Calibration

- Click “pound” on the main screen and move the control you want to calibrate- find the number that is changing- this is your channel
- Click the proper channel “Center-Zero Cal” group “Calib” button to get this dialog:



Minimum Counts: 155
Dead Band Bottom Counts: 1055
dead Band Top Counts: 1096
Maximum Counts: 2027
Raw Counts: 0
Output Reading: 0

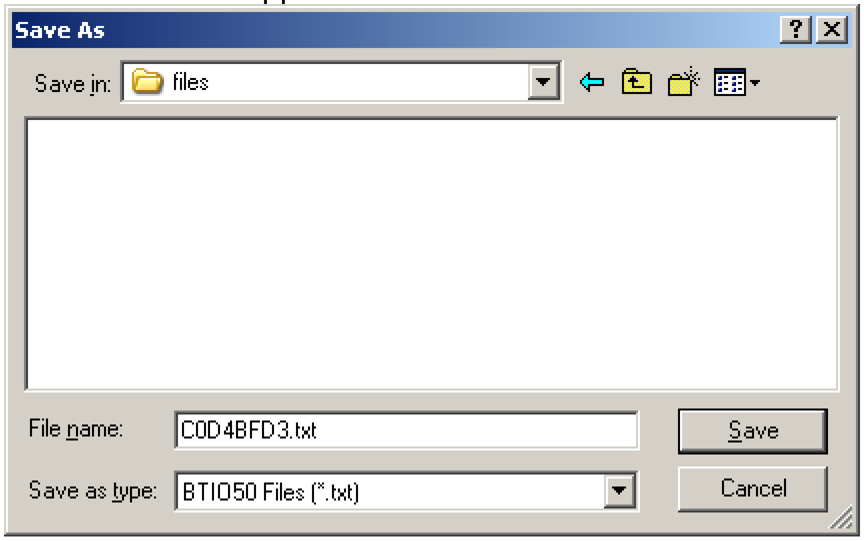
- Click “pound” to get the raw count stream. These numbers show up in the “Raw Counts” window
- Move the control to minimum position and click “Set Minimum”
- Move the control to the maximum position and click “Set Maximum”
- Move the control to the spot just below zero (the bottom of the dead band) and click “Set Low DB”
- Move the control to the spot just above zero where the dead band ends and click “Set High DB”.
- Click “apply” to send the four values to the board
- Click “pound” to start up the communication with the board
- Move the control through its travel and check the “Output Reading”
- If you want to slightly tweak any of the numbers, unclick pound and enter any value into “Minimum Counts” or “Dead Band Bottom Counts ” or “Dead Band Top Counts” or “Maximum Counts” then click “Apply”
- If the readout needs to be in the opposite direction, click “Reverse Direction” and then click “Apply”

### 3.5.5 Save calibration work to the board's flash memory

- Click "**Save To Flash**" on the main screen. If you plan on keeping the work you have done in this application, click this before closing the Calibration application or powering down the board.

### 3.5.6 Save calibration work to a file

- After performing any analog calibration or digital setup changes, the board's calibration parameters should be saved to a file for future use.
- Click the button "Save Buffer To File". The following dialog appears:



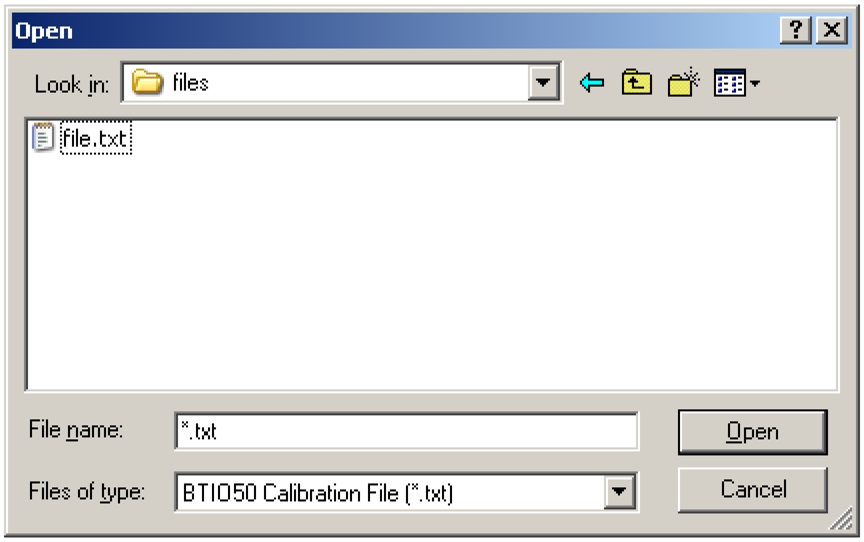
File name: C0D4BFD3.txt
Save as type: BTIO50 Files (*.txt)

