


# USBee AX-Pro Data Extractor Users Manual

**Parallel**
**Serial**
**USB**
**SPI**
**CAN**
**I2C**
**I2S**
**SMBus**
**1-Wire**
**Async**

**Version 1.0**

# USBee AX-Pro Data Extractor Users Manual

**CWAV**
**www.usbee.com**
**(951) 693-3065**
**support@usbee.com**

**USBee AX Data Extractor License Agreement**

The following License Agreement is a legal agreement between you (either an individual or entity), the end user, and CWAV. You have received the USBee Data Extractor Package, which consists of the USBee Data Extractor Software and Documentation. If you do not agree to the terms of the agreement, return the unopened USBee Pod and the accompanying items to CWAV for a full refund. Contact support@usbee.com for the return address.

***By installing and using the USBee Pod and Data Extractors, you agree to be bound by the terms of this Agreement.***

**Grant of License**
CWAV provides Software, both in the USBee Data Extractor Package and on-line at www.usbee.com, for use with the USBee Pod and grants you license to use this Software under the following conditions: a) You may use the USBee Software only in conjunction with the USBee Pod, or in demonstration mode with no USBee Pod connected, b) You may not use this Software in conjunction with any pod providing similar functionality made by other than CWAV, and c) You may not sell, rent, transfer or lease the Software to another party.

**Copyright**
No part of the USBee Package (including but not limited to manuals, accompanying software or source code) may be reproduced, stored in a retrieval system, or transcribed, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior written permission of CWAV, with the sole exception of making backup copies of the diskettes for restoration purposes. You may not reverse engineer, decompile, disassemble, merge or alter the USBee Software or USBee Pods in any way.

**Limited Warranty**
The USBee Package and related contents are provided “as is” without warranty of any kind, either expressed or implied, including but not limited to the implied warranties of merchantability and fitness for a particular purpose, with the sole exception of manufacturing failures in the Package Manual or CD. CWAV warrants the Data Extractor physical diskettes to be free from defects in materials and workmanship for a period of 12 (twelve) months from the purchase date. If during this period a defect in the above should occur, the defective item may be returned to the place of purchase for a replacement. After this period a nominal fee will be charged for replacement parts. You may, however, return the entire Data Extractor Package (as long as you include the associated USBee AX-Pro Pod) within 30 days from the date of purchase for any reason for a full refund as long as the contents are in the same condition as when shipped to you. **In order to return the Data Extractor Software you must return the USBee AX-Pro Pod as well.** Damaged or incomplete USBee Packages will not be refunded.

The information in the Software and Documentation is subject to change without notice and, except for the warranty, does not represent a commitment on the part of CWAV. CWAV cannot be held liable for any mistakes in these items and reserves the right to make changes to the product in order to make improvements at any time.

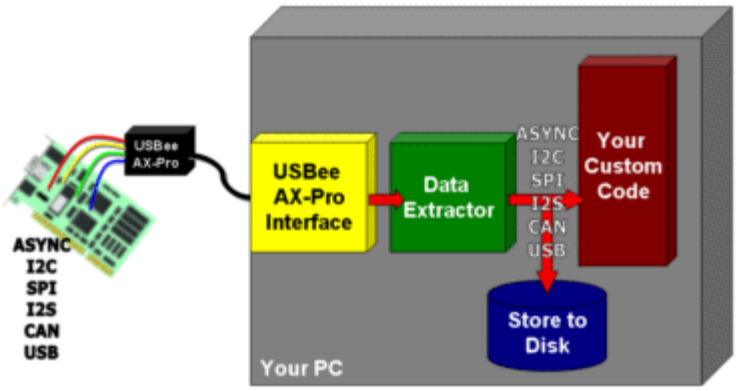
IN NO EVENT WILL CWAV BE LIABLE TO YOU FOR DAMAGES, DIRECT, INDIRECT, INCIDENTAL OR CONSEQUENTIAL, INCLUDING DAMAGES FOR ANY LOST PROFITS, LOST SAVINGS OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES, ARISING OUT OF THE USE OR INABILITY TO USE SUCH USBEE PODS, SOFTWARE AND DOCUMENTATION, EVEN IF CWAV HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES OR FOR ANY CLAIM BY ANY OTHER PARTY. SOME STATES DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY FOR CONSEQUENTIAL OR INCIDENTAL DAMAGES, SO THE ABOVE LIMITATION MAY NOT APPLY TO YOU. IN NO EVENT WILL CWAV’S LIABILITY FOR DAMAGES TO YOU OR ANY OTHER PERSON EVER EXCEED THE AMOUNT OF THE PURCHASE PRICE PAID BY YOU TO CWAV TO ACQUIRE THE USBEE AND SOFTWARE, REGARDLESS OF THE FORM OF THE CLAIM.

**Term**
This license agreement is effective until terminated. You may terminate it at any time by returning the USBee Package (together with the USBee Pod, Software and Documentation) to CWAV. It will also terminate upon conditions set forth elsewhere in this agreement or if you fail to comply with any term or condition of this agreement. You agree that upon such termination you will return the USBee Package, together with the USBee Pod, Software and Documentation, to CWAV.

# Table of Contents

# 1 Overview

The Data Extractors are an option software product for use with the USBee AX-Pro Test Pod that allows engineers to extract the raw data from various embedded busses to store off to disk or stream to another application. The Data Extractors will collect the raw data from Parallel, Serial, I2C, I2S, Async, USB Full and Low Speed, SMBus, 1-Wire or CAN busses and store the data to disk or pass it to your own processing application in real-time.




## *1.1 Data Extractor Features*

- Uses the USBee AX-Pro pod to stream data from your embedded design into your PC
- Captures continuous real-time bus data
- Extracts the transaction data on the fly
- Stores data to disk or process it in real-time
- Runs indefinitely
- Captures entire test sequences
- Monitors embedded system data flows during normal operation
- Processes or stores Megabytes, Gigabytes or Terabytes of data
- Runs as a Windows Command Line executable from the Command Prompt and can be executed from Batch files containing the desired parameters
- Special Viewer to view and search through the extracted data files quickly
- Lets you write your own software to further process the extracted data using the Extractor API libraries.

## *1.2 Bus Types Decoded*

- **Parallel** (internal or external clocking up to 12MHz)
- **Serial** (internal or external clocking up to 12MHz)
- **Async** (up to 12Mbaud)
- **I2C** (SCL up to 4MHz)
- **SPI** (SPI Clock up to 12MHz)
- **1-Wire** (Standard 1-Wire bit rates)
- **I2S** (bit clock up to 12MHz)
- **USB** (Low 1.5Mbps and Full Speed 12Mbps USB)
- **CAN** (up to 12Mbps)
- **SM Bus** (SM Clock up to 12MHz)

## *1.3 Your Testing System*

The typical challenge in embedded streaming bus systems is to get the data out of your embedded system quickly and easily so that you can process it, either to capture a bug in progress or to evaluate performance. In any case, this can be done with the USBee AX-Pro Data Extractor System.

The USBee AX-Pro pod is used to stream raw sample data from its 8 digital input lines directly into the PC. The Data Extractor software modules then take that streaming data and extract your desired data out of the raw stream using the extractor processing threads. Our sample command line application, as well as any custom application you write, interfaces to the extractor through a simple Windows DLL consisting of five function calls. These calls are used to start and extraction, stop an extraction, gather the data (and how much data) and check for error status. Below is a figure of the system and how the pieces fit together.

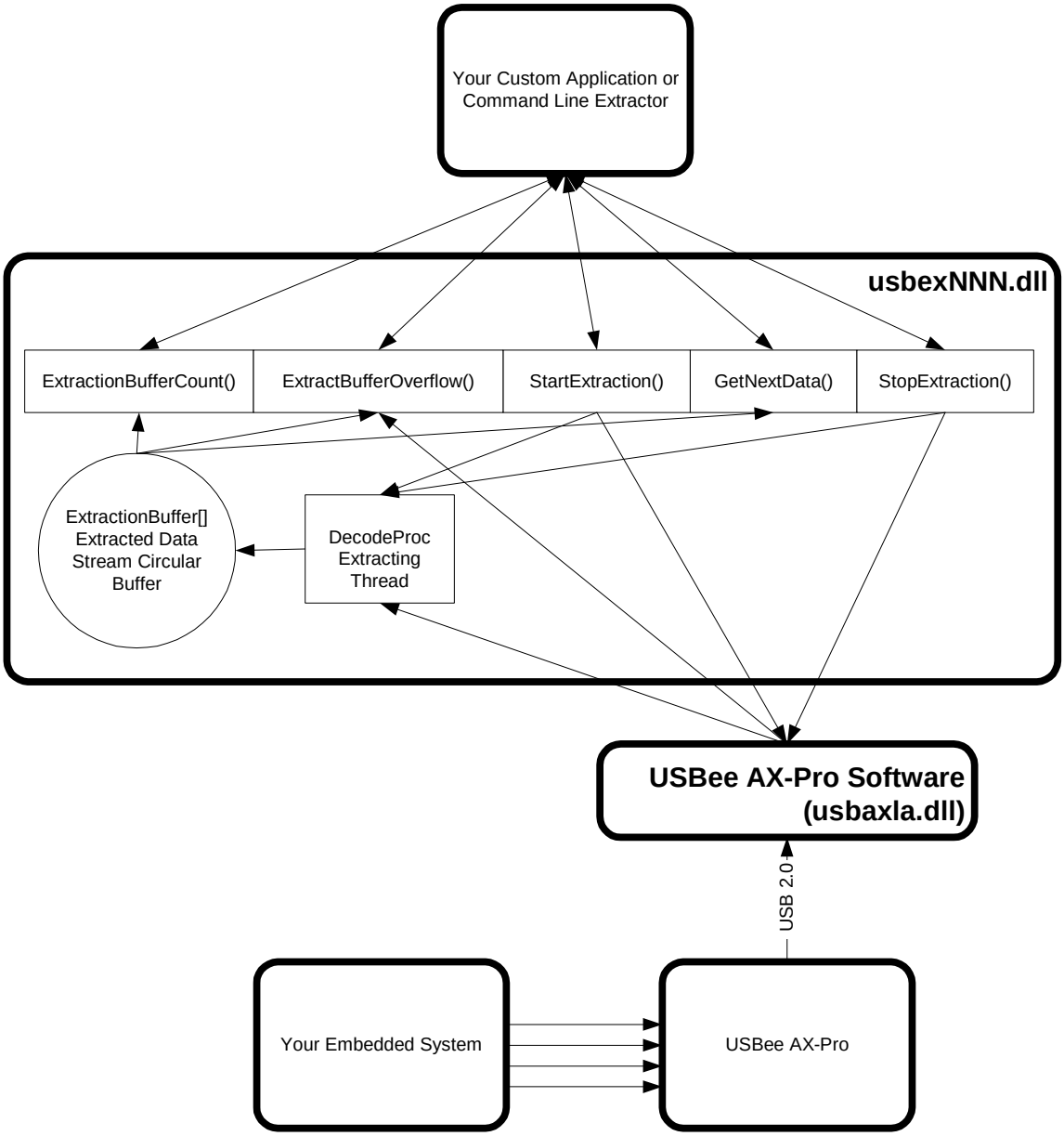



**Figure 1. Data Extractor System Architecture**

## *1.4 System Requirements*

The USBee AX-Pro Data Extractors require the following PC configuration:

- Windows® XP or Windows® 2000 operating system
- Pentium or higher processor
- One USB2.0 High Speed enabled port. It will not run on USB 1.1 Full Speed ports.
- 32MBytes of RAM
- 125MBytes of Hard disk space

It is HIGHLY recommended that the USBee AX-Pro and Data Extractors be run together on a separate PC than the PC controlling the system under test. If your PC is also controlling the system under test you may not be able to get the maximum sample rates needed for some of the extractors.

After installing the software as below, you can determine the maximum sample rate your system can achieve by plugging in the USBee AX-Pro, run the Logic Analyzer Application and choosing the Setup, Sample Rate Test menu option. The sample rate test may take up to 20 seconds. Once the sample rate test is complete, the Sample Rate drop down box will be filled with the available sample rates for you machine. The highest sample rate is what your PC can achieve.

To get the highest sample rates, you will want to use a Desktop PC with native USB 2.0 ports on the motherboard. Some modern Laptops can achieve the maximum of 24Msps, but you will want to disable all power saving features and run your laptop from the power supply, not the batteries.

## *1.5 System Setup*

To configure a system to run these extractors you need the following:

1 USBee AX-Pro Software Installed (follow instructions on the CD)
2 USBee Data Extractors Software Installed (follow instructions on the CD)
3 License File (license.txt) in the working directory of the Data Extractor executables.
4 V File Viewer
5 USBee AX-Pro Pod plugged into a USB 2.0 port on your PC.

### 1.5.1 Installing The USBee AX-Pro CD

Do not plug in the USBee AX-Pro until after you install the USBee AX-Pro CD. Place the USBee AX-Pro CD in the drive and run the setup.exe. This will install all of the drivers and application programs in the proper directories. Choose the default settings for all intallation screens.

### 1.5.2 Installing The USBee AX-Pro Data Extractor CD

Place the USBee AX-Pro Data Extractor CD in the drive and run the setup.exe. This will install all of the drivers and application programs in the proper directories. Choose the default settings for all intallation screens.

### 1.5.3 Installing your License File

The License.txt file must contain a line of text for each extractor that you want to run. These License lines are sent to you in the order shipment as well as emailed to you in your Order Confirmation email. If you are missing your license information, please contact us at support@usbee.com. To create your license file, open the License.txt file (Start|ProgramFiles|USBee AX-Pro Data Extractors|License File" and type in the text lines exactly as shown in your license. You can also cut and paste these lines from your order confirmation email. Then choose "File, Save" in the Notepad menu and save the file as "license.txt" in the directory of the executables (\Program Files\USBee AX-Pro Data Extractors by default). The extractor executables will look for this license file each time it is run.

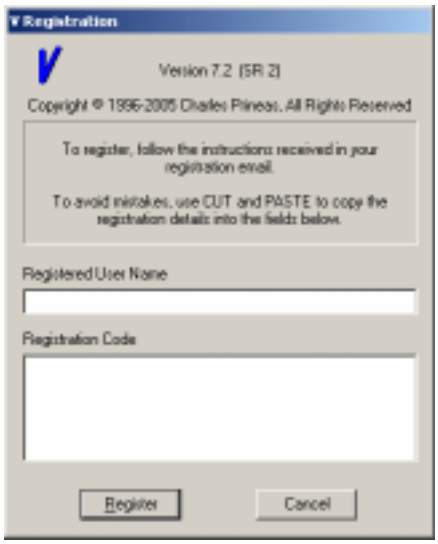
## 1.5.4 Installing the V File Viewer

The files that are created by the Data Extractor can be very large and require a special file viewer that can handle enormous files quickly and easily, both in ASCII text and binary Hexadecimal formats. With the Data Extractor comes a license for the V File Viewer which efficiently views huge data files and allows for quick searching through your data to find the events you are looking for..

To install the V File Viewer, you can either run the v72.exe file from the Data Extractor CD or you can download it. To download the V File Viewer, go to http://www.fileviewer.com/Download.html and download the v72.exe file. This is a self-installing program that installs the V File Viewer.

For help on using the V File Viewer, please refer to the Help included with the viewer.

Once you run the V File Viewer for the first time, you will need to register the product using the registration code that we provide in your order confirmation email. Go to the Help Menu, click Register and enter the 3 text lines into the Registration Code field. The Registered User Name should be left blank.



## 1.5.5 Running the Command Line Extractors

Once these components are installed correctly you can run the Extractor command prompt application .exe files. Each of the executables requires a series of command line parameters that tell the extractor how to process the bus data.

To run the programs, you can do one of two options:

1) Open a Windows Command Prompt Window, change directory (cd) to your \ProgramFiles\USBeeAXProDataExtractors, and enter the command line including all desired parameters.

or

2) Edit the batch files (goUSB.bat, goI2C.bat. etc.) to include the parameters you desire. You can then simply click on the Start Menu items ("Run I2C Batch File etc.) or double click on the batch files themselves in the Windows Explorer.

For all of the extractors you will need to use the USBee Pod ID on your Pod (on the back of the unit) as a command line parameter.

## 1.5.6 Building Your Own Programs Using the API

You can also start to build your own processing programs using the source code for the command prompt applications as a reference point. Each Extractor has a sample project (Visual Studio C++ 6.0) in the \Program Files\USBee AX-Pro Data Extractors directory for you to start with.

In order for your programs to run, you must have a license.txt file in the working directory of your application, and have installed both the USBee AX-Pro CD and the Data Extractors CD on that same machine.

**WARNING**

As with all electronic equipment where you are working with live voltages, it is possible to hurt yourself or damage equipment if not used properly. Although we have designed the USBee AX pod for normal operating conditions, you can cause serious harm to humans and equipment by using the pod in conditions for which it is not specified.

Specifically:

- ALWAYS connect at least one GND line to your circuits ground
- NEVER connect the digital signal lines (0 thru 7, TRG and CLK) to any voltage other than between 0 to 5 Volts
- NEVER connect the analog signal lines (CH1 and CH2) to any voltage other than between -10 and +10 Volts
- The USBee AX actively drives Pod signals 0 through 7 in some applications. Make sure that these pod test leads are either unconnected or connected to signals that are not also driving. Connecting these signals to other active signals can cause damage to you, your circuit under test or the USBee AX test pod, for which CWAV is not responsible.
- Plug in the USBee AX Pod into a powered PC BEFORE connecting the leads to your design.

# 2 Async Data Extractor

The Async Bus Data Extractor takes the real-time streaming data from up to 8 embedded asynchronous buses (UART), formats it and allows you to save the data to disk or process it as it arrives.
The USBee AX Streaming Data Extractors are optional software modules for use with the USBee AX-Pro Test Pod (required) which must be purchased separately.

## *2.1 Async Bus Data Extractor Specifications*

- Continuous Real-Time Data Streaming
- 8 digital channels
- TTL Level inputs (**0-5V max**, Vih = 2.0V, Vil = 0.8V)
- Baud Rates from 1200 baud to 12 Mbaud *
- Data Bit Settings (5, 6, 7 or 8)
- Parity Bit Settings (Mark, Space, Odd, Even, Ignore, None)
- Time Stamps of start of bytes or packets
- Output to Text File (Hex, Decimal, Binary or ASCII)*
- Output to Screen*
- Comma, Space, or Newline Delimited files
- Output File Viewer (including binary, text, search and export functions)
- Extractor API libraries interface directly to your own software to further process the extracted data. Any language that supports calls to DLLs is supported.

* - output bandwidths are dependent on PC USB hardware, hard disk and/or screen throughput.

## *2.2 Hardware Setup*

To use the Data Extractor you need to connect the USBee AX-Pro Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

**Please note that the USBee AX-Pro Test Pod inputs are strictly 0-5V levels. Any voltage outside this range on the signals will damage the pod and may damage your hardware. If your system uses different voltage levels, you must buffer the signals externally to the USBee AX-Pro Test Pod before connecting the signals to the unit.**

The Async Bus Data Extractor uses any of the 8 signal lines (0 thru 7) and the GND (ground) line. Connect any of the 8 signal lines to an Async data bus. Connect the GND line to the digital ground of your system.

## *2.3 Extractor Command Line Program*

The Async Bus Data Extractor includes a Windows Command Prompt executable that lets you operate the Data Extractor without writing any software. The program is executed in a Command Prompt window and is configured using command line arguments. The extracted data is then stored to disk or outputted to the screen depending on these parameters.

To run the Data Extractor:

1) Install the USBee AX-Pro software on your PC
2) Install the Data Extractor software on your PC
3) Plug in your USBee AX-Pro Test Pod into your PC using a USB 2.0 High Speed Port
4) Open a Windows Command Prompt window by clicking Start, All Programs, Accessories, Command Prompt.
5) Change the working directory to the Data Extractor directory
6) (“cd \program files\USBee Data Extractor\Async”)
7) Run the executable using the following command line arguments:

```
AsyncExtractor [-?SADHBICGNX] [-R BaudRate] [-E DataBits] [-L Parity] [-M SignalMask] [-Q
NumberOfBytes] [-T BytesPesLine] [-V Timestamp] [-O filename] -P PodID
```

? - Display this help screen

USBee AX-Pro Pod to Use

P - Pod ID (required)

Output Location Flags

O - Output to filename (default off)
S - Output to the screen (default off)

When to Quit Flags

Q - Number of output values (default = until keypress)

Input Format Flags

R - Baud Rate (9600 baud default)
E - Number of Data Bits (5,6,7,8-default)
L - Parity Type (0=none(default), 1=mark, 2=space, 3=even, 4=odd)
M - Which Signals to capture (1=signal0, 128=signal7, 255=all, 0=none (default))

Output Number Format Flags

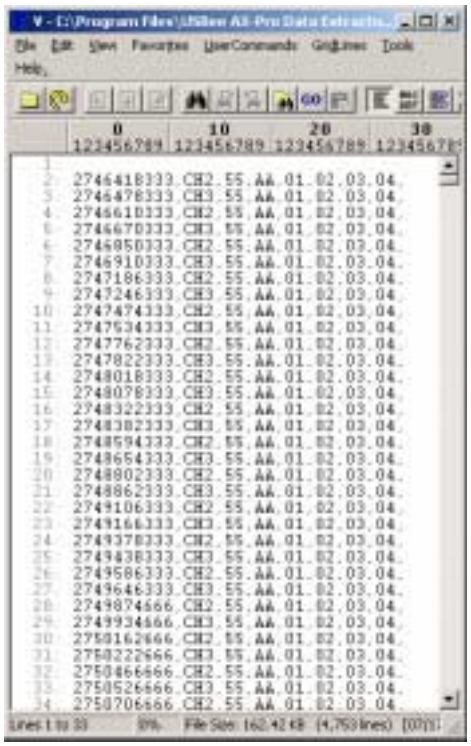
A - ASCII Text Values ("1")
D - Decimal Text Values ("49")
H - Hex Text Values ("31") default
B - Binary Text Values ("00110001")
I - Binary Values (49)
C - Comma Delimited
G - Space Delimited (default)
N - Newline Delimited
X - No Delimeter
T - Force Bytes Per Line (no force default)

Timestamp

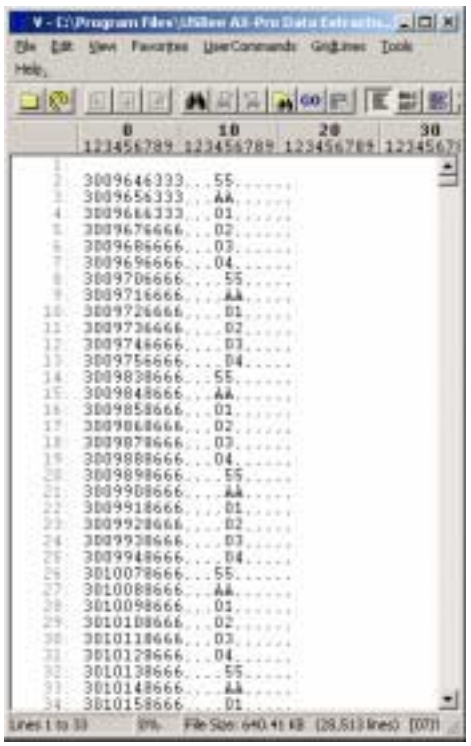
V - Timestamps (0=off, 1=each byte, 2=each channel start)

## 2.3.1 Example Output Files

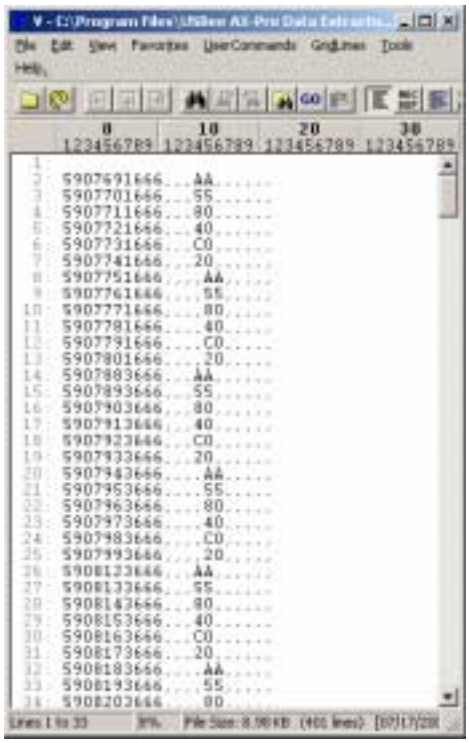
AsyncExtractor -O output.dex -P 3209 -C -Q 100000 -R 1000000 -E 8 -L 0 -M 255 -H -V 2



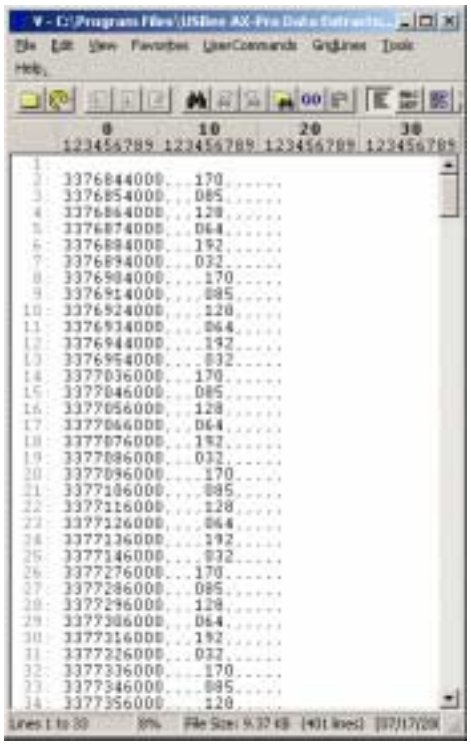

AsyncExtractor -O output.dex -P 3209 -C -Q 100000 -R 1000000 -E 8 -L 0 -M 255 -H -V 1



AsyncExtractor -S -O output.dex -P 3209 -C -Q 400 -R 1000000 -E 8 -L 0 -M 255 -Z -H -V 1



AsyncExtractor -S -O output.dex -P 3209 -C -Q 400 -R 1000000 -E 8 -L 0 -M 255 -Z -D -V 1



AsyncExtractor -S -O output.dex -P 3209 -Q 400 -R 1000000 -E 8 -L 0 -M 255 -Z -H -G -V 3

## *2.4 Extractor API*

The Data Extractor is implemented using a Windows DLL that interfaces to the existing USBee AX-Pro DLL and drivers. This DLL can be called using any software language that supports calls to DLLs. Below are the details of this DLL interface and the routines that are available for your use.

### 2.4.1 DLL filename:

`usbexAsync.dll in \Windows\System32`

### 2.4.2 DLL Exported Functions and parameters

**ExtractionBufferCount** – Returns the number of bytes that have been extracted from the data stream so far and are available to read using GetNextData.

```
CWAV_EXPORT unsigned long CWAV_API ExtractionBufferCount(void)
```
Returns:

0 – No data to read yet
other – number of bytes available to read

**GetNextData** – Copies the extracted data from the extractor into your working buffer

```
CWAV_EXPORT char CWAV_API GetNextData(unsigned char *buffer, unsigned long length);
```

buffer:
pointer to where you want the extracted data to be placed
length:
number of bytes you want to read from the extraction DLL
Returns:

0 – No data to read yet
1 – Data was copied into the buffer

**StartExtraction** – Starts the Data Extraction with the given parameters.

```
CWAV_EXPORT int CWAV_API StartExtraction(unsigned long PodNumber, unsigned long BaudRate,
unsigned int DataBits, unsigned int Parity, unsigned char Channels, unsigned char
MSFirst, unsigned char StopBits)
```

PodNumber:
Pod ID on the back of the USBee AX-Pro Test Pod
BaudRate:
Baud rate of the async channels. All channels are decoded at the same rate.
Data Bits:
Number of Data bits (5, 6, 7 or 8)
Parity:
0 = No parity bit
1 = Mark Parity
2 = Space Parity
3 = Even Parity
4 = Odd Parity
MSFirst:
0 = Least Significant Bit first
1 = Most Significant Bit first
Channels:
Bit mask for which channels to decode (1 = signal 0, 128 = signal 7)
StopBits:
2 = 1 Stop Bit time
3 = 1.5 Stop Bit times
4 = 2 Stop Bit times

Returns:
1 – if Start was successful
0 – if Pod failed initialization

**StopExtraction** – Stops the extraction in progress

```
CWAV_EXPORT int CWAV_API StopExtraction( void );
```

Returns:
1 – always

**ExtractBufferOverflow** – Returns the state of the overflow conditions

```
CWAV_EXPORT char CWAV_API ExtractBufferOverflow(void);
```

Return:
0 – No overflow
1 – Overflow Occurred. ExtractorBuffer Overflow condition cleared.
2 – Overflow Occurred. Raw Stream Buffer Overflow

## 2.4.3 Extraction Data Format

The GetNextData routine gets a series of bytes that represent the extracted data stream and places these bytes into the buffer pointed to by the *buffer parameter.
The Async Bus Extractor uses the following format for the data in this buffer:

Byte 0: Timestamp LSByte (in nanoseconds since start)
Byte 1: Timestamp
Byte 2: Timestamp
Byte 3: Timestamp
Byte 4: Timestamp
Byte 5: Timestamp
Byte 6: Timestamp
Byte 7: Timestamp MSByte
Byte 8: Record Type (bit 1 = 1 means character data is valid)
Byte 9: Channel number (0 thru 7)
Byte 10: Character
Byte 11: Errors (Bit 0 = Parity Error, Bit 1 = Framing (Stop) error)
Byte 12: Control Signal States (all 8 signal bits except async channels)
Byte 13: reserved
Byte 14: reserved
Byte 15: reserved
(repeat) …

