BG SYSTEMS, INC.

LV824 SOFTWARE MANUAL

EPROM REVISION 3.08

Contents

CHAPTER 1 OVERVIEW

This manual provides the technical information required to write software to communicate with the LV824 circuit board at the heart of all BG Systems FlyBox®, BeeBox™ and CerealBox® products. The basic function of the LV824 is to take analog and digital inputs, and convert them to a character string that can be sent to the host computer.

The software examples used in this manual are generic and could be used with any BG Systems hardware product. Where applicable, the "Hardware Manual" will have additional software examples specific to the particular hardware. Our primary development platform is a Silicon Graphics workstation running IRIX (Unix), and so the examples presented here are based on the SGI version of our software.

The "bg" library functions have been ported to both the Apple MacOS and the Microsoft Win32 architecture, and these libraries are available. Sections 4.2 and 4.3 point out the key differences.

1.1. WHAT'S NEW

1.1.1 New Documentation Format

If you have already used BG Systems products, you will notice that there is now a separate "Hardware Manual" for each product. This "LV824 Software Manual" covers the common software used for all of the products. This makes it easier to document software changes without having to update each hardware product manual.

In order to support an on-line "pdf" version of the software manual, the page numbers have been changed so that page 1 is now the cover page.

1.1.2 EPROM Revision 3.06

The introduction by Silicon Graphics of the O2 and Octane workstations required the LV824 EPROM software to be updated to revision 3.06 to accommodated the higher baud rates supported by the new SGI hardware.

The release of the SGI IRIX 6.4 operating system provides support for the high baud rates, but in order to do this certain Unix system "include" files had to be changed. This forced changes to the BG software on the host side. See section 2.3 for more detailed information.

We have made every effort to ensure that there is backwards compatibility between older versions of our software (both on the host and on the EPROM), but of course certain things will not work. For example, if you have a revision 3.05 EPROM you will not be able to set the baud rate to 115200 on an SGI Octane.

Since the EPROM was being upgraded and it was necessary to modify the key data structure in the host software, we decided to add capabilities which are not yet in the hardware. This should allow the next generation of BG hardware to use this EPROM revision.

1.1.3 EPROM Revision 3.07

The latest version of the CerealBox (LV824-H) has 8 analog output channels. In order to control these channels, a new EPROM release was required. Corresponding modifications to some of the library functions were required.

This revision of the software manual documents the use of encoder counters. This is a feature that has been in the 3.07 EPROM, but has not previously been available in the hardware.

As always, every effort has been made to ensure that new EPROMs work with old software and new software works with old EPROMs. Most of the changes (unless you are using an LV824-H) are transparent to the user.

The following functions have been modified:

```
check_rev.c Now recognizes -H, -J and
            -K CerealBoxes.  Checks
            for presence of encoders.
pack.c      Sets analog outputs 4-8
            for LV824-H
init_lv.c   Major cleanup, and new
            initialization string for
            3.07 EPROMs
```

The only situation in which incompatibility will arise is if you try to initialise an LV824-H (or later) with 3.06 host software.

1.1.4 EPROM Revision 3.08

The LV824 board has a new A/D converter chip which can be programmed to accept several input voltage ranges. The default on power up is for the range to be 0 to +5vDc as on all previous LV824s. Now the polarity can be set so that the range is +/-, and the range can be extended to 10 vDC. So you can now go from -10vDC to +10vDC.

The new A/D converter uses an internal 5 vDC reference. The old circuit board used the supply voltage as the reference. This has an implication which may cause a perceived difference in operation. With an old CerealBox, the supply voltage may have been less than 5.00 vDC. If we use 4.90 as an example., the A/D would read the maximum value of 4095 when the supply voltage is reached. With a new CerealBox, if the input signal has a maximum of 4.90 vDC, then the maximum A/D reading will be (4.9 * 4095 / 5.00) = 4013, or about 98% of full scale..

1.2 Conventions Used In This Manual

Various typefaces are used in this manual to refer to the name of something on the computer, to highlight an important piece of information, etc.

Italics are used in the body of the text to indicate computer terminology - typically a function or file name:

open_lv()

A monospace font is used whenever a code fragment is presented:

```
tios.c_flag = CS8 | CREAD | CLOCAL;
```

A monospace font preceded by a % is used for Unix shell commands:

```
%  cd CerealBox/
```

When the font is preceded by a #, it means that system administrator (or root) privileges are required for the operation

```
#  chmod a+rw /dev/ttyd2
```

Things that are really important are set apart with a "lightening bold" next to them, and an italic font is used:

WARNING.

Things that are should be noted, but are not dangerous if ignored, are set apart with the information sign:

NOTE.

1.3 A Little Bit of Background Reading

Over the course of the development of our product line, the circuit board at the heart of the products has changed from the LV816 to the LV824, and the corresponding "on-board" EPROM software has changed. The revision 3.0x EPROM software has been very stable since the introduction of the LV824 in January of 1995.

As the EPROM revisions started to change, and new features were added, it was clear that programmers would need to know what revision of EPROM they were communicating with. EPROM identification is now built into the initialization process which prevent the host software from trying to use a function that is not supported by the hardware.

The host side software has also undergone changes, and not always in step with the EPROM. In the same way, the documentation has also been revised and there is no correlation between the "revision" number on many of the FlyBox and CerealBox Owner's Guides and the software revision on the EPROM.

With the new format of a separate Software Manual, we hope we will reduce any confusion. The software manual will have the appropriate EPROM revision printed on the cover.

It is also assumed that you have a product with an LV824 circuit board (check the cover page of this manual). Although the software we provide will still work with the old LV816 with revision 2.31 EPROM's, we are no longer documenting this.

1.4 Where To Go Next

If you are familiar with BG software, can probably skim through much of this manual and go straight to the *lv3.h* header file to see what has changed. However, since there are some changes related to IRIX 6.4 and the basic *bglv* data structure, you should take a look at section 2.3.

If you have not used our products before, but you are comfortable with Unix, check out section 2.1 to see what's on the tape, and then browse through the tutorial. Of course you could just take a look at some of the examples and figure it all out.

CHAPTER 2 THE BASICS

This chapter covers the following topics: software installation, connecting the hardware, changes for IRIX 6.04, and a short discussion of typical operation. A quick review of this chapter should answer some of the common questions that you may have.

We provide source code for a number of example programs that interface to the LV824, which you should be able to modify to suit your needs. Except where indicated, we are assuming that you are installing software and running the programs on an SGI workstation. For those using a different Unix workstation, the differences will be quite minor, and most of you will be able to make the necessary changes. For those working on Macintosh or Windows platforms, there are brief descriptions of the differences in sections 4.2 and 4.3.

