RibEye Communications Protocol
Revision 5
November 10, 2009

Table of Contents

1.0 Overview

This protocol describes RibEye communications over serial or Ethernet ports. Control software for the current RibEye models – 5th Female, 50th Male, SIDIIs, WorldSID, and Polar – support this protocol. The specification covers the protocol needs for all of the current and future RibEye models. The RibEye will be a Slave, and the external device the Master.

The 5th, 50th, and SIDIIs RibEye controllers use RS232 serial communications. These units are supplied with a trunk box that includes a serial-to-Ethernet converter, allowing Ethernet communications to the RibEye. Appendix C shows the hardware interfaces to the RibEye controller and trunk box, and the cable pinouts.

The Polar and WorldSID RibEyes have the serial-to-Ethernet converter built into the controller, so only Ethernet communications is available. However, they can be wired to provide serial communications.

The protocol includes two types of commands and responses:

1. Information commands: These commands request static information about the RibEye and its capabilities.
2. Control and Data Download commands: These are used for arming the RibEye and transferring data from the RibEye to the Master.

2.0 Port Setup

If connecting via a serial port, the serial port should be configured for 115 kBaud, 8 data bits, no parity, 1 stop bit, and no flow control.

If connecting via Ethernet through a trunk box, use the trunk box IP address (default is 192.168.0.240) and port 3000. Be aware that the first character sent after the RibEye has booted might get dropped and result in a bad checksum response (see Section 4.0).

3.0 Generic Command Format

command [#param1] [#param2,]#checksumCRLF
where:

- command is the text command in all caps
- param1 is an ASCII integer, checked that it is within the correct range
- param2 is an ASCII integer, checked that it is within the correct range
- pound symbols (#) are used as delimiters between the command and the parameters and the checksum
- checksum is an 8-bit checksum of the whole line including all # delimiters up to the checksum
- CRLF is a carriage return followed by a linefeed, used as a command terminator.

Note: No additional spaces are allowed.
Note: The parameters are only used on specific commands, and are not optional if the command needs parameters.

4.0 Bad Checksum

If a command is received with a bad checksum the RibEye will respond with ?1.
Note that most versions of the firmware have left debugging code enabled, so a bad checksum response will include the expected checksum, such as:

?1 - should be 0

5.0 Bad Command

If a command is received with a good checksum, but is not recognized, RibEye will respond with ?2.

6.0 Special Cases during Acquisition and post-test writing to Flash, and Erase

While the RibEye is busy during acquisition, during immediate post-test transfer of data to flash, and during flash erase, the RibEye will only respond to certain commands as defined below.

6.1 Acquisition special case

RibEye will only respond to S# (status), T# (force trigger), and D#(disarm) commands.
Due to limited time available for command parsing during acquisition, the RibEye looks for a good command first, and if it receives a command other than S, T, or D it reports a bad command by sending ?2.
If an S, T, or D command is received with a bad checksum, RibEye will respond with ?1.

6.2 Post-test writing data to Flash special case

If acquisition is terminated normally, and not by a D disarm command, it will immediately write a partial data set to flash to prevent loss of data from a loss of power. During this time, commands will be parsed normally, with the checksum tested first. The RibEye will respond to any command except the Status command with ?2 – the bad command response. This is because writing to flash is the highest priority, and no other commands can be accepted.
The RibEye will respond to a status command with S#2#203CRLF, which indicates it is busy.

6.3 Erase command special case

Erasing flash typically takes about 12 seconds, but it can take up to 90 seconds worst case. During this time, commands will be parsed normally, with the checksum tested first. The RibEye will respond to any command except the Status command with ?2 – the bad command response. This is because erasing flash is the highest priority, and no other commands can be accepted.
The RibEye will respond to a status command with S#2#203CRLF to indicate it is busy.
An Erase Status command E#104CRLF has been added to get progress of data flash erasing. It responds with E#p1#p2#checksumCRLF, where p1 is the current sector being erased, and p2 is the total number of sectors to be erased.

7.0 Command Response Times

The longest response should be less than 50 ms except for:

- CURRENT_POSITIONS command can take up to 0.3 second.
- SAVE and SETTESTCOMMENT commands have to erase one sector of flash. A sector erase will typically take less than 1 second, but can take up to 6 seconds worst case.
- ERASE command – see above.

8.0 Static Information Commands

These commands report static information about the RibEye.

8.1 WHO_ARE_YOU

parameters: none
returns: WHO_ARE_YOU#XXXXXXXXXX#checksum

where XXXXXXXXXX is a character string up to 10 characters long, and will be the RibEye type, such as 5th_Female, 50th_Male, SIDIIs, WorldSID50, Rollover, etc.
Example Command: WHO_ARE_YOU#164CRLF
Example Response: WHO_ARE_YOU#5th_Female#129CRLF

8.2 SERIAL_NUMBER

parameters: none
returns: SERIAL_NUMBER#XXXXXXXXXX#checksum
where XXXXXXXXXX is a character string up to 10 characters long
Example Command: SERIAL_NUMBER#11CRLF
Example Response: SERIAL_NUMBER#0075#250CRLF

8.3 CAL_DATE

parameters: none
returns: CAL_DATE#XXXXXXXXXXXXXXXXXXXX#checksum
where XXXXXXXXXXXXXXXXXXXX is a string up to 20 characters long
Example Command: CAL_DATE#112CRLF
Example Response: CAL_DATE#SEPTEMBER 12, 2007#178CRLF

8.4 CAL_LOC

parameters: none
returns: CAL_LOC#XXXXXXXXXXXXXXXXXXXX#checksum
where XXXXXXXXXXXXXXXXXXXX is a string up to 20 characters long
Example Command: CAL_LOC#48CRLF
Example Response: CAL_LOC#R.A. DENTON, MI#12CRLF

8.5 FIRMWARE

parameters: none
returns: FIRMWARE #XXXXXXXXXX#checksum
where XXXXXXXXXX is a string up to 10 characters long
Example Command: FIRMWARE#128CRLF
Example Response: FIRMWARE#5A0002#219CRLF

8.6 HOW_MANY_LEDS

parameters: none
returns: HOW_MANY_LEDS #XX#checksum
where XX is a decimal number of LEDs the RibEye monitors
Example Command: HOW_MANY_LEDS#44CRLF
Example Response: HOW_MANY_LEDS#12#178CRLF

8.7 HOW_MANY_AXES

parameters: none
returns: HOW_MANY_AXES #XX#checksum
where XX is a one digit decimal number of axes for each LED reported (2 or 3)
Example Command: HOW_MANY_AXES#53CRLF
Example Response: HOW_MANY_AXES#3#139CRLF

Note: Number of LEDs x number of axes = number of "data channels" collected.