- The file save dialog is automatically filled in with the board's serial number (the last 4 bytes of the MAC address). Specify the directory to save the file in and click "Save"
- Archive this file for the future cloning of another board, if ever necessary.

### 3.5.7 Read in calibration information from a file- clone a board

By reading in a previous file into this program's buffers and sending them to the board, a board can be setup to replace a previous board.

- Find and Query the new board based on the steps at the start of this section.
- Click "Read File To Buffer", this dialog shows up:



File name: *.txt

Files of type: BTIO50 Calibration File (*.txt)

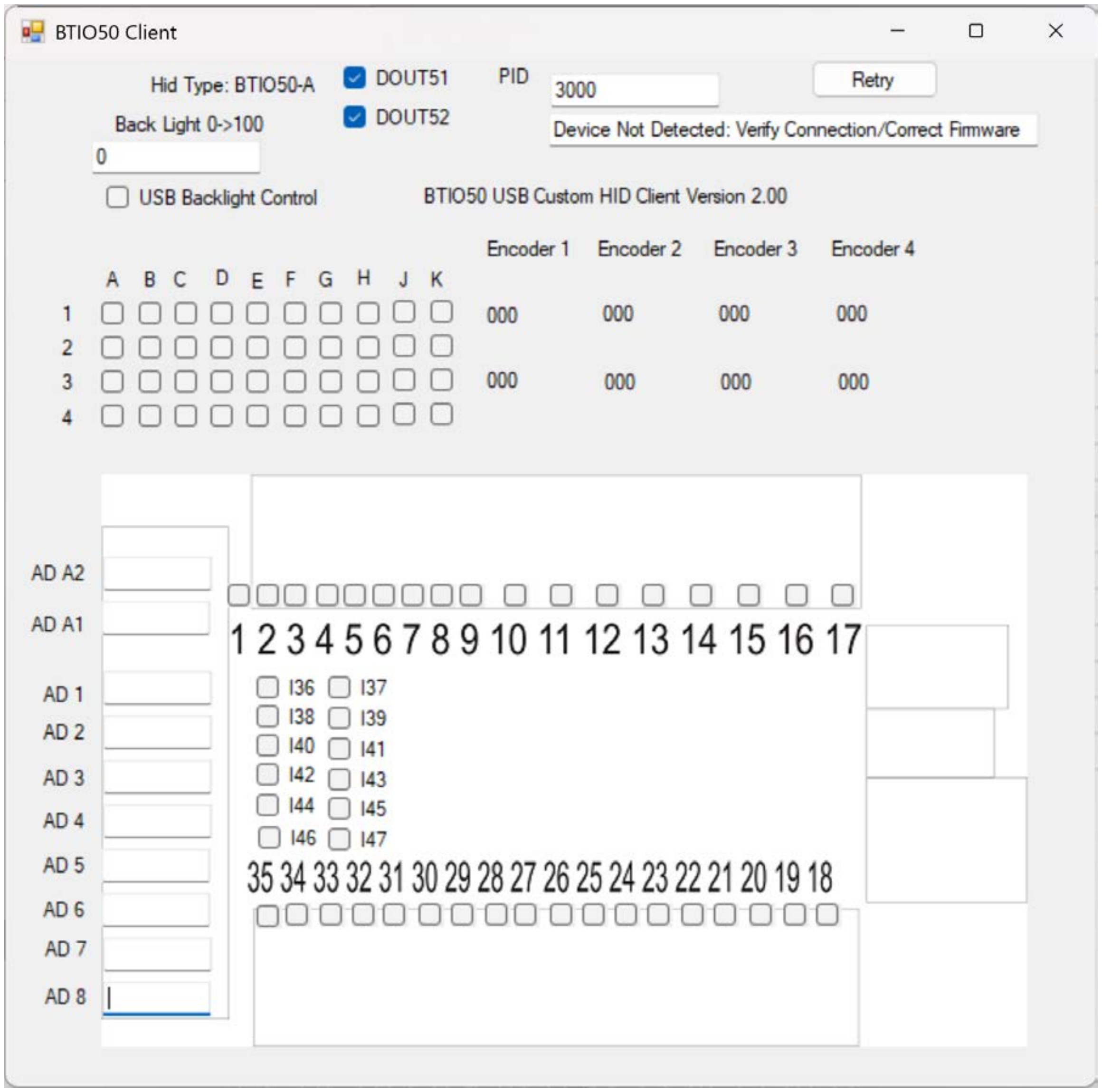
- The file open dialog is automatically filled in with the file type (.txt). Specify the directory that the file is in, click on the file and click “Open”
- Back on the main Calibration screen, click the button “Buffer To Board”
- Now click the button “Save To Flash”
- The board is now a copy of the loaded file’s previous board calibration.