ⓘ *The software provided may be copied and used as needed. The library functions should be changed with caution. The examples are intended to act as templates for your own software applications.*

ⓘ *The LV824 has capabilities that are only used by BG Systems CerealBox products, and are not used with the FlyBox or BeeBox. These capabilities are highlighted in chapter 6.*

2.1 SOFTWARE INSTALLATION

Software is provided on a CD. You should create a new directory for the software and then set it as your working directory. To extract the files from the CD, enter the command:

```
prompt % tar -xvf /CDROM/unix/bg_sw_308.tar
```

The software will be copied from the CD, and several directories will be created:

`bg`	Low level functions in library format
`CerealBox`	Specific examples for the CerealBox
`FineTuning`	SGI specific examples for operation at high speed
`FlyBox`	Specific example for the FlyBox
`JunctionBox`	Specific examples for the JunctionBox
`examples`	Other examples
`sgi`	Some unsupported demos
`xlv`	Motif program with a graphical interface to the LV824
`xmp`	Motif program with a graphical interface to the JunctionBox

The software is also available from our website. Most browsers will retrieve tar files. Then type the command:

```
%  tar -xvf v308.tar
```

This will extract the same files which are on the CD. If you do not need the SGI "flight" demo program, then you can save time by using:

```
v308_short.tar
```

and if you want to get the Windows95 or WindowsNT version of the BG software:

```
bg_win.tar
```

2.2 Connecting the LV824 to a Unix Workstation

The first thing to do is decide which serial port to physically connect the LV824. The serial ports are typically numbered 1-n, with 1 typically reserved for use as an alternate console. Therefore we recommend that you choose port 2 or higher. There is a matching device in the UNIX operating system, in the */dev* directory. The serial ports are conventionally labelled */dev/ttyp1 /dev/ttyd3* etc. Consult your documentation to determine which ports are active on your system.

Once you have chosen the port to connect to, make sure that it is not configured for **terminal** emulation. See your system administrator for help with this.

When a port is configured for terminal emulation there is a UNIX task that runs in the background called a *getty*. This process checks for terminal connection, and is responsible for providing the login prompt to a terminal that is connected. Clearly if you want to talk to an LV824 this process must not be running.

WARNING.

The previous paragraph is very important. A common problem in setting up a VB24 is having the port configured as a 9600 baud terminal.

You should also check the permission on the device that you want to use. Sometimes they are set up so that only "root" can read and write to them. Since it's not a good habit to work as root, the ownership on the */dev/tty* file should be set so that everyone can read and write to it.

```
#  chmod a+w /dev/ttyd2
```

Since you must be logged on as root to do this, you should consult with you system administrator for help (or the root password).

WARNING.

The most common cause for a failure to communicate with the VB24 is a problem with the serial cable. SGI has three different serial connector configurations, so when you move the LV824 to a new machine, please be sure to get the correct type of serial cable. See section 7 for details.

2.2.1 Connecting to a Macintosh

On most Macintosh (or clones) there are two serial ports. You should generally use the modem port, since it is able to operate at higher baud rates. You will also need to have a MiniDIN-8 serial cable in order to connect to the serial port.

2.2.2 Connecting to a Windows PC

On most PC's there are two ports. You should be able to connect to either COM1 or COM2. It will only depend on what other serial devices you have connected. When the software "opens" the serial port, you will need to ensure that the appropriate COM port is specified. A standard PC Null Modem cable is required.

2.3 Updates for IRIX 6.04

In order to allow higher baud rates, SGI was forced to change the *termios* data structure. This means that two files in the BG library have changes. In order to maintain compatibility with older systems, the library files *lv3.h* and *open_lv.c* have the old and new configurations enclosed in *#ifdef* statements. You will need to make sure that the appropriate case is set in your makefiles.

If you have an older version of the BG lv3.a library, software linked with this will execute on a new SGI. However, if you recompile the library on new hardware and then build the application, the hardware will not respond. A quick fix is to define -D_OLD_TERMIOS in your Makefile.

2.3.1 Change to lv3.h

We have added the option for running at a baud rate of 115200. This replaces the 1200 baud rate, which we don't think is too useful. Since we were making changes, we have also added some features to support future plans.

For revision 3.08 the *lv3.h* file has changed again to allow input voltage range and polarity to be set in software for each analog input channel. This means that the *bglv* data structure has changed, although it remains the same size.

If you want to use the "old" *bglv* data structure, you can

```
#define REV300
```

in the *lv3.h* file.

We recommend that you use the new structure, and recompile any files that use it.

2.4 Theory of Operation

It is also useful to understand the basic sequence of events that are required to use the LV824. These steps will be illustrated in more detail in section 3.4. Since there are two computers, the workstation and the LV824, a cable and two pieces of software that must work together, there are a number of things that can go wrong, and this section should help avoid errors.

The first step is to assign the various configurable parameters, then open the device on the host computer side, and then send the parameters to the LV824. If everything is correctly configured, you will be ready to operate.

Since the LV824 is an I/O device, we assume that the host needs to collect the inputs and or send outputs at some regular interval. Some applications will need updated information faster than others, but it is important to understand that the data transfer is being made via RS-232 serial protocol, and there are finite amounts of time involved. The steps required are:

1. Send character to LV824 requesting data
2. Wait for data to arrive
3. Read data from the port

The above sequence seems pretty straightforward, and usually it is. However, it is important that you wait long enough for the data to arrive, and this will also determine the mode in which you want to read the data. The amount of time that you should wait will be determined by the baud rate and the number of channels that you are sampling.

There may be limitations on the workstation side that limit the update frequency

2.5 Extra Credit

The LV824 uses the RS-232 serial communication protocol, and so some knowledge of how UNIX handles serial ports is required. You should either consult the software manuals that came with your computer or the on-line *man* pages for detailed information.

In particular you may want to study: the *serialio* and *termio* pages; the *open*, *read*, and *ioctl* pages; and the *getty* page. Familiarity with these topics will make some of the following material easier to understand.

We also make use of UNIX interval timers. Since this is not a real-time programming guide, we again encourage you to consult the man pages for: *setitimer*, *signal*, *sigset*, *sigpause*, and *sigrelse*. The examples in this chapter make use of the above utilities, and while you can probably get by without a detailed understanding, you should have some understanding of these topics.

CHAPTER 3 TUTORIAL

Perhaps the easiest way to understand how to use the software is to study an example. In the */CerealBox* directory, the source code in *e_test.c* is a fairly simple program that samples the LV824 and prints the values to the terminal.

There is a *Makefile* in the directory, and you should be able to type:

```
% make e_t
```

to compile and link the executable *e_t*

For those of you familiar with earlier releases of BG software, you should still read this section to check for new features. As mentioned in section 2.3.1, the main header file, *lv3.h*, has been modified, and we recommend that you use this new file and recompile anything that has a dependency on this file.