8.8 CURRENT_POSITIONS

parameters: none

returns: CURRENT_POSITIONS#n#CH1, CH2, …,CHn-1, CHn #checksum

where:

n = number of channels

CHi = a decimal number (xxx.x) in millimeters

The channel order for a 2-axis system is: LED1X, LED1Y, LED2X, LED2Y, …,LEDnY

The order for a 3-axis system is: LED1X, LED1Y, LED1Z, LED2X, LED2Y,…,LEDnZ

Example command: CURRENT_POSITIONS#109CRLF

Example Response (for a 6-LED, 3-axis system):

CURRENT_POSITIONS#18#1.0,165.0,-113.5,0.0,167.0,-67.0,0.0,167.0,
-22.5,1.0,166.5,22.5,-1.2, 166.8,66.2,0.0,168.0,112.5#72CRLF

Note: Embedded in the current positions are error codes. For a two-axis system, if the light from a LED can't get to sensor 1, both of the X and Y values will be forced to a 1. If the light can't get to sensor 2, the X and Y values will be forced to 2. If the light can't get to both sensors, the X and Y values will be forced to 3. For a three-axis system the error codes are 1,2,3,4,5,6,7, and ALL axes will report the same number. So if you want to get fancy, you could turn any number on all axes less than10 red to flag the user. The system will return an 8 if it can't resolve a good reading because some other problem occurs that can cause a divide by zero in the code. So you can also flag any 8's that you receive.

The LED positions on the ATD are shown in the tables below.

LED Number	5th Female	50th Male	SIDIIs	Ballistic SIDIIs	Polar
1	Rib 1 Left	Rib 1 Left	Rib 1	Rib 1	Rib 4 Outer
2	Rib 2 Left	Rib 2 Left	Rib 2	Rib 2	Rib 5 Outer
3	Rib 3 Left	Rib 3 Left	Rib 3	Rib 3	Rib 6 Outer
4	Rib 4 Left	Rib 4 Left	Rib 4		Rib 4 Middle
5	Rib 5 Left	Rib 5 Left	Rib 5		Rib 5 Middle
6	Rib 6 Left	Rib 6 Left	Rib 6		Rib 6 Middle
7	Rib 1 Right	Rib 1 Right			
8	Rib 2 Right	Rib 2 Right			
9	Rib 3 Right	Rib 3 Right			
10	Rib 4 Right	Rib 4 Right			
11	Rib 5 Right	Rib 5 Right			
12	Rib 6 Right	Rib 6 Right			

LED Number	WorldSID on ATD Left Side	WorldSID on ATD Right Side
1	Rib 1 Rear	Rib 1 Front
2	Rib 1 Middle	Rib 1 Middle
3	Rib 1 Front	Rib 1 Rear
4	Rib 2 Rear	Rib 2 Front
5	Rib 2 Middle	Rib 2 Middle
6	Rib 2 Front	Rib 2 Rear
7	Rib 3 Rear	Rib 3 Front
8	Rib 3 Middle	Rib 3 Middle

9	Rib 3 Front	Rib 3 Rear
10	Rib 4 Rear	Rib 4 Front
11	Rib 4 Middle	Rib 4 Middle
12	Rib 4 Front	Rib 4 Rear
13	Rib 5 Rear	Rib 5 Front
14	Rib Middle5	Rib Middle5
15	Rib 5 Front	Rib 5 Rear
16	Rib 6 Rear	Rib 6 Front
17	Rib 6 Middle	Rib 6 Middle
18	Rib 6 Front	Rib 6 Rear

Getting the current LED positions is the only way to tell whether the RibEye is performing properly. You can define a range of acceptable numbers for the default unloaded rib positions based on the dummy type.

8.9 SAMPLE_RATE

parameters: none
returns: SAMPLE_RATE #XXXXX#checksumCRLF
where XXXXX is a five digit decimal sample rate in Hz for all channels
Example Command: SAMPLE_RATE#112CRLF
Example Response: SAMPLE_RATE#10000#132CRLF

Note: All current RibEyes run at 10 kHz except the Ballistic SIDIIs (3 LEDs, 3 axes) for weapons testing. The Ballistic SIDIIs runs at 20 kHz.

9.0 Configuration and Control Commands

These commands are for configuring the RibEye, initiating data acquisition, erasing flash, downloading, and checking the RibEye status.

9.1 S (status command)

command has no parameters
returns: S#P1#checksumCRLF
where P1 is:

0 = idle, no data in ram or flash, ready to run a test
1 = if armed and collecting pre-trigger data
2 = if busy. Busy states are:

- collecting post-trigger data
- storing data in flash post-test
- erasing flash pre-test

3 = idle with data in Ram or Flash, ready to download

Example Command: S#118CRLF
Example Response: S#3#204CRLF

9.2 T (force trigger command, only active during acquisition)

command has no parameters
returns: T#119CRLF
Notes:
During acquisition this command will force a T0 event as if a hardware T0 had occurred.
If this command is received at any time other than during data acquisition, a bad command response will be issued (?2).

9.3 D (disarm command during acquisition)

command has no parameters
returns: D#103CRLF
During acquisition, this command will force the RibEye to return to the idle state immediately. NO DATA WILL BE STORED
If this command is received at any time other than during data acquisition, a bad command response will be issued (?2).

9.4 ERASE (erase flash memory)

command has no parameters
response: (after the erase is complete) ERASE#P1#checksumCRLF
where P1 = 0 for a successful erase
P1 >0 if any sector erase failed

Example Command: ERASE#147CRLF
Example Response: ERASE#0#230CRLF

9.5 E (erase status command)

command has no parameters
response: (during erase) ERASE#P1#P2#checksumCRLF
where P1 = current sector being erased
P2 = total number of sectors to be erased (depends on model)

Example Command: E#104CRLF
Example Response: E#31#32#119CRLF
Note that this command is only valid during a flash erase. If issued at other times you will get a bad command response (?2)
Typically you would issue an erase command, then issue erase status commands until the erase is complete. Below is a typical scenario:

```
ERASE#147      //erase command sent
E#104          //erase status command sent
E#1#32#68      //erase status response – erasing sector 1 of 32
E#104          //erase status command sent
E#12#32#118    //erase status response – erasing sector 12 of 32
E#104          //erase status command sent
E#23#32#120    //erase status response – erasing sector 23 of 32
E#104          //erase status command sent
E#31#32#119    //erase status response – erasing sector 31 of 32
ERASE#0#230    //response to original erase command – no errors reported
E#104          //erase status command sent
?2             //bad command response since erase was finished
```

9.6 ARM (initiates data collection)

Mote: RibEye sends an ARM response immediately, then enters acquisition mode
Parameters: Tstop (in integer ms) and Tpost (in integer ms)
returns: ARM#Tstop#Tpost#checksumCRLF
Example Command (for circular buffer, collect for 2 secs after trigger):
ARM#0#2000#59CRLF
Example response: ARM#0#2000#59CRLF

