WinSystems

EPX-GX2
Linux Device Driver for 2.6.x Kernels

Release 1.0 October 27, 2005

1 INTRODUCTION

1.1 This driver has been built and tested on the Linux Kernel 2.6.10 running the SimplyMEPIS 3.3 distribution.

1.2 The driver supports the Linear Technology LTC185X A/D converter chip present on the WinSystems EPX-GX2 board.

1.3 This driver is provided 'as-is' and no warranty as to usability or fitness of purpose is claimed.

1.4 WinSystems does not provide support for the modification of this driver. Bug reports may be sent to linux_drivers@winsystems.com

1.5 This driver is provided under the terms of the GNU General Public License.

2 INSTALLATION

2.1 The driver source code is distributed on an MS-DOS filesystem 1.44MB floppy diskette. The floppy diskette should be mounted and the contents copied to the desired development directory.

2.2 It will be necessary to become the root user to build the driver and the device node.

2.3 The default MAJOR number for this device is 111. It may easily be changed by editing the definition at the beginning of the Makefile. To create the driver, the device node, and the sample programs type :

```
make all
```

2.4 The device driver *gxad.ko*, and the device node *gxad* are created and *chmod* is executed to allow access by all users and groups. These permissions can be changed manually as desired. The device node *gxad* is created in the current directory. A new node can be created manually in */dev* if desired. Four sample programs are also built.

2.5 The driver must be explicitly loaded either through init scripts or manually. In either case *insmod* is used to load the driver. Since the LTC185X chip is not plug-n-play and, I/O probing could be problematic therefore it is required to specify the base address of the LTC chip on the command line when loading the driver. A sample command line loading might look like this :

```
insmod gxad.ko io=0x1e8
```

This would install the driver with a base port of Hex 1E8.

3 DRIVER USAGE

3.1 The LTC185X is accessed in hardware as a byte oriented device. Therefore, the driver is implemented as a character device. Using file I/O i.e. read, write, and seek operations although crudely implemented for compatibility will NOT give the desired results. The driver was designed for maximum flexibility using *ioctl* as its exclusive programming interface.

3.2 The file *gxadio.o* implements the *ioctl* interface and presents the application with a set of standard C functions that may be called directly from the application without any further need for dealing with or understanding of how to access the driver using *ioctl*. An application must merely include *gxad.h* and link to *gxadio.o* to provide this simple interface.

3.3 The LTC185X series converter is very fast with an intrinsic conversion time of about 4uS. When coupled with the state machine hardware, and realistic software, a conversion can be accomplished in under 25uS. Although interrupts are supported in hardware to signal an end of conversion, testing has shown that polled mode is actually faster as interrupt handling adds a significant overhead to the operations. The Linux driver does NOT use interrupts.

4 SAMPLE PROGRAMS

4.1 Also included with the library are 4 sample application programs which utilize the functions in the library. They range in complexity from very simple to much more complex. There is extensive commenting within both the library source file and the sample applications to facilitate understanding of their usage.

4.2 This document will discuss briefly each sample application and then document all of the library functions and exported global variables that may be used by an application.

GETVOLT.EXE

Getvolt.c is the source for sample application number 1. This sample is shown first because it utilizes the highest level function in the library and for a large number of users will be the only function required from the library.

This application is supplied in its source form getvolt.c. It is invoked at the command line as :

```
getvolt x
```

Where x is a value from 0 to 7 indicating the channel number to convert. The voltage on that channel is then displayed. Internally the code calls the high-level function.

```
gxad_auto_get_channel_voltage(channel);
```

This function is an auto ranging function in that it starts out by making a measurement in a ±10V scale, checking the result to see if a more precise value could be obtained by changing scales and if so, making another measurement at the more precise range and returning a floating point voltage to the caller. If absolute speed is not important this is the easiest way to make a reading on a channel.

NOTE : This function was coded for single-ended usage only. Differential inputs would need to set a mode and scale and use the non auto ranging function *gxad_get_channel_voltage*.

GETALL.EXE

The second application uses the *gxad_convert_all_channels* function call to get a snapshot of all of the 8 channels with one call. Unlike *gxad_auto_get_channel_voltage* and *gxad_get_channel_voltage*, the data is returned not in floating point but in an array of raw 16-bit values ranging from 0000H to FFFFH. The program extracts the values one by one from the array, converts them to floating point using global conversion value arrays and then displays the results. Also note that since this is not an auto ranging function it is necessary to call *gxad_set_conversion_mode* for each channel to tell the software the input mode, and range desired. In this sample all channels were set to the same mode but there is no requirement that they all be the same.

REPEAT.EXE

The third application demonstrates the usage of the *gxad_convert_single_repeated* function call. This call is prototyped as:

```
int gxad_convert_single_repeated(int channel, unsigned count, unsigned *buffer);
```

The *channel* number argument is fairly obvious. The *count* value is the number of conversions we want to take on this channel and *buffer* is a pointer to an array large enough to hold the number of samples requested. Upon return the *buffer* array will hold *count* number of conversions which are once again provided in 16-bit values. This sample requests 2000 samples at a time. Once the data is back, it is element by element converted to floating point, displayed and compared against previous minimum and maximum values. Pressing the 'C' key clears the counts and min/max values and pressing 'N' steps to the next channel. Any other key exits.

