Signal Processing and Analysis (vtSPA) APIs

Version 1.2

Note: VIRTINS TECHNOLOGY reserves the right to make modifications to this document at any time without notice. This document may contain typographical errors.

TABLE OF CONTENTS

1. INTRODUCTION 3

2. VTSPA APIS 3

2.1 TIME DOMAIN APIS 3
2.1.1 SPA_MaxMinMeanRMS 3
2.1.2 SPA_FrequencyCounter 4
2.1.3 SPA_Windowing 5
2.2 FREQUENCY DOMAIN APIS 6
2.2.1 SPA_FFT 6
2.2.2 SPA_Nto2NFFT 7
2.2.3 SPA_SpectrumAnalysisSignalChannel 7
2.2.4 SPA_SpectrumAnalysisDualChannel 9
2.2.5 SPA_PeakFrequencyDetection 12
2.2.6 SPA_MFCC 12
2.2.7 SPA_DTW 14
2.2.8 SPA_THD 14
2.3 GENERAL APIS 16
2.3.1 SPA_Unlock 16

3. VTSPA DEVELOPMENT GUIDE 17

3.1 FLOWCHARTS 17
3.2 BASIC FILES 17

4. SAMPLE PROGRAMS 18

4.1 TESTDAQ WRITTEN IN VISUAL C++ 6.0 18
4.2 TESTDAQ WRITTEN IN VISUAL C# 2012 19

1. Introduction

Virtins Technology's Signal Processing and Analysis (vtSPA) Application Programming Interfaces (APIs) provides a suite of generic APIs for data processing and analysis. It contains some unique features / algorithms originated and only available from Virtins Technology.

2. vtSPA APIs

2.1 Time Domain APIs

2.1.1 SPA_MaxMinMeanRMS

The SPA_MaxMinMeanRMS function calculates the maximum, minimum, mean, and RMS values of the input data.

```
int SPA_MaxMinMeanRMS(
double *DataInEU,
DWORD DataCount,
double *Max,
double *Min,
double *Mean,
double *RMS
);
```

Parameters

DataInEU

Pointer to the data to be analyzed (input)

DataCount

Number of data to be analyzed (input)

Max

Pointer to the maximum value of the analyzed data (output)

Min

Pointer to the minimum value of the analyzed data (output)

Mean

Pointer to the mean value of the analyzed data (output)

RMS

Pointer to the RMS value of the analyzed data (output)

Return Values

Reserved.

2.1.2 SPA_FrequencyCounter

The SPA_FrequencyCounter function calculates the frequency, total count, RPM, duty cycle, cycle RMS, cycle mean values of the input data using a software frequency counter algorithm.

```
int SPA_FrequencyCounter(
double *DataInEU,
DWORD DataCount,
double SamplingFrequency,
double Max,
double Min,
double TriggerLevelPercent,
double TriggerHysteresisPercent,
double FrequencyDivider,
double * FrequencyCount,
double * TotalCount,
double * RPM,
double * DutyCycle,
double * CycleRMS,
double * CycleMean
);
```

Parameters

DataInEU

Pointer to the data to be analyzed (input)

DataCount

Number of data to be analyzed (input)

SamplingFrequency

Sampling frequency of the data to be analyzed (input)

Max

Maximum value of the data to be analyzed. It can be obtained through SPA_MaxMinMeanRMS( ). (input)

Min

Minimum value of the data to be analyzed. It can be obtained through SPA_MaxMinMeanRMS( ). (input)

TriggerLevelPercent

Specifies the trigger level percentage (-100%~100%) with regards to Max and Min for the software frequency counter. (input)

TriggerHysteresisPercent

Specifies the trigger hysteresis percentage (0%~100%) with regards to ½ of the difference of Max and Min for the software frequency counter which is equipped with Schmitt Trigger capability. (input)

FrequencyDivider

Specifies the frequency dividing factor for the software frequency counter. (input)

FrequencyCount

Pointer to the counted frequency. (output)

TotalCount

Pointer to the counted total count. (output)

RPM

Pointer to the counted RPM value. (output)

DutyCycle

Pointer to the calculated duty cycle value. (output)

CycleRMS

Pointer to the calculated cycle RMS value. (output)

CycleMean

Pointer to the calculated cycle mean value. (output)

Return Values

Reserved.