## 2.4.4 Example Source Code

```
//*****************************************************
// USBee AX-Pro Data Extractor
// Async Bus Extractor Example Program
// Copyright 2006, CWAV All Rights Reserved.
//*****************************************************

#include "stdafx.h"
#include "stdio.h"
#include "conio.h"
#include "windows.h"
#include <fcntl.h>
#include <io.h>
#include <stdlib.h>
#include <stdio.h>

#define MAJOR_REV 1
#define MINOR_REV 0

//*****************************************************
// Declare the Extractor DLL API routines
//*****************************************************

#define CWAV_API __stdcall
#define CWAV_IMPORT __declspec(dllimport)

CWAV_IMPORT int CWAV_API StartExtraction(unsigned long PodNumber, unsigned long BaudRate, unsigned int DataBits,
unsigned int Parity, unsigned char Channels, unsigned char MSFirst, unsigned char StopBits);
CWAV_IMPORT char CWAV_API GetNextData(unsigned char *buffer, unsigned long length);
CWAV_IMPORT int CWAV_API StopExtraction( void );
CWAV_IMPORT char CWAV_API ExtractBufferOverflow(void);
CWAV_IMPORT unsigned long CWAV_API ExtractionBufferCount(void);

//*****************************************************
// Define the working buffer
//*****************************************************

#define WORKING_BUFFER_SIZE   (65536*8)
unsigned char tempbuffer[WORKING_BUFFER_SIZE];

// Command Line Parameter Settings
unsigned long P_PodID = 0;
unsigned char O_OutputFilename[256] = {0};
unsigned char S_Screen = FALSE;
unsigned char Y_LeastSignificantBitFirst = TRUE;
unsigned char Z_MostSignificantBitFirst = FALSE;
unsigned char A_ASCIITextValues = FALSE;
unsigned char D_DecimalTextValues = FALSE;
unsigned char H_HexTextValues = TRUE;
```

```
unsigned char B_BinaryTextValues = FALSE;
unsigned char I_BinaryValues = FALSE;
unsigned char C_CommaDelimited = FALSE;
unsigned char G_SpaceDelimited = TRUE;
unsigned char N_NewlineDelimited = FALSE;
unsigned char X_NoDelimeter = FALSE;
unsigned long T_ForceBytesPerLine = 0;
unsigned long M_SignalMask = 0xFFFFFFFF;
unsigned long Q_NumberOfBytes = 0;
unsigned long R_BaudRate = 9600;
unsigned long E_DataBits = 8;
unsigned long L_Parity = 0;
unsigned long V_Timestamps = 0;
unsigned long F_StopBits = 2;

typedef struct {

     __int64 TimeStamp;                // 64-bit time stamp at the start of this character or control signal change
     unsigned char RecordType;         // If the Character value is valid (1=Character is good, 0=Character is don't
care)
     unsigned char Signal;             // What channel this was sent on (0-7)
     unsigned char Character;      // Actual character data
     unsigned char Errors;             // Decodng error values (framing error, parity error)
     unsigned char Control;            // Control signal states starting here

} AsyncEvent;

AsyncEvent *AEvent;

void DisplayHelp(void)
{
     fprintf(stdout,"\nAsyncExtractor [-?SADHBICGNXYZ] [-R BaudRate] [-E DataBits] [-L Parity] [-M SignalMask] [-Q
NumberOfBytes] [-V Timestamp] [-O filename] -P PodID\n");

     fprintf(stdout,"\n    ?  - Display this help screen\n");

     fprintf(stdout,"\n  USBee AX-Pro Pod to Use\n");

     fprintf(stdout,"    P  - Pod ID (required)\n");

     fprintf(stdout,"\n  Output Location Flags\n");

     fprintf(stdout,"    O  - Output to filename (default off)\n");
     fprintf(stdout,"    S  - Output to the screen (default off)\n");

     fprintf(stdout,"\n  When to Quit Flags\n");

     fprintf(stdout,"    Q  - Number of output values (default = until keypress)\n");

     fprintf(stdout,"\n  Input Format Flags\n");

     fprintf(stdout,"    R  - Baud Rate (9600 baud default)\n");
     fprintf(stdout,"    E  - Number of Data Bits (5,6,7,8-default)\n");
     fprintf(stdout,"    L  - Parity Type (0=none(default), 1=mark, 2=space, 3=even, 4=odd)\n");
     fprintf(stdout,"    M  - Which Signals to capture (1=signal0, 128=signal7, 255=all, 0=none (default))\n");
     fprintf(stdout,"    Y  - LSBit first (default)\n");
     fprintf(stdout,"    Z  - MSBit first\n");
     fprintf(stdout,"    F  - Number of Stop Bits (2=1 (default), 3=1.5, 4=2)\n");

     fprintf(stdout,"\n  Output Number Format Flags\n");

     fprintf(stdout,"    A  - ASCII Text Values (\"1\")\n");
     fprintf(stdout,"    D  - Decimal Text Values (\"49\")\n");
     fprintf(stdout,"    H  - Hex Text Values (\"31\") default\n");
     fprintf(stdout,"    B  - Binary Text Values (\"00110001\")\n");
     fprintf(stdout,"    I  - Binary Values (49)\n");
     fprintf(stdout,"    C  - Comma Delimited\n");
     fprintf(stdout,"    G  - Space Delimited (default)\n");
     fprintf(stdout,"    N  - Newline Delimited\n");
     fprintf(stdout,"    X  - No Delimeter\n");

     fprintf(stdout,"\n  Timestamp and Channel Labels\n");

     fprintf(stdout,"    V  - Timestamps and Labels (0=Both off(default),1=Time each byte,2=Time and Labels,3=Labels
Only)\n");



}

void Error(char *err)
{
     fprintf(stderr,"Error: ");
     fprintf(stderr,"%s\n",err);
     exit(2);
}


//**************************************************
// Parse all of the command line options
//**************************************************
void ParseCommandLine(int argc, char *argv[])
{
     BOOL cont;
```

```
int       i,j;
DWORD WordExample;
BYTE ByteExample;

for(i=1; i < argc; ++i)
{
     if((argv[i][0] == '-') || (argv[i][0] == '/'))
     {
          cont = TRUE;
          for(j=1;argv[i][j] && cont;++j)     // Cont flag permits multiple commands in a single argv (like -AR)
               switch(toupper(argv[i][j]))
               {
                    case 'P':
                         P_PodID = (WORD)strtol(argv[++i],NULL,0);
                         cont = FALSE;
                         break;
                    case 'O':
                         strcpy((char*)O_OutputFilename, argv[++i]);
                         cont = FALSE;
                         break;
                    case '?':
                         DisplayHelp();
                         exit(0);

                         break;
                    case 'S':
                         S_Screen = TRUE;
                         break;
                    case 'Y':
                         Y_LeastSignificantBitFirst = TRUE;
                         Z_MostSignificantBitFirst = FALSE;
                         break;
                    case 'Z':
                         Z_MostSignificantBitFirst = TRUE;
                         Y_LeastSignificantBitFirst = FALSE;
                         break;
                    case 'A':
                         A_ASCIITextValues = TRUE;
                         H_HexTextValues = FALSE;
                         break;
                    case 'D':
                         D_DecimalTextValues = TRUE;
                         H_HexTextValues = FALSE;
                         break;
                    case 'H':
                         H_HexTextValues = TRUE;
                         break;
                    case 'B':
                         B_BinaryTextValues = TRUE;
                         H_HexTextValues = FALSE;
                         break;
                    case 'I':
                         I_BinaryValues = TRUE;
                         H_HexTextValues = FALSE;
                         break;
                    case 'C':
                         C_CommaDelimited = TRUE;
                         G_SpaceDelimited = FALSE;
                         break;
                    case 'G':
                         G_SpaceDelimited = TRUE;
                         break;
                    case 'N':
                         N_NewlineDelimited = TRUE;
                         G_SpaceDelimited = FALSE;
                         break;
                    case 'X':
                         X_NoDelimeter = TRUE;
                         G_SpaceDelimited = FALSE;
                         break;
                    case 'Q':
                         Q_NumberOfBytes = (DWORD)strtol(argv[++i],NULL,0);
                         cont = FALSE;
                         break;
                    case 'E':
                         E_DataBits = (DWORD)strtol(argv[++i],NULL,0);
                         cont = FALSE;
                         break;
                    case 'M':
                         M_SignalMask = (DWORD)strtol(argv[++i],NULL,0);
                         cont = FALSE;
                         break;
                    case 'F':
                         F_StopBits = (DWORD)strtol(argv[++i],NULL,0);
                         cont = FALSE;
                         break;
                    case 'V':
                         V_Timestamps = (DWORD)strtol(argv[++i],NULL,0);
                         cont = FALSE;
                         break;
                    case 'R':
                         R_BaudRate = (DWORD)strtol(argv[++i],NULL,0);
                         cont = FALSE;
                         break;
                    case 'L':
```

```
                              L_Parity = (BYTE)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'w':
                              WordExample = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'b':
                              ByteExample = (BYTE)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         default:
                              DisplayHelp();
                              fprintf(stdout,"\nCommand line switch %c not recognized\n",toupper(argv[i][j]));
                              Error("Invalid Command Line Switch");
                              exit(0);
                    }
               }
          }


          // Now check to see if they make sense
          if (P_PodID == 0)
          {
               DisplayHelp();
               Error("No Pod Number Specified");
          }

}


//*****************************************************
// Main Entry Point.  The program starts here.
//*****************************************************

int main(int argc, char* argv[])
{
     int RetValue;
     unsigned long totalbytes = 0;
     char *outputstr = new char [256];
     unsigned long ByteCounter = 0;
     unsigned long OutputValue;

     printf("USBee AX Data Extractor\n");
     printf("Async Bus Extractor Version %d.%d\n", MAJOR_REV, MINOR_REV);

     // Parse out the command line options
     ParseCommandLine( argc, argv );

     //*****************************************************
     // Open up a file to store extracted data into
     //*****************************************************

     FILE *fout;
     if (O_OutputFilename[0])
     {
          if (I_BinaryValues)
               fout = fopen((char*)O_OutputFilename, "wb");
          else
               fout = fopen((char*)O_OutputFilename, "w");
     }

     //*****************************************************
     // Start the USBee AX Pod extracting the data we want
     //*****************************************************

     printf("BaudRate=%d DataBits=%d Parity=%d StopBits=%g\n", R_BaudRate, E_DataBits, L_Parity, F_StopBits/2.0);

     RetValue = StartExtraction(P_PodID, R_BaudRate, E_DataBits, L_Parity, M_SignalMask, Z_MostSignificantBitFirst,
F_StopBits);

     if (RetValue == 0)
     {
          printf("Startup failed.  Is the USBee AX-Pro connected and is the PodNumber correct?\n");
          printf("Press any key to continue...");
          getch();
          return(0);
     }


     //*****************************************************
     // Loop and do something with the collected data
     //*****************************************************

     char OldSignal = 99;

     int KeepLooping = TRUE;
     while(KeepLooping)       // Do this forever until we tell it to stop by pressing a key
     {

          if (kbhit())
          {
               KeepLooping = FALSE;          // Stop the processing loop
               StopExtraction();             // Stop the streaming of data from the USBee
          }
```

```
        //*****************************************************
        // If there is data that has come in
        //*****************************************************
        int timeout = 0;
        while (unsigned long length = ExtractionBufferCount())
        {
             if (length > WORKING_BUFFER_SIZE)
                  length = WORKING_BUFFER_SIZE;

             //*****************************************************
             // Get the data into our local working buffer
             //*****************************************************

             GetNextData( tempbuffer, length );

             if (I_BinaryValues)      // Just write out the binary data to a file
             {
                  totalbytes += length;

                  if (O_OutputFilename[0])
                       fwrite(tempbuffer, length, 1,  fout);   // Write it to a file

                  if (Q_NumberOfBytes)
                  {
                       if (Q_NumberOfBytes <= length)
                       {
                            goto Done;          // Done with that many bytes
                       }
                       Q_NumberOfBytes -= length;
                  }

             }
             else      // It's a text output so format it
             {

                  // Now figure out what to send to the output
                  for (unsigned long x = 0; x < length; x += sizeof(AsyncEvent))
                  {
                       AEvent = (AsyncEvent *)&tempbuffer[x];

                       if (AEvent->RecordType != 1)  // This type of record records the edge changes of the other
signals
                       {
                            continue;      // Since we only print out the characters
                       }
                       int Channel = AEvent->Signal;

                       //*****************************************************
                       // Print the Timestamps and Channel Labels (if requested)
                       //*****************************************************

                       if ((V_Timestamps == 1) || ((V_Timestamps >= 2) && (OldSignal != AEvent->Signal)))
                       {
                            if (V_Timestamps == 1)        // Print just the timestamp
                            {
                                 if (C_CommaDelimited)
                                      sprintf(outputstr,"\n%I64d,",AEvent->TimeStamp);

                                 if (G_SpaceDelimited)
                                      sprintf(outputstr,"\n%I64d ",AEvent->TimeStamp);

                                 // Now send it out to the screen or file
                                 if (S_Screen)
                                      fputs(outputstr, stdout);

                                 if (O_OutputFilename[0])
                                      fputs(outputstr, fout);

                                 outputstr[0] = 0;
                            }
                            else if (V_Timestamps == 2)        // Print timestamp and channel number
                            {
                                 if (C_CommaDelimited)
                                      sprintf(outputstr,"\n%I64d,CH%d,",AEvent->TimeStamp,AEvent->Signal);

                                 if (G_SpaceDelimited)
                                      sprintf(outputstr,"\n%I64d CH%d ",AEvent->TimeStamp,AEvent->Signal);

                                 // Now send it out to the screen or file
                                 if (S_Screen)
                                      fputs(outputstr, stdout);

                                 if (O_OutputFilename[0])
                                      fputs(outputstr, fout);

                                 outputstr[0] = 0;
                            }
                            else if (V_Timestamps == 3)        // Print just the channel number
                            {
                                 if (C_CommaDelimited)
                                      sprintf(outputstr,"\nCH%d,",AEvent->Signal);

                                 if (G_SpaceDelimited)
                                      sprintf(outputstr,"\nCH%d ",AEvent->Signal);
```

```
          if (S_Screen)
               fputs(outputstr, stdout);

          if (O_OutputFilename[0])
               fputs(outputstr, fout);

          outputstr[0] = 0;
     }

     OldSignal = AEvent->Signal;
}

//*******************************************************
// Print out the actual Async Channel Data
//*******************************************************

if (V_Timestamps == 1)   // Print the "Timestamp every byte" format
{
     for (int y = 0; y < 8;y++)
     {
          if (Channel == y)        // Print a value here
          {
               OutputValue = AEvent->Character;

               // Now convert the value into the output text
               if (A_ASCIITextValues)
               {
                    outputstr[0] = (unsigned char)OutputValue;
                    outputstr[1] = 0;
               }
               if (D_DecimalTextValues)
               {
                    sprintf(outputstr,"%03d",OutputValue);
               }
               if (B_BinaryTextValues)
               {
                    int count;

                    count = 8;

                    unsigned int mask = 1 << (count - 1);
                    for (int z = 0; z < count; z++)
                    {
                         if (OutputValue & mask)
                              outputstr[z] = '1';
                         else
                              outputstr[z] = '0';
                         mask /= 2;
                    }

                    outputstr[z] = 0;

               }
               if (H_HexTextValues)
               {
                    sprintf(outputstr,"%02X", OutputValue);
               }

               totalbytes++;

               if (Q_NumberOfBytes)
                    if (--Q_NumberOfBytes == 0)
                    {
                         goto Done;          // Done with that many bytes
                    }

          }
          // Now add  delimeters
          if (C_CommaDelimited)
               strcat(outputstr, ",");

          if (G_SpaceDelimited)
               strcat(outputstr, " ");

          if (N_NewlineDelimited)
               strcat(outputstr, "\n");

          // Now send it out to the screen or file
          if (S_Screen)
               fputs(outputstr, stdout);

          if (O_OutputFilename[0])
               fputs(outputstr, fout);

          outputstr[0] = 0;
     }

}
else // Print the "each line is a single channel" format
{

     OutputValue = AEvent->Character;

     // Now convert the value into the output text
```

```
                    if (A_ASCIITextValues)
                    {
                         outputstr[0] = (unsigned char)OutputValue;
                         outputstr[1] = 0;
                    }
                    if (D_DecimalTextValues)
                    {
                         sprintf(outputstr,"%03d",OutputValue);
                    }
                    if (B_BinaryTextValues)
                    {
                         int count;

                         count = 8;

                         unsigned int mask = 1 << (count - 1);
                         for (int z = 0; z < count; z++)
                         {
                              if (OutputValue & mask)
                                   outputstr[z] = '1';
                              else
                                   outputstr[z] = '0';
                              mask /= 2;
                         }

                         outputstr[z] = 0;

                    }
                    if (H_HexTextValues)
                    {
                         sprintf(outputstr,"%02X", OutputValue);
                    }

                    totalbytes++;

                    if (Q_NumberOfBytes)
                         if (--Q_NumberOfBytes == 0)
                         {
                              goto Done;          // Done with that many bytes
                         }

                    // Now add  delimeters
                    if (C_CommaDelimited)
                         strcat(outputstr, ",");

                    if (G_SpaceDelimited)
                         strcat(outputstr, " ");

                    if (N_NewlineDelimited)
                         strcat(outputstr, "\n");

                    // Now send it out to the screen or file
                    if (S_Screen)
                         fputs(outputstr, stdout);

                    if (O_OutputFilename[0])
                         fputs(outputstr, fout);

                    outputstr[0] = 0;

               }
          }

     }

     if (timeout++ > 10  ) break;  // Let up once in a while to let the OS process
}

if (!S_Screen)
     printf("\rProcessed %d output values.", totalbytes);

//*************************************************
// Check to see if we have fallen behind too far
//*************************************************

int y = ExtractBufferOverflow();

if (y == 1)
{
     printf("\nExtractor Buffer Overflow.\nYour data is streaming too fast for your output settings.\nLower
your data rate or change to output binary files.\n");
     goto Done;
}
else if (y == 2)
{
     printf("\nRaw Sample Buffer Overflow.\nYour data is streaming too fast for your output settings.\nLower
your data rate or change to output binary files.\n");
     goto Done;
}

//*************************************************
// Give the OS a little time to do something else
//*************************************************

Sleep(15);
```

```
     }

Done:
     if (!S_Screen)
          printf("\rProcessed %d output values.", totalbytes);

     //**************************************************
     // Close the file
     //**************************************************

     if (O_OutputFilename[0])
          fclose(fout);

     //**************************************************
     // Stop the extraction process
     //**************************************************

     StopExtraction();

     if (kbhit()) getch();
     printf("\nPress any key to continue...");
     getch();

     return 0;
}
```

# 3 Parallel Bus Data Extractor

The Parallel Bus Data Extractor takes the real-time streaming data from an embedded 8-bit parallel bus, formats it and allows you to save the data to disk or process it as it arrives.

## 3.1 Parallel Bus Data Extractor Specifications

- Continuous Real-Time Data Streaming
- 8 digital channels
- TTL Level inputs (**0-5V max**, Vih = 2.0V, Vil = 0.8V)
- Synchronous or Asynchronous Clocking
- Synchronous (external) clock 0 to 16MB/s*
- Asynchronous (internal) clock 1MB/s to 24MB/s*
- Input in 1, 2 or 4 byte serial words
- Little or Big Endian
- Output to Binary File*
- Output to Text File (Hex, Decimal, Binary or ASCII)*
- Output to Screen*
- Comma, Space, or Newline Delimited files
- Output Value Filtering
- Output File Viewer (including binary, text, search and export functions)
- Extractor API libraries interface directly to your own software to further process the extracted data. Any language that supports calls to DLLs is supported.

* - output bandwidths are dependent on PC USB hardware, hard disk and/or screen throughput.

## 3.2 Hardware Setup

To use the Data Extractor you need to connect the USBee AX-Pro Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

***Please note that the USBee AX-Pro Test Pod inputs are strictly 0-5V levels. Any voltage outside this range on the signals will damage the pod and may damage your hardware. If your system uses different voltage levels, you must buffer the signals externally to the USBee AX-Pro Test Pod before connecting the signals to the unit.***

The Parallel Bus Data Extractor uses the 8 signal lines (0 thru 7), the GND (ground) line and optionally the CLK and TRG lines (for external timing). The signal 0 is represented in the bit 0 of each sampled byte. Connect the GND line to the digital ground of your system.

## 3.3 Extractor Command Line Program

The Parallel Bus Data Extractor includes a Windows Command Prompt executable that lets you operate the Data Extractor without writing any software. The program is executed in a Command Prompt window and is configured using command line arguments. The extracted data is then stored to disk or outputted to the screen depending on these parameters.

To run the Data Extractor:

1) Install the USBee AX-Pro software on your PC
2) Install the Data Extractor software on your PC
3) Plug in your USBee AX-Pro Test Pod into your PC using a USB 2.0 High Speed Port
4) Open a Windows Command Prompt window by clicking Start, All Programs, Accessories, Command Prompt.
5) Change the working directory to the Data Extractor directory
6) (“cd \program files\USBee Data Extractor\Parallel”)
7) Run the executable using the following command line arguments:

```
BasicExtractor [-?SADHBICGNX124YZ] [-E clock mode] [-Q NumberOfBytes] [-T BytesPesLine]
[-R SampleRate] [-M SignalMask] [-L FilterValue] [-V FilterMask] [-O filename] -P PodID
```

? - Display this help screen

**USBee AX-Pro Pod to Use**

P - Pod ID (required)

**Output Location Flags**

O - Output to filename (default off)
S - Output to the screen (default off)

**When to Quit Flags**

Q - Number of output values (default = until keypress)

**Input Number Format Flags**

1 - One Byte per value (default)
2 - Two Bytes per value
4 - Four Bytes per value
Y - Least significant byte first
Z - Most significant byte first

**Output Number Format Flags**

A - ASCII Text Values ("1")
D - Decimal Text Values ("49")
H - Hex Text Values ("31") default
B - Binary Text Values ("00110001")
I - Binary Values (49)
C - Comma Delimited
G - Space Delimited (default)
N - Newline Delimited
X - No Delimeter
T - Force Bytes Per Line (no force default)

**Filter Values**

M - Which Signals to capture (1=signal0,255=all(default))
L - Filter Mask (0=no filter,255=filter on all signals)
V - Filter Value (0=store when 0's,255=store when 1's)

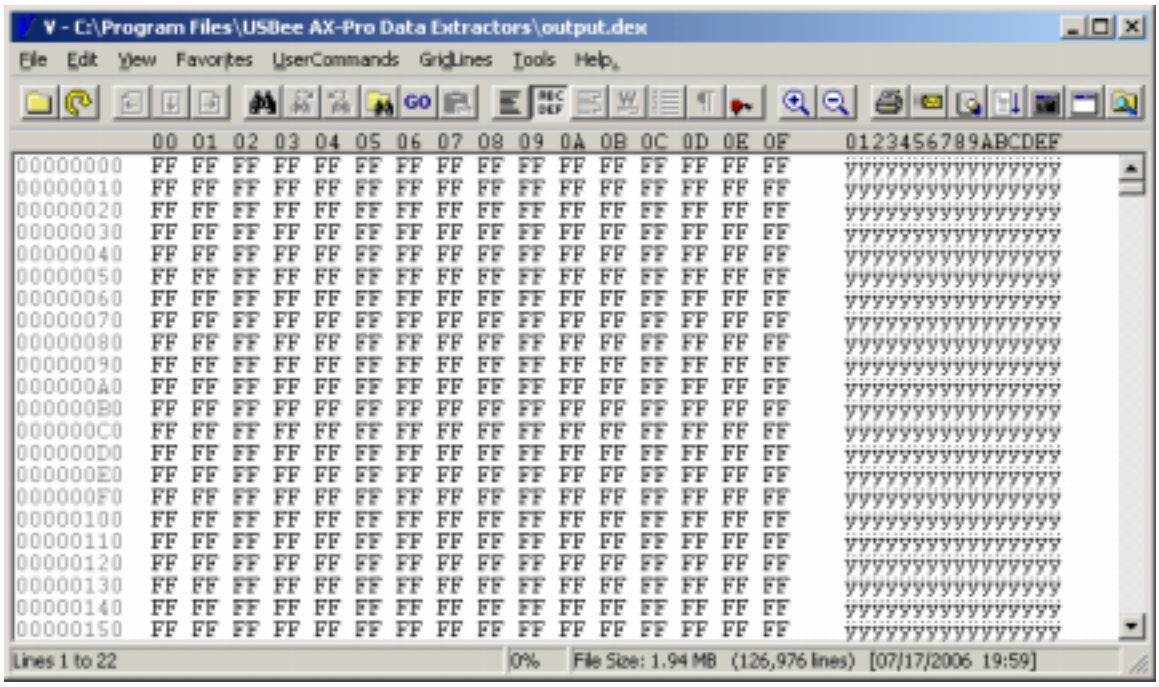
**Clocking Modes**

E - Clocking mode (2=internal (default),
4=CLK rising, 5-CLK falling,
6-CLK rising AND TRG high, 7-CLK falling AND TRG high
8-CLK rising AND TRG low, 9-CLK falling AND TRG low
R - Internal CLK Sample Rate (1Msps default)

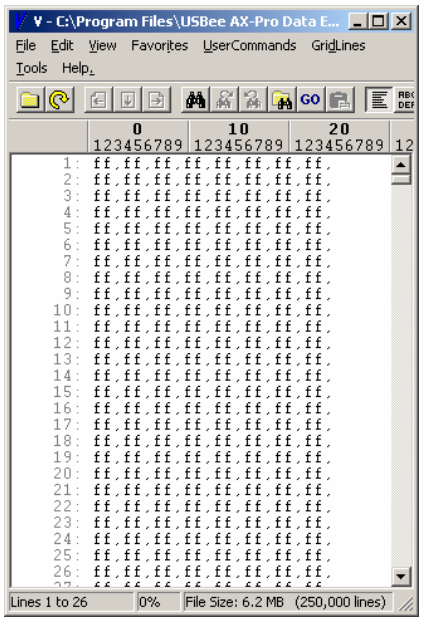
- 247 = 24MHz
- 167 = 16MHz
- 127 = 12MHz
- 87 = 8MHz
- 67 = 6MHz
- 47 = 4MHz
- 37 = 3MHz
- 27 = 2MHz
- 17 = 1MHz (default)

### 3.3.1 Example Output

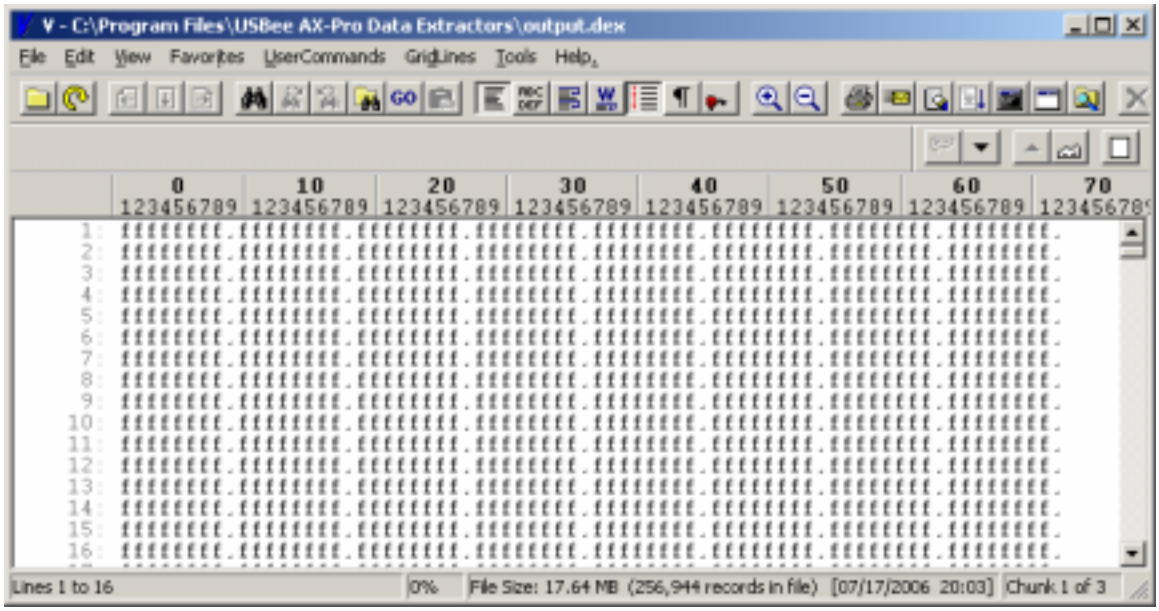
```
BasicExtractor -O output.dex -P 3209 -1 -R 27 -T 8 -Q 2000000 -I
```




```
BasicExtractor -O output.dex -P 3209 -1 -R 27 -T 8 -Q 2000000 -C
```



```
BasicExtractor -O output.dex -P 3209 -1 -R 27 -T 8 -Q 2000000 -C -4
```




## *3.4 Extractor API*

The Data Extractor is implemented using a Windows DLL that interfaces to the existing USBee AX-Pro DLL and drivers. This DLL can be called using any software language that supports calls to DLLs. Below are the details of this DLL interface and the routines that are available for your use.

### 3.4.1 DLL filename:

`usbexBasic.dll in \Windows\System32`

### 3.4.2 DLL Exported Functions and parameters

**ExtractionBufferCount** – Returns the number of bytes that have been extracted from the data stream so far and are available to read using GetNextData.

```
CWAV_EXPORT unsigned long CWAV_API ExtractionBufferCount(void)
```

Returns:
0 – No data to read yet
other – number of bytes available to read

**GetNextData** – Copies the extracted data from the extractor into your working buffer

```
CWAV_EXPORT char CWAV_API GetNextData(unsigned char *buffer, unsigned long length);
```

buffer:
pointer to where you want the extracted data to be placed
length:
number of bytes you want to read from the extraction DLL
Returns:
0 – No data to read yet
1 – Data was copied into the buffer

**StartExtraction** – Starts the Data Extraction with the given parameters.

```
CWAV_EXPORT int CWAV_API StartExtraction( unsigned int SampleRate, unsigned long
PodNumber, unsigned int ClockMode);
```

SampleRate:
- 17 = 1Msps
- 27 = 2Msps
- 37 = 3Msps
- 47 = 4Msps
- 67 = 6Msps
- 87 = 8Msps
- 127 = 12Msps
- 167 = 16Msps
- 247 = 24Msps

PodNumber:
- Pod ID on the back of the USBee AX-Pro Test Pod

ClockMode:
- 2 = Internal Timing as in SampleRate parameter
- 4 – External Timing – sample on rising edge of CLK
- 5 – External Timing – sample on falling edge of CLK
- 6 – External Timing – sample on rising edge of CLK and TRG high
- 7 – External Timing – sample on falling edge of CLK and TRG high
- 8 – External Timing – sample on rising edge of CLK and TRG low
- 9 – External Timing – sample on falling edge of CLK and TRG low

Returns:
- 1 – if Start was successful
- 0 – if Pod failed initialization

**StopExtraction** – Stops the extraction in progress

```
CWAV_EXPORT int CWAV_API StopExtraction( void );
```

Returns:
- 1 – always

**ExtractBufferOverflow** – Returns the state of the overflow conditions

```
CWAV_EXPORT char CWAV_API ExtractBufferOverflow(void);
```

Return:
- 0 – No overflow
- 1 – Overflow Occurred. ExtractorBuffer Overflow condition cleared.
- 2 – Overflow Occurred. Raw Stream Buffer Overflow

### 3.4.3 Extraction Data Format

The GetNextData routine gets a series of bytes that represent the extracted data stream and places these bytes into the buffer pointed to by the *buffer parameter.

