PX1-I440-ADC Windows DLL 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 I440 DLL Package is designed for and has been verified with 32-bit and 64-bit versions of Microsoft WES 7 and Windows 10.

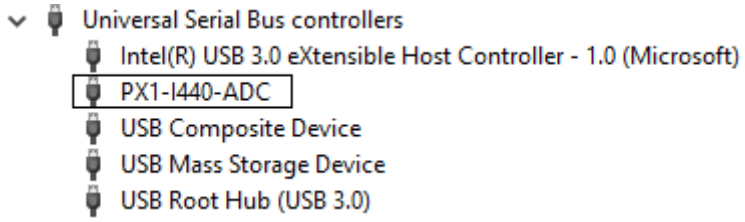
2 Installation

2.1 The driver, support files, and console applications are supplied in the WinSys_Win10_64_i440_0_1.zip file. The following files are included:

- i440DLL.dll – Windows DLL provided by WinSystems
- i440DLL.h – driver include file
- ftd2xx.dll – Windows DLL provided by FTDI
- i440_Setup.exe – FTDI D2XX Driver self-extracting tool
- i440GetSamples.cpp – Windows console application source code
- i440GetSamples.exe – Windows console application
- i440GetSequence.cpp – Windows console application source code
- i440GetSequence.exe – Windows console application
- i440GetSequenceCount.cpp – Windows console application source code
- i440GetSequenceCount.exe – Windows console application
- VC_redist.x86.exe or VC_redist.x64.exe - Microsoft Visual C++ Redistributable

2.2 Before the I440 DLL is usable, the FTDI D2XX driver (*ftdibus.sys*) must be installed on the SBC. The current driver version is 2.12.28. Execute the i440_Setup self-extracting utility file as administrator.

2.3 When the driver is installed, a new PX1-I440-ADC device should appear in the Windows Device Manager under the Universal Serial Bus controllers list.



2.4 The included console applications can be used to verify driver installation and functionality. Usage of the programs is described later in this document.

3 Driver Overview and Architecture

3.1 The file *i440DLL.dll* is a Windows Dynamic-Link Library (DLL) which facilitates access to the underlying hardware through the FTDI D2XX driver.

3.2 The *i440DLL.h* file contains the function definitions to be used by an application to communicate with the D2XX driver. It also provides enumerated data types that can be used as function arguments. See Appendix A: i440DLL.h.

4 Driver Usage

4.1 An application calls the function *InitializeSession* to open the D2XX driver and initialize the hardware. This step is required before any of the other functions can be called. The following code example opens the WinSystems *i440DLL*. If a zero is returned, then the driver 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");
```

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

4.3 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		

4.4 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.

4.5 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 defined compensate for this functionality so that the user does not have to adjust for this delay.

4.5.1 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 dllReturn = SUCCESS;

dllReturn = SetChannelMode(ADC_CHAN4, ADC_BIPOLAR, ADC_TOP_10V);

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

4.5.2 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 dllReturn = SUCCESS;

dllReturn = SetTimeBase(time);

if (dllReturn)
	printf("Error %d setting time base.\n", dllReturn);
else
	printf("Time base set to %d\n", time);
```

4.5.3 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 dllReturn = SUCCESS;

dllReturn = GetChannelData(&adcValue, ADC_CHAN3);

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

4.5.4 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 dllReturn = 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
dllReturn = GetChannelDataCount(Buffer, ADC_CHAN1, &cnt);

if (dllReturn)
	printf("Error %d on channel %d!\n", dllReturn, 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));
}
```

4.5.5 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 dllReturn = SUCCESS;

dllReturn = GetAllChannelData(convValues);

if (dllReturn)
{
    printf("Error performing conversions.\n");
    exit(dllReturn);
}
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));
    }
}
```

4.5.6 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 dllReturn = 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 dllReturn = 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
```

```
dllReturn = GetChannelSequence(Buffer, chSeq);

if (dllReturn)
       printf("\nError %d executing sequence!\n", dllReturn);
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)));
    }
}
```

4.5.7 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 dllReturn = 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
dllReturn = GetChannelSequenceCount(Buffer, chSeq, &cnt);

if (dllReturn != SUCCESS)
{
    printf("\nError %d executing sequence!\n", dllReturn);
}
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));
}
```

4.6 Generic Functions

4.6.1 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 dllReturn = SUCCESS;

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

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

4.6.2 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 dllReturn = SUCCESS;

dllReturn = FlushRxQueue();

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

4.6.3 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");
```

4.7 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.

4.7.1 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.

4.7.2 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.

4.7.3 INVALID_PARAMETER (3)

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

4.7.4 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.

4.7.5 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.

5 Sample Applications

The driver package provides sample Windows console applications for most of the functions provided in the driver package. The source code and an executable file is provided for each application.

5.1 ADC Applications

5.2.1 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.

5.2.2 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.

5.2.3 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.

Appendix A: i440DLL.h

```
#ifndef _I440_DLL_H_
 #define _I440_DLL_H_

#define DECLDIR __declspec(dllexport)

extern "C"
{
    DECLDIR int InitializeSession(void);
    DECLDIR int CloseSession(void);
       DECLDIR int GetSoftwareVersion(unsigned int *pDriverVersion, unsigned int
*pLibVersion);
    DECLDIR int SetChannelMode(unsigned int channel, int duplex, int range);
    DECLDIR int SetTimeBase(unsigned int time);
    DECLDIR int GetChannelData(unsigned int *pValue, unsigned int channel);
    DECLDIR int GetChannelDataCount(unsigned char *pValue, unsigned int channel, unsigned
int *pCount);
    DECLDIR int GetAllChannelData(unsigned char *pValBuf);
    DECLDIR int GetChannelSequence(unsigned char *pValBuf, unsigned int *pChanSeq);
    DECLDIR int GetChannelSequenceCount(unsigned char *pValBuf, unsigned int *pChanSeq,
unsigned int *pCount);
    DECLDIR int ConvertToVolts(float *pVoltage, unsigned int channel, int value);
    DECLDIR int WriteCommands(unsigned char *pValBuf, unsigned char *pCmd, unsigned int
cnt);
    DECLDIR int StartSequence(void);
    DECLDIR int StopSequence(void);
    DECLDIR int FlushRxQueue(void);
}

enum ErrorCodes {
    SUCCESS = 0,
    DRIVER_ERROR = 1,
    INVALID_HANDLE = 2,
    INVALID_PARAMETER = 3,
    INVALID_SEQUENCE = 4,
    TIMEOUT_ERROR = 5
};

enum AdcChannel {
       ADC_CHAN0 = 0,
       ADC_CHAN1 = 1,
       ADC_CHAN2 = 2,
       ADC_CHAN3 = 3,
       ADC_CHAN4 = 4,
       ADC_CHAN5 = 5,
       ADC_CHAN6 = 6,
       ADC_CHAN7 = 7,
       ADC_NO_CHANNEL = 8
};

enum AdcDuplex {
    ADC_UNIPOLAR = 0,
    ADC_BIPOLAR = 2
};

enum AdcRange {
    ADC_TOP_5V = 0,
```

```
        ADC_TOP_5_12V = 1,
        ADC_TOP_10V = 4,
        ADC_TOP_10_24V = 5
};

enum TimeBase {
    TIME_1MHz = 59,
    TIME_500kHz = 119,
    TIME_100kHz = 599,
    TIME_50kHz = 1199,
    TIME_10kHz = 5999,
    TIME_5kHz = 11999,
    TIME_1kHz = 59999,
    TIME_LIMIT = 65533
};

#endif  // _I440_DLL_H_
```