3.1 SETUP

As you go through this example, look at the file *e_test.c* and make sure that it matches this document.

The low level library routines use a data structure, defined as *bglv*, that is defined in *lv3.h*. (Note that this file is in the */bg* directory.) This data structure must be declared in the main file.

```
#include "lv3.h"
bglv bgdata;
// Main BG data structure
```

ⓘ *Note the use of C++ style comments // which are not in the actual code, but are used here to help readability.*

ⓘ *Note that only the following members of the structure need to be set. Those designated CB in the comment, are not used by FlyBoxes or BeeBoxes at all and should be set to zero.*

The following members of the *bglv* data structure must be set before being used.

```
int    analog_in;
// Analog input selector
int    dig_in;
// Digital input selector
int    analog_out;
// Analog out channel selector - CB
int    dig_out;
// Digital output selector - CB
int    aout[8];
// Analog output data - CB
int    dout[3];
// Digital output data - CB
int    baud;
// Baud rate selected
int mp_dig_in;
int mp_dig_out;
// Used with JunctionBox, set to zero
int enc_sel;
//  Selected encoder, set to zero
int enc_type;
//  Absolute/Incremental, set to zero
```

```
int a_range;
//  Analog input channel voltage
range
int a_polarity
// Analog input channel voltage
polarity
```

The remaining members of the structure are computed in *open_lv()* and *init_lv()*. The *bglv* structure is passed to most of the library functions in the call sequence.

In the examples, we call the function *setup_lv()* early in the main program, and we would recommend a similar function be used in your software. This function sets the appropriate members of the data structure, and then opens the serial port, and once the serial port is open, the LV824 can be initialized. This is a function that you will need to modify for your own application to set the required input and output channels.

First we select the number of analog channels to sample:

```
bgdata.analog_in = 0;
bgdata.analog_in = AIC1 | AIC2 |
                   AIC3 | AIC4 | AIC5;
```

The *analog_in* member of the *bgdata* structure sets any combination of the 8 available analog inputs. In the example above, channels 1, 2, 3, 4, and 5 have been requested by "or-ing" the predefined values AIC1 - AIC5. If you have connected an additional analog input to the DB-9 at the back of the LV824, you could add it to the list:

```
bgdata.analog_in |= AIC8;
//  Analog input 8
```

If you examine the definitions in lv3.h, you can of course use the shorthand hex representation to set these parameters.

Note that we use the logical (not C based) counting system that assigns AIC1 to pin 1 on the connector. Of course in C this corresponds to the zero element in the array.

Next we select the digital (discrete) channels:

```
bgdata.dig_in = 0;
bgdata.dig_in = DIC1 | DIC2;
```

In the example above, we have selected digital inputs 1-16. These are selected according to the following table:

DIC1	*0x10*	*Channels 1-8*
DIC2	*0x20*	*Channels 9-16*
DIC3	*0x40*	*Channels 17-24*

Note that since we are using the *e_test.c* program as an example, we are not setting the digital output selector to zero. This is because if we try to set the digital outputs, the *init_lv()* function will fail and report that digital outputs are not supported by the LV824-E.

To avoid problems with uninitialized variables, we now set any unused members to zero:

```
for ( i = 0; i < 8; i++ )
   bgdata.aout[i]  = 0; // set all 8
for ( i = 0; i < 3; i++ )
   bgdata.dout[i]  = 0; // set all 3
for ( i = 0; i < 13; i++ )
   bgdata.mp_dout[i]  = 0; // set all
bgdata.mp_din   = 0;
bgdata.mp_dout  = 0;
bgdata.enc_sel  = 0;
bgdata.enc_type = 0;
bgdata.a_range = 0;
bgdata.a_polarity = 0;
```

Setting these values to zero is good practice and often eliminates "random" problems when an application sometimes works and sometimes doesn't.

3.2 Opening and Initializing

Next we set the baud rate in for the *bgdata* structure:

```
bgdata.baud = BAUD192;
```

Then we open the serial device on the host with the *open_lv()* function. The call takes three arguments: the first is the address of the BG data structure the second specifies the */dev/tty* that you are connected to; and the third specifies whether the read will be blocking or non-blocking. See the man page on *open()* for a detailed discussion of blocking.

```
st = open_lv(&bgdata, "/dev/ttyd2",
FB_NOBLOCK);
```

The above example represents a bad case of hard-coding a magic number. It is used as above only for purpose of illustration. We recommend passing in a string variable, or a null pointer, and using the environment variable FBPORT to specify the port to use. See also section 6.1.

In general for SGI workstations, setting the baud to 19200 as shown above should work well, and the **default on power up for the LV824 is 19200**. This is important when you try to send initialization data to the LV824, since if your serial port is configured for a different baud rate, nothing will get through ! The *open_lv()* function sets the baud rate to 19200 to begin with, and the *init_lv()* function sets the baud rate to the desired operational rate.

Note that when you push the reset button or cycle power, the LV824 reverts to 19200 baud.

If you embed this code in an application that must run in real-time at a desired frame rate, do not block. If you block, and for some reason (UNIX interrupts or latency) the data is not in the serial driver when you expect it, the program will hang there until the data arrives. This is OK if the code is not time critical.

If you do not have read/write permission on the requested serial port, the *open_lv()* call will fail. If the port is configured as a terminal this may happen. Ask your system administrator for help.

During the *open_lv()* call, the software sends a character to the LV824 to retrieve the EPROM version. This is stored in the "*Rev*" structure:

```
bgdata.Rev.year
//  Year the EPROM s/w was written
bgdata.Rev.major
//  Major release  (currently 3)
bgdata.Rev.minor
//  Minor release  (currently 0)
bgdata.Rev.bug
//  Bug release    (currently 7)
bgdata.Rev.alpha
//  EPROM version e, f, g, h, j or k.
```

This data is available for checking by your application, and is used in the *init_lv()* routine to make sure that the requested setup matches the hardware.

The information from the revision and the channel setup is used in the call to initialize the LV824:

```
st = init_lv(&bgdata);
```

This call simply sends a string to the LV824 that tells it what baud rate to run at, and which channels to sample.

Note that the LV824 starts out assuming communication at 19200 baud. If you want to operate at a different baud rate, you can set this when you init_lv() with bgdata.baud

set to BAUD9600 for example. However, be aware that if you don't close_lv(), the LV824 will stay at 9600 baud, and the next time you initialize, the changes will not be received. You need to either push the reset button or turn power off and on to reset the LV824 to 19200.

The *init_lv()* routine performs some compatibility checks. For example, if you have selected:

```
bgdata.dig_in = 0x70;
```

to sample all 24 channels and the major EPROM revision is 2, then the *init_lv()* call will fail.