Example bad command: ARM#-10#2000#153CRLF
Example bad response: ARM#BAD#2000#210CRLF
Example bad command: ARM#0#32000#110CRLF
Example bad response: ARM#0#BAD#64CRLF

Valid Tstop and Tpost are:
Tstop must be >= 0 ms
Tpost must be >=0 and <= 30000 ms

If data memory is not erased, the arm command will return:
ARM#ERROR-NOT_ERASED#225

The RibEye has a 30-second ram buffer, and behaves as follows:

- If Tstop = 0, then the buffer will be treated as a circular buffer. After arming the RibEye will collect data into the circular buffer. When a trigger is received (hardware or T# command) data will be collected for an additional Tpost time. If no trigger is received, the RibEye will just keep collecting forever unless it receives a d# disarm command..
- If Tstop <= 30000 ms, the buffer will be treated as a linear buffer.
  If a trigger is received before Tstop, the RibEye will continue to collect data for Tpost more seconds, or until 30,000 ms has elapsed since the Arm command, whichever is shorter.
  If no trigger is received, data collection will stop Tstop ms after the ARM command
  Note that for DTS interface versions, the ARM command starts acquisition, but the linear buffer timer is not started until the DTS Start signal changes state.
- If Tstop > 30000 ms, the buffer will be treated as a circular buffer.
  If a trigger is received before Tstop, the RibEye will continue to collect data for Tpost more seconds, or until Tstop ms has elapsed since the Arm command, whichever is shorter.
  If no trigger is received, the RibEye will stop collecting data Tstop ms after the ARM command, and the buffer will hold data from (Tstop-30000) to Tstop ms after the Arm command.

Data time definitions:
If a trigger occurs during acquisition, all pre-trigger data is considered negative time, and all post-trigger data is considered positive time.

If no trigger occurs during acquisition, all data times are considered positive. The first sample will be at 0 ms if Tstop <=30000 ms. The first sample will be at (Tstop-30000) ms for Tstop > 30000 ms.

9.7 DUMPINFO (reports data available for download)

no parameters for command
returns: DUMPINFO#T1#T2#checksumCRLF
Where: T1 is the data start time in ms
T2 is the data stop time in ms
Example command: DUMPINFO#133CRLF
Example response: DUMPINFO#-2126#1000#132CRLF
data available from –2126 ms to +1000 ms.

9.8 DUMPBIN (requests data in binary format)

parameters: T1, T2
T1 is the download start time in integer ms
T2 is the download stop time in integer ms
returns: DUMPBIN#X1#X2#checksumCRLF followed by a binary spew.
X1 is the number of data points for each time sample. (24 for the 5th and 50th, 18 for the SIDIIs)
X2 is the number of time samples being sent
A checksum will be inserted at the end of the data points for each time sample.
Each data point will be a 16 bit binary number = millimeter * 100.
Each data point is sent least significant byte first
The checksum is one byte
For 24 data points per sample, 49 bytes are sent (24 * 2 byte + 1 checksum byte)

SIDIIs Example command: DUMPBIN#-90#200#160CRLF
This command requests data from –90 ms to 200 ms for all channels

SIDIIs Example response: DUMPBIN#18#2910#173CRLF
(18 data points per sample, 2910 samples at 10 kHz)
The spew will look like:
LED1X LED1Y LED1Z LED2X ……LED6X LED6Y LED6Z checksum
LED1X LED1Y LED1Z LED2X ……LED6X LED6Y LED6Z checksum
LED1X LED1Y LED1Z LED2X ……LED6X LED6Y LED6Z checksum
no carriage returns or linefeeds are inserted in the spew.

Example Bad Command (assuming data starts at –90 ms):
DUMPBIN#-100#200#200CRLF
Example Bad Response: DUMPBIN#BAD#200#209CRLF

If DUMPINFO reports data start time of TSTART and stop time of TSTOP, then the DUMPBIN command will report errors if:

T1 bad if T1 < TSTART or T1 >= TSTOP
T2 bad if T1 <= T1 or T2 >TSTOP

Note: Error codes are inserted in the data if the LED positions cannot be accurately calculated. The error codes are described in section 8.8 CURRENT_POSITIONS. The error codes will be reported on the X, Y, and Z data for the LED that cannot be resolved.

Figure 1 shows a C program for a PC that reads in the DUMPBIN data that was captured by a terminal program to a file. The program writes out a comma delimited ASCII file of the data. The last column of the output file is a boolean value comparing the calculated checksum to the sent checksum.

Figure 1 – C program on PC for processing DUMPBIN command data

```
#include <stdio.h>

void main (void)
{
FILE *fpin;
FILE *fpout;
int data;
char *pdata;
unsigned char c1, c2, checksum;
int darray[25];
int i,j,k;

fpin = fopen("danbin.txt","r+b");  //data captured to danbin.txt
fpout = fopen("dan.csv","w+");     // converted data put in dan.csv


while(getc(fpin) != '\n');  // throw away the command response


while(!feof(fpin) )
      {
      checksum = 0;
      for(i=0;i<24;i++)  // get 48 chars, process into 24 ints
            {
            pdata = (char*)&data;
            c1=getc(fpin);
            checksum += c1;
            c2=getc(fpin);
            checksum += c2;
            *pdata++ = c1;
            *pdata = c2;
            darray[i]=data;
            }
      c1=getc(fpin);  //read the checksum
      if(feof(fpin)) break;
      darray[24]=(checksum == c1);  // check the checksum
      for(j=0;j<24; j++)fprintf(fpout,"%d,",darray[j]);  //send 24 nums
      fprintf(fpout,"%d\n",darray[24]); //send the checksum check
      }
fclose(fpin);
fclose(fpout);
fprintf(stdout,"all done ..............................");
fflush(stdout);
```

9.9 DUMPBINA (include ambient light data)

This command performs the same as the DUMPBIN command, but it appends the ambient light readings for each sensor. So for a 5th female, instead of getting 24 (12 LEDs X and Y) short ints (16 bits) data points per line, you get 26 short ints, where the last two are the ambient light readings for sensor 1 and sensor 2 respectively. For a SIDIIs, a DUMPBIN command will send 18 data points per line (6 LEDs, 3 axes), while DUMPINA will send 21 data points per line (6 LEDs, 3 axes, + 3 ambient light readings).

Since the ambient light readings could theoretically be full scale, 65,535 counts, and all other data is treated as signed short ints, the ambient light data is divided by two before being sent. On the receiving side, the ambient light data should be multiplied by two to get back to counts.

Note: Error codes are inserted in the data if the LED positions cannot be accurately calculated. The error codes are described in section 8.8 CURRENT_POSITIONS. The error codes will be reported on the X, Y, and Z data for the LED that cannot be resolved.