2.1.3 SPA_Windowing

The SPA_Windowing function imposes window on the input data.

```
double SPA_Windowing(
double *DataInEU,
int WindowType,
DWORD DataCount,
BOOL FilterFlag
);
```

Parameters

DataInEU

Pointer to the data to be processed (input & output)

WindowType

Specifies the window function type:

0: Rectangle 1: Triangle (or Fejer) 2: Hanning 3: Hamming 4: Blackman
5: Exact Blackman 6: Blackman Harris 7: Blackman Nuttall 8: Flat Top
9: Exponetial 0.1 10: Gaussian 2.5 11: Gaussian 3.0 12: Gaussian 3.5
13: Welch (or Riesz) 14: Cosine 1.0 15: Cosine 3.0 16: Cosine 4.0
17: Cosine 5.0 18: Riemann (or Lanczos) 19: Parzen 20: Tukey 0.25
21: Tukey 0.50 22: Tukey 0.75 23: Bohman 24: Poisson 2.0 25: Poisson 3.0
26: Poisson 4.0 27: Hanning-Poisson 0.5 28: Hanning-Poisson 1.0
29: Hanning-Poisson 2.0 30: Cauchy 3.0 31: Cauchy 4.0 32: Cauchy 5.0
33: Bartlett-Hann 34: Kaiser 0.5 35: Kaiser 1 36: Kaiser 2 37: Kaiser 3
38: Kaiser 4 39: Kaiser 40: Kaiser 6 41: Kaiser 7 42: Kaiser 8 43: Kaiser 9
44: Kaiser 10 45: Kaiser 11 46: Kaiser 12 47: Kaiser 13 48: Kaiser 14
49: Kaiser 15 50: Kaiser 16 51: Kaiser 17 52: Kaiser 18 53: Kaiser 19

54: Kaiser 20.
The value behind the window name is the parameter value of that window. Please refer to relevant reference books for the definition of these window functions. A good example can be found at:

http://www.virtins.com/doc/D1003/Evaluation_of_Various_Window_Functions_using_Multi-Instrument_D1003.pdf
(input)

DataCount

Number of data to be processed. (input)

FilterFlag

False: Window for spectral analysis (asymmetric)

True: Window for digital filter design (symmetric)

(input)

Return Values

Total energy of the window function (=DataCount × [RMS of the window function]^2).

2.2 Frequency Domain APIs

2.2.1 SPA_FFT

The SPA_FFT function performs FFT or inverse FFT.

```
SPA_FFT (
double * xr,
double * xi,
long FFTSize,
int InverseFlag
);
```

Parameters

xr

Pointer to the real part of data. (input & output)

xi

Pointer to the imaginary part of data. (input & output)

FFTSize

Number of FFT points. It must be a power of 2. (input)

InverseFlag

0: FFT 1: iFFT

(input)

Return Values

Reserved.

2.2.2 SPA_Nto2NFFT

The SPA_Nto2NFFT function performs 2N-point FFT using N-point FFT. It is faster than performing 2N-point FFT directly. However, it does not support inverse FFT.

```
SPA_Nto2NFFT (
double * xr,
double * xi,
long FFTSize,
int InverseFlag
);
```

Parameters

xr

Pointer to the real part of data. (input & output)

xi

Pointer to the imaginary part of data. (input & output)

FFTSize

Number of FFT points. It must be a power of 2. Internally, FFT will be performed with ½ FFTSize. (input)

InverseFlag

Must be zero. (input)

Return Values

Reserved.

