White Paper

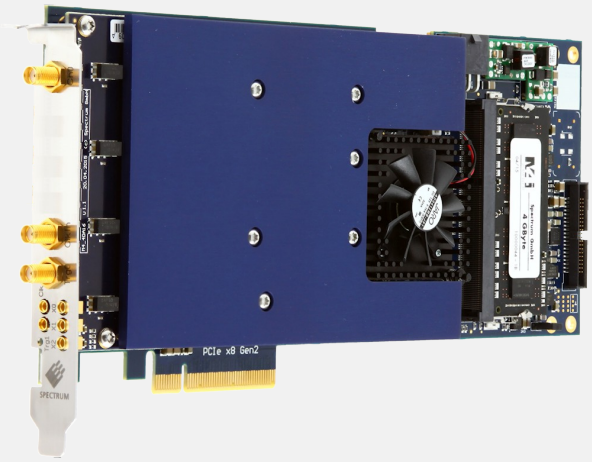
Using software based fast block averaging

Updated Version April 2019 – New Chapter and Results using CUDA-based GPU

Background

The block, or segmented memory, averaging mode is used with Digitizers for different applications where incoherent noise needs to be removed from a signal. Independent of the manufacturer of the digitizer all FPGA based hardware implementations of the block averaging mode limit the maximum size of the segment to be averaged. The limit depends on the capacity of the FPGA and usually ranges from 32k up to 500k samples.

This white paper shows how to use the fast PCIe streaming capabilities of the Spectrum M4i series digitizers to implement block averaging in software to go beyond these limits. Using the M4i.2230-x8 (1 channel, 5 GS/s, 8 bit digitizer with 1.5 GHz bandwidth) results achieved with both the hardware and software block averaging methods are compared.



M4i.2230-x8 - 5 GS/s 8 bit Digitizer with 1.5 GHz BW

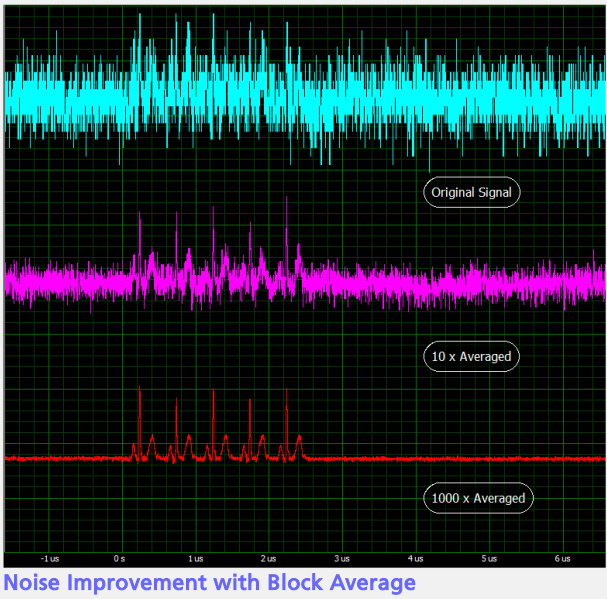
What is Block Averaging?

The Block Averaging mode can be used to improve the fidelity of any repetitive signal by removing its random noise components. The mode allows multiple single acquisitions to be made, accumulated and averaged. The process reduces random noise and improves the visibility of the repetitive signal. The averaged signal has an enhanced measurement resolution and increased signal-to-noise (SNR) ratio.

The Block Averaging mode can be used to improve measurements in a variety of different applications like Radar Test, Mass Spectroscopy, Medical Imaging, Ultrasonic Test, Optical Fiber Test and Laser Ranging.

The right side screenshot shows a low level signal (approximately 2 mV) that is completely overlaid by random noise and the improvement that can be achieved when using different averaging factors. While the source signal is not even visible in the original single-shot acquisition, averaging 10 times shows that there is actually a signal with 5 peaks. Doing a block average of 1000 times improves the signal quality even further revealing the real shape of the signal complete with secondary maximum and minimum peaks.

This example was made using a digitizer with a sampling rate of 500 MS/s (2 ns per point) and 14 bit resolution.



Noise Improvement with Block Average

System Setup

The test system was a standard office PC from the Spectrum development department consisting of the following components:

- Motherboard: Gigabyte GA-H77-D3H
- CPU: Intel i7-3770 4 x 3.4 GHz
- Memory: 8 GByte DDR3 memory
- SSD: 120 GByte Samsung 840 EVO
- Operating System: Windows 7 Professional 64 Bit
- Compiler: Visual Studio 2005 Standard Edition

The motherboard has one free PCIe x8 Gen2 slot which is used by the digitizer card. This slot has a payload of 256 which allows the Spectrum M4i cards to reach a full streaming speed of around 3.4 GByte/s (without any data processing).