9.10 TRIGGERSET (sets the trigger polarity and type, stores the setting in flash)

Parameters: one parameter P1 as follows:

0 – for leading edge, switch or TTL input
1 – for trailing edge, switch or TTL input
3 – for leading edge, differential input
4 – for trailing edge, differential input

returns: TRIGGERSET#P1#checksumCRLF
Where: P1 is the trigger setting as above
Example command: TRIGGERSET#0#118CRLF
Example response: TRIGGERSET#0#118CRLF

Example Bad Command: TRIGGERSET#5#123 CRLF
Example Bad Response: TRIGGERSET#BAD#13

Note: The differential trigger options can only be used for units with a trunk box as the differential input receiver is in the trunk box. For units without a trunk box, selecting a differential input acts the same as a switch or TTL input.

9.11 GETTRIGGER (gets the trigger polarity and type)

Parameters: none
returns: GETTRIGGER#P1#checksumCRLF
Where: P1 is the trigger setting as above
Example command: GETTRIGGER#23CRLF
Example response: GETTRIGGER#0#106CRLF

NOTE: For units such as the Polar, where the trigger comes from the G5DB, the trigger will always be set to 0 for leading edge. It will be set at the factory and won't need to be changed. For units that have a trunk box to interface to any type of DAS, such as the current 5^{th}, 50^{th}, and SIDIIs, all of the trigger options could be used.

9.12 GETTESTCOMMENT (get a string from RibEye flash memory)

An 80-character text string that is stored in flash is called the Test Comment, but it is typically used in my application to store the ATD type and serial number, like 5th#103.

Parameters: none
returns: GETTESTCOMMENT #P1#checksumCRLF
Where: P1 is the text string, up to 80 characters
Example command: GETTESTCOMMENT #86CRLF
Example response: GETTESTCOMMENT #5^{th} s/n 103#106CRLF

Note that the test comment text string can have a # symbol in it when it is written (see Section 9.13, SETTESTCOMMENT). However, when the test comment is sent back from the RibEye, in response to a GETTESTCOMMENT command, it replaces all # symbols with 0x03, so as not to appear as a command delimiter. Your code should look for 0x03 characters and switch them back to #'s.

9.13 SETTESTCOMMENT (set a string in RibEye flash memory)

The behavior of this command is different from other commands.

Command: SETTESTCOMMENT#98
The RibEye will send back: COMMENT?\n
then send the test comment string terminated with a \r
After the RibEye saves the string in flash, it will respond with:
SETTESTCOMMENT#OK#31

Example:
SETTESTCOMMENT#98 {sent to RibEye}
COMMENT?\n {response from RibEye}
Foo bar moo\r {the text string sent to RibEye}
SETTESTCOMMENT#OK#31

Note that the response will be slow since it has to erase an 8k flash block.
Below is a snippet of the firmware that responds to the command:

```
printf("COMMENT?\n");              // ask for the test comment
fflush(stdout);
UART_gets(TestComment, 81);   // get response, no more than 81 chars, terminated by \r
TestComment[strlen(TestComment)-1] = '\0';  // null terminate the string
*(strpbrk(TestComment,"\r")) = '\0';          // get rid of the \r, replace with \0 just in case
save_params();                                // save the parameters in flash
strcpy(response,"SETTESTCOMMENT#OK#");
```

10.0 LOADER (loads a new program into the RibEye)

no parameters for command
Example: LOADER#218

When this command is executed, the RibEye will respond with:
SEND NEW PROGRAM FILE
A program file can then be sent to the RibEye. If it loads properly, the RibEye will respond with:
PROGRAM LOADED SUCESSFULLY, REBOOT TO START NEW PROGRAM
If the file load failed, the RibEye will respond with:
CODE LOAD FAILED

The best way to handle this command, and other future special commands, would be to just open a terminal window to the serial port. If you are running under Windows, you could open a HyperTerminal window to the port, and send the file via the HyperTerminal /Transfer/Send Text File option. The Boxboro Systems RibEye control application can also be used to load new firmware.

Appendix A – Command Summary

Section	Command	Parameter 1	Parameter 2	Function
8.1	WHO_ARE_YOU#164	x	x	reports RibEye model type
8.2	SERIAL_NUMBER#11	x	x	reports serial number
8.3	CAL_DATE#112	x	x	reports calibration date
8.4	CAL_LOC#48	x	x	reports calibration location
8.5	FIRMWARE#128	x	x	reports firmware revision
8.6	HOW_MANY_LEDS#44	x	x	reports number of LEDs
8.7	HOW_MANY_AXES#53	x	x	reports number of axes per LED
8.8	CURRENT_POSITIONS#109	x	x	reports XY or XYZ position of each LED
8.9	SAMPLE_RATE#112	x	x	reports sample rate per LED
9.1	S#118	x	x	reports status
9.2	T#119	x	x	force trigger
9.3	D#103	x	x	disarm
9.4	ERASE#147	x	x	erase memory
9.5	E#104	x	x	erase status
9.6	ARM#0#2000#59	Tstop	Tpost	arm the RibEye
9.7	DUMPINFO#133	start time	end time	reports how much data is in memory
9.8	DUMPBIN#-90#200#160	start time	end time	send data using stored cal info
9.9	DUMPBINA#0#50#78	start time	end time	send data and ambient light, stored cal
9.10	TRIGGERSET#0#118	trigger		set the trigger polarity and type
9.11	GETTRIGGER#23	x	x	set the trigger polarity and type
9.12	GETTESTCOMMENT#86	x		get the text string in flash
9.13	SETTESTCOMMENT#98	x		Set the test string in flash
10.0	LOADER#218	x	x	load new firmware

Appendix B. Behavior on Boot

Upon boot, the RibEye checks the flash for valid data by calculating a checksum. If the flash checksum does not match the checksum stored in the flash, the RibEye will report the data in flash as –

Start Time = –29, 999 ms
Stop Time = -27,9999 ms

There may or may not be any good data in flash, but the data can be downloaded and inspected.

When the RibEye boots, it will flash all LEDs in sequence, starting with LED #1. Each LED will be turned on for approximately 1 second.

Units using Ethernet communications take approximately 10 seconds for the Ethernet processor to be ready to respond to commands. The first character sent after boot may be dropped, resulting in a bad checksum response.

Units using a serial interface will be ready for communications less than 1 second after booting.

The RibEye control processor is set up so that it can boot to one of two versions of the application code stored in flash. This allows it to run DAS-specific code and the normal RibEye code. The processor uses the "Other In" control line to determine which code to boot to – refer to Appendix C for hardware interface information. If a trunk box is used, the Other In control line is connected to the manual arming switch on the trunk box, and is pulled high, booting to the normal RibEye firmware. If the Other In control line is pulled low or left open, the RibEye will boot to alternative firmware, typically used for specific DAS interfaces. Unless a specific DAS interface is selected, the normal RibEye firmware will be installed in both locations in flash.

Appendix C – Hardware Interface Circuits

Please refer to the RibEye hardware manual for RibEye component pictures and a description of your RibEye model. Reference manuals can be downloaded from the Boxboro Systems web site at www.boxborosystems.com. For reference, a picture of the components for a 5th Female RibEye is shown in Figure C1, and a block diagram if the component interconnections is shown in Figure C2.

