

# WinSystems

## EPX-GX A/D Converter Library Routines

Support under DOS for the LTC185X analog to digital converter present on the EPX-GX board is through a library of 'C' language function calls. An application program can either link to the library, or specific functions can be pulled into the application from the supplied library source file. The library file is named *gxa2d.c*. All library and sample programs were compiled and tested using the Borland C/C++ compiler version 3.1. It should take little or no effort to port these routines to other compilers if required.

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 to signal an end of conversion, testing has shown that polled mode is actually faster as interrupt handling add a significant overhead to the operations. The supplied library routines function in either polled or interrupt mode as desired.

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.

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 a pre-compiled form *getvolt.exe* and in its source form *getvolt.c*. It is invoked at the DOS 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  $\pm 10V$  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 0 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 : 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. If interrupt mode is on, it waits for the ISR to set a flag. If polled mode is being used this routine reads the status port until the conversion is complete.

**gxad\_read\_conversion\_data** - Read data from A/D converter

Prototype : unsigned gxad\_read\_conversion\_data(void)

Arguments : none

Return : 16-bit value (signed/unsigned dependent on channel mode)

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 = conversions 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 but that is, in reality, unobtainable in a real mode small model program. 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.

**gxad\_set\_interrupt\_mode** - Turn on interrupt mode

Prototype : int gxad\_set\_interrupt\_mode(int my\_irq)

Arguments : my\_irq - The IRQ number (0-15) to use (Must match hardware)

Return : 1 - Bad IRQ number  
0 - Interrupt enabled

Description : This function enables the end of conversion interrupt on the LTC185X and installs an ISR routine for handling the interrupts. Note that in most cases the extreme speed of the converter will negate any gains by using interrupts. During testing under DOS polled mode resulted in less CPU overhead and faster conversions.

**gxad\_set\_polled\_mode** - Set polled mode for conversion complete status

Prototype : void gxad\_set\_polled\_mode(void)

Arguments : none

Return : none

Description : This function reverses the action of the *gxad\_set\_interrupt\_mode* call. It restores the interrupt vector, masks the interrupt at the PIC and turns off interrupts at the LTC185X. This function does nothing if called without a previous call to *gxad\_set\_interrupt\_mode*.