#### 3.5.8 Display Raw Values checkbox

By checking the box “Display Raw Values” the BTIO50 will send out the raw values read from the A->D channels. The range will be 0..2nnn. The raw values are sent to both analog clients, USB and Ethernet.  To place the board into either raw or scaled mode permanently, choose the mode with the checkbox and click the “Save To FLASH” button.

## 3.6 BTIO50 Custom Hid Client Application

View USB channel digital and analog inputs with Windows application: btio50USBClient200.exe



BTIO50 Client

Hid Type: BTIO50-A

[✓] DOUT51

[✓] DOUT52

PID: 3000

Back Light 0->100: 0

[ ] USB Backlight Control

Device Not Detected: Verify Connection/Correct Firmware

BTIO50 USB Custom HID Client Version 2.00

This is a managed C++ application that can be built with Visual Studio 5.0 and greater. The salient features of this application are:

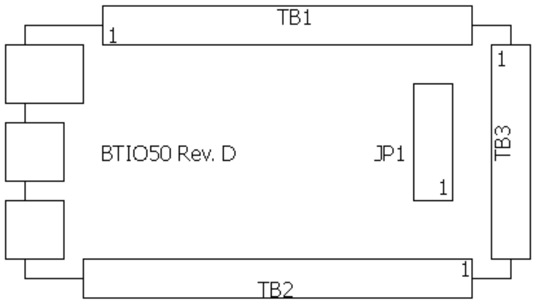
- Discovering the HID device in the OS list
- Unpacking the data transmitted from the BTIO50 board

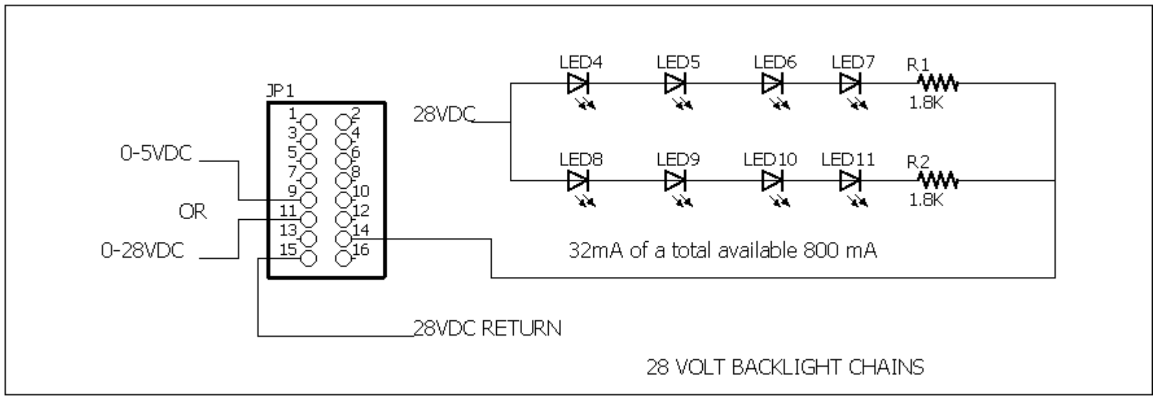
Here is a bit of code from the application that shows the communication core:

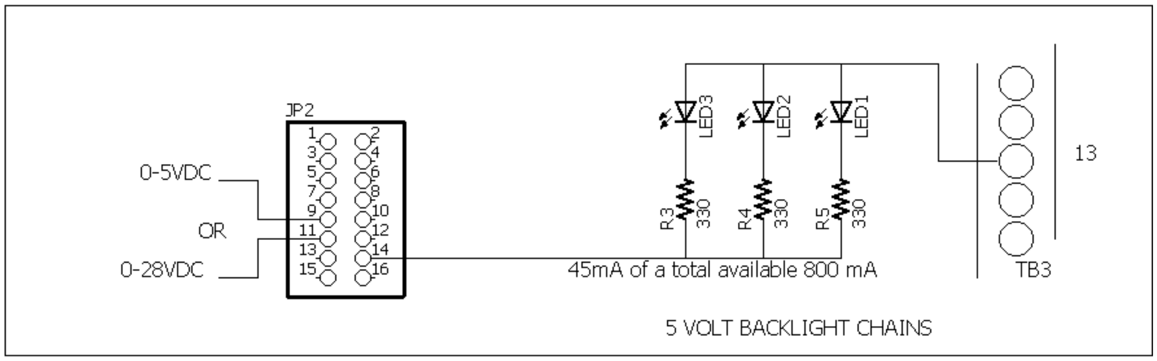
```
//To get the present GET data we send over the SET information
OUTBuffer[0] = 0;
OUTBuffer[1] = 1;
if(WriteFile(WriteHandleToUSBDevice, &OUTBuffer, 65, &BytesWritten, 0))
        //Blocking function, unless an "overlapped" structure is used
{
  INBuffer[0] = 0;
  //Now get the response packet from the firmware.
  if(ReadFile(ReadHandleToUSBDevice, &INBuffer, 65, &BytesRead, 0))
        //Blocking function, unless an "overlapped" structure is used
  {
        //INBuffer[0] is the report ID, which we don't care about.
        //
        wAna[0] = (INBuffer[ 4] << 8) | INBuffer[ 5];
        wAna[1] = (INBuffer[ 6] << 8) | INBuffer[ 7];
        wAna[2] = (INBuffer[ 8] << 8) | INBuffer[ 9];
        wAna[3] = (INBuffer[10] << 8) | INBuffer[11];
        wAna[4] = (INBuffer[12] << 8) | INBuffer[13];
        wAna[5] = (INBuffer[14] << 8) | INBuffer[15];
        wAna[6] = (INBuffer[16] << 8) | INBuffer[17];
        wAna[7] = (INBuffer[18] << 8) | INBuffer[19];
        wAna[8] = (INBuffer[20] << 8) | INBuffer[21];
        wAna[9] = (INBuffer[22] << 8) | INBuffer[23];

        bDigitals[0] = INBuffer[24]; // dig
        bDigitals[1] = INBuffer[25]; // dig
        bDigitals[2] = INBuffer[26]; // dig
        bDigitals[3] = INBuffer[27]; // dig
        bDigitals[4] = INBuffer[28]; // dig
        bDigitals[5] = INBuffer[29]; // dig
                                          //
        bDigitals[6] = INBuffer[30]; // scan
        bDigitals[7] = INBuffer[31]; // scan
        bDigitals[8] = INBuffer[32]; // scan
        bDigitals[9] = INBuffer[33]; // scan
        bDigitals[10] = INBuffer[34]; // scan
                                          //
        bDigitals[11] = INBuffer[35]; // future
        bDigitals[12] = INBuffer[36]; // future
                                          //
        iEncoder1Abs = INBuffer[37]; // encoder 1 abs
        iEncoder2Abs = INBuffer[38]; // encoder 2 abs
        iEncoder1Rel = INBuffer[39]; // encoder 1 rel
        iEncoder2Rel = INBuffer[40]; // encoder 2 rel
}
```