If all goes well, you will see something like:

```
% ./e_t
Copyright (c), BG Systems 1997  Revision 3.07e
LV824-E
Setup OK
-1.00  -1.00  -1.00  -1.00  00000000
00000000  00000000
```

Note that you will see a copyright string if the call to check_rev() succeeded, and if the requested configuration is valid, then you will see "Setup OK".

3.3 The Interval Timer

This next section is operating system dependent, and is based on SGI IRIX. Other operating systems will vary the specifics of this section, but the basic ideas will still hold.

We now call the function *init_timer()* which sets up the interval timer, to trigger an alarm every 50,000 microseconds (50 ms). For most SGI systems, you should be able to set this to 20 ms (sampling at 50Hz). Check your man pages for the clock resolution of the system. (Setting the timer to go off every microsecond is unlikely to work !). Depending on the baud rate and the channels sampled, you may be able to sample as fast as every 10 ms on a UNIX machine. (On a Macintosh the serial port can handle much faster update rates — over 300 Hz for a single channel).

```
itv.it_interval.tv_sec  = 0;
itv.it_interval.tv_usec = 50000;
itv.it_value            =
itv.it_interval;
setitimer(ITIMER_REAL, &itv,
          (struct itimerval *)0 );
```

Next we set up to catch the alarm signal sent by the interval timer. (The routine that we call does nothing.)

```
sigset(SIGALRM,  catcher);
```

This *init_timer()* is simply an example of using interval timers and alarms that has worked well for us.

ⓘ *If you need to run the program in debug, this can be very annoying, so you may want to just use an sginap(3) instead. Using the alarm gives more reliable real-time performance.*

Using the interval timers is simply a method that we have found to be reliable. On high end Silicon

Graphics machines with multiple processors, you may be able to get more accurate or fine grained timing by using the graphics subsystem clock. You might look into the Performer software system calls to see how to get a more accurate clock.

3.4 CYCLING

The normal cycle looks something like this:

```
st = w_lv(bgdata.sp_fd, "o");
// request data
sigpause(SIGALRM);
// wait for the alarm
st = r_lv(&bgdata);
// get the data
```

After reading the data, we choose to print it to the terminal in this example. This is of course where you would choose to do something interesting with the data.

The input values are computed within the *r_lv()* call, and stored in the *bgdata.ain[]* array and the *bgdata.din[]* array. The analog values are scaled between -1.0 and 1.0 for convenience. The digital values are packed into the integers in the array, so you need to mask (as in the example) to determine which bits (channels) are set.

When you run the e_t program, you will see numbers printed on the screen, which will change as the input voltage changes. If you have nothing connected, just brushing a finger over the analog input DB-9 will cause the values to change.

If you want to run this as a single process, you would usually put the intensive processing just before the *sigpause()*. The *sigpause()* suspends your process until the signal is caught, so you should do your processing before you are suspended.

w_lv() should always be followed by r_lv(). If you are using analog or digital outputs (LV824-F or higher), send_outputs must be followed by check_inputs(). Do not mix w_lv() with check_inputs().

CHAPTER 4 ADVANCED TOPICS

The previous chapter explained by way of an example what is involved in writing software to communicate with an LV824 based product. In this chapter some more advanced material is covered.

Section 4.1 explains how to get better performance from an SGI IRIX system. The general principles in that section would probably apply to other UNIX systems, but of course some of the low level functions would be different.

Section 4.2 highlights the modifications to the library to run under the Macintosh operating system.

Likewise, section 4.3 highlights the changes needed for a Windows based system. Due to limited resources and experience with the Windows platform, this section isn't really advanced, and any comments from experienced Windows programmers will be welcomed.

4.1 HIGH UPDATE RATES

You've just installed your blazingly fast multiprocessor system, and you want to see just how fast you can sample your FlyBox. You go ahead and compile the software but it will only run at 50 Hz ! There are a few things that you will need to know in order to improve on this performance.

4.1.1 Theoretical Limits

When you sample a FlyBox (or BeeBox or CerealBox), there a number of characters involved in each data transmission. Sending the "o" to the LV824 to request data takes 2 characters, and the characters sent back include a 'B', a '\n' and two characters for every analog channel and 2 characters for each group of 8 digital channels. Each character is 8 bits, so the update rate is simply

```
HZ = baud /( chars * 8 )
   = baud /(4+2*n_analog+2*n_dig/8)*8
```

So for example, if you are sampling 5 analog channels and 16 digital channels at 19200 baud, the theoretical maximum update rate is:

```
HZ = 19200/((4+10+4)*8) = 133
```

Of course this assumes that everything happens instantaneously on the host and also on the LV824. On the LV824 we don't have a MIPS processor (it's an Intel 8031) and we do have to convert the analog channels to digital, so there are typically from 2 - 10 ms of overhead depending on the number of channels to be sampled. But it is still possible to sample a FlyBox at more than 50 Hz. But how

4.1.2 Tuning Your Software

As an example, lets sample 8 analog channel at 19200. The maximum update rate is about 60 - 80 Hz

(including the overhead). However, if you set up the *CerealBox/e_test.c* program to sample with:

```
itimerval.usec = 10000;
```

(100 HZ), you will probably observe the following when you run *CerealBox/e_t*. Pretty early on, there will be several:

```
200 Read attempts
```

and also a "dropped frame",

```
Frame Dropped
```

This means that *e_t* was checking the serial port too often. In the *r_lv()* function, if no data is read, we keep "polling" the serial port for up to an additional *MAX_TRIES*. At that point we just continue, but now in *e_test.c* we send another "o" to the CerealBox, and now we can run with no more "200 read attempts" messages. This is because we have dropped a frame, and while there is data in the serial port it is now from the previous frame. If you can live with a 10 ms lag, this isn't really a problem. However, you would be better setting the itimerval to 20000 (50Hz).

4.1.3 Running Faster

Let's now look at the case where you want to sample fewer channels faster, or the same number of channels at a higher baud rate. Theoretically, sampling 5 analog channels at 38400 baud is possible well over 150Hz. If we set up *e_test.c* with this configuration, and set the itimerval to 6700, and then run the program, the update rate refuses to go higher than 100 Hz, even though we have set the timer to go off at 150 Hz.

The problem is that the process doesn't have access to the *fastimer*, and is limited to a 10 ms (100Hz) capability. This can be fixed by running the process at a higher priority (see the *npri* man page). The program *fast_test.c* sets this in the *init_timer()* call – assuming that you are running with *root* privileges.

Now that we have *root* privileges, we can get the alarm to go off at the requested interval and we are ready to fly !

Well not so fast. If you try to run in this configuration you will again be confronted with the "Dropped Frame" message.

