PX1-I440-ADC Linux Library Package

1 Introduction

1.1 The PX1-I440-ADC is an industrial data acquisition module for embedded systems with PCIe/104 OneBank expansion. This module features eight differential ADC (analog-to-digital converter) inputs based on the Analog Devices LTC2335-16 ADC.

1.2 The LTC-2335 provides an eight-channel 16-bit Analog-to-Digital (A/D) converters with sample-and-hold-circuit support. Input ranges supported are: 0-5V, 0-5.12V, 0-10V, 0-10.24V, ±5V, ±5.12V, ±10V, and ±10.24V. The data sheet is found at this link: LTC-2335 Data Sheet.

1.3 The library was built and tested on a Linux based Ubuntu 18.04.4 LTS distributions. This release corresponds to Linux kernel version 5.3.0.

1.4 This driver is provided on an 'as-is' basis and no warranty as to usability or fitness of purpose is inferred or claimed.

1.5 WinSystems, Inc. does not provide support for the modification of this driver. Customer application specific queries can be sent to: support@winsystems.com.

1.6 This work is provided under the terms of the GNU General Public License (GPL).

2 Installations and Build

2.1 The library and sample applications are provided in source code form as a compressed tarball archive.

2.2 The build process must run with root permissions in order to copy the shared library into its runtime location (/usr/local/bin).

2.3 The *lsusb* Linux command should return a device with VID:PID = 0328c:0440. The Bus and Device numbers may vary based on the specific port selected.

```
pauld@ubuntu:~/i440/I440LinuxRepo/i440_apps/i440Test$ lsusb
Bus 001 Device 014: ID 328c:0440
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 004: ID 0e0f:0008 VMware, Inc.
Bus 002 Device 003: ID 0e0f:0002 VMware, Inc. Virtual USB Hub
Bus 002 Device 002: ID 0e0f:0003 VMware, Inc. Virtual Mouse
Bus 002 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
```

2.4 To build and install the I440 library, execute the following in the library directory:

sudo make

The library ***libi440.so.1.0*** is created and moved to the appropriate directory. The sample applications are also built.

sudo make libi440 - Builds the library without installing it
sudo make install - Installs the library
sudo make clean - Installs the library

2.5 After installation of the shared library, it may be necessary to update the system's ldconfig configuration file to include the /usr/local/lib directory. The configuration files can be found in the directory ld.so.conf.d located in /etc (/etc/ld.so.conf.d). As root user, edit one of the files located in /etc/ld.so.conf.d and add the line /usr/local/lib to the end of the file, save the file, and exit. Execute the following command:

sudo ldconfig

2.6 Before using the library, it is necessary to unload a pair of kernel modules that are automatically loaded when the I440 card is plugged into a Linux machine. The modules to be removed from the running kernel are *ftdi_sio* and *usbserial*. To unload these modules, as root user, run the shell script *setup.sh* located in the top level I440 directory. This script executes the following commands:

rmmod ftdi_sio
rmmod usbserial

3 Driver Usage

3.1 An application must call the function *InitializeSession* to open the library and initialize the hardware. The following code example opens the WinSystems library. If a zero is returned, then the library has been successfully initialized. Any other returned value indicates that an error has occurred and the device is unusable.

```
if (InitializeSession())
       printf("Error opening device\n");
```

3.2 Once the hardware is initialized, the other ADC functions can be used to control the I440 and retrieve sample data.

3.3 All samples returned are 24-bit values for the specified ADC channel. The first two bytes contain the 16-bit conversion data and the third byte contains the span and channel information of the sample data. The bit fields are defined here:

23 .. 8	7	6	5	4	3	2	1	0
D15 .. D0	0	0	Channel			Span		

3.4 All functions below will follow the same return value model. If a zero is returned, the function was successful. Otherwise there was an error and the function did not complete. Specific error codes are defined later in this document.

3.5 ADC Functions

The I440 uses the Linear Tech LTC-2335 8-channel A/D converter. An 8-bit command is shifted into the ADC interface to configure it for the next conversion. At the same time, the data from the previous conversion is shifted out of device. Consequently, the conversion result is delayed by one conversion from the current command word. All of the functions compensate for this functionality so that the user does not have to adjust for this delay.

3.5.1 int SetChannelMode(unsigned int channel, int duplex, int range)

Configures the allowable voltage range of the selected channel. The duplex variable configures the channel for a unipolar or bipolar conversion. The valid selections are ADC_UNIPOLAR and ADC_BIPOLAR. The range variable determines the input span for the conversion. The valid selections are ADC_TOP_5V, ADC_TOP_5_12V, ADC_TOP_10V, and ADC_TOP_10_24V.

```
// configure channel 4 for ±10V
int Status = SUCCESS;

Status = SetChannelMode(ADC_CHAN4, ADC_BIPOLAR, ADC_TOP_10V);

if (Status)
	printf("Error %d configuring ADC channel %d\n", Status, ADC_CHAN4);
else
	printf("ADC channel %d configured\n", ADC_CHAN4);
```

