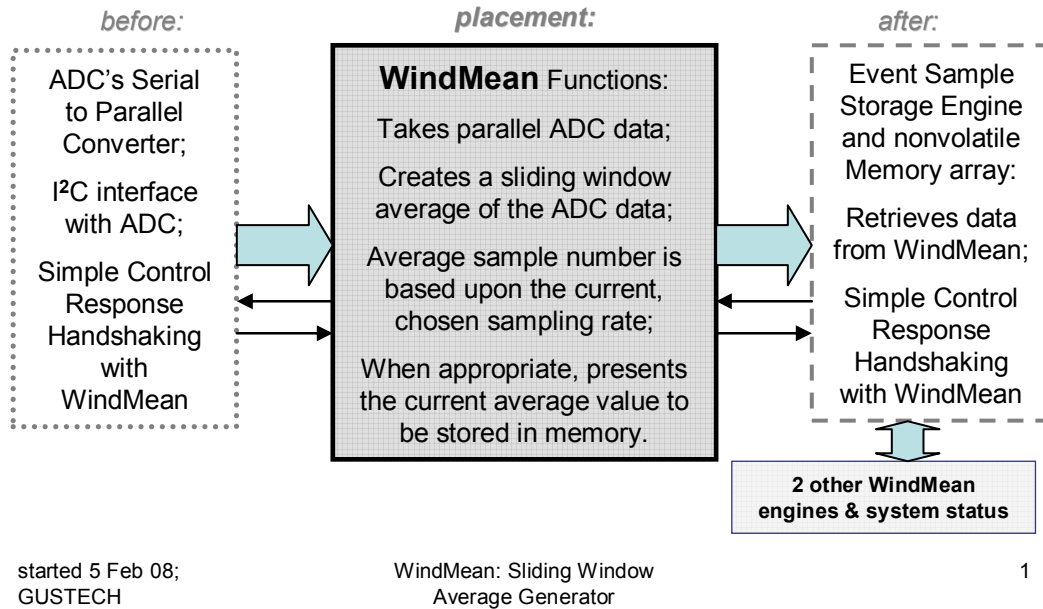


## Functional and Interface Specification for the WindMean™ Data Sampler Averager

This document is the functional and operational description of the WindMean™<sup>1</sup> with its module input and output signal specification. It is being developed as part of the SER (Shock Event Recorder) project, as presented on the web site: [www.gustech.biz](http://www.gustech.biz), as the “Hatching DASlings” online presentation. This particular module is contained within a large CPLD on the Digital Board of the SER.

Spatially the WindMean is positioned (see figure 1) between the module that converts the ADC information from serial to parallel format and the event storage memory. The ADC’s run at a fixed 500,000 samples per second so that the anti-aliasing filters can remain fixed at their maximum bandwidth of about 40 kHz. This prevents aliasing, regardless of the apparent sampling rate that is created by the rate the information is actually stored in memory.

# SER Data WindMean™ Idea 1



**Figure 1: WindMean Module's position in the Data Flow**

The primary function of this module is to create an average to be stored in memory whose number of samples is based upon the desired sampling rate using a sliding window averaging technique. The new “average” value to be stored is simply the summation of the samples within the window divided by the number of samples.

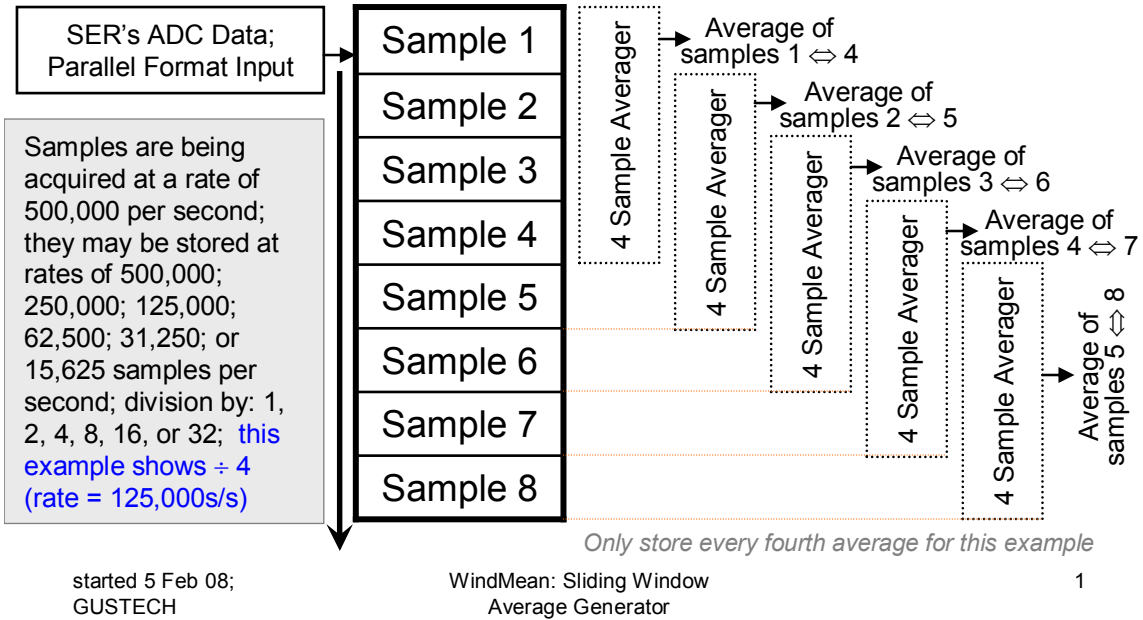
The WindMean architecture consists of a 32 sample circular buffer memory block, an adder/subtractor with an accumulator, and bit-steering logic for generating the “new average” data to be stored in memory. There is a running total to which is added the new ADC data and from which is taken the oldest data value within the summation/averaging window, which is then divided by the number of samples by the bit-steering logic.