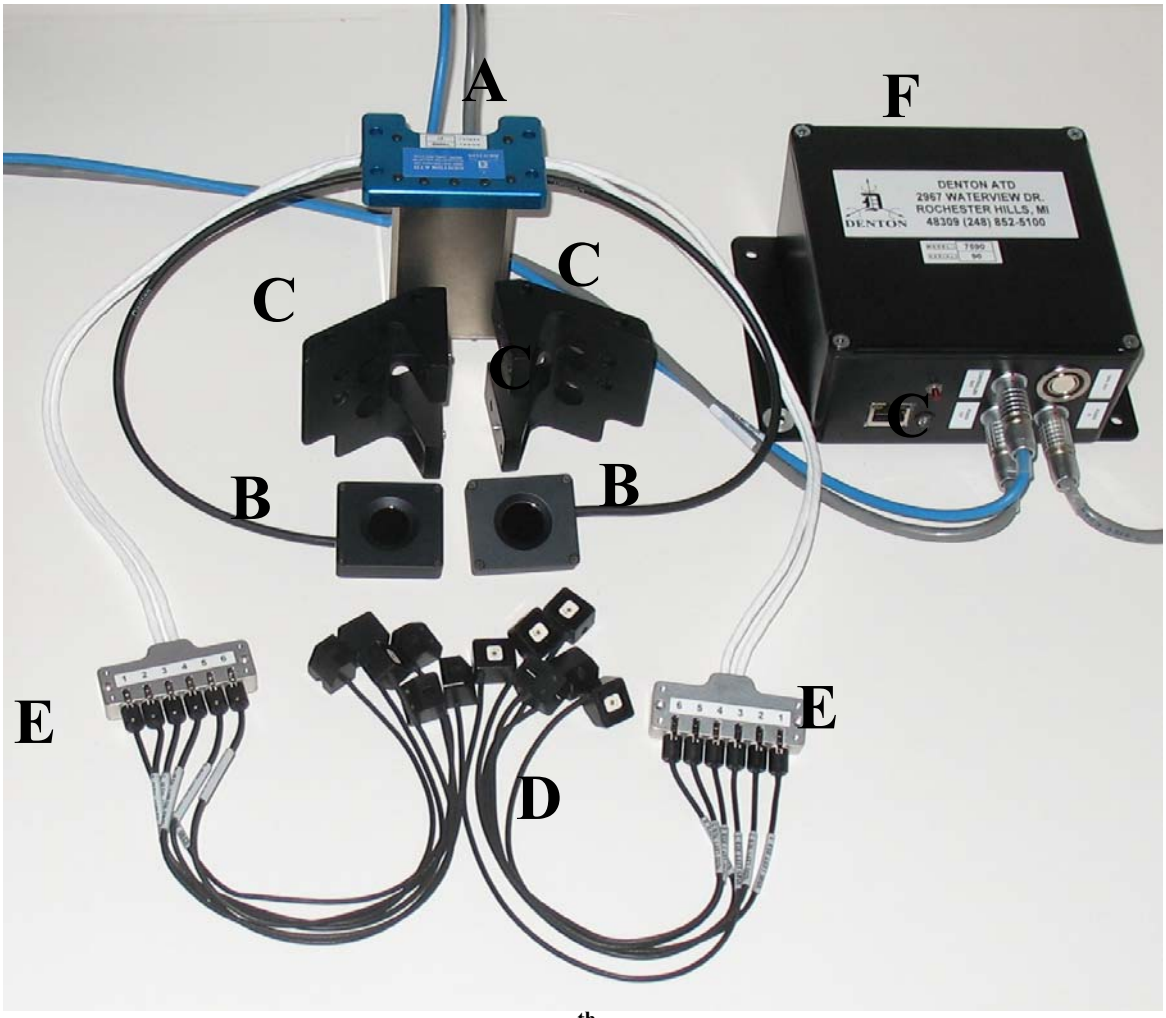

Figure C1. RibEye 5th Female components

The RibEye has the following components:

- The RibEye controller, also called the spine box ("A"), located in the ATD spine.
- Two optical sensor heads ("B") that will attach to sensor head mounting flanges ("C").
- Twelve light-emitting diodes (LEDs), two sets of 6 each ("D"), mounted on the ribs or sternum. Some of the LEDs have an angled mounting surface that aims the LED toward the sensor head to minimize power requirements.
  DANGER: *The LEDs are very bright when driven at full power. Do not look directly at the LEDs.*

- Two LED connector blocks (“E”) that will be mounted on the back of the sensor head mounting flanges
- The interface box (“F”), also called the trunk box because it is typically placed in the trunk of the vehicle.
- Two cables run from the trunk box to the RibEye controller – 1) the gray cable is the power cable and uses a 2-pin Lemo connector, and 2) the blue cable is the communications/control cable and uses 14 pin Lemo connectors.
- The trunk box has a power in cable with a 2-pin Lemo connector, and a 24-pin DAS/Trigger cable for interfacing to DAS systems.