The problem here is that the serial port has not been configured to respond any faster than 50 Hz (which is the SGI default). So the alarm is going off in your software at 150 Hz, but the system is not interrupting itself to update the serial port any faster than 50 Hz.

If you want to run faster, you need to fine tune the Unix kernel. This is actually not very difficult.

4.1.4 Really Running Faster

YOU NEED TO BE ROOT TO CONTINUE.

Failing this, discuss this with your friendly system administrator

You may be able to improve the latency of the serial port by including the line:

```
ioctl(SIOC_TIMER, 0);
```

in your initialization routine. This has the same effect as the kernel change described below, but it is not permanent.

To get faster response from the serial port, you need to fine tune a system parameter. (See the SGI man page on *serial*). SGI indicates that this may impact overall system performance, so if all you really need to run at is 50 Hz, this may not be worthwhile.

In the */var/sysgen/master.d* directory is a file called *sduart*. Make a copy of this, change it to u+w so you can edit it, and change the (HZ/50) to (HZ/300). Versions of these files are included in the *FineTuning* directory. Then make a copy of */unix* (*/unix.sav*). Run *autoconfig* which will create a file called */unix.install*. Then reboot your system. Really, it's that easy.

You should now be able to sample at 250 Hz (depending on baud rate and number of channels).

Make sure that you do this either on a machine for which you are the only user, or with the permission of your friendly system administrator

4.2 Running Under MacOS

Several low-level pieces of the software are quite different for the Macintosh. Opening and configuring the serial port is obviously quite different, as are the read and write functions.

Routines such as *open_lv.c* and *read_write.c* have been modified to call MacOS functions where appropriate, and these MacOS functions are in: *mac_rw.c* and *mac_serial.c*.

A main function is included which sets up the software assuming that some form of console output is supported. It does not include any fancy GUI software. The software was developed using the Symantec C development kit.

This software is available on request by sending e-mail to *john@bgsystems.com*.

4.3 Running Under Windows95 or WindowsNT

The Unix software has been ported to Windows95 and WindowsNT using the Microsoft development environment and the Win32 API. This API is fairly close to Unix, in the sense that read() and write() type functions are supported.

The software will be provided on floppy if requested, and can also be down-loaded from *www.bgsystems.com* (see section 2.1).

While the software is quite close to the Unix version of the bg library, you will need to apply your own expertise as far as getting better performance from your application. In particular, the software developed uses timers in a very simple fashion.

It is hoped that at some point in the future the ActiveX or DirectX interface will be supported. When this happens, updates will be posted on the web.

4.4 Using The Encoder Counters

A new feature on the LV824 is the availability of encoder counters on the board. This features is available for all LV824 CerealBoxes (specify a -4e or -8e when ordering), and is also used in some custom BeeBox and FlyBox applications in which an encoder is mounted. An exmaple of setting up and sampling the encoder counters is in the file *CerealBox/e4_test.c* .

You will need to decide whether to use absolute or incremental encoder values. Absolute encoders are represented as 24 bit numbers. Incremental values are represented as smaller numbers, since they contain only the number of counts since the last sample.

In the example, two encoders are selected for sampling (0x01 and 0x02):

```
bgdata.enc_sel = 0x01 | 0x02;
```

And encoder counter number 2 is selected to be incremental:

```
bgdata.enc_type = 0x02;
```

The default is for the encoder counter to be absolute. To set an incremental counter you need to set the appropriate bit.

When software is cycling, the encoder counter values are stored by the r_lv() call, so the values we are sampling are stored in:

```
bgdata.enc_abs_val[0];  // encoder 1
bgdata.enc_inc_val[1];  //  encoder 2
```

You should be aware that if you are using the absolute values the counters initialise to zero, if you turn the encoder anti-clockwise you will get a value of $2^{24.}$

4.5 Input Voltage Selection

With version 3.08 of the EPROM software, a new A/D converter chip is used which allows each input voltage to be programmed to accept several input voltage ranges. The default on power up is for the range to be 0 to +5vDc as on all previous LV824s. Now the polarity can be set so that the range is +/-, and the range can be extended to 10 vDC. So you can now go from -10vDC to +10vDC.

This means that the *lv3.h* header contains some changes to the bglv data structure. The size of the data structure does not change, but some of the "spares" are now assigned.

The polarity for each channel is contained in the member:

```
int  bglv.a_polarity
```

The voltage range for each channel is contained in the member:

```
int bglv.a_range
```

Each of the eight analog input channels can be set to sample a different range, and this is done by "or-ing" in one bit into the members of the data structure. If the bit is not set, then the polarity is 0 to + voltage, and the range is 5 vDC. If the polarity bit is set, then sampling is for a +/- range, and if the range bit is set then sampling is to 10 vDC.

As an example, we set channel 1 to sample 0 to +5vDC, channel 2 to sample -5 to +5 vDC, channel 5 to sample 0 to +10 vDC, and channel 8 to sample -10 to +10 vDC.

```
bgp->analog_range = 0;
bgp->analog_range = AIN5 | AIN8;
```

this would set analog channels 5 and 8 to a10vDC range.

To set a +/- range, you set:

```
bgp->analog_polarity = 0;
bgp->analog_polarity = AIN2 | AIN8;
```

This has set the polarity to +/- on channels 2 and 8, and so channel 8 now samples from -10 vDC to +10 vDC.

In the *convert.c* file, all values are still scaled between -1.0 to +1.0. If you want to use the raw data, you can check:

```
bgp->sparef[0-7]
```

Note that when +/- polarity is used the least significant bit is set for negative values.

CHAPTER 5 OTHER SOFTWARE

Other demos and examples are provided for your entertainment and edification. There are examples provided for all the different hardware products that BG Systems manufactures. The programs described in this chapter are generic.

5.1 EXAMPLES/CHECK_EPROM.C

This is a simple program that opens the serial port, sends a "T" to the LV824, and waits for an answer. It should respond with a string that looks something like:

```
%   ./eprom
Attempting to open the serial port
Copyright (c), BG Systems 1997  Revision 3.07e
        OK
```

If this program produces no response, turn to Chapter 7 for troubleshooting tips, or review the setup procedure in section 3.1 and 3.2.

5.2 SGI/FLIGHT

This is a version of SGI's flight demo that takes inputs from the a FlyBox or BeeBox and can be run by typing:

```
% flight -S flybox
```

The joystick, levers, and buttons on the FlyBox or BeeBox control *flight* according to the following table:

data	Function	FlyBox or BeeBox Control
ain[0]	Roll	Joystick left/right
ain[1]	Pitch	Joystick fore/aft
ain[2]	Yaw	Joystick twist (FlyBox)
ain[4]	Throttle	Left Lever
ain[5]	Flaps	Right Lever (FlyBox
din[0] 0x01	Gear	Top Left Button
din[0] 0x40	Rocket	Bottom Left Button
din[0] 0x80	Missile	Bottom Right Button
din[1] 0x01	Cannon	Trigger (FlyBox)

If you have a cockpit of your own and have connected flight controls via a CerealBox, then you will be able to use this software as long as you have connected the controls according to the table above.