<sup>1</sup> WindMean is the trade name being assigned to the Sliding Window Averager that is being discussed in this document.

Functional and Interface Specification for the  
**WindMean™** Data Sampler Averager

Figure 2 (below) presents the basic concept of the sliding window averaging technique for a window width of 4 samples, which corresponds to a sampling rate of 125,000s/s.

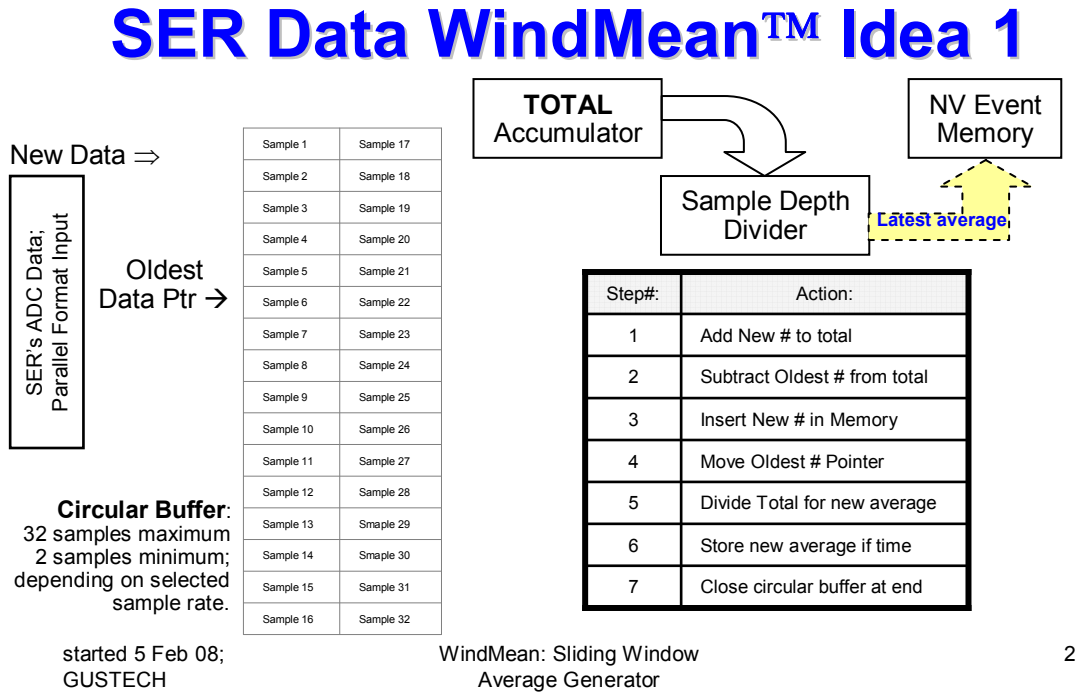
## SER Data WindMean™ Idea 1



**Figure 2: Sliding Window Averaging (x4) example**

## Functional and Interface Specification for the WindMean™ Data Sampler Averager

The general architecture of the Circular Buffer, the TotalSum accumulator, and bit-steering “sample depth divider” is depicted in Figure 3, showing the basic actions performed by the WindMean.