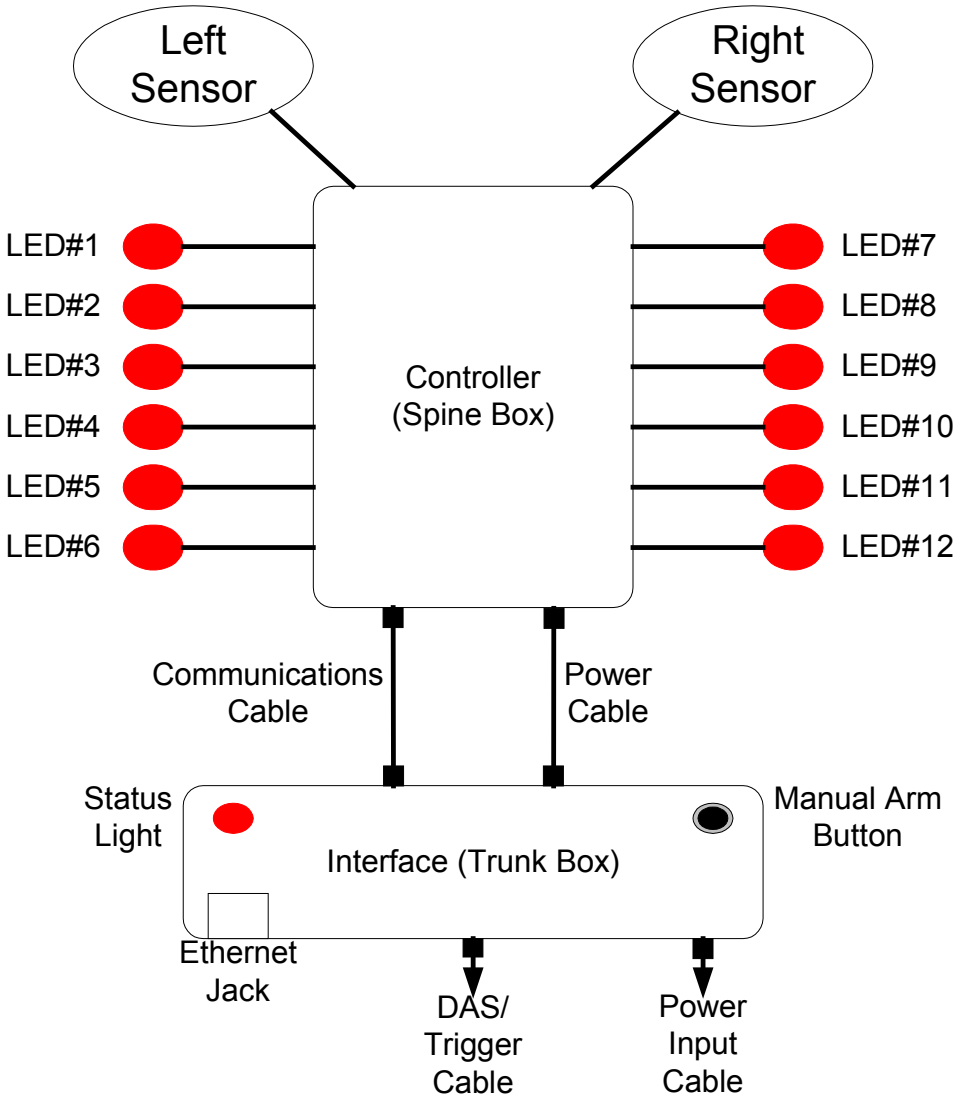

Figure C2. RibEye connections

If you want to connect to the RibEye via RS232, you do not need the trunk box and can connect directly to the blue cable for communications and control. Note that the only ground connection is in the gray cable.

To connect to the RibEye via Ethernet, you must use the trunk box, and control signals such as trigger are available on the 24-pin DAS connector.

Note that some signals are currently not used or are reserved for Boxboro Systems. These signals can be programmed to provide necessary control signal interfaces to your DAS.

Table C1 shows the pin descriptions for the 14-pin blue cable from the RibEye controller. Figure C3 shows the interface circuits inside the RibEye controller. Figure C4 shows the 14-pin blue cable wiring.

Figure C5 shows the 2-pin gray power cable pinouts. Red is positive, Black is ground, 12 to 36 VDC.

Table C1 – Blue Cable Pinouts

Lemo Pin #	cable color	signal	RibEye in/out
1.	blue	TRIGOUT	out (oc)
2.	Light green	ARMOUT	out (oc)
3.	blk/stripe	OTHEROUT	out (oc)
4.	white	STATUSOUT	out (oc)
5.	black	OTHERIN	in (diode)
6.	red/stripe	CONTROL SIGNAL COM	
7.	yellow	RS232 RTS	input
8.	orange	RS232 TX	output
9.	red	RS232 RX	input
10.	brown	RS232 CTS	output
11.	violet	ARMIN	in (diode)
12.		No Connection	
13.	pink	TRIGIN+	in (diode)
14.	gray	TRIGIN-	in (diode)

Signals highlighted in gray are reserved for Boxboro Systems use or are not currently programmed in the firmware. The wires should be insulated and not connected to anything.