This is included purely for entertainment.

This version of flight was compiled for IRIX 5.2. If you are running IRIX 4.0.5 this may not work. In this case, do NOT call BG Systems !

Likewise, if you are running on IRIX 6.xx, you may also have problems.

5.3 XLV

The *xlv* program is a Motif application that gives a graphical interface to any of the LV824 based products. It's handy for checking the behaviour of the hardware, since it's easy to see if it responding correctly.

To use *xlv*, you will need to move the *Xlv* preferences file to the appropriate place in your Unix environment - otherwise the buttons and sliders will not look very good.

To run the application, you simply type

```
prompt % xlv
```

at the prompt. When the main screen comes up, you need to push the OPEN button. This attempts to open the serial port, and then check the revision of the EPROM. If the device (FlyBox, BeeBox, or CerealBox) is not connected, or does not have power, a dialog box will come up and suggest that there is a problem. Quit, make sure the connections are OK, and try again.

Once the serial port is open, you will need to push the INIT button. This sends a string to the LV824 with instructions as to the channels to sample. Under the configuration menu there are preset choices for FlyBox, BeeBox, and CerealBox. The CerealBox configuration actually brings up a dialog box from which you can select the channels to sample.

At this point you can press the RUN and STOP buttons to start and suspend sampling. If all is well, the sliders should move as the joystick (or other input) is moved, and the buttons should change colour depending on the state of the input.

For EPROM 3.07, the layout is slightly different to accomodate the 8 analog output sliders.

CHAPTER 6 THE LIBRARY FUNCTIONS

In the *bg* directory there are several C files that you should examine. These files contain the low level functions that are used to set up and communicate with the LV824. You may be able to get by without looking too closely at these files (just use the *e_test.c* program as template).

The source code is provided so that you can examine the procedure used, but you should always retain the original version of the files.

6.1 OPEN_LV()

```
int open_lv(bglv *bgp, char *pt,
            int flag)
bglv  *bgp;
// a pointer to the bg data structure
char  *pt;
// a pointer to a character string
int    flag;
// can have value of 1 or 2
```

This routine sets up the serial port for use by the LV824. If the second argument is NULL the routine checks the environment variable FBPORT, and if that is not set tries to open */dev/ttyd2*. If your RS-232 cable is connected to a different port, you should use that as the second argument, i.e.:

```
st = open_lv(&bgdata, "/dev/ttyd1",
1);
```

or from the UNIX shell

```
% setenv FBPORT /dev/ttyd4
```

ⓘ *Note that if you compile your program with a serial port passed to open_lv() (as shown above), and you move to a different workstation on which you must use a different port, the program will not work. (The technical term for this is hard-coding a magic number). We recommend leaving the second argument NULL, and using the environment variable (which can be set in your .cshrc file). If you don't like using environment variables, then our second suggestion would be to set a character string based on a command line argument to pass in to open_lv().*

ⓘ *open_lv() first checks for the environment variable FBPORT, then checks the string passed in as the second argument, and if all else fails uses /dev/ttyd2 as a default.*

If the port is not opened, an error message is printed, and a value of -1 is returned.

The third argument to the *open_lv()* call should be set to:

```
flag = FB_NOBLOCK;  or    flag = 1;
flag = FB_BLOCK;    or    flag = 2;
```

If FB_BLOCK is used, any reads of the serial port will wait if there is no data. This ensures that the read will return with data, but may cause undetermined delays, and should not be used in real-time critical sections of code.

If FB_NOBLOCK is set, reads will continue even if there is no data to read. This means that there will be no delays in the application, but there may be no valid data. In general we recommend this option if you allow sufficient time between writing to the LV824 and reading the serial port.

In order to support older and newer SGI hardware (and include files) it was necessary to modify the *set_baud()* function. The new version sets the

```
tios.c_flag = CS8|CREAD|CLOCAL
tios.o_speed = B19200
```

whereas the old version sets

```
tios.c_flag = B19200|CS8|CREAD|CLOCAL
```

The new SGI libraries set the baud in the new member *o_speed*. If you compile the *lv3.a* library and link to new SGI libraries, your application will stop working.

If you are not sure which SGI library is on your system, you can look at the file:

```
/usr/include/sys/termios.h
```

to see if it has the *o_speed* member.

open_lv() must be called before trying to collect data.

6.2 CHECK_REV()

```
int check_rev(bglv *bgp)
bglv *bgp;
// A pointer to the bg data structure
```

This function sends a 'T' to the LV824 and waits for a 44 character string to come back. This string should contain the copyright information as well as the software revision on the EPROM, and the model number of the LV824. This function also checks for the presence of encoder counters on the LV824.

```
bgdata.Rev.major;
//  Major s/w revision
bgdata.Rev.minor;
//  Minor s/w revision
bgdata.Rev.bug;
//  Bug fix. (we never have these...)
bgdata.Rev.alpha;
//  e, f, g, h, j or k, depending on
//  model.
```

WARNING

If this function fails, you should check your connections (cables) and the configuration of your host computer. Make sure that the serial port on the host is NOT configured as a terminal. (See section 7.0)

This function is embedded in the *open_lv()* call, since the EPROM revision is required by the *init_lv()* function for consistency checking. As new products or features are added, we rely on this check, so you edit it at your own risk.

3.06 Problem

If you use version 3.06 of this routine, it will not recognize LV824-H or newer. Version 3.07 of this routine recognizes -H, -J, and -K CerealBoxes.

6.3 INIT_LV()

```
int init_lv(bglv *bgp)
bglv *bgp;
// A pointer to the bg data structure
```

If all the parameters have been set in the *setup_lv()* function, and *check_rev()* has returned successfully, then the *init_lv()* function sends the setup information to the LV824.

The channels that you have requested to sample in the *setup_lv()* function are checked against the EPROM revision to make sure that you are not about to make an impossible request of the LV824.

For revision 3 EPROM's, *init_lv()* also expects an acknowledge from the LV824. If the acknowledge comes back with an 'f' then the setup failed. Generally this should not happen, because of the consistency check, but if it occurs you may simply need to push the reset button on the LV824.

init_lv() sets various members of the *bglv* data structure, such as the computed number of characters to read.

init_lv() must be called before trying to collect data.

init_lv() has been restructured to make it clearer. For revision 3.07, a new initialization string is used (starts with a 'c') and allows for future expansion. The logic in this routine maintains compatibility with all EPROMs from revision 2.20 and higher.