WindMean: Sliding Window  
Average Generator

2

**Figure 3: Circular Buffer, Accumulator/Divider, and steps involved.**

This WindMean module contains an embedded sequence controller (WndMnCtrl) that performs the tasks listed in figure 3, which is based upon VHDL. It performs handshake functions between the two modules that source and take data from the WindMean, and performs all of the proper registration and enabling operations for both external interfacing and internal operations. The handshake techniques used are classic **COMMAND** or request signaling with corresponding status/state **RESPONSE** signaling.

The primary signal groups for interfacing with the WindMean are:

- ◆ new ADC data with a request for accepting it (from the ADC’s serial-to-parallel converter), with a response “pulser” letting the ADC’s serial-to-parallel converter know that the last (new) ADC sample has been successfully retrieved for use by the WindMean.
- ◆ latest Average sample to be stored at the selected sampling rate with a request for accepting it (to the event storage memory), with a response “pulser” input letting the WindMean module know that the last average has been successfully retrieved for storage.
- ◆ the current desired “sampling rate” which determines the number of raw samples that are averaged and when the current average is to be stored in memory. The sample rate is subject to change, on-the-fly, depending upon the programming of hard, soft, and timer triggers.
- ◆ a clock to run the WindMean module engine.

## Functional and Interface Specification for the WindMean™ Data Sampler Averager

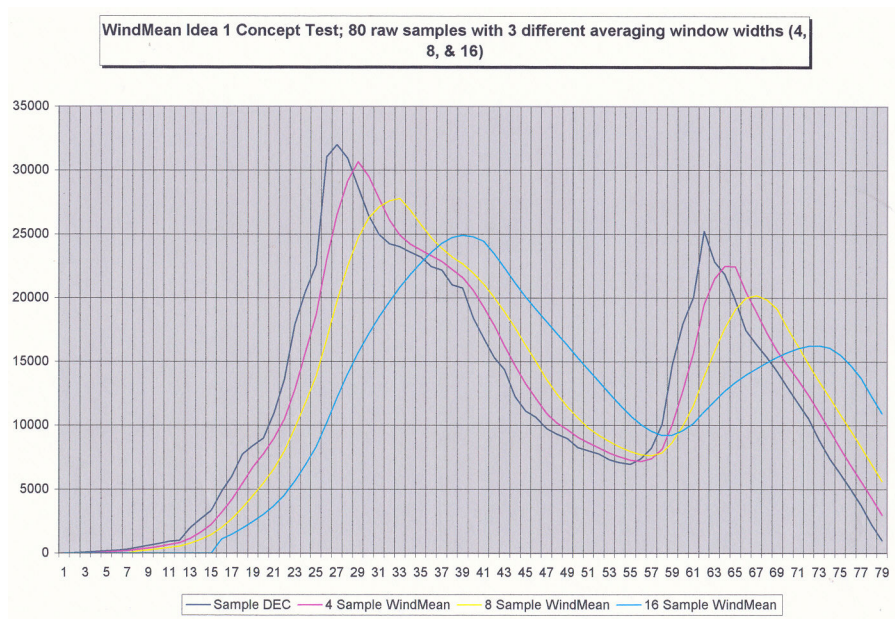
The most important control inputs to the WindMean module are the quasi-static Rate Select signals, 3 bits that define 8 different sampling rates and corresponding sliding window widths, as presented in table 1.

RateSelect[2..0]	Storage Sampling Rate	Averaging Notes	Time/sample
0 0 0	500,000s/s	Un-averaged, raw	2μS
0 0 1	250,000s/s	Averaging 2 samples	4μS
0 1 0	125,000s/s	Averaging 4 samples	8μS
0 1 1	62,500s/s	Averaging 8 samples	16μS
1 0 0	31,250s/s	Averaging 16 samples	32μS
1 0 1	15,625s/s	Averaging 32 samples	64μS
1 1 0	7,812.5s/s	Summation/64, averaging ?? samples	128μS
1 1 1	3,906.25s/s	Summation/128, averaging ?? samples	256μS

**Table 1: Sampling Rate, sample number, time/sample**

The last two “RateSelect[2..0]” values of 6h & 7h corresponding to 7,812.5s/s and 3,906.25s/s respectively, are (for now) reserved for future use. Most high-shock event applications won’t be able to use this low of sampling rate. Additionally, the number of samples that are averaged will need to be ascertained for these last two reserved low-end sampling rates. 32 samples averaged will probably be sufficient so that the circular buffer size will not have to be increased.