3.5.2 int SetTimeBase(unsigned int time)

Selects the rate at which conversions occur. The default rate is 1 MHz which is also the maximum rate. The *time* value for this setting is 59. The minimum rate is 915.54 Hz, and the *time* value for this setting is 65534. An enumerated variable (TimeBase) is provided which provides some common sampling rates. The *time* value to enter can be calculated from the following formula:

Time = (60 MHz / Desired Rate) - 1

```
// select 50 kHz as the time base
unsigned int time = TIME_50kHz;
int Status = SUCCESS;

Status = SetTimeBase(time);

if (Status)
	printf("Error %d setting time base.\n", Status);
else
```

```
        printf("Time base set to %d\n", time);
```

3.5.3 int GetChannelData(unsigned int *value, unsigned int channel)

Measures and returns the 24-bit value for the specified ADC channel and stores it in the memory location provided by the parameter *value*.

```
// read ADC value on channel 3
unsigned int adcValue;
int Status = SUCCESS;

Status = GetChannelData(&adcValue, ADC_CHAN3);

if (Status)
       printf("Error %d reading ADC value\n", Status);
else
       printf("ADC channel %d value is %04x\n",
              (adcValue >> 3) & 0x7,
              (adcValue & 0xFFFF));
```

3.5.4 int GetChannelDataCount(unsigned char *valBuf, unsigned int channel, unsigned int *count)

Measures and returns a preconfigured number of 24-bit values for a specific channel. The conversion values are stored in the memory buffer provided by the parameter *valBuf*. The *count* variable specifies the number of conversions to perform.

If the function fails due to a time-out error, the number of successful conversions before the time-out is returned in the count variable.

```
// obtain 100 samples for channel 1
unsigned char *Buffer;
unsigned int cnt = 100;
unsigned int totalBytes = cnt * 3; // 3 bytes per conversion
int Status = SUCCESS;

// allocate memory for samples
Buffer = (unsigned char *)malloc(totalBytes);

if (Buffer == NULL)
{
    printf("Insufficient memory available!\n");
    exit(1);
}
else
       printf("%d bytes allocated for sample storage.\n", totalBytes);

// get samples
Status = GetChannelDataCount(Buffer, ADC_CHAN1, &cnt);

if (Status)
       printf("Error %d on channel %d!\n", Status, ADC_CHAN1);
else
{
```

```
    // display samples, ignore first sample
    for (unsigned int i = 0; i < totalBytes; i += 3)
        printf("ADC channel %d value is %04x\n",
             (*(Buffer + i + 2) >> 3) & 0x7,
             (*(Buffer + i) << 8) | *(Buffer + i + 1));
}
```

3.5.5 int GetAllChannelData(unsigned char *valBuf)

Measures and returns the 24-bit value for all eight channels in sequential order. The conversion values are stored in the memory array provided by the parameter *valBuf*.

```
// get conversion data for all channels
unsigned char convValues[27];  // 9 samples, 3 bytes per sample
int Status = SUCCESS;

Status = GetAllChannelData(convValues);

if (Status)
{
    printf("Error performing conversions.\n");
    exit(Status);
}
else
{
    for (int i = 1; i <= 8; i++)
    {
        printf("Channel %d: Conversion data = 0x%02x%02x\n",
            *(convValues + (i * 3) + 2) >> 3 & 0x7,
            *(convValues + (i * 3)),
            *(convValues + (i * 3) + 1));
    }
}
```

3.5.6 int GetChannelSequence(unsigned int *valBuf, unsigned int *chanSeq)

Measures and returns the 24-bit values for the specified channel sequence and stores the conversion values in the memory space provided by the parameter *valBuf*. A channel sequence consists of an array of up to 16 channels followed by a byte of value 0xFF which indicates the end of the sequence.

```
// convert a sequence of channels
unsigned int chSeq[] = { 1, 3, 5, 7, 0xff };
unsigned char *Buffer;
unsigned int totalBytes = sizeof(chSeq) / sizeof(unsigned int) * 3 - 1;
float convVoltage;

int Status = SUCCESS;// convert a sequence of channels
unsigned int chSeq[] = { 1, 3, 5, 7, 0xff };
unsigned char *Buffer;
unsigned int totalBytes = sizeof(chSeq) / sizeof(unsigned int) * 3 - 1;
float convVoltage;
int Status = SUCCESS;

// allocate memory for samples
```

```
Buffer = (unsigned char *)malloc(totalBytes);

if (Buffer == NULL)
       printf("Insufficient memory available!\n");
else
       printf("%d bytes allocated for sample storage.\n", totalBytes);

// get samples
Status = GetChannelSequence(Buffer, chSeq);

if (Status)
       printf("\nError %d executing sequence!\n", Status);
else
{
    // display samples
    for (unsigned int i = 3; i < totalBytes; i += 3)
    {
        printf("Sample %5d - Channel %d - Value = %04x\n",
            i / 3,
            (*(Buffer + i + 2) >> 3) & 0x7,
            (*(Buffer + i) << 8) | (*(Buffer + i + 1)));
    }
}
```