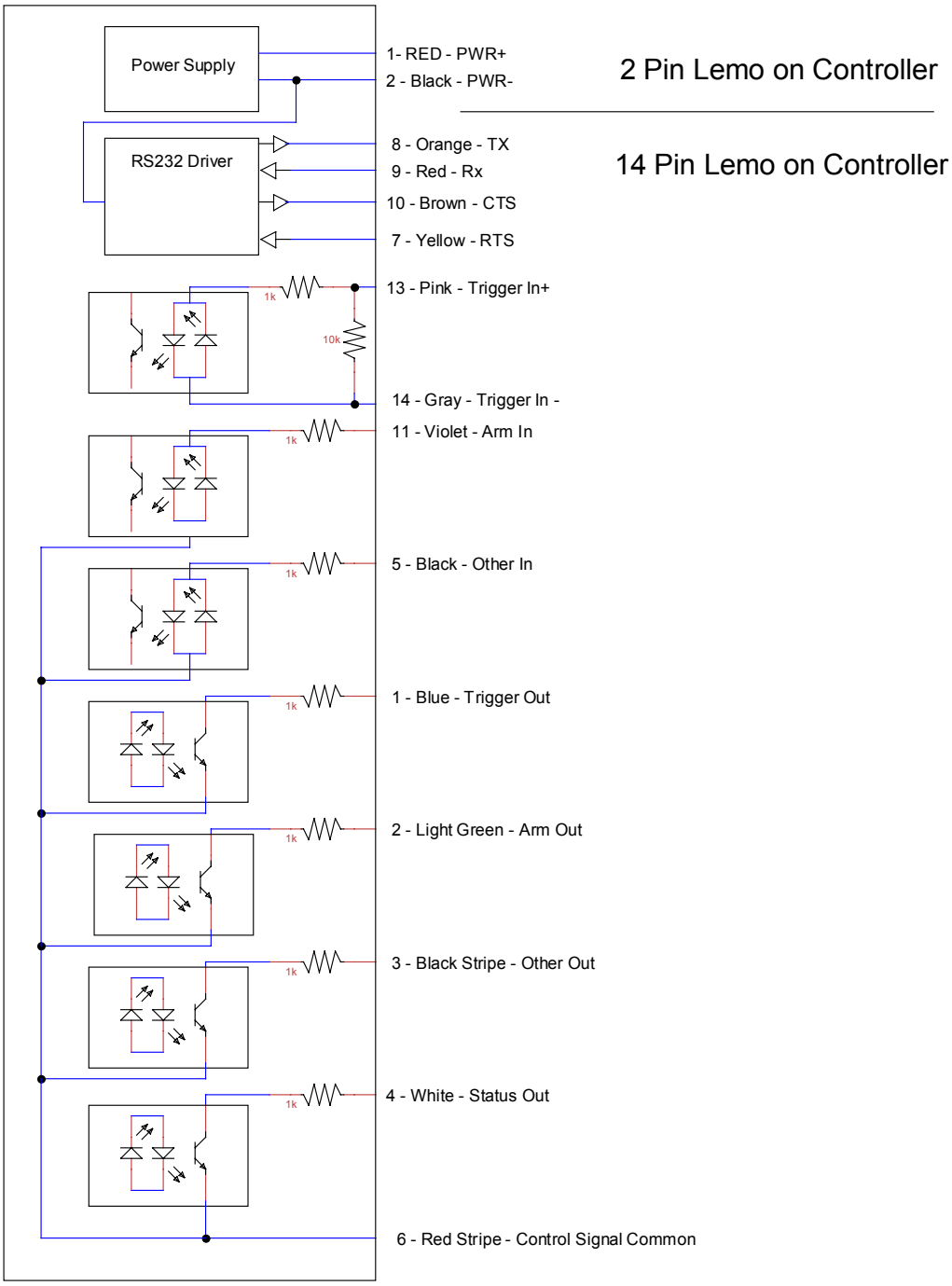

Figure C3. 14-Pin Lemo Connector on Controller, Interface Circuits

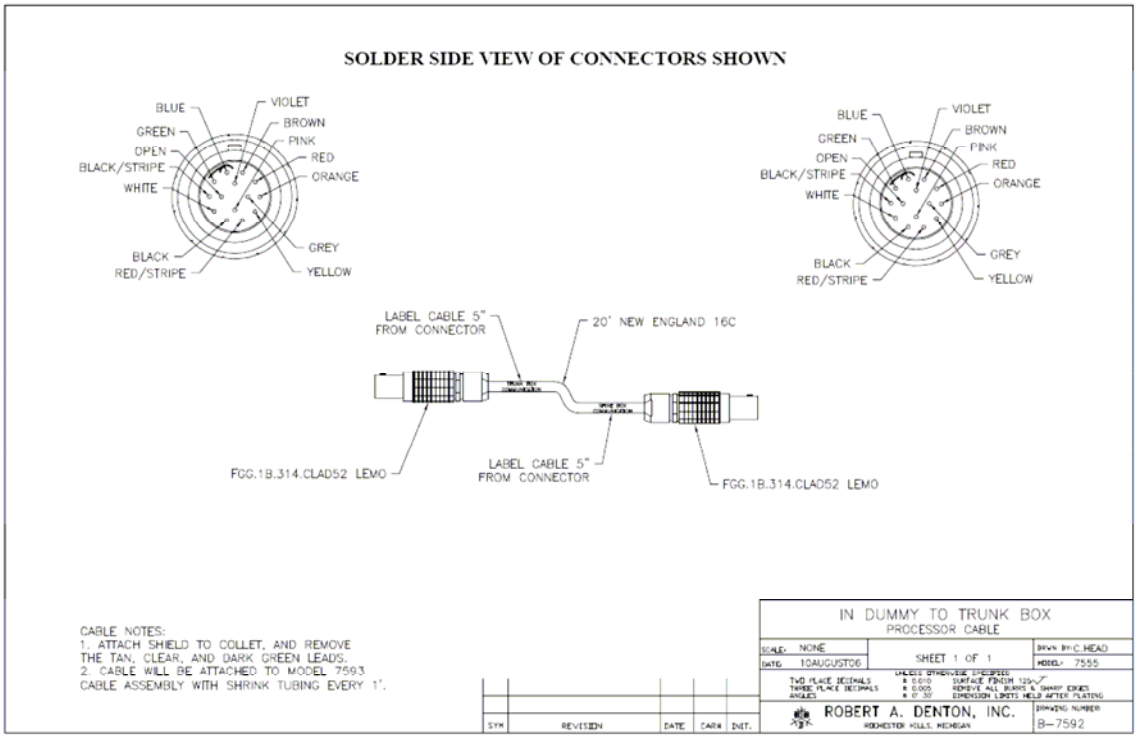

Figure C4 – 14-pin connector cable

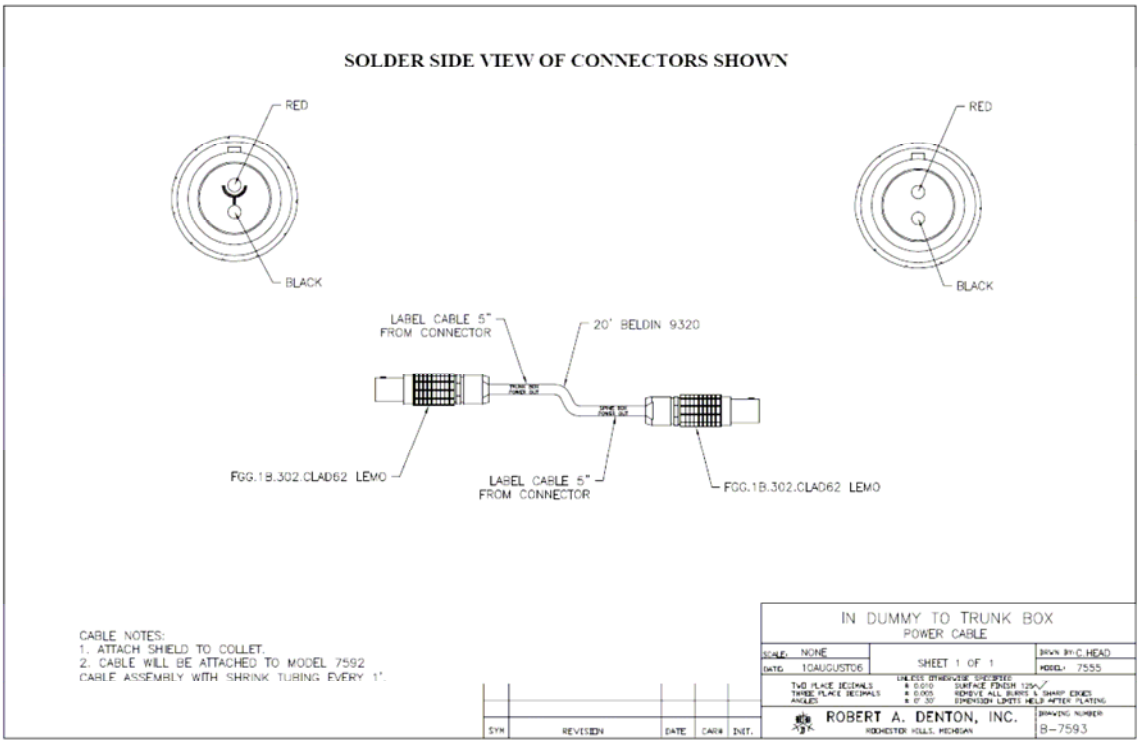

Figure C5 – Power Cable

Table C2 shows the pin descriptions for the 24-pin DAS connector on the trunk box. Figure C6 shows the interface circuits in the RibEye viewed from the 24-pin connector as currently configured.

Table C2 - 24 Pin DAS connector

PIN #	Color Code	signal
2	brown	
3	red	
4	orange	
5	yellow	
13	red/stripe	TRIG OUT
1	black	ARM OUT
10	white	TRIG IN+
11	black/stripe	DIFF Trig In–-
6	green	DIFF Trig In+
7	blue	GND
8	violet	+5
9	grey	Diff ARM In– / 485 Rx-
16	green/stripe	Diff ARM In +/ 485Rx+
17	blue/stripe	ARM IN
14	orange/stripe	485 TX+
15	yellow/stripe	485 TX–
19	pink	OTHER OUT
20	tan	5 V P.U.
12	brown/stripe	
18	violet/stripe	
21	grey/stripe	
22	open	
23	open	
24	open	

Signals highlighted in gray are reserved for Boxboro Systems use or are not currently programmed in the firmware. The wires should be insulated and not connected to anything.