2.2.3 SPA_SpectrumAnalysisSignalChannel

The SPA_SpectrumAnalysisSignalChannel function performs single-channel analysis for amplitude spectrum, phase spectrum, auto correlation, depending on the analysis mode selected.

```
SPA_SpectrumAnalysisSignalChannel(
double* xr,
double * xi,
double * xp,
double * DataInEU,
long FFTSize,
DWORD DataCount,
double Mean,
int WindowType,
double WindowOverlapPercent,
int AnalysisMode
);
```


Parameters

xr

Pointer to the real part of FFT/iFFT intermediate / final result. (output)

xi

Pointer to the imaginary part of FFT/iFFT intermediate / final result. (output)

xp

Pointer to the spectrum analysis result. (output)

DataInEU

Pointer to the data to be analyzed. (input)

FFTSize

Number of FFT points. It must be a power of 2. (input)

DataCount

Number of data to be analyzed (input)

Mean

Mean value of the data to be analyzed. It can be obtained through SPA_MaxMinMeanRMS( ). (input)

It is used to remove the mean in the data in time domain before spectral analysis.

WindowType

Specifies the window function type. (refer to the same parameter in SPA_Windowing()). (input)

WindowOverlapPercent

Specifies the window overlap percentage. Its value should be equal to or greater than 0 but less than 1. (input)

AnalysisMode

Analysis mode. (input)

0: Amplitude Spectrum
xr: real part of FFT intermediate / final result (FFTSize points)
xi: imaginary part of FFT intermediate / final result (FFTSize points)
xp: RMS amplitude spectrum result (FFTSize / 2 + 1 points)

1: Phase Spectrum
xr: real part of FFT intermediate / final result (FFTSize points)
xi: imaginary part of FFT intermediate / final result (FFTSize points)
xp: phase spectrum result in degree (FFTSize / 2 + 1 points)

2: Auto Correlation
xp: auto correlation result (FFTSize - 1 points)

If *FFTSize* is greater than *DataCount*, then zeros will be padded at the end of *DataInEU* during FFT computation.

If *FFTSize* is less than *DataCount*, then *DataInEU* will be split into different segments with the size of each segment equal to *FFTSize*. The last segment of data will be dropped if its size is not equal to *FFTSize*. The final result will be obtained by averaging the FFT results from all segments. It should be noted that this approach is used for Amplitude Spectrum, Auto Correlation Function, except Phase Spectrum where only the first segment of data is used.

Return Values

Reserved.

2.2.4 SPA_SpectrumAnalysisDualChannel

The SPA_SpectrumAnalysisDualChannel function performs dual-channel analysis for amplitude spectrum, phase spectrum, auto correlation, cross correlation, coherence function, transfer function, impulse response, depending on the analysis mode selected.

```
SPA_SpectrumAnalysisDualChannel(
double* xr1,
double * xi1,
double* xr2,
double * xi2,
double * xp1,
double * xp2,
double * DataInEU1,
double * DataInEU2,
long FFTSize,
DWORD DataCount,
double Mean1,
double Mean2,
int WindowType,
double WindowOverlapPercent,
int AnalysisMode
);
```

Parameters

xr1

Pointer to the real part of FFT/iFFT intermediate / final result for Channel 1. (output)

xi1

Pointer to the imaginary part of FFT/iFFT intermediate / final result for Channel 1. (output)

xr2

Pointer to the real part of FFT/iFFT intermediate / final result for Channel 2. (output)

Xi2

Pointer to the imaginary part of FFT/iFFT intermediate / final result for Channel 2. (output)

xp1

Pointer to the spectrum analysis result for Channel 1. (output)

Xp2

Pointer to the spectrum analysis result for Channel 2. (output)

DataInEU1

Pointer to the data of Channel 1 to be analyzed. (input)

DataInEU2

Pointer to the data of Channel 2 to be analyzed. (input)

FFTSize

Number of FFT points. It must be a power of 2. (input)

DataCount

Number of data per channel to be analyzed (input)

Mean1

Mean value of Channel 1 data to be analyzed. It can be obtained through SPA_MaxMinMeanRMS( ). (input)

It is used to remove the mean in the data in time domain before spectral analysis.