3.5.7 GetChannelSequenceCount(unsigned char *valBuf, unsigned int *chanSeq, unsigned int *count)

Measures and returns a preconfigured number of 24-bit values for the specified channel sequence and stores the conversion values in the memory space provided by the parameter *valBuf*. A channel sequence consists of an array of up to 16 channels followed by a byte of value 0xFF which indicates the end of the sequence. The *count* variable sets the total number of conversions required. To convert a sequence of four channels five times, the count value should equal 20.

```
// obtain 50 samples for defined sequence
unsigned int chSeq[] = { 0, 2, 4, 6, 0xff };
unsigned char *Buffer;
unsigned int cnt = 50;
unsigned int totalBytes = cnt * 3; // 3 bytes per conversion
int Status = SUCCESS;

// allocate memory for samples
Buffer = (unsigned char *)malloc(totalBytes);

if (Buffer == NULL)
    printf("Insufficient memory available!\n");
else
    printf("%d bytes allocated for sample storage.\n", totalBytes);

// get samples
Status = GetChannelSequenceCount(Buffer, chSeq, &cnt);

if (Status != SUCCESS)
{
```

```
        printf("\nError %d executing sequence!\n", Status);
    }
    else
    {
        // display samples, ignore first sample
        for (unsigned int i = 3; i < totalBytes; i += 3)
            printf("ADC channel %d value is %04x\n",
                (*(Buffer + i + 2) >> 3) & 0x7,
                (*(Buffer + i) << 8) | *(Buffer + i + 1));
    }
```

3.6 Generic Functions

3.6.1 ConvertToVolts(float *voltage, unsigned int channel, int value)

This function converts a 16-bit measured value to a voltage. The conversion uses the specific channel configuration settings. The floating-point value is returned in the variable *voltage*.

```
// convert to volts
unsigned char convValues[27];
float convVoltage;
int Status = SUCCESS;

Status = ConvertToVolts(&convVoltage,
       *(Buffer + (i * 3) + 2) >> 3 & 0x7,
       (*(Buffer + i) << 8) | *(Buffer + i + 1));

printf("  Voltage = %.3f\n\n", convVoltage);
```

3.6.2 int FlushRxQueue (void)

This function empties the receive queue on the FPGA. This can be used if an error occurs to ensure the queue does not contain any sample remnants.

```
int Status = SUCCESS;

Status = FlushRxQueue();

if (Status != SUCCESS)
       printf("Error flushing queue with code %d\n", Status);
```

3.6.3 int CloseSession(unsigned int device)

This function is used to disable the I440 device and close the driver when complete. If a zero is returned, the device is shut down. Otherwise there was an error and the driver is still open.

```
if (CloseSession())
       printf("Error closing driver\n");
```

3.7 Return Values

Every function returns a zero or a positive integer value indicating success or failure. If a zero is returned, the function has completed successfully. If a failure occurs, the specific value returned provides more clarity as to the failure mechanism.

3.7.1 DRIVER_ERROR (1)

This error indicates that some function within the driver has failed. Using Windows Device Manager, verify that the drivers are loaded and have no resource conflicts.

3.7.2 INVALID_HANDLE (2)

This error indicates that the driver has not initialized or closed. The driver attempts to open an existing handle to the I440 device, and this error indicates that the handle does not exist. Verify that the drivers have loaded successfully.

3.7.3 INVALID_PARAMETER (3)

This error indicates that one of the parameters in a DLL function is out of bounds.

3.7.4 INVALID_SEQUENCE (4)

This error indicates that the sequence provided to the function is invalid. A channel sequence consists of an array of up to 16 channels followed by a byte of value 0xFF which indicates the end of the sequence. Verify that the sequence defined follows these rules.

3.7.5 TIMEOUT_ERROR (5)

This error indicates that the driver has exceeded the required time for a response from the I440 device. All active sample requests are terminated. For some functions the number of valid samples is returned in the count variable.

4 Sample Applications

4.1 GetSamples

The *GetSamples* sample application configures the selected channel for ±10V bi-polar and reads the voltage on that channel using the GetChannelDataCount function. The channel number, number of samples, and time base are provided as command line arguments. If any error occurs during execution, the error is reported and the application is terminated.

4.2 GetSequence

The *GetSequence* sample application configures all eight channels for ±10.24V bi-polar and reads the voltage on a pre-configured sequence of channels using the GetChannelSequence function. The time base is provided as a command line argument. If any error occurs during execution, the error is reported and the application is terminated.

4.3 GetSequenceCount

The *GetSequenceCount* sample application configures all eight channels for ±10.24V bi-polar and reads the voltage on a pre-configured sequence of channels using the GetChannelSequenceCount function. The number of samples and time base are provided as command line arguments. If any error occurs during execution, the error is reported and the application is terminated.