The Parallel Bus Extractor uses the following format for the data in this buffer:

Byte 0: Byte 0 of the sampled data
Byte 1: Byte 1 of the sampled data
Byte 2: Byte 2 of the sampled data
Byte 3: Byte 3 of the sampled data
...
Byte N: Byte N of the sampled data

### 3.4.4 Example Source Code

```
//*****************************************************
// USBee AX-Pro Data Extractor
// Parallel Bus Extractor Example Program
// Copyright 2006, CWAV All Rights Reserved.
//*****************************************************

#include "stdafx.h"
#include "stdio.h"
#include "conio.h"
#include "windows.h"
#include <fcntl.h>
#include <io.h>
#include <stdlib.h>
#include <stdio.h>

#define MAJOR_REV 1
#define MINOR_REV 0

//*****************************************************
// Declare the Extractor DLL API routines
//*****************************************************

#define CWAV_API __stdcall
#define CWAV_IMPORT __declspec(dllimport)

CWAV_IMPORT int CWAV_API StartExtraction( unsigned int SampleRate, unsigned long PodNumber, unsigned int ClockMode );

CWAV_IMPORT char CWAV_API GetNextData(unsigned char *buffer, unsigned long length);
CWAV_IMPORT int CWAV_API StopExtraction( void );
CWAV_IMPORT char CWAV_API ExtractBufferOverflow(void);
CWAV_IMPORT unsigned long CWAV_API ExtractionBufferCount(void);

//*****************************************************
// Define the working buffer
//*****************************************************

#define WORKING_BUFFER_SIZE   (65536*8)
unsigned char tempbuffer[WORKING_BUFFER_SIZE];

// Command Line Parameter Settings
unsigned long P_PodID = 0;
unsigned char O_OutputFilename[256] = {0};
unsigned char S_Screen = FALSE;
unsigned char _1_BytePerValue = TRUE;
unsigned char _2_BytePerValue = FALSE;
unsigned char _4_BytePerValue = FALSE;
unsigned char Y_LeastSignificantByteFirst = FALSE;
unsigned char Z_MostSignificantByteFirst = TRUE;
unsigned char A_ASCIITextValues = FALSE;
unsigned char D_DecimalTextValues = FALSE;
unsigned char H_HexTextValues = TRUE;
unsigned char B_BinaryTextValues = FALSE;
unsigned char I_BinaryValues = FALSE;
unsigned char C_CommaDelimited = FALSE;
unsigned char G_SpaceDelimited = TRUE;
unsigned char N_NewlineDelimited = FALSE;
unsigned char X_NoDelimeter = FALSE;
unsigned long T_ForceBytesPerLine = 0;
unsigned long M_SignalMask = 0xFFFFFFFF;
unsigned long L_FilterMask = 0;
unsigned long V_FilterValue = 0;
unsigned char E_ExternalClockMode = 2;
unsigned char R_SampleRate = 17;
unsigned long Q_NumberOfBytes = 0;
// Not used yet J,K,Q,U,W
```

```
void DisplayHelp(void)
{
     fprintf(stdout,"\nBasicExtractor [-?SADHBICGNX124YZ] [-Q NumberOfBytes] [-T BytesPesLine] [-R SampleRate] [-M
SignalMask] [-L FilterValue] [-V FilterMask] [-O filename] -P PodID\n\n");
     fprintf(stdout,"    ?  - Display this help screen\n");

     fprintf(stdout,"\n  USBee AX-Pro Pod to Use\n");
     fprintf(stdout,"    P  - Pod ID (required)\n");

     fprintf(stdout,"\n  Output Location Flags\n");
     fprintf(stdout,"    O  - Output to filename (default off)\n");
     fprintf(stdout,"    S  - Output to the screen (default off)\n");

     fprintf(stdout,"\n  When to Quit Flags\n");
     fprintf(stdout,"    Q  - Number of output values (default = until keypress)\n");

     fprintf(stdout,"\n  Input Number Format Flags\n");
     fprintf(stdout,"    1  - One   Byte  per value (default)\n");
     fprintf(stdout,"    2  - Two   Bytes per value\n");
     fprintf(stdout,"    4  - Four  Bytes per value\n");
     fprintf(stdout,"    Y  - Least significant byte first\n");
     fprintf(stdout,"    Z  - Most significant byte first\n");

     fprintf(stdout,"\n  Output Number Format Flags\n");
     fprintf(stdout,"    A  - ASCII Text Values (\"1\")\n");
     fprintf(stdout,"    D  - Decimal Text Values (\"49\")\n");
     fprintf(stdout,"    H  - Hex Text Values (\"31\") default\n");
     fprintf(stdout,"    B  - Binary Text Values (\"00110001\")\n");
     fprintf(stdout,"    I  - Binary Values (49)\n");
     fprintf(stdout,"    C  - Comma Delimited\n");
     fprintf(stdout,"    G  - Space Delimited (default)\n");
     fprintf(stdout,"    N  - Newline Delimited\n");
     fprintf(stdout,"    X  - No Delimeter\n");
     fprintf(stdout,"    T  - Force Bytes Per Line (no force default)\n");

     fprintf(stdout,"\n  Filter Values\n");
     fprintf(stdout,"    M  - Which Signals to capture (1=signal0,255=all(default))\n");
     fprintf(stdout,"    L  - Filter Mask (0=no filter,255=filter on all signals)\n");
     fprintf(stdout,"    V  - Filter Value (0=store when 0's,255=store when 1's)\n");

     fprintf(stdout,"\n  Clocking Modes\n");
     fprintf(stdout,"    E  - Clocking mode (2=internal (default),\n");
     fprintf(stdout,"                        4=CLK rising,5-CLK falling,\n");
     fprintf(stdout,"                        6-CLK rising AND TRG high,7-CLK falling AND TRG high\n");
     fprintf(stdout,"                        8-CLK rising AND TRG low,9-CLK falling AND TRG low\n");
     fprintf(stdout,"    R  - Internal CLK Sample Rate (1Msps default)\n");


     exit(0);
}

void Error(char *err)
{
     fprintf(stderr,"Error: ");
     fprintf(stderr,"%s\n",err);
     exit(2);
}


//*************************************************
// Parse all of the command line options
//*************************************************
void ParseCommandLine(int argc, char *argv[])
{
     BOOL cont;
     int       i,j;
     DWORD WordExample;
     BYTE ByteExample;

     for(i=1; i < argc; ++i)
     {
          if((argv[i][0] == '-') || (argv[i][0] == '/'))
          {
               cont = TRUE;
               for(j=1;argv[i][j] && cont;++j)     // Cont flag permits multiple commands in a single argv (like -AR)
                    switch(toupper(argv[i][j]))
                    {
                         case 'P':
                              P_PodID = (WORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'O':
                              strcpy((char*)O_OutputFilename, argv[++i]);
                              cont = FALSE;
                              break;
                         case '?':
                              DisplayHelp();
                              break;
                         case 'S':
                              S_Screen = TRUE;
                              break;
                         case '1':
                              _1_BytePerValue = TRUE;
                              break;
```

```
                    case '2':
                         _2_BytePerValue = TRUE;
                         _1_BytePerValue = FALSE;
                         break;
                    case '4':
                         _4_BytePerValue = TRUE;
                         _1_BytePerValue = FALSE;
                         break;
                    case 'Y':
                         Y_LeastSignificantByteFirst = TRUE;
                         Z_MostSignificantByteFirst = FALSE;
                         break;
                    case 'Z':
                         Z_MostSignificantByteFirst = TRUE;
                         Y_LeastSignificantByteFirst = FALSE;
                         break;
                    case 'A':
                         A_ASCIITextValues = TRUE;
                         H_HexTextValues = FALSE;
                         break;
                    case 'D':
                         D_DecimalTextValues = TRUE;
                         H_HexTextValues = FALSE;
                         break;
                    case 'H':
                         H_HexTextValues = TRUE;
                         break;
                    case 'B':
                         B_BinaryTextValues = TRUE;
                         H_HexTextValues = FALSE;
                         break;
                    case 'I':
                         I_BinaryValues = TRUE;
                         H_HexTextValues = FALSE;
                         break;
                    case 'C':
                         C_CommaDelimited = TRUE;
                         G_SpaceDelimited = FALSE;
                         break;
                    case 'G':
                         G_SpaceDelimited = TRUE;
                         break;
                    case 'N':
                         N_NewlineDelimited = TRUE;
                         G_SpaceDelimited = FALSE;
                         break;
                    case 'X':
                         X_NoDelimeter = TRUE;
                         G_SpaceDelimited = FALSE;
                         break;
                    case 'T':
                         T_ForceBytesPerLine = (DWORD)strtol(argv[++i],NULL,0);
                         cont = FALSE;
                         break;
                    case 'Q':
                         Q_NumberOfBytes = (DWORD)strtol(argv[++i],NULL,0);
                         cont = FALSE;
                         break;
                    case 'M':
                         M_SignalMask = (DWORD)strtol(argv[++i],NULL,0);
                         cont = FALSE;
                         break;
                    case 'L':
                         L_FilterMask = (DWORD)strtol(argv[++i],NULL,0);
                         cont = FALSE;
                         break;
                    case 'V':
                         V_FilterValue = (DWORD)strtol(argv[++i],NULL,0);
                         cont = FALSE;
                         break;
                    case 'E':
                         E_ExternalClockMode = (DWORD)strtol(argv[++i],NULL,0);
                         cont = FALSE;
                         break;
                    case 'R':
                         R_SampleRate = (BYTE)strtol(argv[++i],NULL,0);
                         cont = FALSE;
                         break;
                    case 'w':
                         WordExample = (DWORD)strtol(argv[++i],NULL,0);
                         cont = FALSE;
                         break;
                    case 'b':
                         ByteExample = (BYTE)strtol(argv[++i],NULL,0);
                         cont = FALSE;
                         break;
                    default:
                         DisplayHelp();
                         Error("Invalid Command Line Switch");
               }
          }
     }

     // Now check to see if they make sense
```

```
     if (P_PodID == 0)
     {
          DisplayHelp();
          Error("No Pod Number Specified");
     }

}

unsigned long StartTime;

void StartTimer()
{

     StartTime = GetTickCount();
}

void StopTimer()
{

     printf(" \nTime Delta = %d\n",GetTickCount() - StartTime);

}

//**************************************************
// Main Entry Point.  The program starts here.
//**************************************************

int main(int argc, char* argv[])
{
     int RetValue;
     unsigned long totalbytes = 0;
     char *outputstr = new char [256];
     unsigned long ByteCounter = 0;
     unsigned long OutputValue;

     printf("USBee AX Data Extractor\n");
     printf("Parallel Bus Extractor Version %d.%d\n", MAJOR_REV, MINOR_REV);

     // Parse out the command line options
     ParseCommandLine( argc, argv );

     //**************************************************
     // Open up a file to store extracted data into
     //**************************************************

     FILE *fout;
     if (O_OutputFilename[0])
     {
          if (I_BinaryValues)
               fout = fopen((char*)O_OutputFilename, "wb");
          else
               fout = fopen((char*)O_OutputFilename, "w");
     }

     //**************************************************
     // Start the USBee AX Pod extracting the data we want
     //**************************************************


     RetValue = StartExtraction( R_SampleRate, P_PodID, E_ExternalClockMode );

     if (RetValue == 0)
     {
          printf("Startup failed.  Is the USBee AX-Pro connected and is the PodNumber correct?\n");
          printf("Press any key to continue...");
          getch();
          return(0);
     }

     printf("Processing and Saving Data to Disk.\n");

     //**************************************************
     // Loop and do something with the collected data
     //**************************************************

     int KeepLooping = TRUE;
     while(KeepLooping)       // Do this forever until we tell it to stop by pressing a key
     {

          if (kbhit())
          {
               KeepLooping = FALSE;          // Stop the processing loop
               StopExtraction();             // Stop the streaming of data from the USBee
          }

          //**************************************************
          // If there is data that has come in
          //**************************************************
          int timeout = 0;
          while (unsigned long length = ExtractionBufferCount())
          {
               if (length > WORKING_BUFFER_SIZE)
                    length = WORKING_BUFFER_SIZE;

               //**************************************************
```

```
// Get the data into our local working buffer
//*************************************************
StartTimer();

GetNextData( tempbuffer, length );

if (I_BinaryValues)      // Just write out the binary data to a file
{
     totalbytes += length;

     if (O_OutputFilename[0])
          fwrite(tempbuffer, length, 1,  fout);   // Write it to a file

     if (Q_NumberOfBytes)
     {
          if (Q_NumberOfBytes <= length)
          {
               goto Done;          // Done with that many bytes
          }
          Q_NumberOfBytes -= length;
     }

}
else      // It's a text output so format it all pretty-like
{

     // Now figure out what to send to the output
     for (unsigned long x = 0; x < length;)
     {
          // First get the value to print out
          if (_1_BytePerValue)
          {
               OutputValue = tempbuffer[x];
               x++;
          }
          if (_2_BytePerValue)
          {
               if (Y_LeastSignificantByteFirst)
                    OutputValue = (tempbuffer[x+1] << 8) + tempbuffer[x+0];
               else
                    OutputValue = (tempbuffer[x+0] << 8) + tempbuffer[x+1];
               x += 2;
          }
          if (_4_BytePerValue)
          {
               if (Y_LeastSignificantByteFirst)
                    OutputValue =  (tempbuffer[x+3] << 24) +
                                   (tempbuffer[x+2] << 16) +
                                   (tempbuffer[x+1] << 8) +
                                   tempbuffer[x+0];
               else
                    OutputValue =  (tempbuffer[x+0] << 24) +
                                   (tempbuffer[x+1] << 16) +
                                   (tempbuffer[x+2] << 8) +
                                   tempbuffer[x+3];

               x += 4;
          }

          // Perform the Masking
          OutputValue &= M_SignalMask;

          // Perform the filtering
          if ((OutputValue & L_FilterMask) != V_FilterValue)
               continue;      // Not for use to save so move on.

          // Now convert the value into the output text
          if (A_ASCIITextValues)
          {
               outputstr[0] = (unsigned char)OutputValue;
               outputstr[1] = 0;
          }
          if (D_DecimalTextValues)
          {
               ultoa(OutputValue,outputstr,10);
               // sprintf(outputstr,"%d",OutputValue);
          }
          if (B_BinaryTextValues)
          {
               int count;

               if (_1_BytePerValue)
                    count = 8;
               if (_2_BytePerValue)
                    count = 16;
               if (_4_BytePerValue)
                    count = 32;

               unsigned int mask = 1 << (count - 1);
               for (int z = 0; z < count; z++)
               {
                    if (OutputValue & mask)
                         outputstr[z] = '1';
                    else
                         outputstr[z] = '0';
```

```
                              mask /= 2;
                         }
                    }
                    if (H_HexTextValues)
                    {
                         if (_1_BytePerValue)
                              ultoa(OutputValue, outputstr, 16);
                              //sprintf(outputstr,"%02X", OutputValue);
                         if (_2_BytePerValue)
                              ultoa(OutputValue, outputstr, 16);
                              //sprintf(outputstr,"%04X", OutputValue);
                         if (_4_BytePerValue)
                              ultoa(OutputValue, outputstr, 16);
                              //sprintf(outputstr,"%08X", OutputValue);
                    }

                    // Now add any delimeters to the end of the value
                    if (C_CommaDelimited)
                         strcat(outputstr, ",");

                    if (G_SpaceDelimited)
                         strcat(outputstr, " ");

                    if (N_NewlineDelimited)
                         strcat(outputstr, "\n");

                    if (T_ForceBytesPerLine)
                    {
                         if (++ByteCounter >= T_ForceBytesPerLine)
                         {
                              ByteCounter = 0;
                              strcat(outputstr, "\n");
                         }
                    }

                    if (S_Screen)
                         fputs(outputstr, stdout);

                    if (O_OutputFilename[0])
                         fputs(outputstr, fout);

                    totalbytes++;

                    if (Q_NumberOfBytes)
                         if (--Q_NumberOfBytes == 0)
                         {
                              goto Done;          // Done with that many bytes
                         }

               }

          }

          // StopTimer();

          if (timeout++ > 10  ) break;  // Let up once in a while to let the OS process
     }

     if (!S_Screen)
          printf("\rProcessed %d output values.", totalbytes);

     //*************************************************
     // Check to see if we have fallen behind too far
     //*************************************************

     int y = ExtractBufferOverflow();

     if (y == 1)
     {
          printf("\nExtractor Buffer Overflow.\nYour data is streaming too fast for your output settings.\nLower
your data rate or change to output binary files.\n");
          goto Done;
     }
     else if (y == 2)
     {
          printf("\nRaw Sample Buffer Overflow.\nYour data is streaming too fast for your output settings.\nLower
your data rate or change to output binary files.\n");
          goto Done;
     }

     //*************************************************
     // Give the OS a little time to do something else
     //*************************************************

     Sleep(15);

}

Done:
     if (!S_Screen)
          printf("\rProcessed %d output values.", totalbytes);

     //*************************************************
     // Close the file
     //*************************************************
```

```
     if (O_OutputFilename[0])
          fclose(fout);

     //*************************************************
     // Stop the extraction process
     //*************************************************

     StopExtraction();

     if (kbhit()) getch();
     printf("\nPress any key to continue...");
     getch();

     return 0;
}
```

# 4 Serial Bus Data Extractor

The Serial Bus Data Extractor takes the real-time streaming data from up to 8 serial data lines, formats it and allows you to save the data to disk or process it as it arrives.

## *4.1 Serial Bus Data Extractor Specifications*

- Continuous Real-Time Data Streaming
- 8 digital channels
- TTL Level inputs (**0-5V max**, Vih = 2.0V, Vil = 0.8V)
- Synchronous or Asynchronous Clocking
- Synchronous (external) clock 0 to 16MB/s*
- Asynchronous (internal) clock 1MB/s to 24MB/s*
- Input in 1, 2 or 4 byte serial words
- Little or Big Endian
- Output to Binary File*
- Output to Text File (Hex, Decimal, Binary or ASCII)*
- Output to Screen*
- Comma, Space, or Newline Delimited files
- Output Value Filtering
- Output File Viewer (including binary, text, search and export functions)
- Extractor API libraries interface directly to your own software to further process the extracted data. Any language that supports calls to DLLs is supported.

* - output bandwidths are dependent on PC USB hardware, hard disk and/or screen throughput.

## *4.2 Hardware Setup*

To use the Data Extractor you need to connect the USBee AX-Pro Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

***Please note that the USBee AX-Pro Test Pod inputs are strictly 0-5V levels. Any voltage outside this range on the signals will damage the pod and may damage your hardware. If your system uses different voltage levels, you must buffer the signals externally to the USBee AX-Pro Test Pod before connecting the signals to the unit.***

The Serial Bus Data Extractor uses any of the 8 signal lines (0 thru 7), the GND (ground) line and optionally the CLK and TRG lines (for external timing). Connect the GND line to the digital ground of your system.

## *4.3 Extractor Command Line Program*

The Serial Bus Data Extractor includes a Windows Command Prompt executable that lets you operate the Data Extractor without writing any software. The program is executed in a Command Prompt window and is configured using command line arguments. The extracted data is then stored to disk or outputted to the screen depending on these parameters.

To run the Data Extractor:

1) Install the USBee AX-Pro software on your PC
2) Install the Data Extractor software on your PC
3) Plug in your USBee AX-Pro Test Pod into your PC using a USB 2.0 High Speed Port
4) Open a Windows Command Prompt window by clicking Start, All Programs, Accessories, Command Prompt.
5) Change the working directory to the Data Extractor directory
6) (“cd \program files\USBee Data Extractor\Serial”)
7) Run the executable using the following command line arguments:

```
SerialExtractor [-?SADHBICGNX124YZ] [-Q NumberOfBytes] [-T BytesPesLine] [-R SampleRate]
[-E ClockingMode] [-M SignalMask] [-J ChannelAlign] [-L SignalLevel] [-V AlignmentValue]
[-O filename] -P PodID
```

? - Display this help screen

**USBee AX-Pro Pod to Use**

P - Pod ID (required)

**Output Location Flags**

O - Output to filename (default off)
S - Output to the screen (default off)

**When to Quit Flags**

Q - Number of output values (default = until keypress)

**Input Number Format Flags**

1 - One Byte per value (default)
2 - Two Bytes per value
4 - Four Bytes per value
Y - Least significant bit first
Z - Most significant bit first

**Output Number Format Flags**

A - ASCII Text Values ("1")
D - Decimal Text Values ("49")
H - Hex Text Values ("31") default
B - Binary Text Values ("00110001")
I - Binary Values (49)
C - Comma Delimited
G - Space Delimited (default)
N - Newline Delimited
X - No Delimeter
T - Force Bytes Per Line (no force default)

**Filter Values**

M - Which Signals to capture (1=signal0,255=all(default))

**Bit 0 Alignment**

V - Align on Value
L - Align on Signal Level (0=low,1=high)
J - Which signal to use for alignment (1=signal0,128=signal7)

**Clocking Modes**

E - Clocking mode (2=internal (default),
4=CLK rising, 5-CLK falling,
6-CLK rising AND TRG high, 7-CLK falling AND TRG high
8-CLK rising AND TRG low, 9-CLK falling AND TRG low
R - Internal CLK Sample Rate (1Msps default)

- 247 = 24MHz
- 167 = 16MHz
- 127 = 12MHz
- 87 = 8MHz
- 67 = 6MHz
- 47 = 4MHz
- 37 = 3MHz
- 27 = 2MHz

- 17 = 1MHz (default)

## 4.4 Extractor API

The Data Extractor is implemented using a Windows DLL that interfaces to the existing USBee AX-Pro DLL and drivers. This DLL can be called using any software language that supports calls to DLLs. Below are the details of this DLL interface and the routines that are available for your use.

### 4.4.1 DLL filename:

`usbexSerial.dll in \Windows\System32`

### 4.4.2 DLL Exported Functions and parameters

**ExtractionBufferCount** – Returns the number of bytes that have been extracted from the data stream so far and are available to read using GetNextData.

```
CWAV_EXPORT unsigned long CWAV_API ExtractionBufferCount(void)
```

Returns:
0 – No data to read yet
other – number of bytes available to read

**GetNextData** – Copies the extracted data from the extractor into your working buffer

```
CWAV_EXPORT char CWAV_API GetNextData(unsigned char *buffer, unsigned long length);
```

buffer:
pointer to where you want the extracted data to be placed
length:
number of bytes you want to read from the extraction DLL
Returns:
0 – No data to read yet
1 – Data was copied into the buffer

**StartExtraction** – Starts the Data Extraction with the given parameters.

```
CWAV_EXPORT int CWAV_API StartExtraction( unsigned int SampleRate, unsigned long
PodNumber, unsigned int ClockMode, unsigned long AlignValue, unsigned char SignalLevel,
unsigned char AlignChannel, unsigned char BytePerValue);
```

SampleRate:
17 = 1Msps
27 = 2Msps
37 = 3Msps
47 = 4Msps
67 = 6Msps
87 = 8Msps
127 = 12Msps
167 = 16Msps
247 = 24Msps
PodNumber:
Pod ID on the back of the USBee AX-Pro Test Pod
ClockMode:
2 = Internal Timing as in SampleRate parameter
4 – External Timing – sample on rising edge of CLK
5 – External Timing – sample on falling edge of CLK
6 – External Timing – sample on rising edge of CLK and TRG high

7 – External Timing – sample on falling edge of CLK and TRG high
8 – External Timing – sample on rising edge of CLK and TRG low
9 – External Timing – sample on falling edge of CLK and TRG low

AlignValue:
Value which the extractor syncs with to define bit 0 alignment.

SignalLevel:
Level, 0 or 1, which the extractor syncs with to define bit 0 aligment

AlignChannel:
Which signal the extractor uses for alignment, either via value or signal

BytesPerValue:
1, 2, or 4. Used for Value alignment size.

Returns:
1 – if Start was successful
0 – if Pod failed initialization

**StopExtraction** – Stops the extraction in progress

```
CWAV_EXPORT int CWAV_API StopExtraction( void );
```

Returns:
1 – always

**ExtractBufferOverflow** – Returns the state of the overflow conditions

```
CWAV_EXPORT char CWAV_API ExtractBufferOverflow(void);
```

Return:
0 – No overflow
1 – Overflow Occurred. ExtractorBuffer Overflow condition cleared.
2 – Overflow Occurred. Raw Stream Buffer Overflow

## 4.4.3 Extraction Data Format

The GetNextData routine gets a series of bytes that represent the extracted data stream and places these bytes into the buffer pointed to by the *buffer parameter.

The Serial Bus Extractor uses the following format for the data in this buffer:

Byte 0: Channel 0, first byte extracted
Byte 1: Channel 1, first byte extracted
Byte 2: Channel 2, first byte extracted
Byte 3: Channel 3, first byte extracted
Byte 4: Channel 4, first byte extracted
Byte 5: Channel 5, first byte extracted
Byte 6: Channel 6, first byte extracted
Byte 7: Channel 7, first byte extracted
Byte 8: Channel 0, second byte extracted
Byte 9: Channel 1, second byte extracted
...
Byte N: Channel (N mod 8), byte (N/8)+1 extracted