Mean2

Mean value of Channel 2 data to be analyzed. It can be obtained through SPA_MaxMinMeanRMS( ). (input)

It is used to remove the mean in the data in time domain before spectral analysis.

WindowType

Specifies the window function type. (refer to the same parameter in SPA_Windowing()). (input)

WindowOverlapPercent

Specifies the window overlap percentage. Its value should be equal to or greater than 0 but less than 1. (input)

AnalysisMode

Analysis mode. (input)

0: Amplitude Spectrum
- xr1: real part of FFT intermediate / final result for Channel 1 (FFTSize points)
- xi1: imaginary part of FFT intermediate / final result for Channel 1 (FFTSize points)
- xr2: real part of FFT intermediate / final result for Channel 2 (FFTSize points)
- xi2: imaginary part of FFT intermediate / final result for Channel 2 (FFTSize points)

- xp1: RMS amplitude spectrum result for Channel 1 (FFTSize / 2 + 1 points)
- xp2: RMS amplitude spectrum result for Channel 2 (FFTSize / 2 + 1 points)

1: Phase Spectrum
- xr1: real part of FFT intermediate / final result for Channel 1 (FFTSize points)

xi1: imaginary part of FFT intermediate / final result for Channel 1 (FFTSize points)
xr2: real part of FFT intermediate / final result for Channel 2 (FFTSize points)
xi2: imaginary part of FFT intermediate / final result for Channel 2 (FFTSize points)

xp1: phase spectrum result in degree for Channel 1 (FFTSize / 2 + 1 points)
xp2: phase spectrum result in degree for Channel 2 (FFTSize / 2 + 1 points)

2: Auto Correlation
xp1: auto correlation result for Channel 1 (FFTSize - 1 points)
xp2: auto correlation result for Channel 2 (FFTSize - 1 points)

3: Cross Correlation
xp1: cross correlation result for Channel 1 (FFTSize - 1 points)
xp2: cross correlation result for Channel 2 (FFTSize - 1 points)

4: Coherence Function
xp1: coherence function result (FFTSize / 2 + 1 points)

5: Transfer Function
xp1: Gain result (FFTSize / 2 + 1 points)
xp2: Phase result (FFTSize / 2 + 1 points)

6: Impulse Response
xp1: impulse response result (FFTSize points)

If `FFTSize` is greater than `DataCount`, then zeros will be padded at the end of `DataInEU` during FFT computation.

If `FFTSize` is less than `DataCount`, then `DataInEU` will be split into different segments with the size of each segment equal to `FFTSize`. The last segment of data will be dropped if its size is not equal to `FFTSize`. The final result will be obtained by averaging the FFT results from all segments. It should be noted that this approach is used for Amplitude Spectrum, Auto Correlation Function, Cross Correlation Function, Coherence Function, Transfer function, and Impulse Response, except Phase Spectrum where only the first segment of data is used.

The following table listed the averaging method used for each analysis mode:

Amplitude Spectrum	Phase Spectrum	Auto Correlation	Cross Correlation	Coherence Function	Transfer Function	Impulse Response
Power Average	No	Power Average	Power Average	Power Average	Power Average	Power Average

Power Average: The averaging is performed in power spectrum, such as auto power spectrum or cross power spectrum, during the computing process. For amplitude spectrum, it is often called "RMS average".

Normal Average: arithmetic average.

Return Values

Reserved.