BUFFERED.EXE

The fourth and final sample uses the gxad_buffered_channel_conversions call to program a series of high-speed conversions with the results being stored in a specified buffer. The function prototype is :

```
gxad_buffered_channel_Conversions(unsigned char *input_channel_buffer, unsigned *buffer);
```

The *input_channel_buffer* is an array of channel numbers built by the user as a to-do list of conversions. It is terminated with a 0FFH value. The *buffer* array must be large enough to hold the requested number of conversions. In our sample we load the *input_channel_buffer* with zero 500 times, one 500 times, two 500 times, and three 500 times for a total of 2000 conversions. Actually our input_channel_buffer is 2001 characters long to make space for the terminating 0ffh character. The output buffer is 2000 unsigned integers long which will hold the results. Upon return from this function we have 500 conversions each on the first 4 channels. The program sorts them out, converts them to voltages and displays the values. Any key press exits the program.

FUNCTION LIST

gxad_auto_get_channel_voltage - Get Channel voltage auto ranging

Prototype : float gxad_auto_get_channel_voltage(int channel)

Arguments : channel - The channel to be converted

Return : Floating point value = to voltage at input channel pin

Description : This function returns the voltage on the current input channel pin. It works for single-ended inputs only. It could make as many as Four conversion requests before returning a final value. This Function is the simplest interface to the hardware.

gxad_get_channel_voltage - Get Channel Voltage

Prototype : float gxad_get_channel_voltage(int channel)

Arguments : channel - The channel to be converted

Return : A Floating point value = to voltage at channel input pin.

Description : This function like *gxad_auto_get_channel_voltage* returns the voltage on the specified channel's input pin. The value returned is only valid for the range specified with a preceding *gxad_set_channel_mode*.

gxad_set_channel_mode - Set Channel input mode and range

Prototype : int gxad_set_channel_mode(int channel, int input_mode, int duplex, int range)

Arguments : channel - The channel number to set (0-7)

: input_mode - Input type
GXAD_SINGLE_ENDED
GXAD_DIFFERENTIAL

: duplex - The swing of the input voltage
GXAD_UNIPOLAR
GXAD_BIPOLAR

: range - The input voltage top end
GXAD_TOP_5V
GXAD_TOP_10V

Return : 1 = An argument error occurred.
0 = Function completed successfully.

Description : This function is used to set the input mode for a given channel. Once a channel's mode has been set it will remain until changed or until the application exits. The mode must be set before making any conversion calls except for *gxad_auto_get_channel_voltage* which will change the mode to the one most appropriate for the current input.

gxad_start_conversion - Start a conversion on a channel

Prototype : int gxad_start_conversion(int channel)

Arguments : channel - The channel number (0-7)

Return : 1 = Bad channel number
0 = Conversion started

Description : This function starts an A/D conversion on the specified channel number and returns immediately.

gxad_wait_ready - Wait for conversion complete

Prototype : int gxad_wait_ready(void)

Arguments : none

Return : 1 = Timeout error occurred
0 = Data ready to be read

Description : This function is used to wait for conversions to complete. It reads the status port until the conversion is complete or a timeout error occurs.

gxad_read_conversion_data - Read data from A/D converter

Prototype : __u16 gxad_read_conversion_data(void)

Arguments : none

Return : 16-bit value

Description : The function reads out the data from the second to last conversion. It is important to recognize that with each conversion the converter delivers the data from the previous conversion meaning that if a current reading is required it's necessary to do two conversions. Look at the source code for the sample programs to see how this is accomplished.

gxad_convert_all_channels - Convert all channels

Prototype : int gxad_convert_all_channels(unsigned *buffer)

Arguments : Pointer to an 8 element unsigned array for return of values.

Return : 0 = all conversiona complete

Description : This function is used to snapshot all 8 channels as quickly as possible. The results are stored in an 8-element array provided by the calling program. The values provided are 16-bits (signed/unsigned) in length for each element.

gxad_convert_single_repeated - Multiple conversions on a single channel

Prototype : int gxad_convert_single_repeated(int channel, unsigned count, unsigned *buffer)

Arguments : channel - The channel number (0-7)

: count - The number of desired conversions.

: buffer - A pointer to an array of count elements to hold the results

Return : 0 = conversions complete

Description : The function allows for repetitive high-speed conversions on a single channel. The array pointer buffer must be of sufficient size to hold the results, i.e. count elements long. The absolute maximum count is 65536. Counts from 2 to 16384 are more realistic. The values are returned in the array in 16-bit integers which are signed or unsigned dependent upon the channel mode used.

gxad_buffered_channel_conversions - Programmable conversion sequence

Prototype : int gxad_buffered_channel_conversions(unsigned char input_channel_buffer, unsigned *buffer)

Arguments : input_channel_buffer - Pointer to an array of channel numbers to Be converted. Terminated with 0ffH.

: buffer - Pointer to an array to receive the results.

Return : 0 = conversions complete.

Description : This function allows for high speed multiple channel conversions. The input is an array of channel numbers in any order repetitive or not as desired. The function will start each conversion immediately after completing the previous one without further application intervention. The list is terminated with a 0ffH value.
The buffer argument should point to an adequately sized array to hold all of the specified conversion results.

GLOBAL VARIABLES

The GXAD support routines export several global variables which are usable by the application program. The most commonly used globals are defined here :

int gxad_error_code

Description : gxad_error_code is set to a non-zero value when an error occurs within the support library. The error codes are defined in the file gxad.h. See gxad_error_string for string error messages.

char gxad_error_string[ ]

This string holds a valid error string whenever gxadio_error_code is non zero. It can be used for displaying consistent error messages from within an application program.

__u16 gxad_adjust[channel]

float gxad_bitval[channel]

float gxad_offset[channel]

These three arrays are used to convert a raw 16-bit value from the converter routines to a floating point voltage. They are initialized with a *gxad_set_channel_mode* call for a channel and make repetitive conversions much simpler with less math involved. The formula for the conversion is as follows :

voltage = ((raw16 + gxad_adjust[channel]) * gxad_bitval[channel]) + gxad_offset[channel]

Refer to the source code for the example programs for illustrated usage of these global variable.