6.4 W_LV()

```
w_lv(int, char *)
int fd;
// serial file descriptor
(bgdata.sp_fd) char *string;
// Character(s) to send to the LV824
```

This is a low level routine that allow you to send data to the LV824

The character string sent with *w_lv()* should be an "o" for normal operation. There are several characters that you can send with the *w_lv()* call:

- *c Setup for revision 3.07 and later EPROMs.*
- *o Send data. First character returned is a 'B'.*
- *p This is for the CerealBox models LV824-F and LV824-G models. Should not be used with a LV824-E.*
- *R Reset. Reset the LV824 according to the following characters. See init_lv() for details. (rev 2.2 EPROM)*
- *r Reset for revision 2.3 EPROM.*
- *s Setup for revision 3.0 to 3.06 EPROMs. There are now three characters to send at setup, and the LV824 returns a character.*
- *T Test. Return a fixed length (44 characters) string. The LV824 will blink the yellow LED with this command.*

Note that with Revision 3.0 software, the reset or setup character is "hidden" within the init_lv() call. It is listed here primarily for completeness.

Always follow w_lv() with a call to r_lv(). Do not follow it with a call to check_inputs().

6.5 R_LV()

```
r_lv(bglv *)
bglv *bgp; // pointer to bg data structure
```

To read data from the LV824:

```
st = r_lv(&bgdata);
```

This function reads whatever is in the serial port. If the expected number of characters are there, then the character string is passed to the *convert_serial()* function to turn the character string into useful numbers in the data structure.

The analog inputs are scaled between -1.0 to 1.0 for convenience. The digital inputs are packed into the three integers in the *bgdata.din[]* array, and should be masked off as needed.

r_lv() should always follow a call to w_lv(). If you are using send_outputs() for analog or digital outputs, do NOT call r_lv(), but call check_inputs().

6.6 CLOSE_LV()

```
void close_lv(bglv *bgp)
bglv *bgp; // pointer to bg data structure
```

This should be called when your application is cleaning up, and resets the baud on the LV824 to 19200, closes the file descriptor, and prints statistics.

If you are running at a baud rate other than 19200 you MUST call this routine to set the baud rate back to 19200 on the LV824.

6.7 SEND_OUTPUTS() - CEREALBOX LV824-F, -G AND -H

```
int send_outputs(bglv *bgp)
bglv *bgp;
// pointer to data structure
```

This function sends data to the CerealBox for digital and/or analog outputs. It also requests input data on return.

The data should be set in the main program, or function. The data is set as bits for the digital outputs, and integers between 0-4095 for the analog outputs.

The digital outputs are set by "or-ing" the appropriate bits to create a hex number:

```
pin             bit         hex
1               0001        0x1
2               0010        0x2
3               0100        0x4
4               1000        0x8
```

Combing these bits, you can set the appropriate outputs:

```
1+2             0011        0x3
3+4             1100        0xc
2,3,4           1110        0xe
```

The analog outputs are a bit easier, since they scale 1 mVolt per bit:

```
bgdata.a_out[0] = 1023;
// analog out 1 to 1/4 scale 1.023 V
bgdata.a_out[1] = 4095;
// analog out 2 to full scale 4.095 V
```

For the LV824-H CerealBox, an additional 5 analog outputs may be set.

send_outputs() must be followed by a call to check_inputs(). Do not call r_lv() after a call to send_outputs().

6.8 CHECK_INPUTS() - CEREALBOX LV824-F, -G AND -H

```
int check_inputs(bglv *bgp)
bglv *bgp;
// pointer to data structure
```

This function is used to read inputs when outputs have been sent to the CerealBox. The reason to use this function instead of *r_lv()* is that if data being sent to the LV824 gets "lost", the CerealBox will send a request for a handshake from the host. This is to try to reduce the number of errors in transmission, and to prevent the host and the CerealBox from getting out of sync.

The input values are stored in the *bgdata.ain[]* and *bgdata.din[]* arrays just as in the *r_lv()* call.

check_inputs() must follow a call to send_outputs().

Chapter 7 Trouble Shooting

We test every product prior to shipping, and have confidence that when you connect to a device which uses the LV824 it will work. However, since we have two pieces of hardware (host computer and LV824), three pieces of software (host application, LV824 embedded software, and the host operating system), and a cable, we realize that there will be occasions when you connect a LV824 it will appear that nothing is happening.

7.1 The LV824 Is Not Responding

You have plugged everything in, and you try to run a test program (*eprom* or *e_test*), and the program exits with the message:

```
% eprom
Attempting to open the serial port

Writing a 'T' to the Box produced no answer.

The expected string was not returned from the BG box.
Here are some possible problems:
   1. Check power to Box
   2. Check the serial cable
   3. Check the environment variable FBPORT
      - does it match the connected serial port ?
   4. Is the serial port configured as a terminal ?
     - if so use "System Manager" to disconnect the port
   5. You have an old FlyBox (serial no. less than 60)
       which has a revision 1.0 EPROM.  Call BG Systems.

         Unable to open port
bye
```

There are several possible reasons for this, and only rarely is it a hardware failure on the LV824.

7.1.1 Power

The first thing to check is to make sure that there is power to the LV824. There is a green LED on the LV824 right next to the RS-232 connector. This should be on when there is power to the LV824. If the green LED is not on, you need to check the power connection of the device in question.

> ⓘ *On a FlyBox, you can check the red LED on the front cover, which should glow when there is power. If this LED is on, and the green LED is off, then for some reason power to the LV824 within the FlyBox is not getting through. Contact BG Systems.*

If you are sure that power is being supplied to the device, and the green LED is not on, then there is a problem and you should contact BG Systems.

If the green LED is on, then you should turn power off and then back on, and look at the yellow LED next to the green one. This should blink one time for an LV824-E, twice for an LV824-F, three times for an LV824-G, and four times for an LV824-H. If you have power (green LED), and the yellow LED doesn't blink then there is a problem with the board and you should contact BG Systems.

If the yellow LED blinks on power up, then the CPU and memory on the LV824 are functioning correctly.

7.1.2 Serial Cable

The most common problem is when there is a problem with the serial cable. This problem is compounded by the fact that SGI uses three different types of serial connector (although it appears they have finally settled on the PC DB-9). We make every effort to ship the correct serial cable with the products, but if you move the BG device to a new workstation for which you do not have a BG serial cable proceed with caution. The tables below list the functions on the pins for the various connectors:

Signal	SG-DB-9	MiniDIN-8	PC-DB-9
Rx	3	3	2
Tx	2	5	3
Gnd	7	4	5

Signal	LV824
Rx	2
Tx	3
Gnd	5 and 7

Make sure to connect Rx on the LV824 to Tx on the serial cable !