2.2.5 SPA_PeakFrequencyDetection

The SPA_PeakFrequencyDetection function detects the peak frequency as well as its RMS amplitude and phase from the RMS amplitude spectrum result.

```
SPA_PeakFrequencyDetection(
double * xr,
double * xi,
double * xp,
long FFTSize,
double SamplingFrequency,
double * PeakFrequency,
double *PeakFrequencyRMS,
double * PeakFrequencyPhase
);
```

Parameters

xr

Pointer to the real part of FFT/iFFT intermediate / final result. (input)

xi

Pointer to the imaginary part of FFT/iFFT intermediate / final result. (input)

xp

Pointer to the RMS amplitude spectrum result. (input)

FFTSize

Number of FFT points. It must be a power of 2. (input)

SamplingFrequency

Sampling frequency of the data to be analyzed (input)

PeakFrequency

Pointer to the calculated peak frequency. (output)

PeakFrequencyRMS

Pointer to the calculated RMS value of the peak frequency. (output)

PeakFrequencyPhase

Pointer to the calculated phase value (in degree) of the peak frequency. (output)

Return Values

Reserved.

2.2.6 SPA_MFCC

The SPA_MFCC function calculates the Mel-Frequency Cepstrum Coefficients from RMS amplitude spectrum data.


```
SPA_MFCC(
double *xp,
long FFTSize,
double SamplingFrequency,
double FrequencyLowerLimit,
double FrequencyUpperLimit,
long MelBandCount,
double * MelCenterF,
double * MelBand,
double * MFCC,
int FilterMode,
double RelativeTroughLevelIndB
)
```

Parameters

xp

Pointer to the RMS amplitude spectrum result. (input)

FFTSize

Number of FFT points. It must be a power of 2. (input)

SamplingFrequency

Sampling frequency of the RMS amplitude spectrum data to be analyzed (input)

FrequencyLowerLimit

Frequency lower limit of the Mel frequency bands (input)

FrequencyUpperLimit

Frequency upper limit of the Mel frequency bands (input)

MelBandCount

Number of Mel frequency bands in the specified frequency range (input)

MelCenterF

Pointer to an array of center frequencies of Mel frequency bands (output)

Note that: [number of center frequencies] = MelBandCount + 2;

MelBand

Pointer to an array of energy contained in each Mel band. (output)

Note that: [number of Mel Bands] = MelBandCount;

MFCC

Pointer to an array of MFCC. (output)

Note that: [number of MFCCs] = MelBandCount;

FilterMode

Mel-scale filter bank mode (input)
0 - filter on power spectrum 1 – filter on amplitude spectrum

RelativeTroughLevelIndB

Forced minimum trough level in dB relative the peak level in Mel frequency bands.

Return Values

Reserved.

2.2.7 SPA_DTW

The SPA_DTW function computes the distance between two 2-D arrays using Dynamic Time Warping. For example, it can be used to compute the similarity between two 2-D arrays of MFCCs.

```
double SPA_DTW(
double **x,
double **y,
long XCount,
long YCount,
long MFCCCountChosen
)
```

Parameters

***x*

Pointer to the first 2-D array. (input)

***y*

Pointer to the second 2-D array. (input)

Xcount

Number of points in the first array

Ycount

Number of points in the second array

MFCCCountChosen

Number of MFCC used for distance calculation.
[MFCCCountChosen] <= [MFCC Band Count]

Return Values

Distance between the two arrays.

2.2.8 SPA_THD

The SPA_THD function calculates THD, THD+N, SINAD, SNR from the RMS amplitude spectrum result.

```
SPA_THD(
double * xr,
double * xi,
double * xp,
long FFTSize,
double SamplingFrequency,
double THDFreqLowerLimit,
double THDFreqUpperLimit,
```



```
double * PeakFrequency,
double * THD,
double * THDN,
double * SINAD,
double * SNR
);
```

Parameters

xr

Pointer to the real part of FFT/iFFT intermediate / final result. (input)

xi

Pointer to the imaginary part of FFT/iFFT intermediate / final result. (input)

xp

Pointer to the RMS amplitude spectrum result. (input)

FFTSize

Number of FFT points. It must be a power of 2. (input)

SamplingFrequency

Sampling frequency of the data to be analyzed (input)

THDFreqLowerLimit

THD, THD+N, SINAD, SNR calculation frequency band lower limit (input)

THDFreqUpperLimit

THD, THD+N, SINAD, SNR calculation frequency band upper limit (input)