## 4.4.4 Example Source Code

```
//**************************************************
// USBee AX-Pro Data Extractor
// Serial Bus Extractor Example Program
// Copyright 2006, CWAV All Rights Reserved.
//**************************************************

#include "stdafx.h"
#include "stdio.h"
#include "conio.h"
#include "windows.h"
#include <fcntl.h>
#include <io.h>
#include <stdlib.h>
#include <stdio.h>

#define MAJOR_REV 1
#define MINOR_REV 0

//**************************************************
// Declare the Extractor DLL API routines
//**************************************************

#define CWAV_API __stdcall
#define CWAV_IMPORT __declspec(dllimport)

CWAV_IMPORT int CWAV_API StartExtraction( unsigned int SampleRate, unsigned long PodNumber, unsigned int ClockMode,
unsigned long AlignValue,
                                                  unsigned char SignalLevel, unsigned char AlignChannel, unsigned char
BytePerValue);
CWAV_IMPORT char CWAV_API GetNextData(unsigned char *buffer, unsigned long length);
CWAV_IMPORT int CWAV_API StopExtraction( void );
CWAV_IMPORT char CWAV_API ExtractBufferOverflow(void);
CWAV_IMPORT unsigned long CWAV_API ExtractionBufferCount(void);

//**************************************************
// Define the working buffer
//**************************************************

#define WORKING_BUFFER_SIZE   (65536*8)
unsigned char tempbuffer[WORKING_BUFFER_SIZE];

// Command Line Parameter Settings
unsigned long P_PodID = 0;
unsigned char O_OutputFilename[256] = {0};
unsigned char S_Screen = FALSE;
unsigned char BytePerValue = 1;
unsigned char Y_LeastSignificantByteFirst = FALSE;
unsigned char Z_MostSignificantByteFirst = TRUE;
unsigned char A_ASCIITextValues = FALSE;
unsigned char D_DecimalTextValues = FALSE;
unsigned char H_HexTextValues = TRUE;
unsigned char B_BinaryTextValues = FALSE;
unsigned char I_BinaryValues = FALSE;
unsigned char C_CommaDelimited = FALSE;
unsigned char G_SpaceDelimited = TRUE;
unsigned char N_NewlineDelimited = FALSE;
unsigned char X_NoDelimeter = FALSE;
unsigned long T_ForceBytesPerLine = 0;
unsigned long M_SignalMask = 0xFFFFFFFF;
unsigned char L_SignalLevel = 0;
unsigned long V_AlignValue = 0;
unsigned char E_ExternalClockMode = 2;
unsigned char J_ChannelAlign = 0;
unsigned char R_SampleRate = 17;
unsigned long Q_NumberOfBytes = 0;

void DisplayHelp(void)
{
     fprintf(stdout,"\nSerialExtractor [-?SADHBICGNX124YZ] [-Q NumberOfBytes] [-T BytesPesLine] [-R SampleRate] [-E
ClockingMode] [-M SignalMask] [-J ChannelAlign] [-L SignalLevel] [-V AlignmentValue] [-O filename] -P PodID\n\n");
     fprintf(stdout,"    ?  - Display this help screen\n");

     fprintf(stdout,"\n  USBee AX-Pro Pod to Use\n");
     fprintf(stdout,"    P  - Pod ID (required)\n");

     fprintf(stdout,"\n  Output Location Flags\n");
     fprintf(stdout,"    O  - Output to filename (default off)\n");
     fprintf(stdout,"    S  - Output to the screen (default off)\n");

     fprintf(stdout,"\n  When to Quit Flags\n");
     fprintf(stdout,"    Q  - Number of output values (default = until keypress)\n");

     fprintf(stdout,"\n  Input Number Format Flags\n");
     fprintf(stdout,"    1  - One   Byte  per value (default)\n");
     fprintf(stdout,"    2  - Two   Bytes per value\n");
     fprintf(stdout,"    4  - Four  Bytes per value\n");
     fprintf(stdout,"    Y  - Least significant byte first\n");
     fprintf(stdout,"    Z  - Most significant byte first\n");

     fprintf(stdout,"\n  Output Number Format Flags\n");
```

```
     fprintf(stdout,"    A  - ASCII Text Values (\"1\")\n");
     fprintf(stdout,"    D  - Decimal Text Values (\"49\")\n");
     fprintf(stdout,"    H  - Hex Text Values (\"31\") default\n");
     fprintf(stdout,"    B  - Binary Text Values (\"00110001\")\n");
     fprintf(stdout,"    I  - Binary Values (49)\n");
     fprintf(stdout,"    C  - Comma Delimited\n");
     fprintf(stdout,"    G  - Space Delimited (default)\n");
     fprintf(stdout,"    N  - Newline Delimited\n");
     fprintf(stdout,"    X  - No Delimeter\n");
     fprintf(stdout,"    T  - Force Bytes Per Line (no force default)\n");

     fprintf(stdout,"\n  Filter Values\n");
     fprintf(stdout,"    M  - Which Signals to capture (1=signal0,255=all(default))\n");

     fprintf(stdout,"\n  Clocking Modes\n");
     fprintf(stdout,"    E  - Clocking mode (2=internal (default),\n");
     fprintf(stdout,"                        4=CLK rising,5-CLK falling,\n");
     fprintf(stdout,"                        6-CLK rising AND TRG high,7-CLK falling AND TRG high\n");
     fprintf(stdout,"                        8-CLK rising AND TRG low,9-CLK falling AND TRG low\n");
     fprintf(stdout,"    R  - Internal CLK Sample Rate (1Msps default)\n");

     fprintf(stdout,"\n  Bit Zero Alignment Setting\n");
     fprintf(stdout,"    V  - Align on Value\n");
     fprintf(stdout,"    L  - Align on Signal Level (0=Low, 1=High)\n");
     fprintf(stdout,"    J  - Align on Which Channel (1=Ch 0, 128=Ch 7)\n");

     exit(0);
}

void Error(char *err)
{
     fprintf(stderr,"Error: ");
     fprintf(stderr,"%s\n",err);
     exit(2);
}


//*****************************************************
// Parse all of the command line options
//*****************************************************
void ParseCommandLine(int argc, char *argv[])
{
     BOOL cont;
     int       i,j;
     DWORD WordExample;
     BYTE ByteExample;

     for(i=1; i < argc; ++i)
     {
          if((argv[i][0] == '-') || (argv[i][0] == '/'))
          {
               cont = TRUE;
               for(j=1;argv[i][j] && cont;++j)     // Cont flag permits multiple commands in a single argv (like -AR)
                    switch(toupper(argv[i][j]))
                    {
                         case 'P':
                              P_PodID = (WORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'O':
                              strcpy((char*)O_OutputFilename, argv[++i]);
                              cont = FALSE;
                              break;
                         case '?':
                              DisplayHelp();
                              break;
                         case 'S':
                              S_Screen = TRUE;
                              break;
                         case '1':
                              BytePerValue = 1;
                              break;
                         case '2':
                              BytePerValue = 2;
                              break;
                         case '4':
                              BytePerValue = 4;
                              break;
                         case 'Y':
                              Y_LeastSignificantByteFirst = TRUE;
                              Z_MostSignificantByteFirst = FALSE;
                              break;
                         case 'Z':
                              Z_MostSignificantByteFirst = TRUE;
                              Y_LeastSignificantByteFirst = FALSE;
                              break;
                         case 'A':
                              A_ASCIITextValues = TRUE;
                              H_HexTextValues = FALSE;
                              break;
                         case 'D':
                              D_DecimalTextValues = TRUE;
                              H_HexTextValues = FALSE;
                              break;
                         case 'H':
```

```
                    H_HexTextValues = TRUE;
                    break;
               case 'B':
                    B_BinaryTextValues = TRUE;
                    H_HexTextValues = FALSE;
                    break;
               case 'I':
                    I_BinaryValues = TRUE;
                    H_HexTextValues = FALSE;
                    break;
               case 'C':
                    C_CommaDelimited = TRUE;
                    G_SpaceDelimited = FALSE;
                    break;
               case 'G':
                    G_SpaceDelimited = TRUE;
                    break;
               case 'N':
                    N_NewlineDelimited = TRUE;
                    G_SpaceDelimited = FALSE;
                    break;
               case 'X':
                    X_NoDelimeter = TRUE;
                    G_SpaceDelimited = FALSE;
                    break;
               case 'T':
                    T_ForceBytesPerLine = (DWORD)strtol(argv[++i],NULL,0);
                    cont = FALSE;
                    break;
               case 'Q':
                    Q_NumberOfBytes = (DWORD)strtol(argv[++i],NULL,0);
                    cont = FALSE;
                    break;
               case 'M':
                    M_SignalMask = (DWORD)strtol(argv[++i],NULL,0);
                    cont = FALSE;
                    break;
               case 'L':
                    L_SignalLevel = (BYTE)strtol(argv[++i],NULL,0);
                    cont = FALSE;
                    break;
               case 'V':
                    V_AlignValue = (DWORD)strtol(argv[++i],NULL,0);
                    cont = FALSE;
                    break;
               case 'E':
                    E_ExternalClockMode = (DWORD)strtol(argv[++i],NULL,0);
                    cont = FALSE;
                    break;
               case 'J':
                    J_ChannelAlign = (BYTE)strtol(argv[++i],NULL,0);
                    cont = FALSE;
                    break;
               case 'R':
                    R_SampleRate = (BYTE)strtol(argv[++i],NULL,0);
                    cont = FALSE;
                    break;
               case 'w':
                    WordExample = (DWORD)strtol(argv[++i],NULL,0);
                    cont = FALSE;
                    break;
               case 'b':
                    ByteExample = (BYTE)strtol(argv[++i],NULL,0);
                    cont = FALSE;
                    break;
               default:
                    DisplayHelp();
                    Error("Invalid Command Line Switch");
          }
     }
}


     // Now check to see if they make sense
     if (P_PodID == 0)
     {
          DisplayHelp();
          Error("No Pod Number Specified");
     }

}

unsigned long StartTime;

void StartTimer()
{

     StartTime = GetTickCount();
}

void StopTimer()
{

     printf(" \nTime Delta = %d\n",GetTickCount() - StartTime);
```

```
}

//*****************************************************
// Main Entry Point.  The program starts here.
//*****************************************************

int main(int argc, char* argv[])
{
     int RetValue;
     unsigned long totalbytes = 0;
     char *outputstr = new char [256];
     unsigned long ByteCounter = 0;
     unsigned long OutputValue;

     printf("USBee AX Data Extractor\n");
     printf("Serial Bus Extractor Version %d.%d\n", MAJOR_REV, MINOR_REV);

     // Parse out the command line options
     ParseCommandLine( argc, argv );

     //*****************************************************
     // Open up a file to store extracted data into
     //*****************************************************

     FILE *fout;
     if (O_OutputFilename[0])
     {
          if (I_BinaryValues)
               fout = fopen((char*)O_OutputFilename, "wb");
          else
               fout = fopen((char*)O_OutputFilename, "w");
     }

     //*****************************************************
     // Start the USBee AX Pod extracting the data we want
     //*****************************************************


     RetValue = StartExtraction( R_SampleRate, P_PodID, E_ExternalClockMode, V_AlignValue, L_SignalLevel,
J_ChannelAlign, BytePerValue);

     if (RetValue == 0)
     {
          printf("Startup failed.  Is the USBee AX-Pro connected and is the PodNumber correct?\n");
          printf("Press any key to continue...");
          getch();
          return(0);
     }

     printf("Processing and Saving Data to Disk.\n");

     //*****************************************************
     // Loop and do something with the collected data
     //*****************************************************

     int KeepLooping = TRUE;
     printf("BytePerValue = %d, M_SignalMask = %d\n",BytePerValue, M_SignalMask);
     while(KeepLooping)       // Do this forever until we tell it to stop by pressing a key
     {

          if (kbhit())
          {
               KeepLooping = FALSE;          // Stop the processing loop
               StopExtraction();             // Stop the streaming of data from the USBee
          }

          //*****************************************************
          // If there is data that has come in
          //*****************************************************
          int timeout = 0;
          while (unsigned long length = ExtractionBufferCount())
          {
               if (length > WORKING_BUFFER_SIZE)
                    length = WORKING_BUFFER_SIZE;

               //*****************************************************
               // Get the data into our local working buffer
               //*****************************************************
               StartTimer();

               GetNextData( tempbuffer, length );

               if (I_BinaryValues)      // Just write out the binary data to a file
               {
                    totalbytes += length;

                    if (O_OutputFilename[0])
                         fwrite(tempbuffer, length, 1,  fout);   // Write it to a file

                    if (Q_NumberOfBytes)
                    {
                         if (Q_NumberOfBytes <= length)
                         {
                              goto Done;          // Done with that many bytes
                         }
```

```
                    Q_NumberOfBytes -= length;
               }

          }
          else      // It's a text output so format it all pretty-like
          {

               // Now figure out what to send to the output
               for (unsigned long x = 0; x < length; x+=(8 * BytePerValue))         //Do multiple of 8 values at
a time becuase each one is a data line
               {
                    sprintf(outputstr, "\n%08X: ",x);
                    fputs(outputstr, fout);
                    //First, check which lines we want
                    for (unsigned char y = 0; y < 8; y++)
                    {
                         sprintf(outputstr, "%02X ",tempbuffer[x+y]);
                         fputs(outputstr, fout);

                         if (M_SignalMask & (2^y))     //Check mask value
                         {
                              // First get the value to print out
                              if (BytePerValue == 1)
                              {
                                   OutputValue = tempbuffer[x + y];
                              }
                              if (BytePerValue == 2)
                              {
                                   if (Y_LeastSignificantByteFirst)
                                        OutputValue = (tempbuffer[x+8+y] << 8) + tempbuffer[x+0+y];
                                   else
                                        OutputValue = (tempbuffer[x+0+y] << 8) + tempbuffer[x+8+y];
                              }
                              if (BytePerValue == 4)
                              {
                                   if (Y_LeastSignificantByteFirst)
                                        OutputValue =  (tempbuffer[x+32+y] << 24) +
                                                       (tempbuffer[x+16+y] << 16) +
                                                       (tempbuffer[x+8+y] << 8) +
                                                       tempbuffer[x+0+y];
                                   else
                                        OutputValue =  (tempbuffer[x+0+y] << 24) +
                                                       (tempbuffer[x+8+y] << 16) +
                                                       (tempbuffer[x+16+y] << 8) +
                                                       tempbuffer[x+32+y];

                              }

                              // Now convert the value into the output text
                              if (A_ASCIITextValues)
                              {
                                   outputstr[0] = (unsigned char)OutputValue;
                                   outputstr[1] = 0;
                              }
                              if (D_DecimalTextValues)
                              {
                                   ultoa(OutputValue,outputstr,10);
                                   // sprintf(outputstr,"%d",OutputValue);
                              }
                              if (B_BinaryTextValues)
                              {
                                   int count;

                                   if (BytePerValue == 1)
                                        count = 8;
                                   if (BytePerValue == 2)
                                        count = 16;
                                   if (BytePerValue == 4)
                                        count = 32;

                                   unsigned int mask = 1 << (count - 1);
                                   for (int z = 0; z < count; z++)
                                   {
                                        if (OutputValue & mask)
                                             outputstr[z] = '1';
                                        else
                                             outputstr[z] = '0';
                                        mask /= 2;
                                   }
                              }
                              if (H_HexTextValues)
                              {
                                   if (BytePerValue == 1)
                                        ultoa(OutputValue, outputstr, 16);
                                        //sprintf(outputstr,"%02X", OutputValue);
                                   if (BytePerValue == 2)
                                        ultoa(OutputValue, outputstr, 16);
                                        //sprintf(outputstr,"%04X", OutputValue);
                                   if (BytePerValue == 4)
                                        ultoa(OutputValue, outputstr, 16);
                                        //sprintf(outputstr,"%08X", OutputValue);
                              }

                              // Now add any delimeters to the end of the value
                              if (C_CommaDelimited)
```

```
                                        strcat(outputstr, ",");

                                    if (G_SpaceDelimited)
                                        strcat(outputstr, " ");

                                    if (N_NewlineDelimited)
                                        strcat(outputstr, "\n");

                                    if (T_ForceBytesPerLine)
                                    {
                                        if (++ByteCounter >= T_ForceBytesPerLine)
                                        {
                                            ByteCounter = 0;
                                            strcat(outputstr, "\n");
                                        }
                                    }

                                    if (S_Screen)
                                        fputs(outputstr, stdout);

                                    if (O_OutputFilename[0])
                                        fputs(outputstr, fout);

                                    totalbytes++;

                                    if (Q_NumberOfBytes)
                                        if (--Q_NumberOfBytes == 0)
                                        {
                                            goto Done;          // Done with that many bytes
                                        }
                                }
                            }
                        }

                    }

                    // StopTimer();

                    if (timeout++ > 10  ) break;  // Let up once in a while to let the OS process
                }

                if (!S_Screen)
                    printf("\rProcessed %d output values.", totalbytes);

                //*****************************************************
                // Check to see if we have fallen behind too far
                //*****************************************************

                int y = ExtractBufferOverflow();

                if (y == 1)
                {
                    printf("\nExtractor Buffer Overflow.\nYour data is streaming too fast for your output settings.\nLower
your data rate or change to output binary files.\n");
                    goto Done;
                }
                else if (y == 2)
                {
                    printf("\nRaw Sample Buffer Overflow.\nYour data is streaming too fast for your output settings.\nLower
your data rate or change to output binary files.\n");
                    goto Done;
                }

                //*****************************************************
                // Give the OS a little time to do something else
                //*****************************************************

                Sleep(15);

            }

Done:
            if (!S_Screen)
                printf("\rProcessed %d output values.", totalbytes);

            //*****************************************************
            // Close the file
            //*****************************************************

            if (O_OutputFilename[0])
                fclose(fout);

            //*****************************************************
            // Stop the extraction process
            //*****************************************************

            StopExtraction();

            if (kbhit()) getch();
            printf("\nPress any key to continue...");
            getch();

            return 0;
}
```

# 5 I2C Data Extractor

The I²C Bus Data Extractor takes the real-time streaming data from the I2C bus, formats it and allows you to save the data to disk or process it as it arrives.

## *5.1 I2C Data Extractor Specifications*

- Continuous Real-Time Data Streaming
- Monitors one I²C Bus
- TTL Level inputs (**0-5V max**, Vih = 2.0V, Vil = 0.8V)
- Time Stamp for each packet
- Output to Text File*
- Output to Screen*
- Comma or Space Delimited files
- Output File Viewer (including binary, text, search and export functions)
- Extractor API libraries interface directly to your own software to further process the extracted data. Any language that supports calls to DLLs is supported.

* - output bandwidths are dependent on PC USB hardware, hard disk and/or screen throughput.

## *5.2 Hardware Setup*

To use the Data Extractor you need to connect the USBee AX-Pro Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

***Please note that the USBee AX-Pro Test Pod inputs are strictly 0-5V levels. Any voltage outside this range on the signals will damage the pod and may damage your hardware. If your system uses different voltage levels, you must buffer the signals externally to the USBee AX-Pro Test Pod before connecting the signals to the unit.***

The I²C Bus Data Extractor connects to the SDA and SCL lines of the I²C bus. Use one signal as the SDA data line and one signal as the SCL clock line. Also connect the GND line to the digital ground of your system. Connect these signals to the I²C bus using the test clips provided.

## *5.3 Extractor Command Line Program*

The I²C Bus Data Extractor includes a Windows Command Prompt executable that lets you operate the Data Extractor without writing any software. The program is executed in a Command Prompt window and is configured using command line arguments. The extracted data is then stored to disk or outputted to the screen depending on these parameters.

To run the Data Extractor:

1) Install the USBee AX-Pro software on your PC
2) Install the Data Extractor software on your PC
3) Plug in your USBee AX-Pro Test Pod into your PC using a USB 2.0 High Speed Port
4) Open a Windows Command Prompt window by clicking Start, All Programs, Accessories, Command Prompt.
5) Change the working directory to the Data Extractor directory
6) (“cd \program files\USBee Data Extractor\I2C”)
7) Run the executable using the following command line arguments:

```
I2CExtractor [-?SDHICGAB] [-Q NumberOfBytes] [-V Timestamp] [-O
filename] [-M SDA] [-N SCL] -P PodID
```

? - Display this help screen

**USBee AX-Pro Pod to Use**

P - Pod ID (required)

**Output Location Flags**

O - Output to filename (default off)
S - Output to the screen (default off)

**When to Quit Flags**

Q - Number of output values (default = until keypress)

**Input Format Flags**

M - SDA signal Mask (1-Ch0, 128=Ch7, Ch0 default)
N - SCL signal Mask (1-Ch0, 128=Ch7, Ch1 default)

**Output Format Flags**

A - All Packet Fields are output (default)
B – Only Data Bytes are output
D - Decimal Text Values ("49")
H - Hex Text Values ("31") default
I - Binary Values (49)
C - Comma Delimited
G - Space Delimited (default)

**Timestamps**

V - Timestamps (0=off, 1=each packet start)

## *5.4 Extractor API*

The Data Extractor is implemented using a Windows DLL that interfaces to the existing USBee AX-Pro DLL and drivers. This DLL can be called using any software language that supports calls to DLLs. Below are the details of this DLL interface and the routines that are available for your use.

### 5.4.1 DLL filename:

```
usbexI2C.dll in \Windows\System32
```

### 5.4.2 DLL Exported Functions and parameters

**ExtractionBufferCount** – Returns the number of bytes that have been extracted from the data stream so far and are available to read using GetNextData.

```
CWAV_EXPORT unsigned long CWAV_API ExtractionBufferCount(void)
```

Returns:
0 – No data to read yet
other – number of bytes available to read

**GetNextData** – Copies the extracted data from the extractor into your working buffer

```
CWAV_EXPORT char CWAV_API GetNextData(unsigned char *buffer, unsigned
long length);
```

buffer:
pointer to where you want the extracted data to be placed
length:
number of bytes you want to read from the extraction DLL
Returns:
0 – No data to read yet
1 – Data was copied into the buffer

**StartExtraction** – Starts the Data Extraction with the given parameters.

```
CWAV_EXPORT int CWAV_API StartExtraction(unsigned long PodNumber,
unsigned char All, unsigned char Decimal, unsigned char Hex, unsigned
char Binary, unsigned char Comma, unsigned char Space, unsigned char
Timestamps, unsigned long SDAMask, unsigned long SCLMask)
```

PodNumber:
Pod ID on the back of the USBee AX-Pro Test Pod

All:
0 – Only the data payload bytes are returned
1 – All I2C packet fields are returned
Decimal:
1 – Decimal Values (text) are output for the data bytes
Hex:
1 – Hex Values (text) are output for the data bytes
Binary:
1 – All data is in binary form, not text
Comma:
1 – Commas are placed between each field/data byte
Space:
1 – Spaces are placed between each field/data byte
Timestamp:
1 – Print Timestamps at the start of each packet
SDAMask:
The mask for the channel to use for SDA
(1 = Ch0, 128 = Ch7)
SCLMask:
The mask for the channel to use for SCL
(1 = Ch0, 128 = Ch7)
Returns:
1 – if Start was successful
0 – if Pod failed initialization

**StopExtraction** – Stops the extraction in progress

```
CWAV_EXPORT int CWAV_API StopExtraction( void );
```

Returns:
1 – always

**ExtractBufferOverflow** – Returns the state of the overflow conditions

```
CWAV_EXPORT char CWAV_API ExtractBufferOverflow(void);
```

Return:

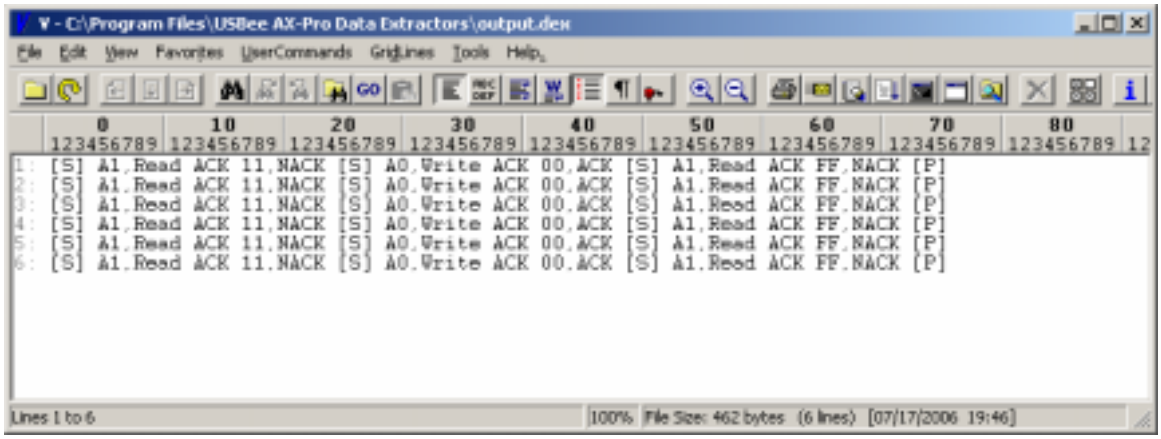
0 – No overflow
1 – Overflow Occurred. ExtractorBuffer Overflow condition cleared.
2 – Overflow Occurred. Raw Stream Buffer Overflow

## 5.4.3 Extraction Data Format

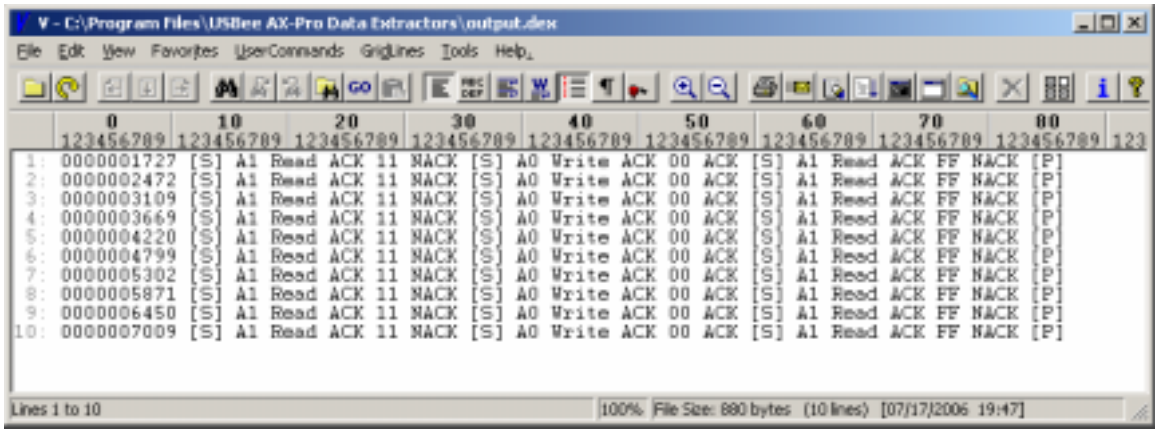
The GetNextData routine gets a series of bytes that represent the extracted data stream and places these bytes into the buffer pointed to by the *buffer parameter.

The I²C Bus Extractor DLL sends the extracted data through the *buffer in the requested form based on the parameters in the StartExtraction call. For example, if Binary is set to a 0, then the *buffer will receive the binary bytes that make up the data stream. If Hex is set to a 1, the *buffer will contain a text string which is the data of the I2C traffic in Hex text form, separated by any specified delimiters.

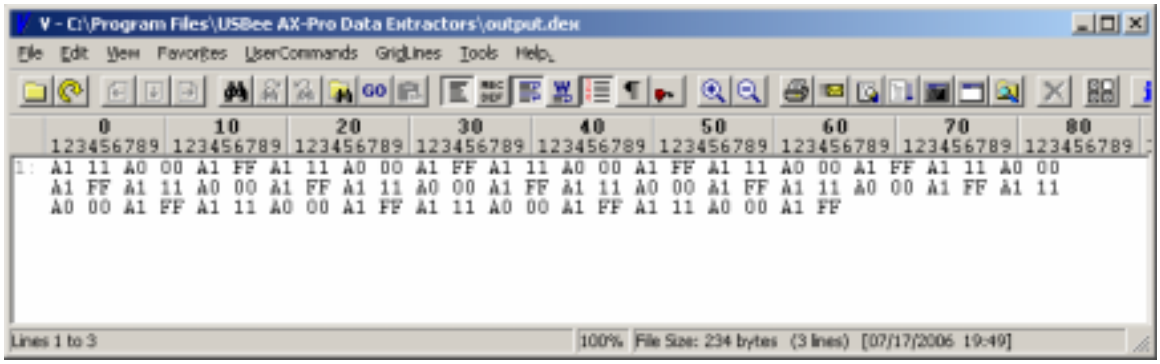
```
I2CExtractor -O output.dex -P 3209 -Q 5000 -H -C -M 2 -N 1 -V 0
```



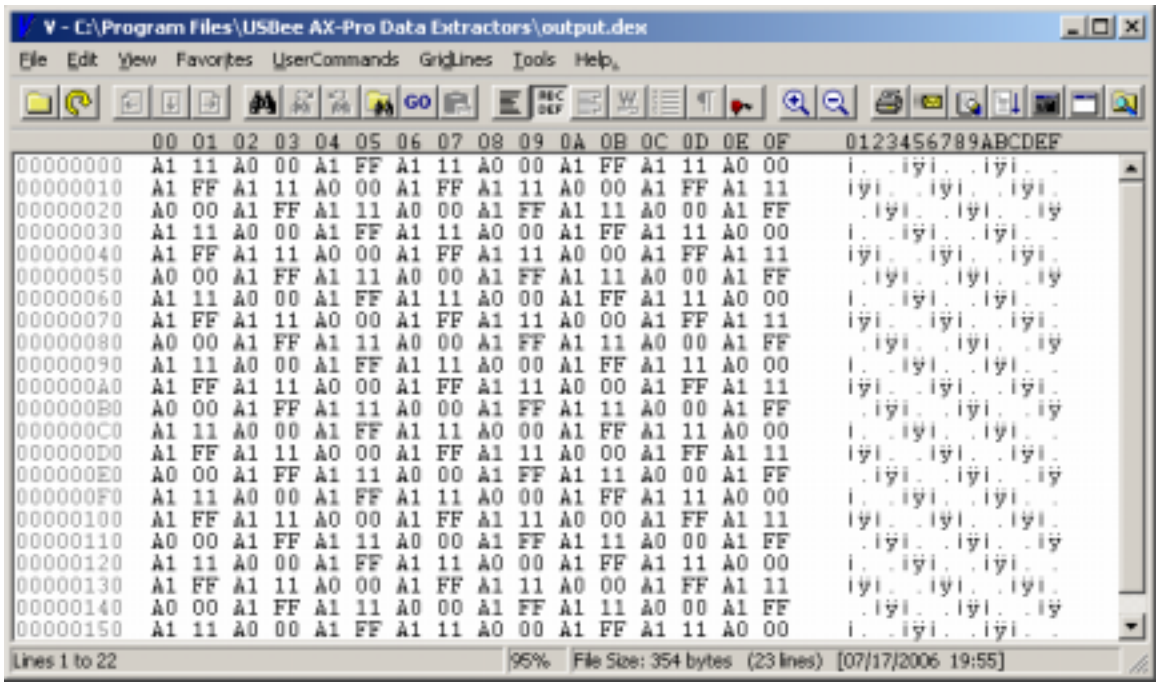

```
I2CExtractor -O output.dex -P 3209 -Q 5000 -H -G -M 2 -N 1 -V 1
```



```
I2CExtractor -O output.dex -P 3209 -Q 5000 -B -M 2 -N 1
```




```
I2CExtractor -O output.dex -P 3209 -Q 5000 -I -M 2 -N 1
```



## 5.4.4 Example Source Code

```
//**************************************************
// USBee AX-Pro Data Extractor
// I2C Bus Extractor Example Program
// Copyright 2006, CWAV All Rights Reserved.
//**************************************************

#include "stdafx.h"
#include "stdio.h"
#include "conio.h"
#include "windows.h"
#include <fcntl.h>
#include <io.h>
#include <stdlib.h>
#include <stdio.h>

#define MAJOR_REV 1
#define MINOR_REV 0

//**************************************************
// Declare the Extractor DLL API routines
//**************************************************

#define CWAV_API __stdcall
#define CWAV_IMPORT __declspec(dllimport)

CWAV_IMPORT int CWAV_API StartExtraction(unsigned long PodNumber, unsigned char All, unsigned char Decimal, unsigned
char Hex, unsigned char Binary, unsigned char Comma, unsigned char Space, unsigned char Timestamps, unsigned long
SDA,unsigned long SCL);
CWAV_IMPORT char CWAV_API GetNextData(unsigned char *buffer, unsigned long length);
CWAV_IMPORT int CWAV_API StopExtraction( void );
CWAV_IMPORT char CWAV_API ExtractBufferOverflow(void);
CWAV_IMPORT unsigned long CWAV_API ExtractionBufferCount(void);

//**************************************************
// Define the working buffer
//**************************************************

#define WORKING_BUFFER_SIZE   (65536*8)
unsigned char tempbuffer[WORKING_BUFFER_SIZE];

// Command Line Parameter Settings
unsigned long P_PodID = 0;
unsigned char O_OutputFilename[256] = {0};
unsigned char S_Screen = FALSE;
unsigned char A_All = TRUE;
unsigned char B_DataOnly = FALSE;
unsigned char D_DecimalTextValues = FALSE;
unsigned char H_HexTextValues = TRUE;
unsigned char I_BinaryValues = FALSE;
unsigned char C_CommaDelimited = FALSE;
unsigned char G_SpaceDelimited = FALSE;
unsigned long Q_NumberOfBytes = 0;
unsigned long V_Timestamps = TRUE;
unsigned long M_SDA = 1;
unsigned long N_SCL = 2;


void DisplayHelp(void)
{
     fprintf(stdout,"\nI2CExtractor [-?SDHICGAB] [-Q NumberOfBytes] [-V Timestamp] [-O filename] [-M SDAMask] [-N
SCLMask] -P PodID\n");

     fprintf(stdout,"\n    ?  - Display this help screen\n");

     fprintf(stdout,"\n  USBee AX-Pro Pod to Use\n");

     fprintf(stdout,"    P  - Pod ID (required)\n");

     fprintf(stdout,"\n  Output Location Flags\n");

     fprintf(stdout,"    O  - Output to filename (default off)\n");
     fprintf(stdout,"    S  - Output to the screen (default off)\n");

     fprintf(stdout,"\n  When to Quit Flags\n");

     fprintf(stdout,"    Q  - Number of output values (default = until keypress)\n");

     fprintf(stdout,"\n  Input Format Flags\n");

     fprintf(stdout,"    R  - Bus Speed in bits/second (default = 250000)\n");

     fprintf(stdout,"\n  Output Number Format Flags\n");

     fprintf(stdout,"    A  - All Packet Fields are output (default)\n");
     fprintf(stdout,"    B  - Only data bytes are output\n");
     fprintf(stdout,"    D  - Decimal Text Values (\"49\")\n");
     fprintf(stdout,"    H  - Hex Text Values (\"31\") default\n");
     fprintf(stdout,"    I  - Binary Values (49)\n");
     fprintf(stdout,"    C  - Comma Delimited\n");
     fprintf(stdout,"    G  - Space Delimited (default)\n");
```

```
     fprintf(stdout,"    V  - Timestamps (0=off(default),1=Timestamp on\n");
     fprintf(stdout,"    M  - SDA signal (1=ch0, 128=ch7, ch0 default)\n");
     fprintf(stdout,"    N  - SCL signal (1=ch0, 128=ch7, ch1 default)\n");


}

void Error(char *err)
{
     fprintf(stderr,"Error: ");
     fprintf(stderr,"%s\n",err);
     exit(2);
}


//*************************************************
// Parse all of the command line options
//*************************************************
void ParseCommandLine(int argc, char *argv[])
{
     BOOL cont;
     int       i,j;
     DWORD WordExample;
     BYTE ByteExample;

     for(i=1; i < argc; ++i)
     {
          if((argv[i][0] == '-') || (argv[i][0] == '/'))
          {
               cont = TRUE;
               for(j=1;argv[i][j] && cont;++j)     // Cont flag permits multiple commands in a single argv (like -AR)
                    switch(toupper(argv[i][j]))
                    {
                         case 'P':
                              P_PodID = (WORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'O':
                              strcpy((char*)O_OutputFilename, argv[++i]);
                              cont = FALSE;
                              break;
                         case '?':
                              DisplayHelp();
                              exit(0);

                              break;
                         case 'S':
                              S_Screen = TRUE;
                              break;
                         case 'A':
                              A_All = TRUE;
                              B_DataOnly = FALSE;
                              break;
                         case 'B':
                              A_All = FALSE;
                              B_DataOnly = TRUE;
                              break;
                         case 'D':
                              D_DecimalTextValues = TRUE;
                              H_HexTextValues = FALSE;
                              break;
                         case 'H':
                              H_HexTextValues = TRUE;
                              break;
                         case 'I':
                              I_BinaryValues = TRUE;
                              H_HexTextValues = FALSE;
                              break;
                         case 'C':
                              C_CommaDelimited = TRUE;
                              G_SpaceDelimited = FALSE;
                              break;
                         case 'G':
                              G_SpaceDelimited = TRUE;
                              break;
                         case 'Q':
                              Q_NumberOfBytes = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'V':
                              V_Timestamps = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'M':
                              M_SDA = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'N':
                              N_SCL = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'w':
                              WordExample = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
```

```
                         break;
                    case 'b':
                         ByteExample = (BYTE)strtol(argv[++i],NULL,0);
                         cont = FALSE;
                         break;
                    default:
                         DisplayHelp();
                         fprintf(stdout,"\nCommand line switch %c not recognized\n",toupper(argv[i][j]));
                         Error("Invalid Command Line Switch");
                         exit(0);
               }
          }
     }


     // Now check to see if they make sense
     if (P_PodID == 0)
     {
          DisplayHelp();
          Error("No Pod Number Specified");
     }

}


//**************************************************
// Main Entry Point.  The program starts here.
//**************************************************

int main(int argc, char* argv[])
{
     int RetValue;
     unsigned long totalbytes = 0;
     char *outputstr = new char [256];
     unsigned long ByteCounter = 0;
     unsigned long OutputValue;

     printf("USBee AX Data Extractor\n");
     printf("I2C Bus Extractor Version %d.%d\n", MAJOR_REV, MINOR_REV);

     // Parse out the command line options
     ParseCommandLine( argc, argv );

     //**************************************************
     // Open up a file to store extracted data into
     //**************************************************

     FILE *fout;
     if (O_OutputFilename[0])
     {
          if (I_BinaryValues)
               fout = fopen((char*)O_OutputFilename, "wb");
          else
               fout = fopen((char*)O_OutputFilename, "w");
     }

     //**************************************************
     // Start the USBee AX Pod extracting the data we want
     //**************************************************

     int Endpoint = 999;
     int Device = 999;

     RetValue = StartExtraction(P_PodID, A_All, D_DecimalTextValues, H_HexTextValues, I_BinaryValues, C_CommaDelimited,
G_SpaceDelimited, V_Timestamps, M_SDA, N_SCL)     ;

     if (RetValue == 0)
     {
          printf("Startup failed.  Is the USBee AX-Pro connected and is the PodNumber correct?\n");
          printf("Press any key to continue...");
          getch();
          return(0);
     }


     //**************************************************
     // Loop and do something with the collected data
     //**************************************************

     char OldSignal = 99;

     int KeepLooping = TRUE;
     while(KeepLooping)       // Do this forever until we tell it to stop by pressing a key
     {

          if (kbhit())
          {
               KeepLooping = FALSE;          // Stop the processing loop
               StopExtraction();             // Stop the streaming of data from the USBee
          }

          //**************************************************
          // If there is data that has come in
          //**************************************************
          int timeout = 0;
```

```
          while (unsigned long length = ExtractionBufferCount())
          {
               if (length > WORKING_BUFFER_SIZE)
                    length = WORKING_BUFFER_SIZE;

               //*****************************************************
               // Get the data into our local working buffer
               //*****************************************************

               GetNextData( tempbuffer, length );

               totalbytes += length;

               if (O_OutputFilename[0])
                    fwrite(tempbuffer, length, 1,  fout);   // Write it to a file

               if (S_Screen)
                    fwrite(tempbuffer, length, 1,  stdout); // Write it to the screen

               if (Q_NumberOfBytes)
               {
                    if (Q_NumberOfBytes <= length)
                    {
                         goto Done;          // Done with that many bytes
                    }
                    Q_NumberOfBytes -= length;
               }

               if (timeout++ > 3  ) break;  // Let up once in a while to let the OS process
          }

          if (!S_Screen)
               printf("\rProcessed %d output values.", totalbytes);

          //*****************************************************
          // Check to see if we have fallen behind too far
          //*****************************************************

          int y = ExtractBufferOverflow();

          if (y == 1)
          {
               printf("\nExtractor Buffer Overflow.\nYour data is streaming too fast for your output settings.\nLower
your data rate or change to output binary files.\n");
               goto Done;
          }
          else if (y == 2)
          {
               printf("\nRaw Sample Buffer Overflow.\nYour data is streaming too fast for your output settings.\nLower
your data rate or change to output binary files.\n");
               goto Done;
          }

          //*****************************************************
          // Give the OS a little time to do something else
          //*****************************************************

          Sleep(15);

     }

Done:
     if (!S_Screen)
          printf("\rProcessed %d output values.", totalbytes);

     //*****************************************************
     // Close the file
     //*****************************************************

     if (O_OutputFilename[0])
          fclose(fout);

     //*****************************************************
     // Stop the extraction process
     //*****************************************************

     StopExtraction();

     if (kbhit()) getch();
     printf("\nPress any key to continue...");
     getch();

     return 0;
}
```

# 6 SM Bus Data Extractor

The SM Bus Data Extractor takes the real-time streaming data from the SM bus, formats it and allows you to save the data to disk or process it as it arrives.