## Backlight Hardware

BTIO50 Backlight Support Rev. D and above



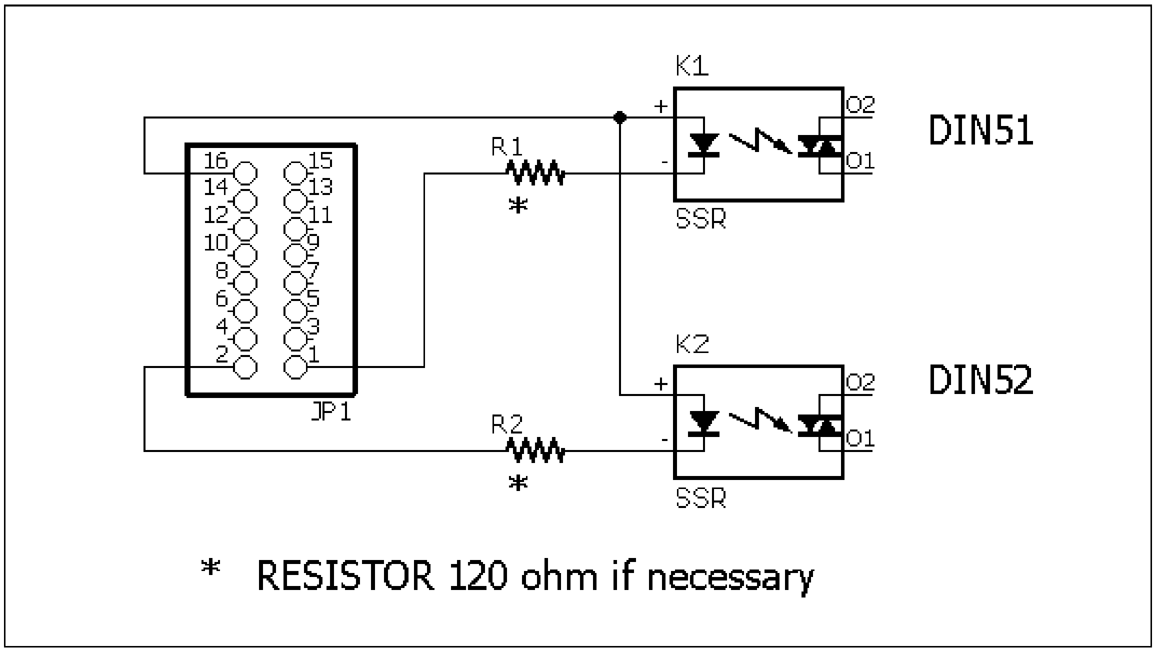




The backlight can be controlled by 0-5 VDC, 0-28VDC, or through the Ethernet interface by software.
The on-board FET is rated at 800mA maximum and provides an open drain pulling down to the common ground- shown here in JP2 pin 15. The FET can switch 28Volts resistive.
If the backlight is driven by the on-board 5 volts care must be taken that the current draw does not violate the host USB current supply value- typically 1 amp.

The connector information: wire housing: TE 1-87133-3 contacts: 1-104479-1

## 4.0 Outputs Example Wiring Diagram



The BTIO50 board provides two digital outputs that are capable of sinking 20 mA or less. The signals are referenced to the +3.3 volt power supply that is available on JP10 pin 16. If the load SSR is current limited, then the resistors shown on the above drawing are not necessary.

## 5.0 BTIO50 Sanitization Operations

### 5.1 BTIO50 Temporary App Sanitization

The BTIO50 can be rendered temporarily non-operational by downloading the application “BTIO50SanitizeApp.hex” (Bugeye number 15B180000-3006) with the Bugeye application BFinder201.exe. The board will still have the Boot Monitor in place, but until the proper application is re-programmed into the BTIO50 with BFinder201.exe, the instrument/board will be non-operational. The steps to temporarily render the BTIO50-based instrument non-operational:

1. Connect the instrument Ethernet connector to your Windows computer.
2. Ensure the Windows Ethernet connector is on the same network as the instrument.
3. Execute “BFinder201.exe”
4. Enter the target instrument IP Address into the “IP Address” line on the BFinder201 main dialog
5. On BFinder201.exe “Browse” to the folder that contains “BTIO50SanitizeApp.hex” and click that same file.
6. Click “Update” on the BFinder201.exe main dialog.