PeakFrequency

Pointer to the calculated peak frequency (fundamental frequency). (output)

THD

Pointer to the calculated THD in percentage (e.g. 0.01 means 1%). (output)

THD+N

Pointer to the calculated THD+N in percentage (e.g. 0.01 means 1%). (output)

SINAD

Pointer to the calculated SINAD in dB. (output)

SNR

Pointer to the calculated SNR in dB. (output)

Return Values

0: Normal
Others: Error

Note: For THD, THD+N, SINAD, SNR measurement, Record Length, FFT size, Window function, etc. need to be set properly in order to get meaningful results. Please refer to Multi-Instrument software manual for details.

2.3 General APIs

2.3.1 SPA_Unlock

The SPA_Unlock function unlocks the vtSPA DLL so that its functions can be used by the calling program. This function must be called before any other API functions can be used.

```
void Unlock(
long nSerialNumberPart1, //serial number part 1
long nSerialNumberPart2, //serial number part 2
long nSerialNumberPart3, //serial number part 3
long nSerialNumberPart4  //serial number part 4
)
```

Parameters

nSerialNumberPart1

Part 1 of the serial number of the vtSPA DLL.

nSerialNumberPart1

Part 2 of the serial number of the vtSPA DLL.

nSerialNumberPart1

Part 3 of the serial number of the vtSPA DLL.

nSerialNumberPart1

Part 4 of the serial number of the vtSPA DLL.

Note that:

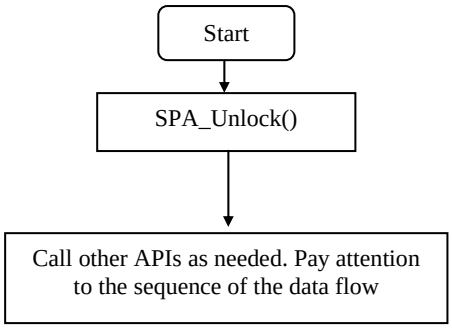
1. The serial number has a format of part1-part2-part3-part4, where each part contains four characters in hex format.

2. For copy-protected vtSPA DLLs, such as the trial version, the softkey activated version, the USB hardkey activated version and the DSO hardware bundled version, a generic serial number 0000-0000-0000-0000 should be used. Note that for the trial version and the softkey activated version, a warning message will pop up showing that the DLL is a trial version. The message will not show up if a USB hardkey or any VT DSO hardware is connected to your computer.

3. For not-copy-protected vtSPA DLLs, which is usually the case for OEM, a customer specific serial number will be given when the DLL is purchased from Virtins Technology.