## *6.1 SM Bus Data Extractor Specifications*

- Continuous Real-Time Data Streaming
- Monitors one SM Bus
- TTL Level inputs (**0-5V max**, Vih = 2.0V, Vil = 0.8V)
- Time Stamp for each packet
- Output to Text File*
- Output to Screen*
- Comma or Space Delimited files
- Output File Viewer (including binary, text, search and export functions)
- Extractor API libraries interface directly to your own software to further process the extracted data. Any language that supports calls to DLLs is supported.

* - output bandwidths are dependent on PC USB hardware, hard disk and/or screen throughput.

## *6.2 Hardware Setup*

To use the Data Extractor you need to connect the USBee AX-Pro Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

***Please note that the USBee AX-Pro Test Pod inputs are strictly 0-5V levels. Any voltage outside this range on the signals will damage the pod and may damage your hardware. If your system uses different voltage levels, you must buffer the signals externally to the USBee AX-Pro Test Pod before connecting the signals to the unit.***

The SM Bus Data Extractor connects to the SMBClk and SMBData lines of the SM Bus. Use one signal as the SMBData line and one signal as the SMBClk line. Also connect the GND line to the digital ground of your system. Connect these signals to the SM Bus using the test clips provided.

## *6.3 Extractor Command Line Program*

The SM Bus Data Extractor includes a Windows Command Prompt executable that lets you operate the Data Extractor without writing any software. The program is executed in a Command Prompt window and is configured using command line arguments. The extracted data is then stored to disk or outputted to the screen depending on these parameters.

To run the Data Extractor:

1) Install the USBee AX-Pro software on your PC
2) Install the Data Extractor software on your PC
3) Plug in your USBee AX-Pro Test Pod into your PC using a USB 2.0 High Speed Port
4) Open a Windows Command Prompt window by clicking Start, All Programs, Accessories, Command Prompt.
5) Change the working directory to the Data Extractor directory
6) (“cd \program files\USBee Data Extractor\SMBus”)
7) Run the executable using the following command line arguments:

```
SMBusExtractor [-?SDHICGAB] [-Q NumberOfBytes] [-V Timestamp] [-O
filename] [-M SMBDatMask] [-N SMBClkMask] -P PodID
```

? - Display this help screen

**USBee AX-Pro Pod to Use**

P - Pod ID (required)

**Output Location Flags**

O - Output to filename (default off)
S - Output to the screen (default off)

**When to Quit Flags**

Q - Number of output values (default = until keypress)

**Input Format Flags**

M - SMBData signal Mask (1-Ch0, 128=Ch7, Ch0 default)
N - SMBClk signal Mask (1-Ch0, 128=Ch7, Ch1 default)

**Output Format Flags**

A - All Packet Fields are output (default)
B – Only Data Bytes are output
D - Decimal Text Values ("49")
H - Hex Text Values ("31") default
I - Binary Values (49)
C - Comma Delimited
G - Space Delimited (default)

**Timestamps**

V - Timestamps (0=off, 1=each packet start)

## *6.4 Extractor API*

The Data Extractor is implemented using a Windows DLL that interfaces to the existing USBee AX-Pro DLL and drivers. This DLL can be called using any software language that supports calls to DLLs. Below are the details of this DLL interface and the routines that are available for your use.

### 6.4.1 DLL filename:

`usbexSMBus.dll in \Windows\System32`

### 6.4.2 DLL Exported Functions and parameters

**ExtractionBufferCount** – Returns the number of bytes that have been extracted from the data stream so far and are available to read using GetNextData.

```
CWAV_EXPORT unsigned long CWAV_API ExtractionBufferCount(void)
```

Returns:
0 – No data to read yet
other – number of bytes available to read

**GetNextData** – Copies the extracted data from the extractor into your working buffer

```
CWAV_EXPORT char CWAV_API GetNextData(unsigned char *buffer, unsigned
long length);
```

buffer:
pointer to where you want the extracted data to be placed
length:
number of bytes you want to read from the extraction DLL
Returns:
0 – No data to read yet
1 – Data was copied into the buffer

**StartExtraction** – Starts the Data Extraction with the given parameters.

```
CWAV_EXPORT int CWAV_API StartExtraction(unsigned long PodNumber,
unsigned char All, unsigned char Decimal, unsigned char Hex, unsigned
char Binary, unsigned char Comma, unsigned char Space, unsigned char
Timestamps, unsigned long SMBData, unsigned long SMBClk);
```

PodNumber:
Pod ID on the back of the USBee AX-Pro Test Pod
All:
0 – Only the data payload bytes are returned
1 – All SMBus packet fields are returned
Decimal:
1 – Decimal Values (text) are output for the data bytes
Hex:
1 – Hex Values (text) are output for the data bytes
Binary:
1 – All data is in binary form, not text
Comma:
1 – Commas are placed between each field/data byte
Space:
1 – Spaces are placed between each field/data byte
Timestamp:
1 – Print Timestamps at the start of each packet
SMBData:
The mask for the channel to use for Data
(1 = Ch0, 128 = Ch7)
SMDClk:
The mask for the channel to use for Clk
(1 = Ch0, 128 = Ch7)
Returns:
1 – if Start was successful
0 – if Pod failed initialization

**StopExtraction** – Stops the extraction in progress

```
CWAV_EXPORT int CWAV_API StopExtraction( void );
```

Returns:
1 – always

**ExtractBufferOverflow** – Returns the state of the overflow conditions

```
CWAV_EXPORT char CWAV_API ExtractBufferOverflow(void);
```

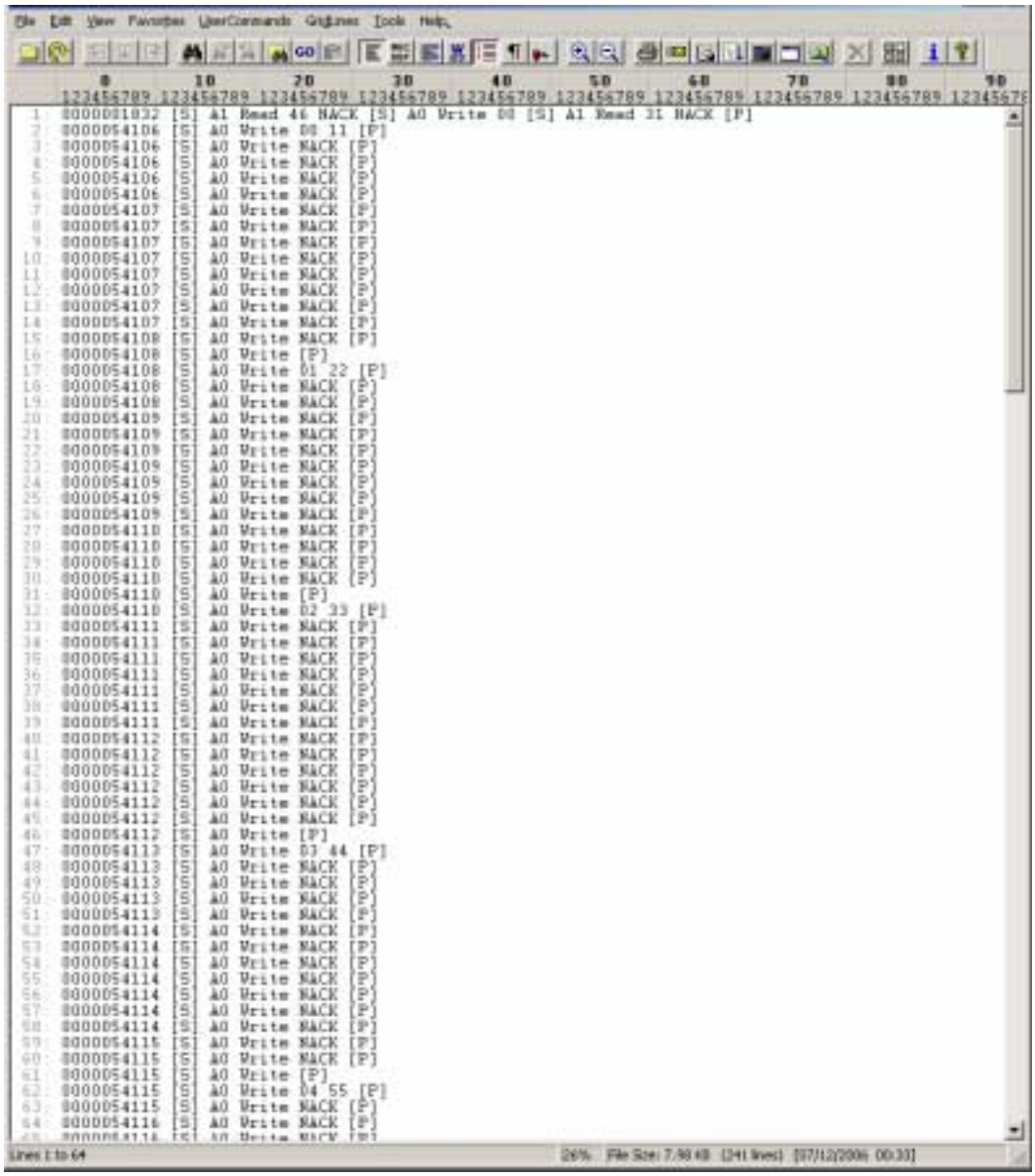
Return:

0 – No overflow
1 – Overflow Occurred. ExtractorBuffer Overflow condition cleared.
2 – Overflow Occurred. Raw Stream Buffer Overflow

## 6.4.3 Extraction Data Format

The GetNextData routine gets a series of bytes that represent the extracted data stream and places these bytes into the buffer pointed to by the *buffer parameter.

The SM Bus Extractor DLL sends the extracted data through the *buffer in the requested form based on the parameters in the StartExtraction call. For example, if Binary is set to a 0, then the *buffer will receive the binary bytes that make up the data stream. If Hex is set to a 1, the *buffer will contain a text string which is the data of the SMBus traffic in Hex text form, separated by any specified delimiters.



## 6.4.4 Example Source Code

```
//*****************************************************
// USBee AX-Pro Data Extractor
// SMBus Extractor Example Program
// Copyright 2006, CWAV All Rights Reserved.
//*****************************************************

#include "stdafx.h"
#include "stdio.h"
#include "conio.h"
#include "windows.h"
#include <fcntl.h>
#include <io.h>
#include <stdlib.h>
#include <stdio.h>

#define MAJOR_REV 1
#define MINOR_REV 0

//*****************************************************
// Declare the Extractor DLL API routines
//*****************************************************

#define CWAV_API __stdcall
#define CWAV_IMPORT __declspec(dllimport)

CWAV_IMPORT int CWAV_API StartExtraction(unsigned long PodNumber, unsigned char All, unsigned char Decimal, unsigned
char Hex, unsigned char Binary, unsigned char Comma, unsigned char Space, unsigned char Timestamps, unsigned long
SMBData,unsigned long SMBClk);
CWAV_IMPORT char CWAV_API GetNextData(unsigned char *buffer, unsigned long length);
CWAV_IMPORT int CWAV_API StopExtraction( void );
CWAV_IMPORT char CWAV_API ExtractBufferOverflow(void);
CWAV_IMPORT unsigned long CWAV_API ExtractionBufferCount(void);

//*****************************************************
// Define the working buffer
//*****************************************************

#define WORKING_BUFFER_SIZE   (65536*8)
unsigned char tempbuffer[WORKING_BUFFER_SIZE];

// Command Line Parameter Settings
unsigned long P_PodID = 0;
unsigned char O_OutputFilename[256] = {0};
unsigned char S_Screen = FALSE;
unsigned char A_All = TRUE;
unsigned char B_DataOnly = FALSE;
unsigned char D_DecimalTextValues = FALSE;
unsigned char H_HexTextValues = TRUE;
unsigned char I_BinaryValues = FALSE;
unsigned char C_CommaDelimited = FALSE;
unsigned char G_SpaceDelimited = FALSE;
unsigned long Q_NumberOfBytes = 0;
unsigned long V_Timestamps = TRUE;
unsigned long M_SDA = 1;
unsigned long N_SCL = 2;


void DisplayHelp(void)
{
     fprintf(stdout,"\nSMBusExtractor [-?SDHICGAB] [-Q NumberOfBytes] [-V Timestamp] [-O filename] [-M SMBDatMask] [-N
SMBClkMask] -P PodID\n");

     fprintf(stdout,"\n    ?  - Display this help screen\n");

     fprintf(stdout,"\n  USBee AX-Pro Pod to Use\n");

     fprintf(stdout,"    P  - Pod ID (required)\n");

     fprintf(stdout,"\n  Output Location Flags\n");

     fprintf(stdout,"    O  - Output to filename (default off)\n");
     fprintf(stdout,"    S  - Output to the screen (default off)\n");

     fprintf(stdout,"\n  When to Quit Flags\n");

     fprintf(stdout,"    Q  - Number of output values (default = until keypress)\n");

     fprintf(stdout,"\n  Output Number Format Flags\n");

     fprintf(stdout,"    A  - All Packet Fields are output (default)\n");
     fprintf(stdout,"    B  - Only data bytes are output\n");
     fprintf(stdout,"    D  - Decimal Text Values (\"49\")\n");
     fprintf(stdout,"    H  - Hex Text Values (\"31\") default\n");
     fprintf(stdout,"    I  - Binary Values (49)\n");
     fprintf(stdout,"    C  - Comma Delimited\n");
     fprintf(stdout,"    G  - Space Delimited (default)\n");
     fprintf(stdout,"    V  - Timestamps (0=off(default),1=Timestamp on\n");
     fprintf(stdout,"    M  - SMBData signal (1=ch0, 128=ch7, ch0 default)\n");
     fprintf(stdout,"    N  - SMBClk signal (1=ch0, 128=ch7, ch1 default)\n");
```

```
}

void Error(char *err)
{
     fprintf(stderr,"Error: ");
     fprintf(stderr,"%s\n",err);
     exit(2);
}


//*************************************************
// Parse all of the command line options
//*************************************************
void ParseCommandLine(int argc, char *argv[])
{
     BOOL cont;
     int       i,j;
     DWORD WordExample;
     BYTE ByteExample;

     for(i=1; i < argc; ++i)
     {
          if((argv[i][0] == '-') || (argv[i][0] == '/'))
          {
               cont = TRUE;
               for(j=1;argv[i][j] && cont;++j)     // Cont flag permits multiple commands in a single argv (like -AR)
                    switch(toupper(argv[i][j]))
                    {
                         case 'P':
                              P_PodID = (WORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'O':
                              strcpy((char*)O_OutputFilename, argv[++i]);
                              cont = FALSE;
                              break;
                         case '?':
                              DisplayHelp();
                              exit(0);
                              break;
                         case 'S':
                              S_Screen = TRUE;
                              break;
                         case 'A':
                              A_All = TRUE;
                              B_DataOnly = FALSE;
                              break;
                         case 'B':
                              A_All = FALSE;
                              B_DataOnly = TRUE;
                              break;
                         case 'D':
                              D_DecimalTextValues = TRUE;
                              H_HexTextValues = FALSE;
                              break;
                         case 'H':
                              H_HexTextValues = TRUE;
                              break;
                         case 'I':
                              I_BinaryValues = TRUE;
                              H_HexTextValues = FALSE;
                              break;
                         case 'C':
                              C_CommaDelimited = TRUE;
                              G_SpaceDelimited = FALSE;
                              break;
                         case 'G':
                              G_SpaceDelimited = TRUE;
                              break;
                         case 'Q':
                              Q_NumberOfBytes = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'V':
                              V_Timestamps = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'M':
                              M_SDA = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'N':
                              N_SCL = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'w':
                              WordExample = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'b':
                              ByteExample = (BYTE)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         default:
                              DisplayHelp();
```

```
                              fprintf(stdout,"\nCommand line switch %c not recognized\n",toupper(argv[i][j]));
                              Error("Invalid Command Line Switch");
                              exit(0);
                         }
               }
          }

          // Now check to see if they make sense
          if (P_PodID == 0)
          {
               DisplayHelp();
               Error("No Pod Number Specified");
          }

}


//*****************************************************
// Main Entry Point.  The program starts here.
//*****************************************************

int main(int argc, char* argv[])
{
     int RetValue;
     unsigned long totalbytes = 0;
     char *outputstr = new char [256];
     unsigned long ByteCounter = 0;
     unsigned long OutputValue;

     printf("USBee AX Data Extractor\n");
     printf("SMBus Extractor Version %d.%d\n", MAJOR_REV, MINOR_REV);

     // Parse out the command line options
     ParseCommandLine( argc, argv );

     //*****************************************************
     // Open up a file to store extracted data into
     //*****************************************************

     FILE *fout;
     if (O_OutputFilename[0])
     {
          if (I_BinaryValues)
               fout = fopen((char*)O_OutputFilename, "wb");
          else
               fout = fopen((char*)O_OutputFilename, "w");
     }

     //*****************************************************
     // Start the USBee AX Pod extracting the data we want
     //*****************************************************

     int Endpoint = 999;
     int Device = 999;

     RetValue = StartExtraction(P_PodID, A_All, D_DecimalTextValues, H_HexTextValues, I_BinaryValues, C_CommaDelimited,
G_SpaceDelimited, V_Timestamps, M_SDA, N_SCL)     ;

     if (RetValue == 0)
     {
          printf("Startup failed.  Is the USBee AX-Pro connected and is the PodNumber correct?\n");
          printf("Press any key to continue...");
          getch();
          return(0);
     }


     //*****************************************************
     // Loop and do something with the collected data
     //*****************************************************

     char OldSignal = 99;

     int KeepLooping = TRUE;
     while(KeepLooping)       // Do this forever until we tell it to stop by pressing a key
     {

          if (kbhit())
          {
               KeepLooping = FALSE;          // Stop the processing loop
               StopExtraction();             // Stop the streaming of data from the USBee
          }

          //*****************************************************
          // If there is data that has come in
          //*****************************************************
          int timeout = 0;
          while (unsigned long length = ExtractionBufferCount())
          {
               if (length > WORKING_BUFFER_SIZE)
                    length = WORKING_BUFFER_SIZE;

               //*****************************************************
               // Get the data into our local working buffer
```

```
               //*****************************************************

               GetNextData( tempbuffer, length );

               totalbytes += length;

               if (O_OutputFilename[0])
                    fwrite(tempbuffer, length, 1,  fout);   // Write it to a file

               if (S_Screen)
                    fwrite(tempbuffer, length, 1,  stdout); // Write it to the screen

               if (Q_NumberOfBytes)
               {
                    if (Q_NumberOfBytes <= length)
                    {
                         goto Done;          // Done with that many bytes
                    }
                    Q_NumberOfBytes -= length;
               }

               if (timeout++ > 3  ) break;  // Let up once in a while to let the OS process
          }

          if (!S_Screen)
               printf("\rProcessed %d output values.", totalbytes);

          //*****************************************************
          // Check to see if we have fallen behind too far
          //*****************************************************

          int y = ExtractBufferOverflow();

          if (y == 1)
          {
               printf("\nExtractor Buffer Overflow.\nYour data is streaming too fast for your output settings.\nLower
your data rate or change to output binary files.\n");
               goto Done;
          }
          else if (y == 2)
          {
               printf("\nRaw Sample Buffer Overflow.\nYour data is streaming too fast for your output settings.\nLower
your data rate or change to output binary files.\n");
               goto Done;
          }

          //*****************************************************
          // Give the OS a little time to do something else
          //*****************************************************

          Sleep(15);

     }

Done:
     if (!S_Screen)
          printf("\rProcessed %d output values.", totalbytes);

     //*****************************************************
     // Close the file
     //*****************************************************

     if (O_OutputFilename[0])
          fclose(fout);

     //*****************************************************
     // Stop the extraction process
     //*****************************************************

     StopExtraction();

     if (kbhit()) getch();
     printf("\nPress any key to continue...");
     getch();

     return 0;
}
```

# 7 SPI Data Extractor

The SPI Bus Data Extractor takes the real-time streaming data from an SPI bus, formats it and allows you to save the data to disk or process it as it arrives.

## *7.1 Serial Bus Data Extractor Specifications*

- Continuous Real-Time Data Streaming
- Monitors one SPI Bus
- TTL Level inputs (**0-5V max**, Vih = 2.0V, Vil = 0.8V)
- SPI Clock speeds up to 12MHz
- Asynchronous (internal) sampling of 1MB/s to 24MB/s*
- Output to Binary File*
- Output to Text File*
- Output to Screen*
- Output File Viewer (including binary, text, search and export functions)
- Extractor API libraries interface directly to your own software to further process the extracted data. Any language that supports calls to DLLs is supported.

* - output bandwidths are dependent on PC USB hardware, hard disk and/or screen throughput.

## *7.2 Hardware Setup*

To use the Data Extractor you need to connect the USBee AX-Pro Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

***Please note that the USBee AX-Pro Test Pod inputs are strictly 0-5V levels. Any voltage outside this range on the signals will damage the pod and may damage your hardware. If your system uses different voltage levels, you must buffer the signals externally to the USBee AX-Pro Test Pod before connecting the signals to the unit.***

The SPI Bus Data Extractor uses any of the 8 signal lines (0 thru 7) and the GND (ground) line. Connect any of the 8 signals lines to Slave Select, MOSI, and MISO. Connect the GND line to the digital ground of your system.

## *7.3 Extractor Command Line Program*

The SPI Bus Data Extractor includes a Windows Command Prompt executable that lets you operate the Data Extractor without writing any software. The program is executed in a Command Prompt window and is configured using command line arguments. The extracted data is then stored to disk or outputted to the screen depending on these parameters.

To run the Data Extractor:

1) Install the USBee AX-Pro software on your PC
2) Install the Data Extractor software on your PC
3) Plug in your USBee AX-Pro Test Pod into your PC using a USB 2.0 High Speed Port
4) Open a Windows Command Prompt window by clicking Start, All Programs, Accessories, Command Prompt.
5) Change the working directory to the Data Extractor directory
6) (“cd \program files\USBee Data Extractor\SPI”)
7) Run the executable using the following command line arguments:

```
SPIExtractor [-?SWT] [-Q NumberOfBytes] [-R SampleRate] [-M SlaveSelect] [-L CLK] [-V
MOSI] [-J MISO] [-K MOSISample] [-U MOSISample] [-O filename] -P PodID
  ? - Display this help screen
```

**USBee AX-Pro Pod to Use**

P - Pod ID (required)

**Output Location Flags**

O - Output to filename (default off)
S - Output to the screen (default off)

**When to Quit Flags**

Q - Number of output values (default = until keypress)

**Signal Selection**

M - Slave Select Signal (1=signal0,128=signal7)
L - Clk Signal (1=signal0,128=signal7)
V - MOSI Signal (1=signal0,128=signal7)
J - MISO Signal (1=signal0,128=signal7)
K - MOSI Sample Time (1=Rising CLK Edge,0=Falling CLK Edge)
U - MISO Sample Time (1=Rising CLK Edge,0=Falling CLK Edge)

**Display Options**

W - Insert Slave Select Boundaries
T - Insert Time Stamps

**Clocking Modes**

R - Internal CLK Sample Rate (16Msps default)

- 247 = 24MHz
- 167 = 16MHz (default)
- 127 = 12MHz
- 87 = 8MHz
- 67 = 6MHz
- 47 = 4MHz
- 37 = 3MHz
- 27 = 2MHz
- 17 = 1MHz

## *7.4 Extractor API*

The Data Extractor is implemented using a Windows DLL that interfaces to the existing USBee AX-Pro DLL and drivers. This DLL can be called using any software language that supports calls to DLLs. Below are the details of this DLL interface and the routines that are available for your use.

### 7.4.1 DLL filename:

`usbexSPI.dll in \Windows\System32`

## 7.4.2 DLL Exported Functions and parameters

**ExtractionBufferCount** – Returns the number of bytes that have been extracted from the data stream so far and are available to read using GetNextData.

```
CWAV_EXPORT unsigned long CWAV_API ExtractionBufferCount(void)
```

Returns:
0 – No data to read yet
other – number of bytes available to read

**GetNextData** – Copies the extracted data from the extractor into your working buffer

```
CWAV_EXPORT char CWAV_API GetNextData(unsigned char *buffer, unsigned long length);
```

buffer:
pointer to where you want the extracted data to be placed
length:
number of bytes you want to read from the extraction DLL
Returns:
0 – No data to read yet
1 – Data was copied into the buffer

**StartExtraction** – Starts the Data Extraction with the given parameters.

```
CWAV_EXPORT int CWAV_API StartExtraction( unsigned int SampleRate, unsigned long
PodNumber, unsigned int ClockMode, unsigned char SlaveSelect, unsigned char CLK, unsigned
char MOSI, unsigned char MISO, unsigned char MOSIEdge, unsigned char MISOEdge, unsigned
char SSInsert, unsigned char Timestamp );
```

SampleRate:
17 = 1Msps
27 = 2Msps
37 = 3Msps
47 = 4Msps
67 = 6Msps
87 = 8Msps
127 = 12Msps
167 = 16Msps
247 = 24Msps
PodNumber:
Pod ID on the back of the USBee AX-Pro Test Pod
ClockMode:
2 = Internal Timing as in SampleRate parameter
SlaveSelect:
Which signal the extractor uses for Slave Select (1=channel0,128=channel7)
CLK:
Which signal the extractor uses for CLK (1=channel0,128=channel7)
MOSI:
Which signal the extractor uses for MOSI (1=channel0,128=channel7)
MISO:
Which signal the extractor uses for MISO (1=channel0,128=channel7)
MOSIEdge:
When the MOSI signal is sampled, 0=Falling CLK Edge, 1=Rising CLK Edge
MISOEdge:
When the MISO signal is sampled, 0=Falling CLK Edge, 1=Rising CLK Edge
SSInsert:
Set to 1 to insert Slave Select boundaries into the extracted data stream
Timestamp:
Set to 1 to insert Time Stamps into the extracted data stream

Returns:
1 – if Start was successful
0 – if Pod failed initialization

**StopExtraction** – Stops the extraction in progress

```
CWAV_EXPORT int CWAV_API StopExtraction( void );
```

Returns:
1 – always

**ExtractBufferOverflow** – Returns the state of the overflow conditions

```
CWAV_EXPORT char CWAV_API ExtractBufferOverflow(void);
```

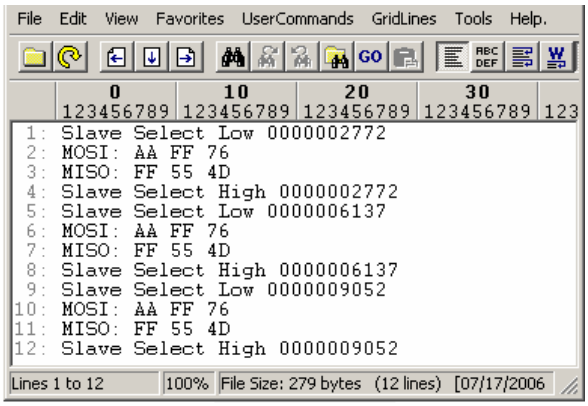
Return:
0 – No overflow
1 – Overflow Occurred. ExtractorBuffer Overflow condition cleared.
2 – Overflow Occurred. Raw Stream Buffer Overflow

## 7.4.3 Extraction Data Format

The GetNextData routine gets a series of bytes that represent the extracted data stream and places these bytes into the buffer pointed to by the *buffer parameter.

The SPI Bus Extractor outputs MOSI and MISO values separated by newline characters with optional Slave Select and Timestamps inserted.