After downloading BTIO50SanitizeApp.hex into the instrument, the Blue and Orange LEDs visible near the Ethernet connector on the instrument will ping pong. The instrument will then be non-operational and it can be moved from the secure area.

The instrument can be brought back online by downloading the application (e.g “gameApp220.hex”) that was shipped with the instrument originally. The steps above are executed again with the original hexfile (e.g. “gameApp220.hex”) substituted in step (5) above. The instrument’s original calibration will still in place in this scenario

## 5.2 BTIO50 Boot Kill Sanitization

The BTIO50 can be rendered completely non-operational by downloading the application “BTIO50SanitizeBoot.hex” (Bugeye number 15B180000-3007) with the Bugeye application BFinder201.exe.  The board will no longer have the Boot Monitor in place and will be completely non-operational. The instrument will need to be shipped back to Bugeye to operate again.

The steps to render the BTIO50-based instrument non-operational and without its Boot Monitor:

1. Connect the instrument Ethernet connector to your Windows computer.
2. Ensure the Windows Ethernet connector is on the same network as the instrument.
3. Execute “BFinder201.exe”
4. Enter the target instrument IP Address into the “IP Address” line on the BFinder201 main dialog
5. On BFinder201.exe “Browse” to the folder that contains “BTIO50SanitizeBoot.hex” and click that same file.
6. Click “Update” on the BFinder201.exe main dialog.

After downloading BTIO50SanitizeBoot.hex into the instrument, the instrument will go dark and will not work any longer. The instrument can then be moved from the secure area.

The instrument can be shipped to Bugeye to bring the unit back online.