If the LV824 stops responding when you connect to a new SGI workstation, please consider the serial cable as a likely problem. The quickest test of the BG hardware is to reconnect it to the last known working machine. If it works with one computer, but not with another, then the BG hardware is OK, and you should double check the serial cable or move on to the next section.

7.1.3 Host Configuration

There are a number of possible problems that can be attributed to the host computer. The first thing to double check is to make sure that the physical serial port matches the environment variable that you have set. So if you are connected to serial port number 4, you should make sure to:

```
setenv FBPORT /dev/ttyd4
```

If you have selected port 4, and it doesn't exist, you will see a message like this:

```
% ./eprom
Attempting to open the serial port
/dev/ttyd4: No such device or address
        Unable to open port
bye
```

When running the eprom test program, if the program prints a message:

```
% ./eprom
Attempting to open the serial port
/dev/ttyd2: Permission denied
        Unable to open port
bye
```

This means that you do not have the appropriate permission to read and write to the serial port that you have selected. Consult with your system administrator to change the permission.

The next thing to check will require the assistance of the system administrator (or the password to log in as root). Using the graphical system administration tools, bring up the ports and terminals screen. Make sure that the port that you wish to use is not configured and shows up as disconnected. This is to prevent a terminal management program (getty) from changing the baud rate unexpectedly.

7.1.4 Failure to close_lv()

If your application sets the baud rate to anything other than the default of 19200, then you must make sure to call the function close_lv() when your application finishes. Otherwise, when you restart your program the

initial communication will be at the wrong baud rate, and the LV824 will not respond because it has been left at the new baud rate. You can cycle power to the LV824 or push the reset button to return it to 19200 baud.

You can also modify the init_lv() function to run through different baud rates until a response is detected.

7.2 Errors in communication

There are generally going to be two types of communication error that you will come across.

The first occurs when you are trying to sample the LV824 too quickly. Remember, serial communication only allows a finite number of bits per second, so if you read immediately after writing, there is no chance for the data to arrive. (Use the *e_test.c* program to experiment with baud rates, update frequencies, and channel selection to find a suitable update rate.)

Section 4.1 describes how to tune your software to get the maximum update rates when required. If you see a message printed:

```
Frame dropped
```

This means that your software has fallen behind, and that the data you are reading from the serial buffer is one "frame" old. This would lead to lags in software that requires real-time performance. Note that many uses of the LV824 involve sampling human interaction with an input device, such as a joystick, and in these situations, an update rate of 30-50 Hz is usually sufficient.

APPENDIX A THE CHARACTER STRING

This appendix explains the details of the character strings sent between the LV824 and the host computer. Understanding of the format of the strings is not required if you are using the BG library functions.

If you need to rewrite the library functions be advised that BG Systems reserves the right to change the format of the character string in future EPROMs and cannot guarantee compatibility.

A.1 Setup String

This string is sent to the LV824 at startup and contains information about which channels are to be sampled, whether outputs are used, what baud rate to communicate at, and the range and polarity of the analog inputs. These details are contained in *init_lv.c*, which contains quite a bit of "historic" software. This function can be used for EPROMs going back to revision 2.0, and contains pieces of code which are not used for current EPROMs but are needed to maintain backward compatibility. If you compile the version 3.08 library, the *init_lv()* function will send the correct string to a 2.20 EPROM.

This section only describes the current setup string for EPROM 3.08. The first character (or character *c0* in C) sent is a 'c'. The next character, *c1*, contains information abou the baud rate and the first 4 analog input channels. Bits 1 to 4 contain the status of analog inputs 1-4, and bits 5-8 contain the baud rate. Character *c2* contains the status of analog inputs 5-8 in bits 1-4. Bits 5-7 contain the status of the digital inputs i

```
 *  If multiplex digital inputs
selected, overwrite c2 and set c3
c1:
 bits 1-4 for analog inputs (1-4)
 bits 5-8 for baud rate
c2:
 bits 1-4 for analog inputs (5-8)
 bits 5-7 for dig input (unless MP board)
 bits 5-7 for MP input board type
selection
C3:
 bits 1-3 for analog outputs (1-3)
 bits 5-7 for dig output selection
 bit  4 for MP request (3.06 and lower)
 bit  4 for analog output 4 (3.07 and
  higher)
 bits 5-7 for MP input board
selection
c4:
 bits 1-4 for analog outputs (5-8)
 bit  5 for MP request (3.07 and higher)
c5:
 bits 1-4 encoder 1-4 selection on/off
c6:
 bits 1-4 encoder 1-4 selection inc/abs
c7:
 bits 1-4 encoder 5-8 selection on/off
c8:
 bits 1-4 encoder 5-8 selection inc/abs
c9:
 bits 1-4 analog input 1-4 polarity
c10:
 bits 1-4 analog input 1-4 range
c11:
 bits 1-4 analog input 5-8 polarity
c12:
 bits 1-4 analog input 5-8 range
```

Each character is then stored into a string with 0x21 added to ensure that no "flow control" characters are sent by mistake. (Sending the wrong character could instruct the LV824 to stop listening for new commands.)

A.2 Incoming String

The string that comes in to the LV824 from the host starts with either a “B” (no digital outputs selected) or a “p” (digital outputs are in use). This character is checked along with the length of the string to make sure that a full string has been received. If the string length is correct, the string is passed to the *convert()* function.

The characters in the string depend on the setup, so if for example no digital inputs have been selected, then character *c1* would contain data pertaining to the analog inputs.

If digital inputs have been selected, then each 8 digital inputs are stored in pairs of characters. Each character has 0x21 subtracted from it, and then is tested against 0xf to see which bits have been set. The second of the characters is shifted to the left by 4 and contains digital inputs 5-8 (or 13-16, or 21-24).

If multiplexed digital inputs are being used (i.e. with a JunctionBox), then up to fourteen pairs of characters are read and stored in the same manner as above.

Once the digital inputs, if any, have been decoded, we move on to the analog inputs. Each 12 bit analog input is stored in two characters. Each character has 0x21 subtracted. The first character is masked against 0x3f to see which of the six bits are set, and then shifted 6 to the left. The second character contains the lower six bits, which are simply masked against 0x3f. These two bit patterns are then stored into a 16 bit integer of which 12 bits are used. This value is then scaled into a -1.0 to 1.0 range. The raw data values are also stored.

With revision 3.08 EPROMs a second pass is needed for the analog inputs. If the polarity is set to +/-, then the least significant bit sets the sign. Then the mapping to -1.0 to 1.0 uses a different formula depending on whether the value is positive or negative.

BG Systems, Inc. Palo Alto, California, USA

Tel: 650-858-2628

Fax: 650-858-2685

URL: www.bgsystems.com