SPIExtractor -O output.dex -P 143 -Q 500000 -M 8 -L 1 -V 2 -J 4 -K 1 -U 0 -W –T



## 7.4.4 Example Source Code

```
//*****************************************************
// USBee AX-Pro Data Extractor
// SPI Bus Extractor Example Program
// Copyright 2006, CWAV All Rights Reserved.
//*****************************************************

#include "stdafx.h"
#include "stdio.h"
#include "conio.h"
#include "windows.h"
#include <fcntl.h>
#include <io.h>
#include <stdlib.h>
#include <stdio.h>

#define MAJOR_REV 1
#define MINOR_REV 0

//*****************************************************
// Declare the Extractor DLL API routines
//*****************************************************

#define CWAV_API __stdcall
#define CWAV_IMPORT __declspec(dllimport)

CWAV_IMPORT int CWAV_API StartExtraction( unsigned int SampleRate, unsigned long PodNumber, unsigned int ClockMode,
                                          unsigned char SlaveSelect, unsigned char CLK, unsigned char MOSI,
                                          unsigned char MISO, unsigned char MOSIEdge, unsigned char MISOEdge,
                                          unsigned char SSInsert, unsigned char Timestamp );
CWAV_IMPORT char CWAV_API GetNextData(unsigned char *buffer, unsigned long length);
CWAV_IMPORT int CWAV_API StopExtraction( void );
CWAV_IMPORT char CWAV_API ExtractBufferOverflow(void);
CWAV_IMPORT unsigned long CWAV_API ExtractionBufferCount(void);

//*****************************************************
// Define the working buffer
//*****************************************************

#define WORKING_BUFFER_SIZE   (65536*8)
unsigned char tempbuffer[WORKING_BUFFER_SIZE];

// Command Line Parameter Settings
unsigned long P_PodID = 0;
unsigned char O_OutputFilename[256] = {0};
unsigned char S_Screen = FALSE;
unsigned char _1_BytePerValue = TRUE;
unsigned char _2_BytePerValue = FALSE;
unsigned char _4_BytePerValue = FALSE;
unsigned char Y_LeastSignificantByteFirst = FALSE;
unsigned char Z_MostSignificantByteFirst = TRUE;
unsigned char A_ASCIITextValues = FALSE;
unsigned char D_DecimalTextValues = FALSE;
unsigned char H_HexTextValues = TRUE;
unsigned char B_BinaryTextValues = FALSE;
unsigned char I_BinaryValues = FALSE;
unsigned char C_CommaDelimited = FALSE;
unsigned char G_SpaceDelimited = TRUE;
unsigned char N_NewlineDelimited = FALSE;
unsigned char X_NoDelimeter = FALSE;
unsigned long T_ForceBytesPerLine = 0;
unsigned char M_SlaveSelect = 0;
unsigned char L_CLK = 0;
unsigned char V_MOSI = 0;
unsigned char J_MISO = 0;
unsigned char K_MOSIEdge = 0;
unsigned char U_MISOEdge = 0;
unsigned char W_SSInsert = 0;
unsigned char T_Timestamp = 0;
unsigned char E_ExternalClockMode = 2;
unsigned char R_SampleRate = 167;
unsigned long Q_NumberOfBytes = 0;

void DisplayHelp(void)
{
     fprintf(stdout,"\nSPIExtractor [-?SWT] [-Q NumberOfBytes] [-R SampleRate] [-M SlaveSelect] [-L CLK] [-V MOSI] [-J
MISO] [-K MOSISample] [-U MOSISample] [-O filename] -P PodID\n\n");
     fprintf(stdout,"    ?  - Display this help screen\n");

     fprintf(stdout,"\n  USBee AX-Pro Pod to Use\n");
     fprintf(stdout,"    P  - Pod ID (required)\n");

     fprintf(stdout,"\n  Output Location Flags\n");
     fprintf(stdout,"    O  - Output to filename (default off)\n");
     fprintf(stdout,"    S  - Output to the screen (default off)\n");

     fprintf(stdout,"\n  When to Quit Flags\n");
     fprintf(stdout,"    Q  - Number of output values (default = until keypress)\n");

     fprintf(stdout,"\n  Signal Selection\n");
     fprintf(stdout,"    M  - Slave Select Signal (1=signal0,128=signal7)\n");
```

```
     fprintf(stdout,"    L  - Clk Signal (1=signal0,128=signal7)\n");
     fprintf(stdout,"    V  - MOSI Signal (1=signal0,128=signal7)\n");
     fprintf(stdout,"    J  - MISO Signal (1=signal0,128=signal7)\n");
     fprintf(stdout,"    K  - MOSI Sample Time (1=Rising CLK Edge,0=Falling CLK Edge)\n");
     fprintf(stdout,"    U  - MISO Sample Time (1=Rising CLK Edge,0=Falling CLK Edge)\n");

     fprintf(stdout,"\n  Clocking Modes\n");
     fprintf(stdout,"    R  - Internal CLK Sample Rate (16Msps default)\n");

     fprintf(stdout,"\n  Display Option\n");
     fprintf(stdout,"    W  - Insert Slave Select Boundaries\n");
     fprintf(stdout,"    T  - Insert Time Stamps\n");

     exit(0);
}
void Error(char *err)
{
     fprintf(stderr,"Error: ");
     fprintf(stderr,"%s\n",err);
     exit(2);
}

//*************************************************
// Parse all of the command line options
//*************************************************
void ParseCommandLine(int argc, char *argv[])
{
     BOOL cont;
     int       i,j;
     DWORD WordExample;
     BYTE ByteExample;

     for(i=1; i < argc; ++i)
     {
          if((argv[i][0] == '-') || (argv[i][0] == '/'))
          {
               cont = TRUE;
               for(j=1;argv[i][j] && cont;++j)     // Cont flag permits multiple commands in a single argv (like -AR)
                    switch(toupper(argv[i][j]))
                    {
                         case 'P':
                              P_PodID = (WORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'O':
                              strcpy((char*)O_OutputFilename, argv[++i]);
                              cont = FALSE;
                              break;
                         case '?':
                              DisplayHelp();
                              break;
                         case 'S':
                              S_Screen = TRUE;
                              break;
                         case 'Q':
                              Q_NumberOfBytes = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'M':
                              M_SlaveSelect = (BYTE)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'L':
                              L_CLK = (BYTE)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'V':
                              V_MOSI = (BYTE)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'J':
                              J_MISO = (BYTE)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'K':
                              K_MOSIEdge = (BYTE)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'U':
                              U_MISOEdge = (BYTE)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'W':
                              W_SSInsert = 1;
                              cont = FALSE;
                              break;
                         case 'T':
                              T_Timestamp = 1;
                              cont = FALSE;
                              break;
                         case 'E':
                              E_ExternalClockMode = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'R':
```

```
                         R_SampleRate = (BYTE)strtol(argv[++i],NULL,0);
                         cont = FALSE;
                         break;
                    case 'w':
                         WordExample = (DWORD)strtol(argv[++i],NULL,0);
                         cont = FALSE;
                         break;
                    case 'b':
                         ByteExample = (BYTE)strtol(argv[++i],NULL,0);
                         cont = FALSE;
                         break;
                    default:
                         DisplayHelp();
                         Error("Invalid Command Line Switch");
               }
          }
     }

     // Now check to see if they make sense
     if (P_PodID == 0)
     {
          DisplayHelp();
          Error("No Pod Number Specified");
     }

}
unsigned long StartTime;

void StartTimer()
{

     StartTime = GetTickCount();
}

void StopTimer()
{
     printf(" \nTime Delta = %d\n",GetTickCount() - StartTime);
}
//***************************************************
// Main Entry Point.  The program starts here.
//***************************************************

int main(int argc, char* argv[])
{
     int RetValue;
     unsigned long totalbytes = 0;
     char *outputstr = new char [256];
     unsigned long ByteCounter = 0;

     printf("USBee AX Data Extractor\n");
     printf("SPI Bus Extractor Version %d.%d\n", MAJOR_REV, MINOR_REV);

     // Parse out the command line options
     ParseCommandLine( argc, argv );

     //***************************************************
     // Open up a file to store extracted data into
     //***************************************************

     FILE *fout;
     if (O_OutputFilename[0])
     {
          if (I_BinaryValues)
               fout = fopen((char*)O_OutputFilename, "wb");
          else
               fout = fopen((char*)O_OutputFilename, "w");
     }

     //***************************************************
     // Start the USBee AX Pod extracting the data we want
     //***************************************************

     RetValue = StartExtraction( R_SampleRate, P_PodID, E_ExternalClockMode, M_SlaveSelect, L_CLK, V_MOSI, J_MISO,
K_MOSIEdge, U_MISOEdge, W_SSInsert, T_Timestamp );

     if (RetValue == 0)
     {
          printf("Startup failed.  Is the USBee AX-Pro connected and is the PodNumber correct?\n");
          printf("Press any key to continue...");
          getch();
          return(0);
     }
     printf("Processing and Saving Data to Disk.\n");

     //***************************************************
     // Loop and do something with the collected data
     //***************************************************

     int KeepLooping = TRUE;
     while(KeepLooping)       // Do this forever until we tell it to stop by pressing a key
     {

          if (kbhit())
          {
               KeepLooping = FALSE;          // Stop the processing loop
```

```
          StopExtraction();             // Stop the streaming of data from the USBee
     }

     //*****************************************************
     // If there is data that has come in
     //*****************************************************
     int timeout = 0;
     while (unsigned long length = ExtractionBufferCount())
     {
          if (length > WORKING_BUFFER_SIZE)
               length = WORKING_BUFFER_SIZE;

          //*****************************************************
          // Get the data into our local working buffer
          //*****************************************************
          StartTimer();

          GetNextData( tempbuffer, length );

          totalbytes += length;

          if (O_OutputFilename[0])
               fwrite(tempbuffer, length, 1,  fout);   // Write it to a file

          if (S_Screen)
               fwrite(tempbuffer, length, 1,  stdout); // Write it to the screen

          if (Q_NumberOfBytes)
          {
               if (Q_NumberOfBytes <= length)
               {
                    goto Done;          // Done with that many bytes
               }
               Q_NumberOfBytes -= length;
          }


          // StopTimer();

          if (timeout++ > 10  ) break;  // Let up once in a while to let the OS process
     }

     if (!S_Screen)
          printf("\rProcessed %d output values.", totalbytes);

     //*****************************************************
     // Check to see if we have fallen behind too far
     //*****************************************************

     int y = ExtractBufferOverflow();

     if (y == 1)
     {
          printf("\nExtractor Buffer Overflow.\nYour data is streaming too fast for your output settings.\nLower
your data rate or change to output binary files.\n");
          goto Done;
     }
     else if (y == 2)
     {
          printf("\nRaw Sample Buffer Overflow.\nYour data is streaming too fast for your output settings.\nLower
your data rate or change to output binary files.\n");
          goto Done;
     }

     //*****************************************************
     // Give the OS a little time to do something else
     //*****************************************************

     Sleep(15);

}

Done:
     if (!S_Screen)
          printf("\rProcessed %d output values.", totalbytes);

     //*****************************************************
     // Close the file
     //*****************************************************

     if (O_OutputFilename[0])
          fclose(fout);

     //*****************************************************
     // Stop the extraction process
     //*****************************************************

     StopExtraction();

     if (kbhit()) getch();
     printf("\nPress any key to continue...");
     getch();

     return 0;}
```

# 8 1-Wire Data Extractor

The 1-Wire Bus Data Extractor takes the real-time streaming data from an 1-Wire bus, formats it and allows you to save the data to disk or process it as it arrives.

## *8.1 1-Wire Bus Data Extractor Specifications*

- Continuous Real-Time Data Streaming
- Monitors one 1-Wire Bus
- TTL Level inputs (**0-5V max**, Vih = 2.0V, Vil = 0.8V)
- Asynchronous (internal) sampling from 1MB/s to 24MB/s*
- Output to Binary File*
- Output to Text File*
- Output to Screen*
- Output File Viewer (including binary, text, search and export functions)
- Extractor API libraries interface directly to your own software to further process the extracted data. Any language that supports calls to DLLs is supported.

* - output bandwidths are dependent on PC USB hardware, hard disk and/or screen throughput.

## *8.2 Hardware Setup*

To use the Data Extractor you need to connect the USBee AX-Pro Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

***Please note that the USBee AX-Pro Test Pod inputs are strictly 0-5V levels. Any voltage outside this range on the signals will damage the pod and may damage your hardware. If your system uses different voltage levels, you must buffer the signals externally to the USBee AX-Pro Test Pod before connecting the signals to the unit.***

The 1-Wire Bus Data Extractor uses any of the 8 signal lines (0 thru 7) and the GND (ground) line. Connect any of the 8 signals lines to the 1-Wire Signal. Connect the GND line to the digital ground of your system.

## *8.3 Extractor Command Line Program*

The 1-Wire Bus Data Extractor includes a Windows Command Prompt executable that lets you operate the Data Extractor without writing any software. The program is executed in a Command Prompt window and is configured using command line arguments. The extracted data is then stored to disk or outputted to the screen depending on these parameters.

To run the Data Extractor:

1) Install the USBee AX-Pro software on your PC
2) Install the Data Extractor software on your PC
3) Plug in your USBee AX-Pro Test Pod into your PC using a USB 2.0 High Speed Port
4) Open a Windows Command Prompt window by clicking Start, All Programs, Accessories, Command Prompt.
5) Change the working directory to the Data Extractor directory
6) (“cd \program files\USBee Data Extractor\1Wire”)
7) Run the executable using the following command line arguments:

```
1WireExtractor [-?STW] [-Q NumberOfBytes] [-R SampleRate] [-M Signal] [-O filename] -P
PodID
```

? - Display this help screen

**USBee AX-Pro Pod to Use**

P - Pod ID (required)

**Output Location Flags**

O - Output to filename (default off)
S - Output to the screen (default off)

**When to Quit Flags**

Q - Number of output values (default = until keypress)

**Signal Selection**

M - 1 Wire Signal Mask (1=channel0,128=channel7)

**Display Options**

W - Insert Reset/Presence Pulse
T - Insert Time Stamps

**Clocking Modes**

R - Internal CLK Sample Rate (16Msps default)

- 247 = 24MHz
- 167 = 16MHz
- 127 = 12MHz
- 87 = 8MHz
- 67 = 6MHz
- 47 = 4MHz
- 37 = 3MHz
- 27 = 2MHz
- 17 = 1MHz (default)

## *8.4 Extractor API*

The Data Extractor is implemented using a Windows DLL that interfaces to the existing USBee AX-Pro DLL and drivers. This DLL can be called using any software language that supports calls to DLLs. Below are the details of this DLL interface and the routines that are available for your use.

### 8.4.1 DLL filename:

```
Usbex1Wire.dll in \Windows\System32
```

### 8.4.2 DLL Exported Functions and parameters

**ExtractionBufferCount** – Returns the number of bytes that have been extracted from the data stream so far and are available to read using GetNextData.

```
CWAV_EXPORT unsigned long CWAV_API ExtractionBufferCount(void)
```

Returns:
0 – No data to read yet
other – number of bytes available to read

**GetNextData** – Copies the extracted data from the extractor into your working buffer

```
CWAV_EXPORT char CWAV_API GetNextData(unsigned char *buffer, unsigned long length);
```

buffer:
pointer to where you want the extracted data to be placed
length:
number of bytes you want to read from the extraction DLL
Returns:
0 – No data to read yet
1 – Data was copied into the buffer

**StartExtraction** – Starts the Data Extraction with the given parameters.

```
CWAV_EXPORT int CWAV_API StartExtraction( unsigned int SampleRate, unsigned long
PodNumber, unsigned int ClockMode, unsigned char Signal, unsigned char SSInsert, unsigned
char Timestamp );
```

SampleRate:
17 = 1Msps
27 = 2Msps
37 = 3Msps
47 = 4Msps
67 = 6Msps
87 = 8Msps
127 = 12Msps
167 = 16Msps
247 = 24Msps
PodNumber:
Pod ID on the back of the USBee AX-Pro Test Pod
ClockMode:
2 = Internal Timing as in SampleRate parameter
Signal:
Which signal the extractor uses for the 1-Wire Signal (1=channel0,128=channel7)
SSInsert:
Set to 1 to insert Reset/Presence boundaries into the extracted data stream
Timestamp:
Set to 1 to insert Time Stamps into the extracted data stream

Returns:
1 – if Start was successful
0 – if Pod failed initialization

**StopExtraction** – Stops the extraction in progress

```
CWAV_EXPORT int CWAV_API StopExtraction( void );
```

Returns:
1 – always

**ExtractBufferOverflow** – Returns the state of the overflow conditions

```
CWAV_EXPORT char CWAV_API ExtractBufferOverflow(void);
```

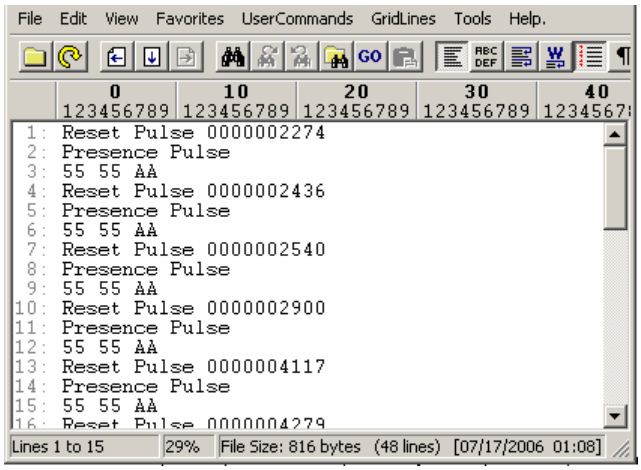
Return:
0 – No overflow
1 – Overflow Occurred. ExtractorBuffer Overflow condition cleared.
2 – Overflow Occurred. Raw Stream Buffer Overflow

## 8.4.3 Extraction Data Format

The GetNextData routine gets a series of bytes that represent the extracted data stream and places these bytes into the buffer pointed to by the *buffer parameter.

The 1-Wire Bus Extractor outputs data values separated by newline characters with option Reset/Presence and Timestamps inserted.

1WireExtractor -O output.dex -P 143 -Q 500000 -M 1 -W -T -R 127




## 8.4.4 Example Source Code

```
//*****************************************************
// USBee AX-Pro Data Extractor
// 1 Wire Bus Extractor Example Program
// Copyright 2006, CWAV All Rights Reserved.
//*****************************************************

#include "stdafx.h"
#include "stdio.h"
#include "conio.h"
#include "windows.h"
#include <fcntl.h>
#include <io.h>
#include <stdlib.h>
#include <stdio.h>

#define MAJOR_REV 1
#define MINOR_REV 0

//*****************************************************
// Declare the Extractor DLL API routines
//*****************************************************

#define CWAV_API __stdcall
#define CWAV_IMPORT __declspec(dllimport)

CWAV_IMPORT int CWAV_API StartExtraction( unsigned int SampleRate, unsigned long PodNumber, unsigned int ClockMode,
                                          unsigned char Signal, unsigned char SSInsert, unsigned char
Timestamp );
CWAV_IMPORT char CWAV_API GetNextData(unsigned char *buffer, unsigned long length);
CWAV_IMPORT int CWAV_API StopExtraction( void );
CWAV_IMPORT char CWAV_API ExtractBufferOverflow(void);
CWAV_IMPORT unsigned long CWAV_API ExtractionBufferCount(void);

//*****************************************************
// Define the working buffer
//*****************************************************

#define WORKING_BUFFER_SIZE   (65536*8)
unsigned char tempbuffer[WORKING_BUFFER_SIZE];

// Command Line Parameter Settings
unsigned long P_PodID = 0;
unsigned char O_OutputFilename[256] = {0};
unsigned char S_Screen = FALSE;
unsigned char _1_BytePerValue = TRUE;
unsigned char _2_BytePerValue = FALSE;
unsigned char _4_BytePerValue = FALSE;
unsigned char Y_LeastSignificantByteFirst = FALSE;
```

```
unsigned char Z_MostSignificantByteFirst = TRUE;
unsigned char A_ASCIITextValues = FALSE;
unsigned char D_DecimalTextValues = FALSE;
unsigned char H_HexTextValues = TRUE;
unsigned char B_BinaryTextValues = FALSE;
unsigned char I_BinaryValues = FALSE;
unsigned char C_CommaDelimited = FALSE;
unsigned char G_SpaceDelimited = TRUE;
unsigned char N_NewlineDelimited = FALSE;
unsigned char X_NoDelimeter = FALSE;
unsigned long T_ForceBytesPerLine = 0;
unsigned char M_Signal = 0;
unsigned char W_SSInsert = 0;
unsigned char T_Timestamp = 0;
unsigned char E_ExternalClockMode = 2;
unsigned char R_SampleRate = 167;
unsigned long Q_NumberOfBytes = 0;
// Not used yet W

void DisplayHelp(void)
{
     fprintf(stdout,"\n1WireExtractor [-?STW] [-Q NumberOfBytes] [-R SampleRate] [-M Signal] [-O filename] -P
PodID\n\n");
     fprintf(stdout,"    ?  - Display this help screen\n");

     fprintf(stdout,"\n  USBee AX-Pro Pod to Use\n");
     fprintf(stdout,"    P  - Pod ID (required)\n");

     fprintf(stdout,"\n  Output Location Flags\n");
     fprintf(stdout,"    O  - Output to filename (default off)\n");
     fprintf(stdout,"    S  - Output to the screen (default off)\n");

     fprintf(stdout,"\n  When to Quit Flags\n");
     fprintf(stdout,"    Q  - Number of output values (default = until keypress)\n");

     fprintf(stdout,"\n  Signal Selection\n");
     fprintf(stdout,"    M  - Signal (1=signal0,128=signal7)\n");

     fprintf(stdout,"\n  Clocking Modes\n");
     fprintf(stdout,"    R  - Internal CLK Sample Rate (16Msps default)\n");

     fprintf(stdout,"\n  Display Option\n");
     fprintf(stdout,"    W  - Insert Reset/Presence\n");
     fprintf(stdout,"    T  - Insert Time Stamps\n");

     exit(0);
}

void Error(char *err)
{
     fprintf(stderr,"Error: ");
     fprintf(stderr,"%s\n",err);
     exit(2);
}


//*************************************************
// Parse all of the command line options
//*************************************************
void ParseCommandLine(int argc, char *argv[])
{
     BOOL cont;
     int       i,j;
     DWORD WordExample;
     BYTE ByteExample;

     for(i=1; i < argc; ++i)
     {
          if((argv[i][0] == '-') || (argv[i][0] == '/'))
          {
               cont = TRUE;
               for(j=1;argv[i][j] && cont;++j)     // Cont flag permits multiple commands in a single argv (like -AR)
                    switch(toupper(argv[i][j]))
                    {
                         case 'P':
                              P_PodID = (WORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'O':
                              strcpy((char*)O_OutputFilename, argv[++i]);
                              cont = FALSE;
                              break;
                         case '?':
                              DisplayHelp();
                              break;
                         case 'S':
                              S_Screen = TRUE;
                              break;
                         case 'Q':
                              Q_NumberOfBytes = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'M':
                              M_Signal = (BYTE)strtol(argv[++i],NULL,0);
                              cont = FALSE;
```

```
                         break;
                    case 'W':
                         W_SSInsert = 1;
                         cont = FALSE;
                         break;
                    case 'T':
                         T_Timestamp = 1;
                         cont = FALSE;
                         break;
                    case 'R':
                         R_SampleRate = (BYTE)strtol(argv[++i],NULL,0);
                         cont = FALSE;
                         break;
                    case 'w':
                         WordExample = (DWORD)strtol(argv[++i],NULL,0);
                         cont = FALSE;
                         break;
                    case 'b':
                         ByteExample = (BYTE)strtol(argv[++i],NULL,0);
                         cont = FALSE;
                         break;
                    default:
                         DisplayHelp();
                         Error("Invalid Command Line Switch");
               }
          }
     }


     // Now check to see if they make sense
     if (P_PodID == 0)
     {
          DisplayHelp();
          Error("No Pod Number Specified");
     }

}

unsigned long StartTime;

void StartTimer()
{

     StartTime = GetTickCount();
}

void StopTimer()
{

     printf(" \nTime Delta = %d\n",GetTickCount() - StartTime);

}

//*****************************************************
// Main Entry Point.  The program starts here.
//*****************************************************

int main(int argc, char* argv[])
{
     int RetValue;
     unsigned long totalbytes = 0;
     char *outputstr = new char [256];
     unsigned long ByteCounter = 0;

     printf("USBee AX Data Extractor\n");
     printf("1Wire Bus Extractor Version %d.%d\n", MAJOR_REV, MINOR_REV);

     // Parse out the command line options
     ParseCommandLine( argc, argv );

     //*****************************************************
     // Open up a file to store extracted data into
     //*****************************************************

     FILE *fout;
     if (O_OutputFilename[0])
     {
          if (I_BinaryValues)
               fout = fopen((char*)O_OutputFilename, "wb");
          else
               fout = fopen((char*)O_OutputFilename, "w");
     }

     //*****************************************************
     // Start the USBee AX Pod extracting the data we want
     //*****************************************************

     RetValue = StartExtraction( R_SampleRate, P_PodID, E_ExternalClockMode, M_Signal, W_SSInsert, T_Timestamp );

     if (RetValue == 0)
     {
          printf("Startup failed.  Is the USBee AX-Pro connected and is the PodNumber correct?\n");
          printf("Press any key to continue...");
          getch();
          return(0);
```

```
    }

    printf("Processing and Saving Data to Disk.\n");

    //*****************************************************
    // Loop and do something with the collected data
    //*****************************************************

    int KeepLooping = TRUE;
    while(KeepLooping)       // Do this forever until we tell it to stop by pressing a key
    {

         if (kbhit())
         {
              KeepLooping = FALSE;          // Stop the processing loop
              StopExtraction();             // Stop the streaming of data from the USBee
         }

         //*****************************************************
         // If there is data that has come in
         //*****************************************************
         int timeout = 0;
         while (unsigned long length = ExtractionBufferCount())
         {
              if (length > WORKING_BUFFER_SIZE)
                   length = WORKING_BUFFER_SIZE;

              //*****************************************************
              // Get the data into our local working buffer
              //*****************************************************
              StartTimer();

              GetNextData( tempbuffer, length );

              totalbytes += length;

              if (O_OutputFilename[0])
                   fwrite(tempbuffer, length, 1,  fout);   // Write it to a file

              if (S_Screen)
                   fwrite(tempbuffer, length, 1,  stdout); // Write it to the screen

              if (Q_NumberOfBytes)
              {
                   if (Q_NumberOfBytes <= length)
                   {
                        goto Done;          // Done with that many bytes
                   }
                   Q_NumberOfBytes -= length;
              }


              // StopTimer();

              if (timeout++ > 10  ) break;  // Let up once in a while to let the OS process
         }

         if (!S_Screen)
              printf("\rProcessed %d output values.", totalbytes);

         //*****************************************************
         // Check to see if we have fallen behind too far
         //*****************************************************

         int y = ExtractBufferOverflow();

         if (y == 1)
         {
              printf("\nExtractor Buffer Overflow.\nYour data is streaming too fast for your output settings.\nLower
your data rate or change to output binary files.\n");
              goto Done;
         }
         else if (y == 2)
         {
              printf("\nRaw Sample Buffer Overflow.\nYour data is streaming too fast for your output settings.\nLower
your data rate or change to output binary files.\n");
              goto Done;
         }

         //*****************************************************
         // Give the OS a little time to do something else
         //*****************************************************

         Sleep(15);

    }

Done:
    if (!S_Screen)
         printf("\rProcessed %d output values.", totalbytes);

    //*****************************************************
    // Close the file
    //*****************************************************
```

```
     if (O_OutputFilename[0])
          fclose(fout);

     //*************************************************
     // Stop the extraction process
     //*************************************************

     StopExtraction();

     if (kbhit()) getch();
     printf("\nPress any key to continue...");
     getch();

     return 0;
}
```

# 9 I2S Data Extractor

The I2S Bus Data Extractor takes the real-time streaming data from an I2S bus, formats it and allows you to save the data to disk or process it as it arrives.

## 9.1 I2S Bus Data Extractor Specifications

- Continuous Real-Time Data Streaming
- Monitors one I2S Bus
- TTL Level inputs (**0-5V max**, Vih = 2.0V, Vil = 0.8V)
- I2S Bit Clock up to 12MHz
- Asynchronous (internal) sampling from 1MB/s to 24MB/s*
- Output to Binary File*
- Output to Text File*
- Output to Screen*
- Output File Viewer (including binary, text, search and export functions)
- Extractor API libraries interface directly to your own software to further process the extracted data. Any language that supports calls to DLLs is supported.

* - output bandwidths are dependent on PC USB hardware, hard disk and/or screen throughput.

## 9.2 Hardware Setup

To use the Data Extractor you need to connect the USBee AX-Pro Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

***Please note that the USBee AX-Pro Test Pod inputs are strictly 0-5V levels. Any voltage outside this range on the signals will damage the pod and may damage your hardware. If your system uses different voltage levels, you must buffer the signals externally to the USBee AX-Pro Test Pod before connecting the signals to the unit.***

The I2S Bus Data Extractor uses any of the 8 signal lines (0 thru 7) and the GND (ground) line. Connect any of the 8 signals lines to Word Select, CLK, and Data. Connect the GND line to the digital ground of your system.

## 9.3 Extractor Command Line Program

The I2S Bus Data Extractor includes a Windows Command Prompt executable that lets you operate the Data Extractor without writing any software. The program is executed in a Command Prompt window and is configured using command line arguments. The extracted data is then stored to disk or outputted to the screen depending on these parameters.

To run the Data Extractor:

1) Install the USBee AX-Pro software on your PC
2) Install the Data Extractor software on your PC
3) Plug in your USBee AX-Pro Test Pod into your PC using a USB 2.0 High Speed Port
4) Open a Windows Command Prompt window by clicking Start, All Programs, Accessories, Command Prompt.
5) Change the working directory to the Data Extractor directory
6) (“cd \program files\USBee Data Extractor\I2S”)
7) Run the executable using the following command line arguments:

```
I2SExtractor [-?ST1234] [-Q NumberOfBytes] [-R SampleRate] [-M WordSelect] [-L CLK] [-V
Data] [-O filename] -P PodID
  ? - Display this help screen
```

**USBee AX-Pro Pod to Use**

P - Pod ID (required)

**Output Location Flags**

O - Output to filename (default off)
S - Output to the screen (default off)

**When to Quit Flags**

Q - Number of output values (default = until keypress)

**Signal Selection**

M - Word Select Signal (1=signal0,128=signal7)
L - Clk Signal (1=signal0,128=signal7)
V - Data Signal (1=signal0,128=signal7)

**Display Options**

T - Insert Word Select Boundaries

**Clocking Modes**

R - Internal CLK Sample Rate (16Msps default)

- 247 = 24MHz
- 167 = 16MHz (default)
- 127 = 12MHz
- 87 = 8MHz
- 67 = 6MHz
- 47 = 4MHz
- 37 = 3MHz
- 27 = 2MHz
- 17 = 1MHz

## *9.4 Extractor API*

The Data Extractor is implemented using a Windows DLL that interfaces to the existing USBee AX-Pro DLL and drivers. This DLL can be called using any software language that supports calls to DLLs. Below are the details of this DLL interface and the routines that are available for your use.