3. vtSPA Development Guide

3.1 Flowcharts

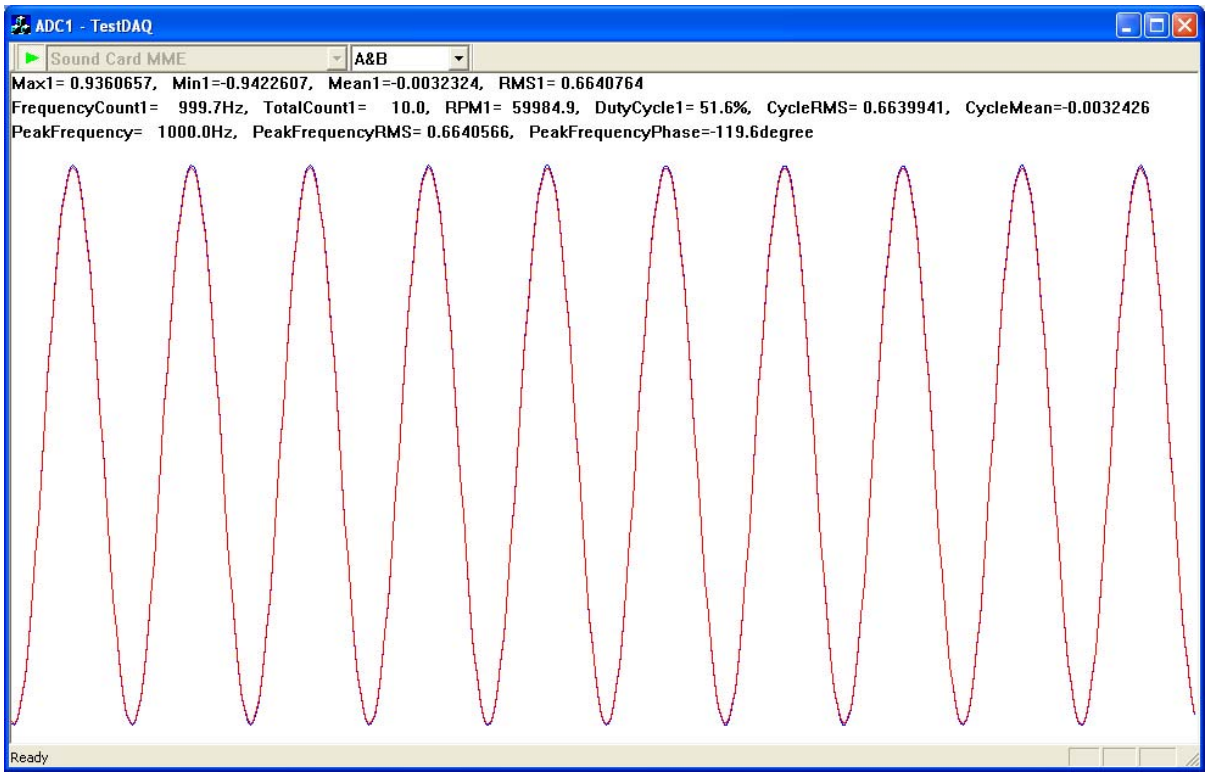


3.2 Basic Files

1. Header file to be included: vtSPA.h
2. vtSPA.dll
3. vtSPA.lib

4. Sample Programs

4.1 TestDAQ written in Visual C++ 6.0



TestDAQ is a sample DAQ back-end program. It demonstrates how to use the vtDAQ interfaces to perform data acquisition. As shown in the above screenshot, there are one Start/Stop button for starting/stopping DAQ, one combo box for selecting vtDAQ interface DLLs, and one combo box for selecting the number of sampling channels. Selection of sampling channels is used to demonstrate how to change a sampling parameter on-the-fly without manually stopping the DAQ first. All other DAQ parameters are set inside the software codes for simplicity purpose. The program also demonstrates how easily a back-end program can interface to a variety of vtDAQ compatible devices, currently including:

- Sound Cards (MME)
- Sound Cards (ASIO)
- NI DAQmx Cards
- VT DSO H1
- VT DSO H2
- VT DSO H3
- VT DSO F1
- VT DAQ 1
- My DAQ Device

To facilitate data processing and analysis after data acquisition, Virtins Technology has also developed and exposed a suite of Signal Processing and Analysis APIs (vtSPA). These APIs are also linked inside the TestDAQ program. To demostrate some of the vtSPA features, TestDAQ calculates the Max, Min, Mean, RMS, frequency count, total count, RPM, duty

cycle, cycle RMS, cycle mean, peak frequency, RMS of peak frequency, and phase of peak frequency. These values are displayed in the upper par of the oscilloscope graph.

(please refer to: http://www.virtins.com/vtDAQ-and-vtDAO-Interfaces.pdf for detailed description of vtDAQ and vt DAO interfaces.)

4.2 TestDAQ written in Visual C# 2012

TestDAQ_CSharp is a sample DAQ back-end program written in Visual C#, with functions similar to its Visual C++ counterpart introduced previously. Instead of calling vtDAQ interface dll directly, it interfaces to vtDAQLV.dll which in turn calls the respective vtDAQ interface dll, thus avoiding using complex data structures in the original vtDAQ APIs. This program also demonstrates how to interface to vtSPA.dll using C#.