Since this is an untested concept at this stage, and since the averaging process inserts a known lump time delay as well as definable “filtering” characteristics (see figure 3 below), it is possible that 32 samples included in the largest samples may be too high for practical applications. Since the averaging window size is based upon the read and write pointers that are defined by the VHDL-based WindMean Controller (WndMnCtrl) module, window size is easy to change based upon future simulations and/or actual hardware testing results.



An Excel-based simulation was performed on a stream of 80 decimal “sample” values, and was plotted (see figure 4) against the corresponding WindMean averaged outputs for window widths of 4, 8 and 16 samples each.

This figure depicts quite well both of the phenomenon mentioned above:

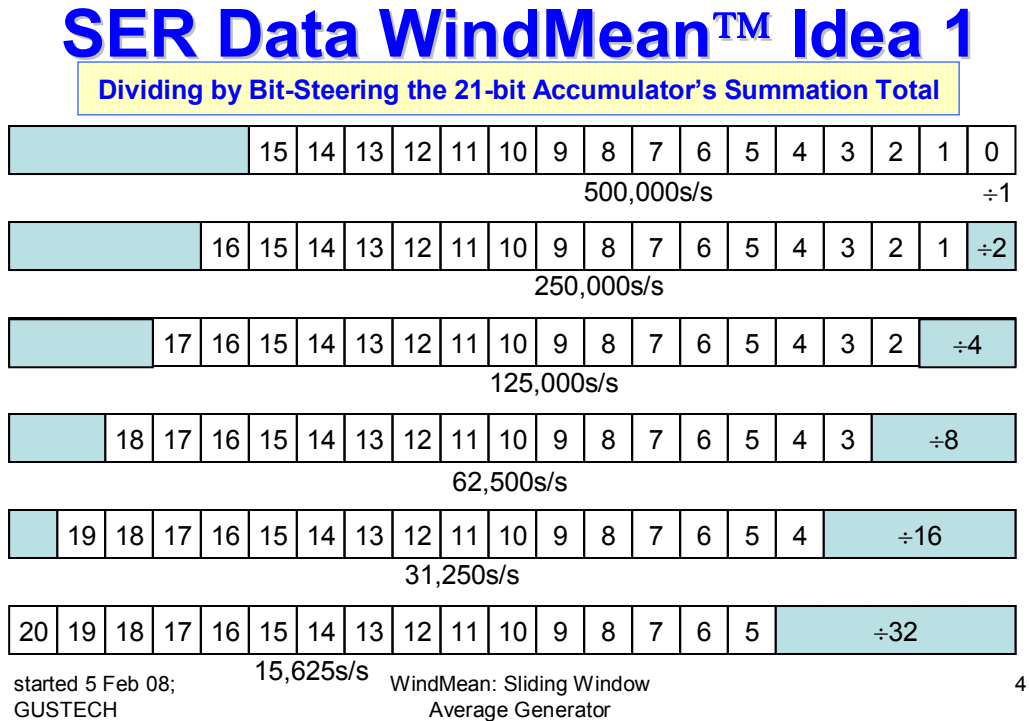
1. known lump time delay (the greater the number of samples the greater the delay); and,
2. the filtering effect where “noise” as quick changes in slope are smoothed out more and more as the number of samples increases.

**Figure 4: WindMean Filtering Simulation Results**

The total time span shown in figure 4 is just 160μS.

Functional and Interface Specification for the  
**WindMean™** Data Sampler Averager

The sampling rates were selected to be a binary regression so that extremely simple “logic” could be used to perform the changes, on the fly as dictated by the preprogrammed trigger-caused mode changes. The method used, called bit steering, shifts the contents of the total/accumulator left to divide by 2, and shifts left again for a division by 4, and again to achieve division by 8, etc. This is depicted by figure 5.



**Figure 5: Bit-steering Division of Accumulator for lower Sampling Rates**

As will be seen in the logical implementation of this circuitry, all of this bit-steering can be easily accomplished with a 6x16-bit multiplexer. To present 500,000s/s to the storage memory, bits 15⇒0 are used every time a raw sample is ready (2μS); to send 31,250s/s to the storage memory, the accumulator's bits 19⇒4 are used every 32μS, etc.

**IMPLEMENTATION NOTES:**

In order to proceed with converting these concepts into a testable (by simulation initially) circuit, a choice needs to be made regarding tools and potential devices. At this juncture, I have chosen to first try developing these circuits inside a large Altera CPLD because it will not require a support memory device for Power-up loading as do FPGA devices. This is only a test and not a commitment to this device or family in the final design instantiation.

The next section will discuss interfacing:

- ◆ to and from the WindMean module; and,
- ◆ to and from the WndMnCtrl VHDL module inside.