### 9.4.1 DLL filename:

`usbexI2S.dll in \Windows\System32`

### 9.4.2 DLL Exported Functions and parameters

**ExtractionBufferCount** – Returns the number of bytes that have been extracted from the data stream so far and are available to read using GetNextData.

```
CWAV_EXPORT unsigned long CWAV_API ExtractionBufferCount(void)
```

Returns:
0 – No data to read yet
other – number of bytes available to read

**GetNextData** – Copies the extracted data from the extractor into your working buffer

```
CWAV_EXPORT char CWAV_API GetNextData(unsigned char *buffer, unsigned long length);
```

buffer:
pointer to where you want the extracted data to be placed
length:
number of bytes you want to read from the extraction DLL
Returns:
0 – No data to read yet
1 – Data was copied into the buffer

**StartExtraction** – Starts the Data Extraction with the given parameters.

```
CWAV_EXPORT int CWAV_API StartExtraction( unsigned int SampleRate, unsigned long
PodNumber, unsigned int ClockMode, unsigned char WordSelect, unsigned char CLK, unsigned
char Data, unsigned char SSInsert, unsigned char BytesPerValue );
```

SampleRate:
17 = 1Msps
27 = 2Msps
37 = 3Msps
47 = 4Msps
67 = 6Msps
87 = 8Msps
127 = 12Msps
167 = 16Msps
247 = 24Msps
PodNumber:
Pod ID on the back of the USBee AX-Pro Test Pod
ClockMode:
2 = Internal Timing as in SampleRate parameter
WordSelect:
Which signal the extractor uses for Word Select (1=channel0,128=channel7)
CLK:
Which signal the extractor uses for CLK (1=channel0,128=channel7)
Data:
Which signal the extractor uses for Data (1=channel0,128=channel7)
SSInsert:
Set to 1 to insert Word Select boundaries into the extracted data stream
BytesPerValue:
1, 2, 3, or 4 bytes per value. Allows capture of 8, 16, 24, or 32 bits of audio data

Returns:
1 – if Start was successful
0 – if Pod failed initialization

**StopExtraction** – Stops the extraction in progress

```
CWAV_EXPORT int CWAV_API StopExtraction( void );
```

Returns:
1 – always

**ExtractBufferOverflow** – Returns the state of the overflow conditions

```
CWAV_EXPORT char CWAV_API ExtractBufferOverflow(void);
```

Return:

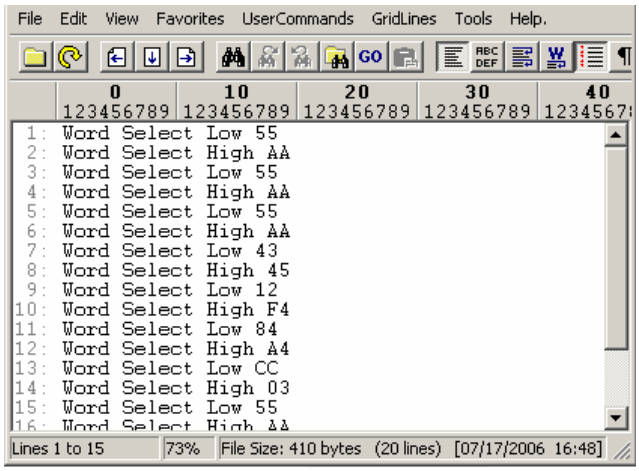
0 – No overflow
1 – Overflow Occurred. ExtractorBuffer Overflow condition cleared.
2 – Overflow Occurred. Raw Stream Buffer Overflow

## 9.4.3 Extraction Data Format

The GetNextData routine gets a series of bytes that represent the extracted data stream and places these bytes into the buffer pointed to by the *buffer parameter.

The I2S Bus Extractor outputs data values separated by newline characters with optional Word Selects inserted.

I2SExtractor -O output.dex -P 143 -Q 500000 -M 8 -L 1 -V 2 –T




## 9.4.4 Example Source Code

```
//*****************************************************
// USBee AX-Pro Data Extractor
// I2S Bus Extractor Example Program
// Copyright 2006, CWAV All Rights Reserved.
//*****************************************************

#include "stdafx.h"
#include "stdio.h"
#include "conio.h"
#include "windows.h"
#include <fcntl.h>
#include <io.h>
#include <stdlib.h>
#include <stdio.h>

#define MAJOR_REV 1
#define MINOR_REV 0

//*****************************************************
// Declare the Extractor DLL API routines
//*****************************************************

#define CWAV_API __stdcall
#define CWAV_IMPORT __declspec(dllimport)

CWAV_IMPORT int CWAV_API StartExtraction( unsigned int SampleRate, unsigned long PodNumber, unsigned int ClockMode,
                                          unsigned char WordSelect, unsigned char CLK, unsigned char Data,
                                          unsigned char SSInsert, unsigned char BytesPerValue );
CWAV_IMPORT char CWAV_API GetNextData(unsigned char *buffer, unsigned long length);
CWAV_IMPORT int CWAV_API StopExtraction( void );
CWAV_IMPORT char CWAV_API ExtractBufferOverflow(void);
CWAV_IMPORT unsigned long CWAV_API ExtractionBufferCount(void);

//*****************************************************
// Define the working buffer
//*****************************************************

#define WORKING_BUFFER_SIZE   (65536*8)
unsigned char tempbuffer[WORKING_BUFFER_SIZE];
```

```
// Command Line Parameter Settings
unsigned long P_PodID = 0;
unsigned char O_OutputFilename[256] = {0};
unsigned char S_Screen = FALSE;
unsigned char BytePerValue = 1;
unsigned char _2_BytePerValue = FALSE;
unsigned char _4_BytePerValue = FALSE;
unsigned char Y_LeastSignificantByteFirst = FALSE;
unsigned char Z_MostSignificantByteFirst = TRUE;
unsigned char A_ASCIITextValues = FALSE;
unsigned char D_DecimalTextValues = FALSE;
unsigned char H_HexTextValues = TRUE;
unsigned char B_BinaryTextValues = FALSE;
unsigned char I_BinaryValues = FALSE;
unsigned char C_CommaDelimited = FALSE;
unsigned char G_SpaceDelimited = TRUE;
unsigned char N_NewlineDelimited = FALSE;
unsigned char X_NoDelimeter = FALSE;
unsigned long T_ForceBytesPerLine = 0;
unsigned char M_WordSelect = 0;
unsigned char L_CLK = 0;
unsigned char V_Data = 0;
unsigned char J_MISO = 0;
unsigned char K_DataEdge = 0;
unsigned char U_MISOEdge = 0;
unsigned char T_SSInsert = 0;
unsigned char E_ExternalClockMode = 2;
unsigned char R_SampleRate = 167;
unsigned long Q_NumberOfBytes = 0;

void DisplayHelp(void)
{
     fprintf(stdout,"\nI2SExtractor [-?ST1234] [-Q NumberOfBytes] [-R SampleRate] [-M WordSelect] [-L CLK] [-V Data] [-
O filename] -P PodID\n\n");
     fprintf(stdout,"    ?  - Display this help screen\n");

     fprintf(stdout,"\n  USBee AX-Pro Pod to Use\n");
     fprintf(stdout,"    P  - Pod ID (required)\n");

     fprintf(stdout,"\n  Output Location Flags\n");
     fprintf(stdout,"    O  - Output to filename (default off)\n");
     fprintf(stdout,"    S  - Output to the screen (default off)\n");

     fprintf(stdout,"\n  When to Quit Flags\n");
     fprintf(stdout,"    Q  - Number of output values (default = until keypress)\n");

     fprintf(stdout,"\n  Signal Selection\n");
     fprintf(stdout,"    M  - Word Select Signal (1=signal0,128=signal7)\n");
     fprintf(stdout,"    L  - Clk Signal (1=signal0,128=signal7)\n");
     fprintf(stdout,"    V  - Data Signal (1=signal0,128=signal7)\n");

     fprintf(stdout,"\n  Number of Bytes to Capture\n");
     fprintf(stdout,"    1  - One   Byte  per value (default)\n");
     fprintf(stdout,"    2  - Two   Bytes per value\n");
     fprintf(stdout,"    3  - Three Bytes per value\n");
     fprintf(stdout,"    4  - Four  Bytes per value\n");

     fprintf(stdout,"\n  Clocking Modes\n");
     fprintf(stdout,"    R  - Internal CLK Sample Rate (16Msps default)\n");

     fprintf(stdout,"\n  Display Option\n");
     fprintf(stdout,"    T  - Insert Word Select Boundaries\n");

     exit(0);
}

void Error(char *err)
{
     fprintf(stderr,"Error: ");
     fprintf(stderr,"%s\n",err);
     exit(2);
}


//*****************************************************
// Parse all of the command line options
//*****************************************************
void ParseCommandLine(int argc, char *argv[])
{
     BOOL cont;
     int       i,j;
     DWORD WordExample;
     BYTE ByteExample;

     for(i=1; i < argc; ++i)
     {
          if((argv[i][0] == '-') || (argv[i][0] == '/'))
          {
               cont = TRUE;
               for(j=1;argv[i][j] && cont;++j)     // Cont flag permits multiple commands in a single argv (like -AR)
                    switch(toupper(argv[i][j]))
                    {
                         case 'P':
                              P_PodID = (WORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
```

```
                    break;
               case 'O':
                    strcpy((char*)O_OutputFilename, argv[++i]);
                    cont = FALSE;
                    break;
               case '?':
                    DisplayHelp();
                    break;
               case 'S':
                    S_Screen = TRUE;
                    break;
               case 'Q':
                    Q_NumberOfBytes = (DWORD)strtol(argv[++i],NULL,0);
                    cont = FALSE;
                    break;
               case '1':
                    BytePerValue = 1;
                    break;
               case '2':
                    BytePerValue = 2;
                    break;
               case '3':
                    BytePerValue = 3;
                    break;
               case '4':
                    BytePerValue = 4;
                    break;
               case 'M':
                    M_WordSelect = (BYTE)strtol(argv[++i],NULL,0);
                    cont = FALSE;
                    break;
               case 'L':
                    L_CLK = (BYTE)strtol(argv[++i],NULL,0);
                    cont = FALSE;
                    break;
               case 'V':
                    V_Data = (BYTE)strtol(argv[++i],NULL,0);
                    cont = FALSE;
                    break;
               case 'J':
                    J_MISO = (BYTE)strtol(argv[++i],NULL,0);
                    cont = FALSE;
                    break;
               case 'K':
                    K_DataEdge = (BYTE)strtol(argv[++i],NULL,0);
                    cont = FALSE;
                    break;
               case 'U':
                    U_MISOEdge = (BYTE)strtol(argv[++i],NULL,0);
                    cont = FALSE;
                    break;
               case 'T':
                    T_SSInsert = 1;
                    cont = FALSE;
                    break;
               case 'E':
                    E_ExternalClockMode = (DWORD)strtol(argv[++i],NULL,0);
                    cont = FALSE;
                    break;
               case 'R':
                    R_SampleRate = (BYTE)strtol(argv[++i],NULL,0);
                    cont = FALSE;
                    break;
               case 'w':
                    WordExample = (DWORD)strtol(argv[++i],NULL,0);
                    cont = FALSE;
                    break;
               case 'b':
                    ByteExample = (BYTE)strtol(argv[++i],NULL,0);
                    cont = FALSE;
                    break;
               default:
                    DisplayHelp();
                    Error("Invalid Command Line Switch");
          }
     }
}


// Now check to see if they make sense
if (P_PodID == 0)
{
     DisplayHelp();
     Error("No Pod Number Specified");
}

}

unsigned long StartTime;

void StartTimer()
{

     StartTime = GetTickCount();
}
```

```
void StopTimer()
{

     printf(" \nTime Delta = %d\n",GetTickCount() - StartTime);

}

//**************************************************
// Main Entry Point.  The program starts here.
//**************************************************

int main(int argc, char* argv[])
{
     int RetValue;
     unsigned long totalbytes = 0;
     char *outputstr = new char [256];
     unsigned long ByteCounter = 0;

     printf("USBee AX Data Extractor\n");
     printf("I2S Bus Extractor Version %d.%d\n", MAJOR_REV, MINOR_REV);

     // Parse out the command line options
     ParseCommandLine( argc, argv );

     //**************************************************
     // Open up a file to store extracted data into
     //**************************************************

     FILE *fout;
     if (O_OutputFilename[0])
     {
          if (I_BinaryValues)
               fout = fopen((char*)O_OutputFilename, "wb");
          else
               fout = fopen((char*)O_OutputFilename, "w");
     }

     //**************************************************
     // Start the USBee AX Pod extracting the data we want
     //**************************************************

     RetValue = StartExtraction( R_SampleRate, P_PodID, E_ExternalClockMode, M_WordSelect, L_CLK, V_Data, T_SSInsert,
BytePerValue );

     if (RetValue == 0)
     {
          printf("Startup failed.  Is the USBee AX-Pro connected and is the PodNumber correct?\n");
          printf("Press any key to continue...");
          getch();
          return(0);
     }

     printf("Processing and Saving Data to Disk.\n");

     //**************************************************
     // Loop and do something with the collected data
     //**************************************************

     int KeepLooping = TRUE;
     while(KeepLooping)       // Do this forever until we tell it to stop by pressing a key
     {

          if (kbhit())
          {
               KeepLooping = FALSE;          // Stop the processing loop
               StopExtraction();             // Stop the streaming of data from the USBee
          }

          //**************************************************
          // If there is data that has come in
          //**************************************************
          int timeout = 0;
          while (unsigned long length = ExtractionBufferCount())
          {
               if (length > WORKING_BUFFER_SIZE)
                    length = WORKING_BUFFER_SIZE;

               //**************************************************
               // Get the data into our local working buffer
               //**************************************************
               StartTimer();

               GetNextData( tempbuffer, length );

               totalbytes += length;

               if (O_OutputFilename[0])
                    fwrite(tempbuffer, length, 1,  fout);   // Write it to a file

               if (S_Screen)
                    fwrite(tempbuffer, length, 1,  stdout); // Write it to the screen

               if (Q_NumberOfBytes)
               {
```

```
                    if (Q_NumberOfBytes <= length)
                    {
                         goto Done;          // Done with that many bytes
                    }
                    Q_NumberOfBytes -= length;
               }


               // StopTimer();

               if (timeout++ > 10  ) break;  // Let up once in a while to let the OS process
          }

          if (!S_Screen)
               printf("\rProcessed %d output values.", totalbytes);

          //*****************************************************
          // Check to see if we have fallen behind too far
          //*****************************************************

          int y = ExtractBufferOverflow();

          if (y == 1)
          {
               printf("\nExtractor Buffer Overflow.\nYour data is streaming too fast for your output settings.\nLower
your data rate or change to output binary files.\n");
               goto Done;
          }
          else if (y == 2)
          {
               printf("\nRaw Sample Buffer Overflow.\nYour data is streaming too fast for your output settings.\nLower
your data rate or change to output binary files.\n");
               goto Done;
          }

          //*****************************************************
          // Give the OS a little time to do something else
          //*****************************************************

          Sleep(15);

     }

Done:
     if (!S_Screen)
          printf("\rProcessed %d output values.", totalbytes);

     //*****************************************************
     // Close the file
     //*****************************************************

     if (O_OutputFilename[0])
          fclose(fout);

     //*****************************************************
     // Stop the extraction process
     //*****************************************************

     StopExtraction();

     if (kbhit()) getch();
     printf("\nPress any key to continue...");
     getch();

     return 0;
}
```

# 10 Low and Full Speed USB Data Extractor

The USB Data Extractor takes the real-time streaming data from the Full or Low Speed bus, formats it and allows you to save the data to disk or process it as it arrives.

## *10.1 USB Data Extractor Specifications*

- Continuous Real-Time Data Streaming
- One USB Bus running at Low (1.5Mbps) or Full Speed (12Mbps) USB (not High Speed)
- TTL Level inputs (**0-5V max**, Vih = 2.0V, Vil = 0.8V)
- Time Stamp for each packet
- Output to Text File*
- Output to Screen*
- Comma, Space, or Newline Delimited files
- Packet filter on Device Address, and/or Endpoint
- Output File Viewer (including binary, text, search and export functions)
- Extractor API libraries interface directly to your own software to further process the extracted data. Any language that supports calls to DLLs is supported.

* - output bandwidths are dependent on PC USB hardware, hard disk and/or screen throughput.

## *10.2 Hardware Setup*

To use the Data Extractor you need to connect the USBee AX-Pro Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

***Please note that the USBee AX-Pro Test Pod inputs are strictly 0-5V levels. Any voltage outside this range on the signals will damage the pod and may damage your hardware. If your system uses different voltage levels, you must buffer the signals externally to the USBee AX-Pro Test Pod before connecting the signals to the unit.***

The USB Bus Data Extractor uses signal 0 and signal 1 as the DPlus and DMinus lines of the USB bus. Connect these signals to the USB bus using the test clips provided. Connect the GND line to the digital ground of your system.

## *10.3 Extractor Command Line Program*

The USB Bus Data Extractor includes a Windows Command Prompt executable that lets you operate the Data Extractor without writing any software. The program is executed in a Command Prompt window and is configured using command line arguments. The extracted data is then stored to disk or outputted to the screen depending on these parameters.

To run the Data Extractor:

1) Install the USBee AX-Pro software on your PC
2) Install the Data Extractor software on your PC
3) Plug in your USBee AX-Pro Test Pod into your PC using a USB 2.0 High Speed Port
4) Open a Windows Command Prompt window by clicking Start, All Programs, Accessories, Command Prompt.
5) Change the working directory to the Data Extractor directory
6) (“cd \program files\USBee Data Extractor\USB”)
7) Run the executable using the following command line arguments:

```
USBExtractor [-?SDHICGAB] [-R USBSpeed] [-Q NumberOfBytes] [-V
Timestamp] [-O filename] -P PodID
```

? - Display this help screen

**USBee AX-Pro Pod to Use**

P - Pod ID (required)

**Output Location Flags**

O - Output to filename (default off)
S - Output to the screen (default off)

**When to Quit Flags**

Q - Number of output values (default = until keypress)

**Input Format Flags**

R - Bus Speed (0=Low Speed USB, 1=Full Speed USB)

**Output Format Flags**

A - All Packet Fields are output (default)
B – Only Data Bytes are output
D - Decimal Text Values ("49")
H - Hex Text Values ("31") default
I - Binary Values (49)
C - Comma Delimited
G - Space Delimited (default)

**Timestamps**

V - Timestamps (0=off, 1=each packet start)

## *10.4 Extractor API*

The Data Extractor is implemented using a Windows DLL that interfaces to the existing USBee AX-Pro DLL and drivers. This DLL can be called using any software language that supports calls to DLLs. Below are the details of this DLL interface and the routines that are available for your use.

### 10.4.1 DLL filename:

`usbexUSB.dll in \Windows\System32`

### 10.4.2 DLL Exported Functions and parameters

**ExtractionBufferCount** – Returns the number of bytes that have been extracted from the data stream so far and are available to read using GetNextData.

```
CWAV_EXPORT unsigned long CWAV_API ExtractionBufferCount(void)
```

Returns:
0 – No data to read yet
other – number of bytes available to read

**GetNextData** – Copies the extracted data from the extractor into your working buffer

```
CWAV_EXPORT char CWAV_API GetNextData(unsigned char *buffer, unsigned
long length);
```

buffer:
pointer to where you want the extracted data to be placed
length:
number of bytes you want to read from the extraction DLL
Returns:
0 – No data to read yet
1 – Data was copied into the buffer

**StartExtraction** – Starts the Data Extraction with the given parameters.

```
CWAV_EXPORT int CWAV_API StartExtraction(unsigned long PodNumber,
unsigned char Speed, unsigned char All, unsigned char Decimal, unsigned
char Hex, unsigned char Binary, unsigned char Comma, unsigned char
Space, unsigned char Timestamps)
```

PodNumber:
Pod ID on the back of the USBee AX-Pro Test Pod
Speed:
0 = Low Speed
1 = Full Speed
All:
0 – Only the data payload bytes are returned
1 – All USB packet fields are returned
Decimal:
1 – Decimal Values (text) are output for the data bytes
Hex:
1 – Hex Values (text) are output for the data bytes
Binary:
1 – All data is in binary form, not text
Comma:
1 – Commas are placed between each field/data byte
Space:
1 – Spaces are placed between each field/data byte
Timestamp:
1 – Print Timestamps at the start of each packet
Returns:
1 – if Start was successful
0 – if Pod failed initialization

**StopExtraction** – Stops the extraction in progress

```
CWAV_EXPORT int CWAV_API StopExtraction( void );
```

Returns:
1 – always

**ExtractBufferOverflow** – Returns the state of the overflow conditions

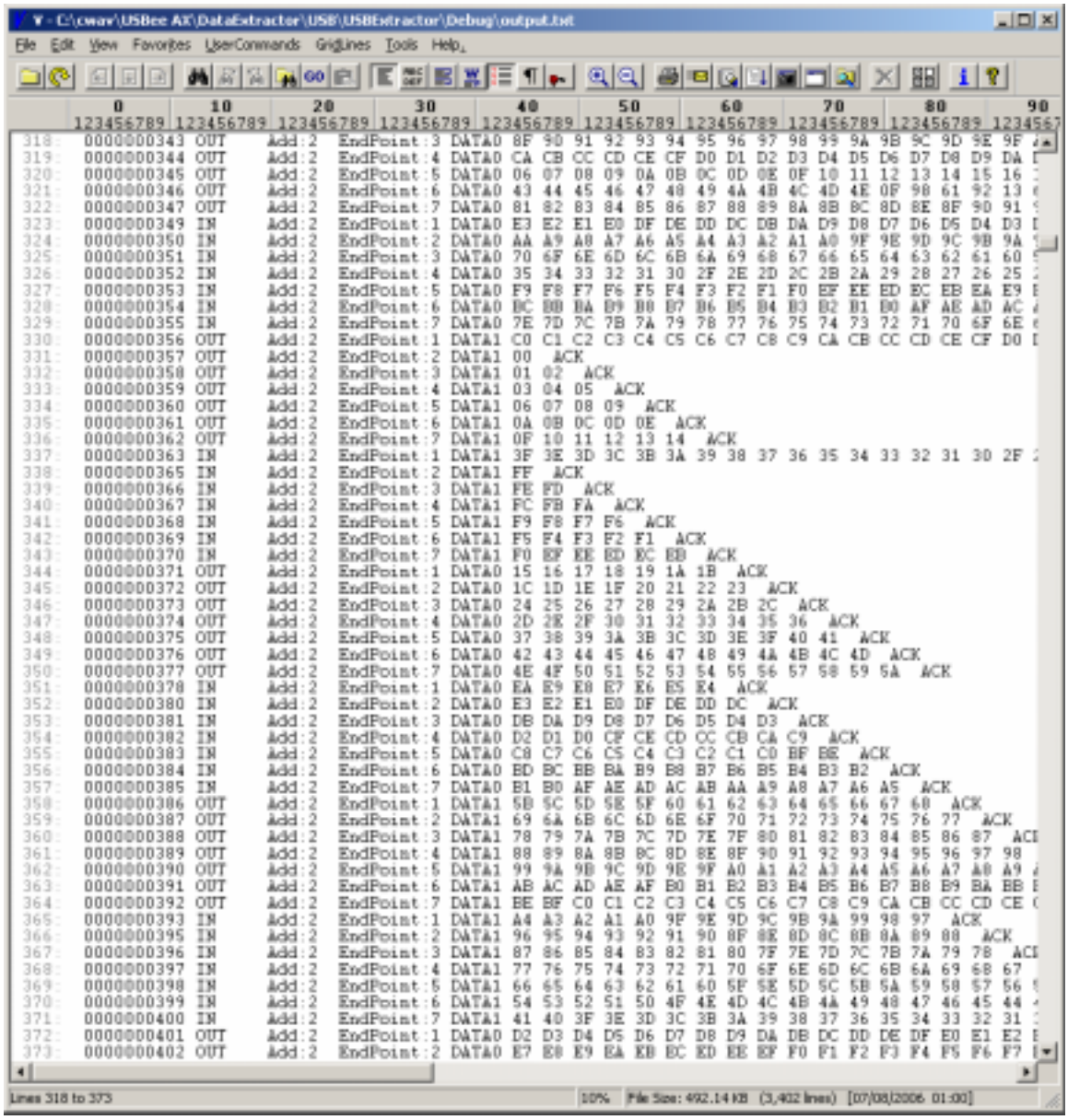
```
CWAV_EXPORT char CWAV_API ExtractBufferOverflow(void);
```

Return:
0 – No overflow
1 – Overflow Occurred. ExtractorBuffer Overflow condition cleared.
2 – Overflow Occurred. Raw Stream Buffer Overflow

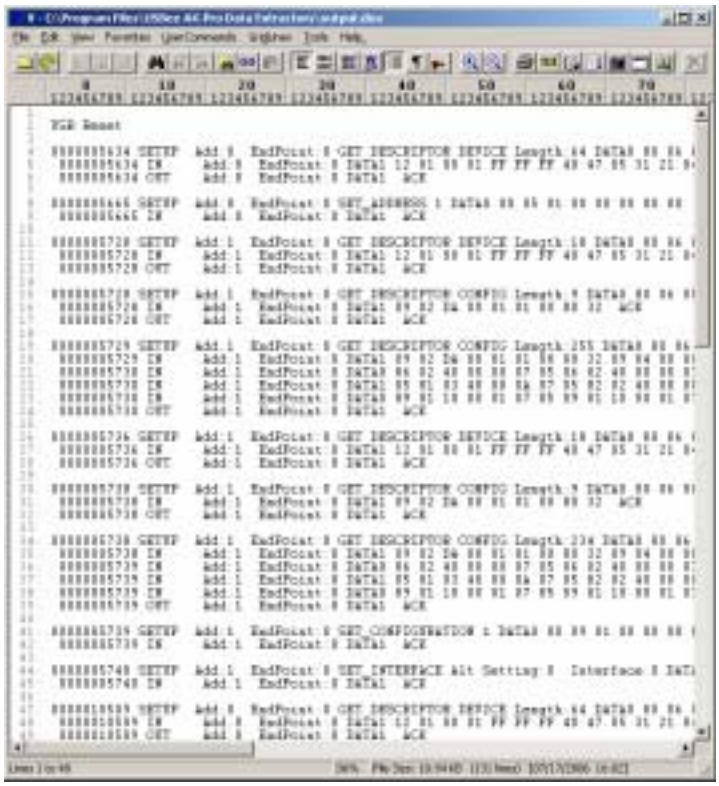
## 10.4.3 Extraction Data Format

The GetNextData routine gets a series of bytes that represent the extracted data stream and places these bytes into the buffer pointed to by the *buffer parameter.

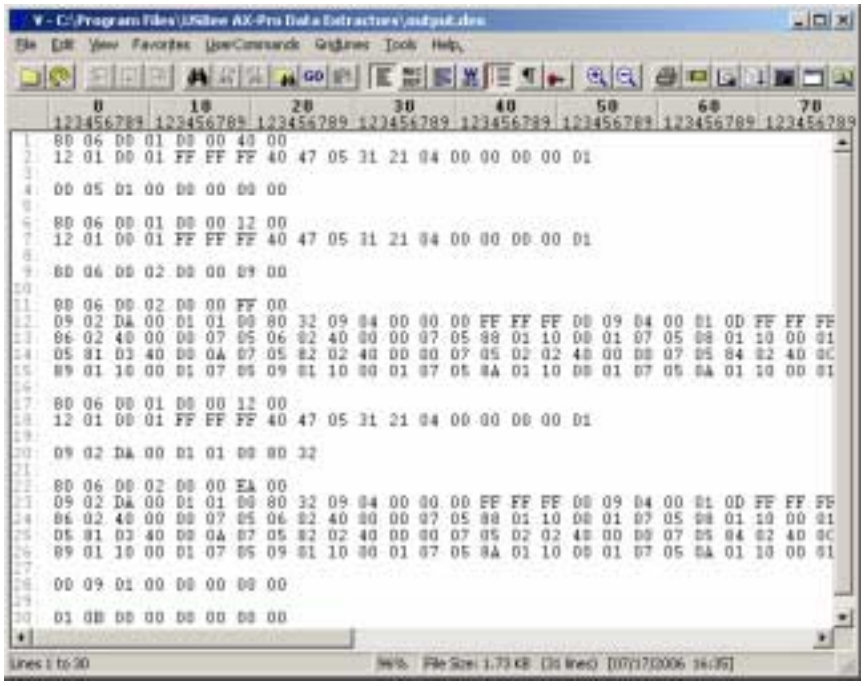
The USB Bus Extractor DLL sends the extracted data through the *buffer in the requested form based on the parameters in the StartExtraction call. For example, if Binary is set to a 0, then the *buffer will receive the binary bytes that make up the data stream. If Hex is set to a 1, the *buffer will contain a text string which is the data of the USB traffic in Hex text form, separated by any specified delimiters.