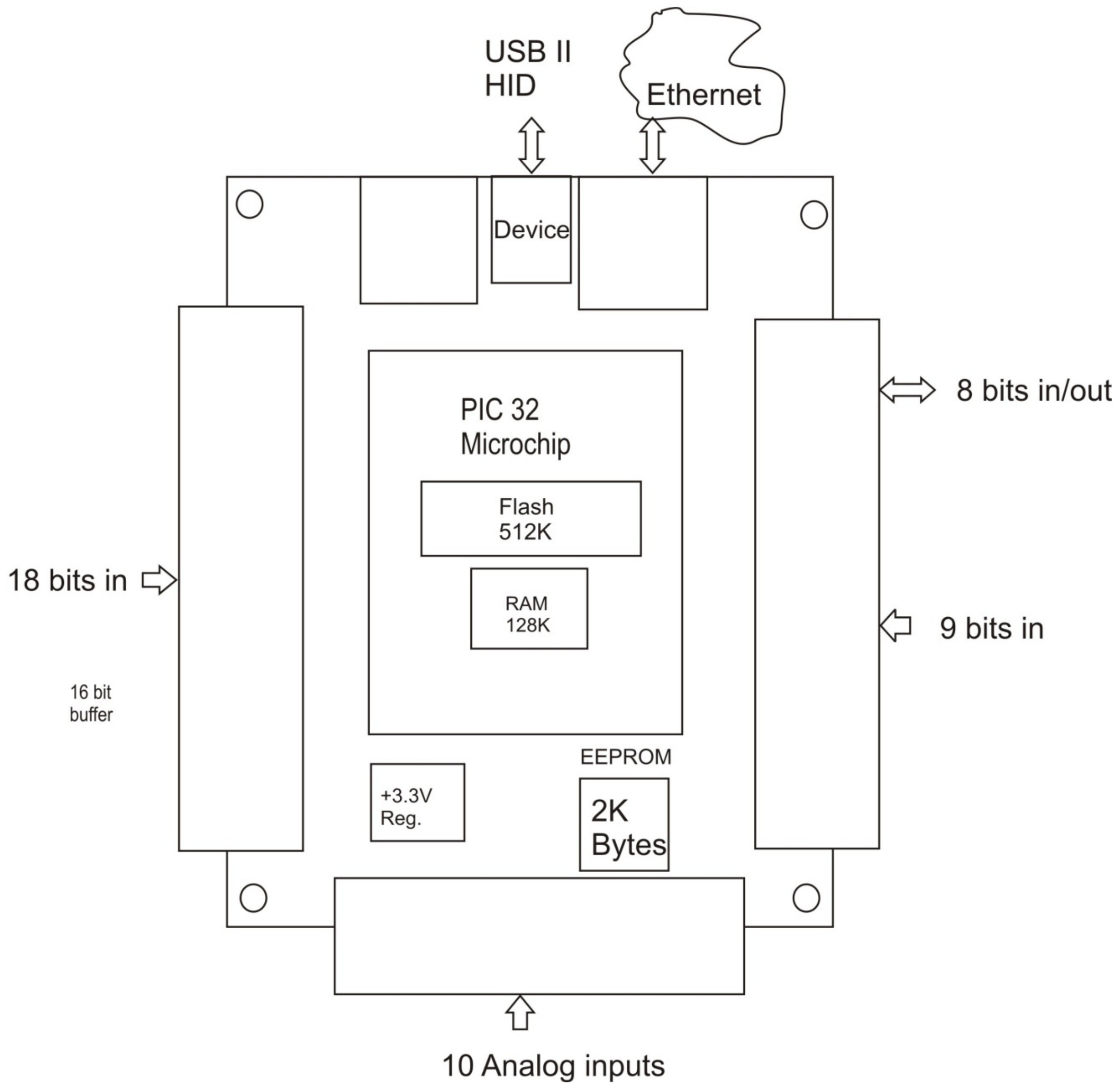
## 6.0 Letter Of Volatility

| | | | | | |
|---|---|---|---|---|---|
| Title: Letter of Volatility, Bugeye BTIO50 | | | | | |
| Prepared By: Chris Ruff | Dept | Date: 4-20-2016 | Checked By: | Dept | Date: |
| Approved by: | Dept | Date: | Quality Assurance Approval: | Dept | Date: |

| Rev | ECN No. | Description of change | Date: | Approved by: |
|---|---|---|---|---|
| A | | Production release | 4-20-2016 | |

BTIO50 - Letter Of Volatility

The following details the non-volatile memory storage that exists within the BTIO50 product.



- 128 K Bytes of RAM

The BTIO50 contains 128K bytes of RAM for system operation. This memory is volatile with the contents persisting only while the product is powered up. Once power is removed, the content of RAM is lost.

- 512K Bytes of FLASH Memory
  The BTIO50 contains 512K bytes of FLASH memory. This memory is non-volatile, meaning that it retains its content through power down and power up events.
  This memory is used for two purposes: Storage of the Boot Monitor in the first 8K block, storage of the product's application firmware in the remainder.

  The FLASH memory can be programmed with executable firmware via the Ethernet port or via an external PIC32 programming device. Random data cannot be written into the flash by any other method. Only valid S-records can be loaded into the main flash area using the monitor firmware or using the PIC32 programmer. The records copied into the flash memory space must be a valid executable program. Access to the in-circuit programming capability requires access to the BTIO50 card, installation (soldering in) of a programming header that is not factory installed in the board, a specialized programming cable, and a PIC32 programming device. The FLASH memory is not directly accessible via the ETHERNET port, and is not accessible at all via the USB device connector.

  The FLASH memory could be sanitized by writing HEX records over the address range desired to be cleaned. NOTE: OVERWRITING THE FLASH MEMORY WILL RENDER THE BTIO50 INOPERATIVE AND WILL NECESSITATE THE RETURN OF THE UNIT TO THE BUGEYE FACTORY FOR SERVICING.

- 2Kbit of EEPROM
  The BTIO50 contains a 2Kbit EEPROM (256 bytes) that contains the calibration parameters of the analog channels and the Ethernet parameters (MAC address, IP Address, etc.). The EEPROM is not available through any port. Only the PIC32 has access to the memory in this small part.

### 3.5.9 Product Information

Product registration and application information

**Serial Number:** ____________________________________________________

**Purchase date:** ____________________________________________________

**Installation Notes:**

## Contact Information

**On the web at:**

**WWW.BUGEYETECH.COM**

support@bugeyetech.com

sales@bugeyetech.com

Bugeye Technologies
7 Progress Pkwy.
Union MO 63084
USA

Telephone 636 257 3530