Functional and Interface Specification for the  
**WindMean™** Data Sampler Averager

**WindMean MODULE INTERFACE SPECIFICATION:**

Table 2 below presents all of the signals, to and from, the WindMean module, and how they are used by outside interfacing circuits.

Outside Module	Signal/Bus	Description/Operation/States/Notes
System	SysClk	Main synchronous clock <b>INPUT</b> for all registration and processes
	reset	Main reset (asserted high) <b>INPUT</b> used by internal VHDL module for reset
	RateSelect[2..0]	3-bit Sampling Rate Select lines <b>INPUT</b> (see Table 1 above for details)
ADC_S2Pconv	ADCdata[15..0]	Latest 16-bit ADC raw data <b>INPUT</b> from ADC's serial-to-parallel converter
	newSampleRdy	<b>INPUT</b> high state indicates that the ADC data is a new sample to be averaged; it is held high until the ADC_S2Pconv module receives the output pulse "ThanksGotIt"
	ThanksGotIt	<b>OUTPUT</b> high pulse indicates that the WindMean module has captured the new ADC Data, and that the ADC_S2Pconv module is clear to acquire a new sample.
StorageEngine	RegAvgSample[15..0]	Latest Averaged Sample <b>OUTPUT</b> to be stored in event memory based upon the current Sampling Rate
	NewStoreRdy	<b>OUTPUT</b> high state indicates that a new averaged sample is ready for event storage; it will remain high until the StorageEngine pulses high the signal "gotStoreDataOK" indicating that the sample has been retrieved.
	gotStoreDataOK	<b>INPUT</b> high pulse indicates that the StorageEngine has successfully retrieved the latest posted average, and that the WindMean module is clear to continue averaging duties.

**Table 2: WindMean Module Interface Specification**

The last portion of this specification identifies the signals that are internal to the WindMean module that need definitions, those specifically for the VHDL controller currently called: WndMnCtrl. These signals generally control signals for the orderly averaging operations, and interface duties for the WindMean modules I/O ports identified in table 2.

Functional and Interface Specification for the  
**WindMean™** Data Sampler Averager

**WndMnCtrl VHDL MODULE INTERFACE SPECIFICATION:**

<b>Signal/Bus</b>	<b>Function/Description/Operation/States/Interface/Notes:</b>
SysClk	This is the main system <b>INPUT</b> clock that drives all VHDL processes
reset	This is an asserted high system reset <b>INPUT</b> that synchronously resets all VHDL processes
RegRateSel[2..0]	Pre-synchronized Rate Select <b>INPUT</b> signals used to determine sliding window sample width and when to post a new average to be stored in memory
newDataWE	Asserted high when it is time to write the new raw ADC sample into the circular buffer after the correct address has been posted
CirBuffAddr[4..0]	5-bit Circular Buffer address for read and write operations
AddSubCtrl	High for Addition and low for Subtraction control line
AccumUpdate	High enable line that updates the accumulators contents being supplied by the adder subtractor circuitry
newSampleRdy	High state <b>INPUT</b> indicating that a new raw ADC sample as been presented
ThanksGotIt	High pulse <b>OUTPUT</b> indicating that the new raw ADC sample has been successfully retrieved
UpdateStoreSamp	High enable line that registers the bit-steered accumulator's contents as a new averaged data that is ready for being stored in memory
NewStoreRdy	High state <b>OUTPUT</b> indicating that a new average has been registered and is ready for retrieval and writing to the event memory
gotStoreDataOK	High pulse <b>INPUT</b> indicating that the last posted average for storage has been successfully retrieved

**Table 3: WndMnCtrl VHDL Module's Interface specification**

***This is the end of this functional and interface specification, for now.  
As the design evolves and simulations provide results, these specifications may change.***

A list of figures and a list of tables are presented on the next, last page.

Functional and Interface Specification for the  
**WindMean™** Data Sampler Averager

TABLE of Tables

Table 1: Sampling Rate, sample number, time/sample .....	4
Table 2: WindMean Module Interface Specification .....	6
Table 3: WndMnCtrl VHDL Module's Interface specification.....	7

TABLE of Figures

Figure 1: WindMean Module's position in the Data Flow .....	1
Figure 2: Sliding Window Averaging (x4) example .....	2
Figure 3: Circular Buffer, Accumulator/Divider, and steps involved.....	3
Figure 4: WindMean Filtering Simulation Results .....	4
Figure 5: Bit-steering Division of Accumulator for lower Sampling Rates.....	5