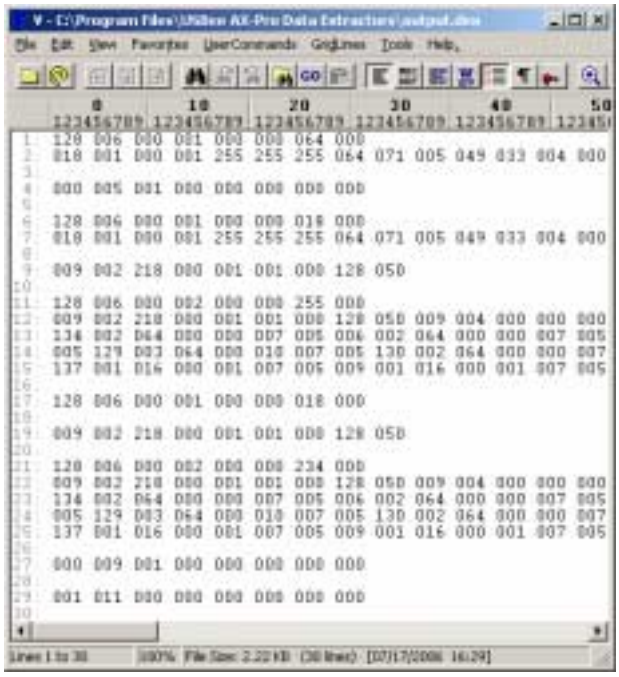
```
USBExtractor  -O output.dex -P 3209 -G -Q 10000 -R 1 -A -H -V 1
```



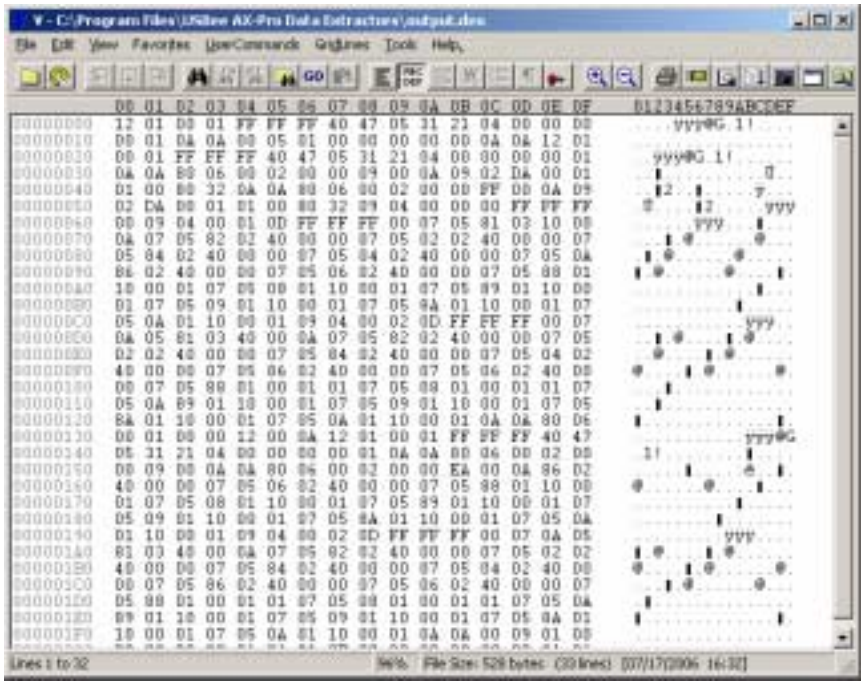
```
USBExtractor  -O output.dex -P 3209 -G -Q 10000 -R 1 -B
```



```
USBExtractor  -O output.dex -P 3209 -G -Q 10000 -R 1 -A -H -V 1 -B –D
```



```
USBExtractor  -O output.dex -P 3209 -G -Q 10000 –R 1 -B -I
```



## 10.4.4 Example Source Code

```
//*****************************************************
// USBee AX-Pro Data Extractor
// USB Bus Extractor Example Program
// Copyright 2006, CWAV All Rights Reserved.
//*****************************************************

#include "stdafx.h"
#include "stdio.h"
#include "conio.h"
#include "windows.h"
#include <fcntl.h>
#include <io.h>
#include <stdlib.h>
#include <stdio.h>

#define MAJOR_REV 1
#define MINOR_REV 0

//*****************************************************
// Declare the Extractor DLL API routines
//*****************************************************

#define CWAV_API __stdcall
#define CWAV_IMPORT __declspec(dllimport)

CWAV_IMPORT int CWAV_API StartExtraction(unsigned long PodNumber, unsigned char Speed, unsigned char All, unsigned char
Decimal, unsigned char Hex, unsigned char Binary, unsigned char Comma, unsigned char Space, unsigned char Timestamps,
unsigned int Endpoint, unsigned int Device)  ;
CWAV_IMPORT char CWAV_API GetNextData(unsigned char *buffer, unsigned long length);
CWAV_IMPORT int CWAV_API StopExtraction( void );
CWAV_IMPORT char CWAV_API ExtractBufferOverflow(void);
CWAV_IMPORT unsigned long CWAV_API ExtractionBufferCount(void);

//*****************************************************
// Define the working buffer
//*****************************************************

#define WORKING_BUFFER_SIZE   (65536*8)
unsigned char tempbuffer[WORKING_BUFFER_SIZE];

// Command Line Parameter Settings
unsigned long P_PodID = 0;
unsigned char O_OutputFilename[256] = {0};
unsigned char S_Screen = FALSE;
unsigned char A_All = TRUE;
unsigned char B_DataOnly = FALSE;
unsigned char D_DecimalTextValues = FALSE;
unsigned char H_HexTextValues = TRUE;
unsigned char I_BinaryValues = FALSE;
unsigned char C_CommaDelimited = FALSE;
unsigned char G_SpaceDelimited = TRUE;
unsigned long Q_NumberOfBytes = 0;
unsigned long R_Speed = 1;                    // Full Speed
unsigned long V_Timestamps = TRUE;


void DisplayHelp(void)
{
     fprintf(stdout,"\nUSBExtractor [-?SDHICGAB] [-R USBSpeed] [-Q NumberOfBytes] [-V Timestamp] [-O filename] -P
PodID\n");

     fprintf(stdout,"\n    ?  - Display this help screen\n");

     fprintf(stdout,"\n  USBee AX-Pro Pod to Use\n");

     fprintf(stdout,"    P  - Pod ID (required)\n");

     fprintf(stdout,"\n  Output Location Flags\n");

     fprintf(stdout,"    O  - Output to filename (default off)\n");
     fprintf(stdout,"    S  - Output to the screen (default off)\n");

     fprintf(stdout,"\n  When to Quit Flags\n");

     fprintf(stdout,"    Q  - Number of output values (default = until keypress)\n");

     fprintf(stdout,"\n  Input Format Flags\n");

     fprintf(stdout,"    R  - Bus Speed (0=Low Speed USB, 1=Full Speed USB)\n");

     fprintf(stdout,"\n  Output Number Format Flags\n");

     fprintf(stdout,"    A  - All Packet Fields are output (default)\n");
     fprintf(stdout,"    B  - Only data bytes are output\n");
     fprintf(stdout,"    D  - Decimal Text Values (\"49\")\n");
     fprintf(stdout,"    H  - Hex Text Values (\"31\") default\n");
     fprintf(stdout,"    I  - Binary Values (49)\n");
     fprintf(stdout,"    C  - Comma Delimited\n");
     fprintf(stdout,"    G  - Space Delimited (default)\n");
```

```
     fprintf(stdout,"    V  - Timestamps (0=off(default),1=Timestamp on\n");


}

void Error(char *err)
{
     fprintf(stderr,"Error: ");
     fprintf(stderr,"%s\n",err);
     exit(2);
}


//*****************************************************
// Parse all of the command line options
//*****************************************************
void ParseCommandLine(int argc, char *argv[])
{
     BOOL cont;
     int       i,j;
     DWORD WordExample;
     BYTE ByteExample;

     for(i=1; i < argc; ++i)
     {
          if((argv[i][0] == '-') || (argv[i][0] == '/'))
          {
               cont = TRUE;
               for(j=1;argv[i][j] && cont;++j)     // Cont flag permits multiple commands in a single argv (like -AR)
                    switch(toupper(argv[i][j]))
                    {
                         case 'P':
                              P_PodID = (WORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'O':
                              strcpy((char*)O_OutputFilename, argv[++i]);
                              cont = FALSE;
                              break;
                         case '?':
                              DisplayHelp();
                              exit(0);

                              break;
                         case 'S':
                              S_Screen = TRUE;
                              break;
                         case 'A':
                              A_All = TRUE;
                              B_DataOnly = FALSE;
                              break;
                         case 'B':
                              A_All = FALSE;
                              B_DataOnly = TRUE;
                              break;
                         case 'D':
                              D_DecimalTextValues = TRUE;
                              H_HexTextValues = FALSE;
                              break;
                         case 'H':
                              H_HexTextValues = TRUE;
                              break;
                         case 'I':
                              I_BinaryValues = TRUE;
                              H_HexTextValues = FALSE;
                              break;
                         case 'C':
                              C_CommaDelimited = TRUE;
                              G_SpaceDelimited = FALSE;
                              break;
                         case 'G':
                              G_SpaceDelimited = TRUE;
                              break;
                         case 'Q':
                              Q_NumberOfBytes = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'V':
                              V_Timestamps = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'R':
                              R_Speed = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'w':
                              WordExample = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'b':
                              ByteExample = (BYTE)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         default:
```

```
                              DisplayHelp();
                              fprintf(stdout,"\nCommand line switch %c not recognized\n",toupper(argv[i][j]));
                              Error("Invalid Command Line Switch");
                              exit(0);
                         }
               }
          }


          // Now check to see if they make sense
          if (P_PodID == 0)
          {
               DisplayHelp();
               Error("No Pod Number Specified");
          }

}


//*****************************************************
// Main Entry Point.  The program starts here.
//*****************************************************

int main(int argc, char* argv[])
{
     int RetValue;
     unsigned long totalbytes = 0;
     char *outputstr = new char [256];
     unsigned long ByteCounter = 0;
     unsigned long OutputValue;

     printf("USBee AX Data Extractor\n");
     printf("USB Bus Extractor Version %d.%d\n", MAJOR_REV, MINOR_REV);

     // Parse out the command line options
     ParseCommandLine( argc, argv );

     //*****************************************************
     // Open up a file to store extracted data into
     //*****************************************************

     FILE *fout;
     if (O_OutputFilename[0])
     {
          if (I_BinaryValues)
               fout = fopen((char*)O_OutputFilename, "wb");
          else
               fout = fopen((char*)O_OutputFilename, "w");
     }

     //*****************************************************
     // Start the USBee AX Pod extracting the data we want
     //*****************************************************

     int Endpoint = 999;
     int Device = 999;

     RetValue = StartExtraction(P_PodID, R_Speed, A_All, D_DecimalTextValues, H_HexTextValues, I_BinaryValues,
C_CommaDelimited, G_SpaceDelimited, V_Timestamps, Endpoint, Device)   ;

     if (RetValue == 0)
     {
          printf("Startup failed.  Is the USBee AX-Pro connected and is the PodNumber correct?\n");
          printf("Press any key to continue...");
          getch();
          return(0);
     }


     //*****************************************************
     // Loop and do something with the collected data
     //*****************************************************

     char OldSignal = 99;

     int KeepLooping = TRUE;
     while(KeepLooping)       // Do this forever until we tell it to stop by pressing a key
     {

          if (kbhit())
          {
               KeepLooping = FALSE;          // Stop the processing loop
               StopExtraction();             // Stop the streaming of data from the USBee
          }

          //*****************************************************
          // If there is data that has come in
          //*****************************************************
          int timeout = 0;
          while (unsigned long length = ExtractionBufferCount())
          {
               if (length > WORKING_BUFFER_SIZE)
                    length = WORKING_BUFFER_SIZE;

               //*****************************************************
```

```
               // Get the data into our local working buffer
               //*************************************************

               GetNextData( tempbuffer, length );

               totalbytes += length;

               if (O_OutputFilename[0])
                    fwrite(tempbuffer, length, 1,  fout);   // Write it to a file

               if (S_Screen)
                    fwrite(tempbuffer, length, 1,  stdout); // Write it to the screen

               if (Q_NumberOfBytes)
               {
                    if (Q_NumberOfBytes <= length)
                    {
                         goto Done;          // Done with that many bytes
                    }
                    Q_NumberOfBytes -= length;
               }

               if (timeout++ > 3  ) break;  // Let up once in a while to let the OS process
          }

          if (!S_Screen)
               printf("\rProcessed %d output values.", totalbytes);

          //*************************************************
          // Check to see if we have fallen behind too far
          //*************************************************

          int y = ExtractBufferOverflow();

          if (y == 1)
          {
               printf("\nExtractor Buffer Overflow.\nYour data is streaming too fast for your output settings.\nLower
your data rate or change to output binary files.\n");
               goto Done;
          }
          else if (y == 2)
          {
               printf("\nRaw Sample Buffer Overflow.\nYour data is streaming too fast for your output settings.\nLower
your data rate or change to output binary files.\n");
               goto Done;
          }

          //*************************************************
          // Give the OS a little time to do something else
          //*************************************************

          Sleep(15);

     }

Done:
     if (!S_Screen)
          printf("\rProcessed %d output values.", totalbytes);

     //*************************************************
     // Close the file
     //*************************************************

     if (O_OutputFilename[0])
          fclose(fout);

     //*************************************************
     // Stop the extraction process
     //*************************************************

     StopExtraction();

     if (kbhit()) getch();
     printf("\nPress any key to continue...");
     getch();

     return 0;
}
```

# 11 CAN Data Extractor

The CAN Bus Data Extractor takes the real-time streaming data from the CAN bus, formats it and allows you to save the data to disk or process it as it arrives.

## *11.1 CAN Data Extractor Specifications*

- Continuous Real-Time Data Streaming
- Monitors one CAN Bus
- TTL Level inputs (**0-5V max**, Vih = 2.0V, Vil = 0.8V) – intended to be used on the digital side of a CAN bus transceiver (such as the Microchip MCP2551)
- 11 or 29-bit identifier supported
- Time Stamp for each packet
- Output to Text File*
- Output to Screen*
- Comma or Space Delimited files
- Packet filter on Identifier
- Output File Viewer (including binary, text, search and export functions)
- Extractor API libraries interface directly to your own software to further process the extracted data. Any language that supports calls to DLLs is supported.

* - output bandwidths are dependent on PC USB hardware, hard disk and/or screen throughput.

## *11.2 Hardware Setup*

To use the Data Extractor you need to connect the USBee AX-Pro Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

***Please note that the USBee AX-Pro Test Pod inputs are strictly 0-5V levels. Any voltage outside this range on the signals will damage the pod and may damage your hardware. If your system uses different voltage levels, you must buffer the signals externally to the USBee AX-Pro Test Pod before connecting the signals to the unit.***

The CAN Bus Data Extractor connects to the digital side of your CAN bus transceiver and only needs to listen to the receiving side of the transceiver (such as the RxD pin on the Microchip MCP2551 CAN bus transceiver chip). Use signal 0 as the RxD data line and connect the GND line to the digital ground of your system. Connect these signals to the CAN bus transceiver IC using the test clips provided.

## *11.3 Extractor Command Line Program*

The CAN Bus Data Extractor includes a Windows Command Prompt executable that lets you operate the Data Extractor without writing any software. The program is executed in a Command Prompt window and is configured using command line arguments. The extracted data is then stored to disk or outputted to the screen depending on these parameters.

To run the Data Extractor:

1) Install the USBee AX-Pro software on your PC
2) Install the Data Extractor software on your PC
3) Plug in your USBee AX-Pro Test Pod into your PC using a USB 2.0 High Speed Port
4) Open a Windows Command Prompt window by clicking Start, All Programs, Accessories, Command Prompt.
5) Change the working directory to the Data Extractor directory
6) (“cd \program files\USBee Data Extractor\CAN”)
7) Run the executable using the following command line arguments:

```
CANExtractor [-?SDHICGAB] [-R CANSpeed] [-Q NumberOfBytes] [-V
Timestamp] [-O filename] [-M MaxID] [-N MinID] -P
```

? - Display this help screen

**USBee AX-Pro Pod to Use**

P - Pod ID (required)

**Output Location Flags**

O - Output to filename (default off)
S - Output to the screen (default off)

**When to Quit Flags**

Q - Number of output values (default = until keypress)

**Input Format Flags**

R - Bus Speed in bits/second (default = 250000)

**Output Format Flags**

A - All Packet Fields are output (default)
B – Only Data Bytes are output
D - Decimal Text Values ("49")
H - Hex Text Values ("31") default
I - Binary Values (49)
C - Comma Delimited
G - Space Delimited (default)
M - Maximum Identifier Filter
N - Minimum Identifier Filter

**Timestamps**

V - Timestamps (0=off, 1=each packet start)

## *11.4 Extractor API*

The Data Extractor is implemented using a Windows DLL that interfaces to the existing USBee AX-Pro DLL and drivers. This DLL can be called using any software language that supports calls to DLLs. Below are the details of this DLL interface and the routines that are available for your use.

### 11.4.1 DLL filename:

`usbexCAN.dll in \Windows\System32`

### 11.4.2 DLL Exported Functions and parameters

**ExtractionBufferCount** – Returns the number of bytes that have been extracted from the data stream so far and are available to read using GetNextData.

```
CWAV_EXPORT unsigned long CWAV_API ExtractionBufferCount(void)
```

Returns:
0 – No data to read yet
other – number of bytes available to read

**GetNextData** – Copies the extracted data from the extractor into your working buffer

```
CWAV_EXPORT char CWAV_API GetNextData(unsigned char *buffer, unsigned
long length);
```

buffer:
pointer to where you want the extracted data to be placed
length:
number of bytes you want to read from the extraction DLL
Returns:
0 – No data to read yet
1 – Data was copied into the buffer

**StartExtraction** – Starts the Data Extraction with the given parameters.

```
CWAV_EXPORT int CWAV_API StartExtraction(unsigned long PodNumber,
unsigned long Speed, unsigned char All, unsigned char Decimal, unsigned
char Hex, unsigned char Binary, unsigned char Comma, unsigned char
Space, unsigned char Timestamps, unsigned long MaxIDFilter, unsigned
long MinIDFilter)
```

PodNumber:
Pod ID on the back of the USBee AX-Pro Test Pod
Speed:
Bit rate of the CAN bus in bits per second
All:
0 – Only the data payload bytes are returned
1 – All CAN packet fields are returned
Decimal:
1 – Decimal Values (text) are output for the data bytes
Hex:
1 – Hex Values (text) are output for the data bytes
Binary:
1 – All data is in binary form, not text
Comma:
1 – Commas are placed between each field/data byte
Space:
1 – Spaces are placed between each field/data byte
Timestamp:
1 – Print Timestamps at the start of each packet
MaxIDFilter:
The Maximum Identifier to log (0xFFFFFFFF default)
MinIDFilter:
The Minimum Identifier to log (0 default)
Returns:
1 – if Start was successful
0 – if Pod failed initialization

**StopExtraction** – Stops the extraction in progress

```
CWAV_EXPORT int CWAV_API StopExtraction( void );
```

Returns:
1 – always

**ExtractBufferOverflow** – Returns the state of the overflow conditions

```
CWAV_EXPORT char CWAV_API ExtractBufferOverflow(void);
```

Return:

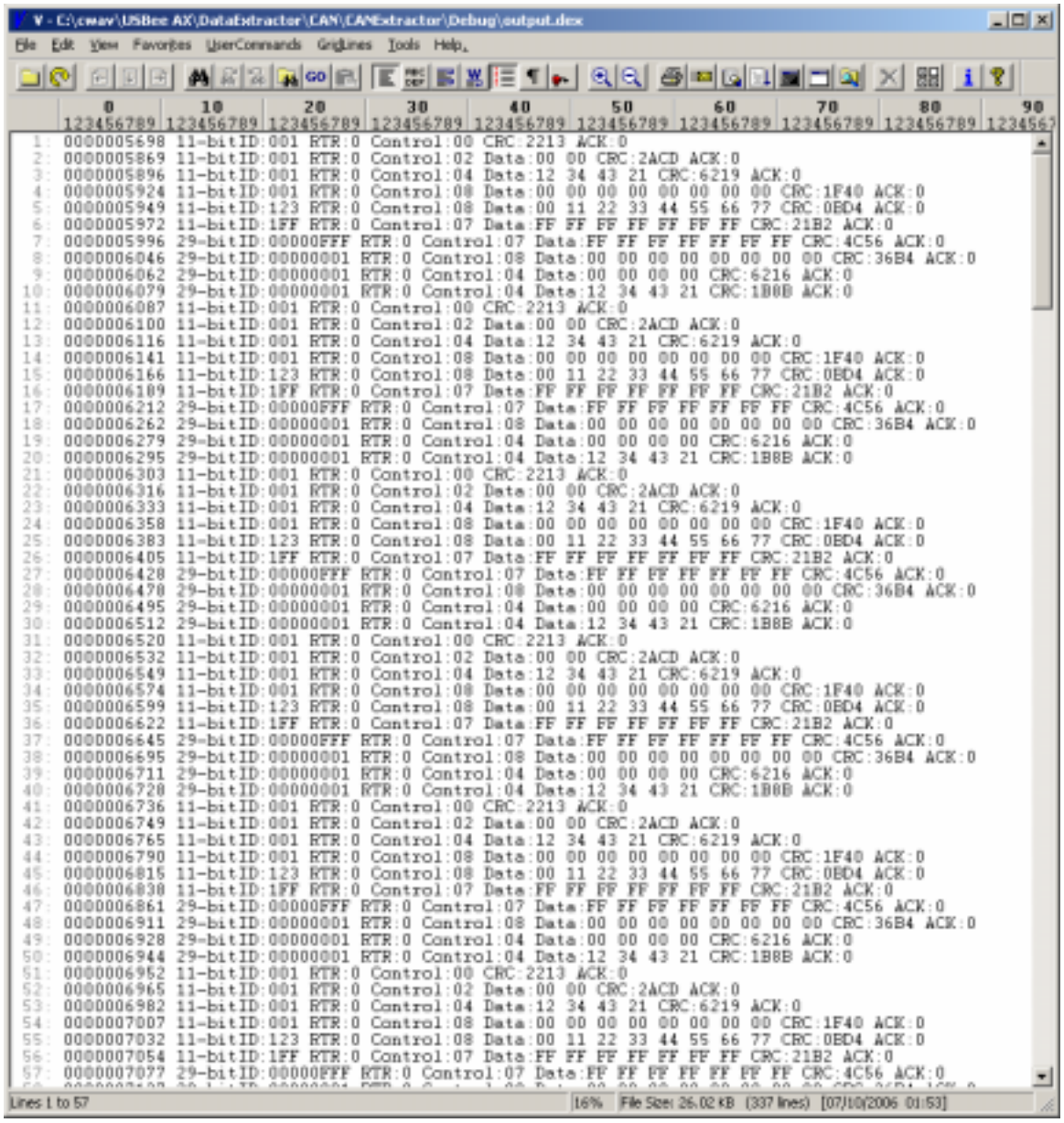
0 – No overflow
1 – Overflow Occurred. ExtractorBuffer Overflow condition cleared.
2 – Overflow Occurred. Raw Stream Buffer Overflow

### 11.4.3 Extraction Data Format

The GetNextData routine gets a series of bytes that represent the extracted data stream and places these bytes into the buffer pointed to by the *buffer parameter.

The CAN Bus Extractor DLL sends the extracted data through the *buffer in the requested form based on the parameters in the StartExtraction call. For example, if Binary is set to a 0, then the *buffer will receive the binary bytes that make up the data stream. If Hex is set to a 1, the *buffer will contain a text string which is the data of the CAN traffic in Hex text form, separated by any specified delimiters.

```
CANExtractor -O output.dex -S -P 3209 -Q 500000 -R 250000 -A -H -V 1
```



## 11.4.4 Example Source Code

```
//*****************************************************
// USBee AX-Pro Data Extractor
// CAN Bus Extractor Example Program
// Copyright 2006, CWAV All Rights Reserved.
//*****************************************************

#include "stdafx.h"
#include "stdio.h"
#include "conio.h"
#include "windows.h"
#include <fcntl.h>
#include <io.h>
#include <stdlib.h>
#include <stdio.h>

#define MAJOR_REV 1
#define MINOR_REV 0

//*****************************************************
// Declare the Extractor DLL API routines
//*****************************************************

#define CWAV_API __stdcall
#define CWAV_IMPORT __declspec(dllimport)

CWAV_IMPORT int CWAV_API StartExtraction(unsigned long PodNumber, unsigned long Speed, unsigned char All, unsigned char
Decimal, unsigned char Hex, unsigned char Binary, unsigned char Comma, unsigned char Space, unsigned char Timestamps,
unsigned long MaxID,unsigned long MinID);
CWAV_IMPORT char CWAV_API GetNextData(unsigned char *buffer, unsigned long length);
CWAV_IMPORT int CWAV_API StopExtraction( void );
CWAV_IMPORT char CWAV_API ExtractBufferOverflow(void);
CWAV_IMPORT unsigned long CWAV_API ExtractionBufferCount(void);

//*****************************************************
// Define the working buffer
//*****************************************************

#define WORKING_BUFFER_SIZE   (65536*8)
unsigned char tempbuffer[WORKING_BUFFER_SIZE];

// Command Line Parameter Settings
unsigned long P_PodID = 0;
unsigned char O_OutputFilename[256] = {0};
unsigned char S_Screen = FALSE;
unsigned char A_All = TRUE;
unsigned char B_DataOnly = FALSE;
unsigned char D_DecimalTextValues = FALSE;
unsigned char H_HexTextValues = TRUE;
unsigned char I_BinaryValues = FALSE;
unsigned char C_CommaDelimited = FALSE;
unsigned char G_SpaceDelimited = FALSE;
unsigned long Q_NumberOfBytes = 0;
unsigned long R_Speed = 250000;
unsigned long V_Timestamps = TRUE;
unsigned long M_ID = 0xFFFFFFFF;
unsigned long N_ID = 0;


void DisplayHelp(void)
{
     fprintf(stdout,"\nCANExtractor [-?SDHICGAB] [-R CANSpeed] [-Q NumberOfBytes] [-V Timestamp] [-O filename] [-M
MaxID] [-N MinID] -P PodID\n");

     fprintf(stdout,"\n    ?  - Display this help screen\n");

     fprintf(stdout,"\n  USBee AX-Pro Pod to Use\n");

     fprintf(stdout,"    P  - Pod ID (required)\n");

     fprintf(stdout,"\n  Output Location Flags\n");

     fprintf(stdout,"    O  - Output to filename (default off)\n");
     fprintf(stdout,"    S  - Output to the screen (default off)\n");

     fprintf(stdout,"\n  When to Quit Flags\n");

     fprintf(stdout,"    Q  - Number of output values (default = until keypress)\n");

     fprintf(stdout,"\n  Input Format Flags\n");

     fprintf(stdout,"    R  - Bus Speed in bits/second (default = 250000)\n");

     fprintf(stdout,"\n  Output Number Format Flags\n");

     fprintf(stdout,"    A  - All Packet Fields are output (default)\n");
     fprintf(stdout,"    B  - Only data bytes are output\n");
     fprintf(stdout,"    D  - Decimal Text Values (\"49\")\n");
     fprintf(stdout,"    H  - Hex Text Values (\"31\") default\n");
     fprintf(stdout,"    I  - Binary Values (49)\n");
     fprintf(stdout,"    C  - Comma Delimited\n");
```

```
     fprintf(stdout,"    G  - Space Delimited (default)\n");
     fprintf(stdout,"    V  - Timestamps (0=off(default),1=Timestamp on\n");
     fprintf(stdout,"    M  - Maximum Identifier Filter\n");
     fprintf(stdout,"    N  - Minimum Identifier Filter\n");


}

void Error(char *err)
{
     fprintf(stderr,"Error: ");
     fprintf(stderr,"%s\n",err);
     exit(2);
}


//*****************************************************
// Parse all of the command line options
//*****************************************************
void ParseCommandLine(int argc, char *argv[])
{
     BOOL cont;
     int       i,j;
     DWORD WordExample;
     BYTE ByteExample;

     for(i=1; i < argc; ++i)
     {
          if((argv[i][0] == '-') || (argv[i][0] == '/'))
          {
               cont = TRUE;
               for(j=1;argv[i][j] && cont;++j)     // Cont flag permits multiple commands in a single argv (like -AR)
                    switch(toupper(argv[i][j]))
                    {
                         case 'P':
                              P_PodID = (WORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'O':
                              strcpy((char*)O_OutputFilename, argv[++i]);
                              cont = FALSE;
                              break;
                         case '?':
                              DisplayHelp();
                              exit(0);

                              break;
                         case 'S':
                              S_Screen = TRUE;
                              break;
                         case 'A':
                              A_All = TRUE;
                              B_DataOnly = FALSE;
                              break;
                         case 'B':
                              A_All = FALSE;
                              B_DataOnly = TRUE;
                              break;
                         case 'D':
                              D_DecimalTextValues = TRUE;
                              H_HexTextValues = FALSE;
                              break;
                         case 'H':
                              H_HexTextValues = TRUE;
                              break;
                         case 'I':
                              I_BinaryValues = TRUE;
                              H_HexTextValues = FALSE;
                              break;
                         case 'C':
                              C_CommaDelimited = TRUE;
                              G_SpaceDelimited = FALSE;
                              break;
                         case 'G':
                              G_SpaceDelimited = TRUE;
                              break;
                         case 'Q':
                              Q_NumberOfBytes = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'V':
                              V_Timestamps = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'R':
                              R_Speed = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'M':
                              M_ID = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'N':
                              N_ID = (DWORD)strtol(argv[++i],NULL,0);
```

```
                              cont = FALSE;
                              break;
                         case 'w':
                              WordExample = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'b':
                              ByteExample = (BYTE)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         default:
                              DisplayHelp();
                              fprintf(stdout,"\nCommand line switch %c not recognized\n",toupper(argv[i][j]));
                              Error("Invalid Command Line Switch");
                              exit(0);
                    }
               }
          }


          // Now check to see if they make sense
          if (P_PodID == 0)
          {
               DisplayHelp();
               Error("No Pod Number Specified");
          }

}


//**************************************************
// Main Entry Point.  The program starts here.
//**************************************************

int main(int argc, char* argv[])
{
     int RetValue;
     unsigned long totalbytes = 0;
     char *outputstr = new char [256];
     unsigned long ByteCounter = 0;
     unsigned long OutputValue;

     printf("USBee AX Data Extractor\n");
     printf("CAN Bus Extractor Version %d.%d\n", MAJOR_REV, MINOR_REV);

     // Parse out the command line options
     ParseCommandLine( argc, argv );

     //**************************************************
     // Open up a file to store extracted data into
     //**************************************************

     FILE *fout;
     if (O_OutputFilename[0])
     {
          if (I_BinaryValues)
               fout = fopen((char*)O_OutputFilename, "wb");
          else
               fout = fopen((char*)O_OutputFilename, "w");
     }

     //**************************************************
     // Start the USBee AX Pod extracting the data we want
     //**************************************************

     int Endpoint = 999;
     int Device = 999;

     RetValue = StartExtraction(P_PodID, R_Speed, A_All, D_DecimalTextValues, H_HexTextValues, I_BinaryValues,
C_CommaDelimited, G_SpaceDelimited, V_Timestamps, M_ID, N_ID)    ;

     if (RetValue == 0)
     {
          printf("Startup failed.  Is the USBee AX-Pro connected and is the PodNumber correct?\n");
          printf("Press any key to continue...");
          getch();
          return(0);
     }


     //**************************************************
     // Loop and do something with the collected data
     //**************************************************

     char OldSignal = 99;

     int KeepLooping = TRUE;
     while(KeepLooping)       // Do this forever until we tell it to stop by pressing a key
     {

          if (kbhit())
          {
               KeepLooping = FALSE;          // Stop the processing loop
               StopExtraction();             // Stop the streaming of data from the USBee
          }
```

```
          //*****************************************************
          // If there is data that has come in
          //*****************************************************
          int timeout = 0;
          while (unsigned long length = ExtractionBufferCount())
          {
               if (length > WORKING_BUFFER_SIZE)
                    length = WORKING_BUFFER_SIZE;

               //*****************************************************
               // Get the data into our local working buffer
               //*****************************************************

               GetNextData( tempbuffer, length );

               totalbytes += length;

               if (O_OutputFilename[0])
                    fwrite(tempbuffer, length, 1,  fout);   // Write it to a file

               if (S_Screen)
                    fwrite(tempbuffer, length, 1,  stdout); // Write it to the screen

               if (Q_NumberOfBytes)
               {
                    if (Q_NumberOfBytes <= length)
                    {
                         goto Done;          // Done with that many bytes
                    }
                    Q_NumberOfBytes -= length;
               }

               if (timeout++ > 3  ) break;  // Let up once in a while to let the OS process
          }

          if (!S_Screen)
               printf("\rProcessed %d output values.", totalbytes);

          //*****************************************************
          // Check to see if we have fallen behind too far
          //*****************************************************

          int y = ExtractBufferOverflow();

          if (y == 1)
          {
               printf("\nExtractor Buffer Overflow.\nYour data is streaming too fast for your output settings.\nLower
your data rate or change to output binary files.\n");
               goto Done;
          }
          else if (y == 2)
          {
               printf("\nRaw Sample Buffer Overflow.\nYour data is streaming too fast for your output settings.\nLower
your data rate or change to output binary files.\n");
               goto Done;
          }

          //*****************************************************
          // Give the OS a little time to do something else
          //*****************************************************

          Sleep(15);

     }

Done:
     if (!S_Screen)
          printf("\rProcessed %d output values.", totalbytes);

     //*****************************************************
     // Close the file
     //*****************************************************

     if (O_OutputFilename[0])
          fclose(fout);

     //*****************************************************
     // Stop the extraction process
     //*****************************************************

     StopExtraction();

     if (kbhit()) getch();
     printf("\nPress any key to continue...");
     getch();

     return 0;
}
```

Printed in the USA

Version 1.0