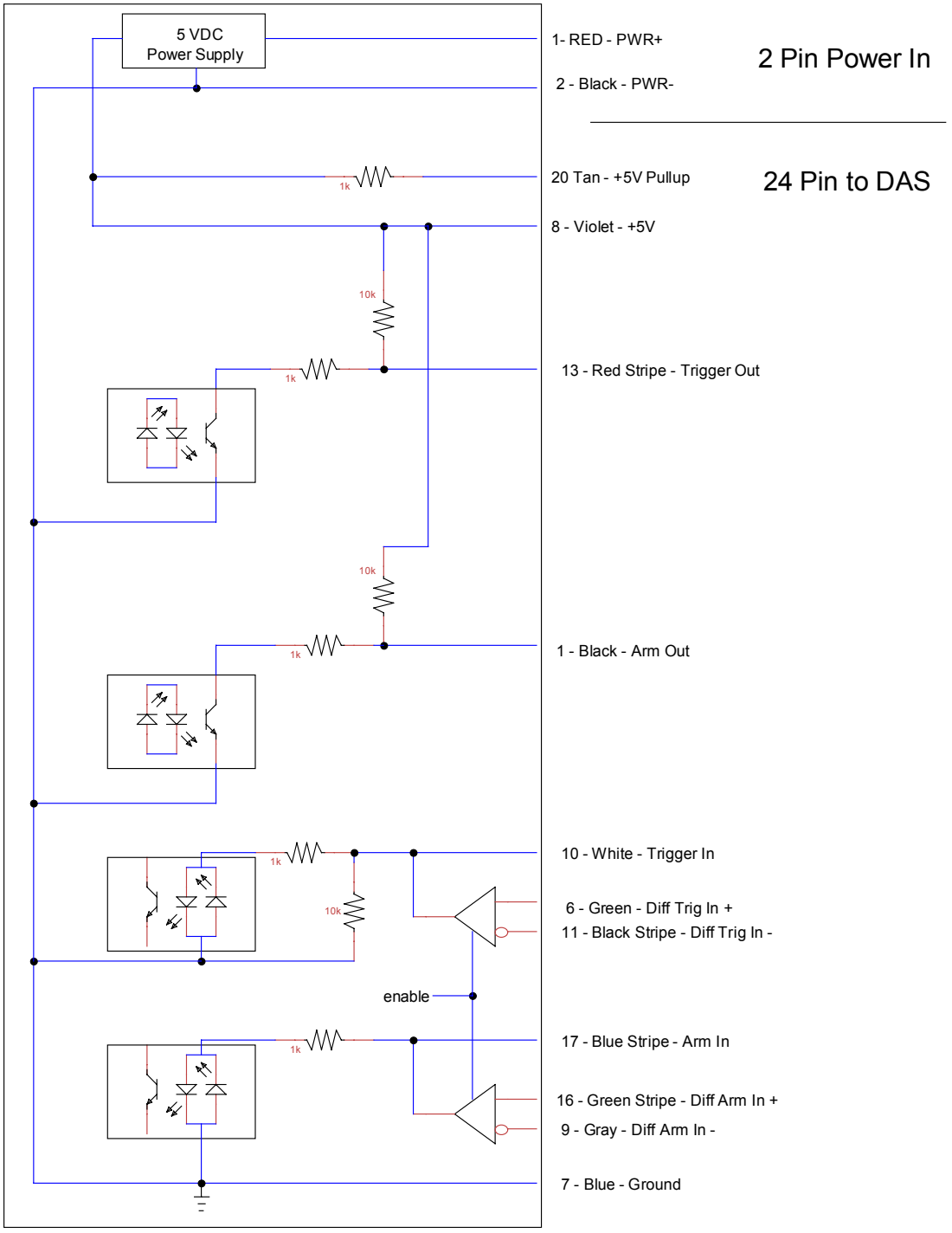

Figure C6 - Trunk Box Interface Circuits, 24-Pin Connector

Appendix D – Document Change History

Rev 5 Changes from Rev 4:

- major cleanup and reformatting
- eliminated the DUMPBINX and DUMPBINXA commands
- for the DUMPBIN and DUMPBINA commands each data point will be a 16 bit binary number = millimeter * 100, not mm*10
- the DUMPBIN and DUMPBINA commands all report error codes in the data as described in the CURRENT-POSITIONS command in section 8.8
- since all users do not change the LED positions – the LEDs are always run in the default positions - the LED_POSN, SHOW-LEDS , and SAVE commands were removed.
- added note to TRIGGERSET command that the differential input receiver is in the Trunk box, so RibEyes without a Trunk box treat a differential trigger setting the same as a switch input.
- fixed typo in GETTRIGGER
- added GETTESTCOMMENT
- added SETTESTCOMMENT
- updated command response times section 7.0
- updated section 2 port setup to add Ethernet connections
- updated section 4, bad checksum
- updated appendix A, command summary
- updated ARM command for DTS interface versions
- added Appendix B Behavior on Boot
- added Appendix C Hardware Interfaces
- moved revision history to Appendix D

Rev 4 changes from Rev 3:

- fixed formatting error in section 9.1
- updated behavior at boot in Appendix C

Rev 3 Changes from Rev 2:

- Added Erase Status (E)command
- added DUMPBINA and DUMPBINXA commands
- added TRIGGERSET command
- added GETTRIGGER command
- updated the SAVE command with Polar, WorldSID, and Ballistic SIDIIs
- updated appendix A, command summary

Rev 2 Changes from Rev1:

- Added Appendix A, command summary
- Added Appendix B, connector pinouts and wiring for CrashLink Hub
- Added Appendix C, protocol test rig behaviors
- Added documentation of ARM response when memory not erased
- Added Loader command to load new code
- Added DUMBINX command

Rev 1 Changes from Rev 0:

- added status command (S) to report current status
- added force trigger command (T) to force a trigger during acquisition
- added disarm command (D) to stop acquisition immediately
- added DUMPINFO command to report available data for downloading
- added SHOW_LEDS command to show the current LED position codes
- added SAVE command to save and LED position changes in flash
- added ?1 response to a bad checksum
- added ?2 response for a bad command
- eliminated trigger setting command (TRIGGER#) and put the acquisition times (Tstop and Tpost) parameters into the ARM command.
- Added Error response to ARM command if flash is not erased
- changed DUMPALL to DUMPBIN to allow for possible future DUMPCSV command to dump the data in CSV format, and future DUMPISO command to dump in ISO format.
- changed ERASE command to include a parameter to indicate successful erase or failure.
- fixed checksums in some command examples
- fixed LED position codes for 5th female