Software Implementation

The test software was done in plain C++ and is based on the Spectrum streaming examples. The test card was fed with an external trigger and acquired one segment of data on every trigger event. Data was stored in the cards on-board memory and transferred by scatter-gather DMA directly into PC memory where it was accumulated to perform the block averaging. Different setups and improvement methods have been tested to see what performance levels could be achieved.

```
SPCM_THREAD_RETURN SPCM_THREAD_CALLTYPE pvAverageSegmentPart (void*
pvArguments)
    {
    SPCM_AVERAGE_DATA* pstData = (SPCM_AVERAGE_DATA*) pvArguments;
    int32  i, j;
    int32  lStart;
    int32  lEnd;
    int32  lNumSegments =    pstData->lNumSegments;
    int32* plAverageData =   pstData->plAverageData;
    int8*  pbyData =         pstData->pbyData;

    while (pstData->bRunNotQuit)
        {
        spcm_vWaitEvent (&pstData->hStart);

        for (j=0; j<lNumSegments; j++)
            {
            lStart = j * pstData->lSegmentsize + pstData->lStartOffset;
            lEnd =   pstData->lStartOffset + pstData->lAveragesize;

            for (i=lStart; i < lEnd; i++)
                plAverageData[i] += (int32) pbyData[i];
            }

        spcm_vSignalEvent (&pstData->hEnd);
        }

    return 0;
    }
```

The small source code excerpt shows the threaded version of the main summation loop. This is the crucial and speed determining part of the software.

The following list gives information and comments on the different aspects of the implementation found in the results section:

- Segmentsize: the number of samples for one data segment that will be acquired after receiving a trigger event.
- Averages: the number of averages (summations) that are performed until a segment is stored and the average process is restarted.
- Notifysize: the amount of data after which an interrupt is generated by the PC hardware. This notifysize defines the pace of the complete average loop. If the notifysize is larger than the segmentsize multiple segments are summarized on one interrupt. This reduces the overhead for thread communication and interrupt handling.

- Buffersize: the overall target buffer in memory for the DMA transfer. In our example the buffer is a fixed size of 16 times the notifysize.
- Triggerrate: the repetition rate of the external signal generator. In the results we show the maximum achieved triggerrate without filling up (overflowing) the buffers.
- Threads: to speed up the summation process we parallelized this task by splitting the summation into a number of different software threads as shown on the previous page. If Threads is shown as zero the summation process does not use threading but runs directly inline in a loop.
- CPU Load: as the average process is done in software the CPU(s) need to do all the work. Luckily modern CPU's consist of multiple cores allowing an easy way to share working tasks between them.
- SSE/SSE2 commands: on a first look these commands seem to be perfectly suited to parallelize the summation process and speed up the software without the need of any thread based programming. However, unfortunately the SSE command set is all based on data of the same type. As the acquired data is 8 bit wide and the average buffer is 32 bit wide this is not a solution that can be used here.

Results

All measurements are made with a digitizer using 1 channel sampling at 5 GS/s, with 8 bit resolution and an external trigger. The table also lists different program settings to show the result differences. The best result for each segmentsize is marked yellow in the table.

Samplerate	Segmentsize	Averages	Notifysize	Mode	Threads	Max Triggerrate	CPU Load
5 GS/s	32 kSamples	1000	1 MByte	Hardware	-	150 kHz	< 5%
5 GS/s	128 kSamples	1000	1 MByte	Hardware	-	38 kHz	< 5%
5 GS/s	256 kSamples	1000	256 kByte	Software	2	10.3 kHz	25%
5 GS/s	256 kSamples	1000	1 MByte	Software	2	12.6 kHz	17%
5 GS/s	256 kSamples	1000	1 MByte	Software	4	12.8 kHz	16%
5 GS/s	256 kSamples	1000	1 MByte	Software	-	6.4 kHz	14%
5 GS/s	512 kSamples	1000	512 kByte	Software	2	5.9 kHz	25%
5 GS/s	512 kSamples	1000	512 kByte	Software	4	6.0 kHz	29%
5 GS/s	512 kSamples	1000	1 MByte	Software	4	6.4 kHz	23%
5 GS/s	512 kSamples	1000	2 MByte	Software	4	6.4 kHz	23%
5 GS/s	512 kSamples	1000	8 MByte	Software	4	6.4 kHz	14%
5 GS/s	512 kSamples	1000	8 MByte	Software	-	3.4 kHz	14%
5 GS/s	1 MSamples	1000	1 MByte	Software	-	1.5 kHz	16%
5 GS/s	1 MSamples	1000	1 MByte	Software	2	2.9 kHz	24%
5 GS/s	1 MSamples	1000	1 MByte	Software	4	2.9 kHz	23%
5 GS/s	1 MSamples	100	1 MByte	Software	4	2.9 kHz	30%
5 GS/s	1 MSamples	10000	1 MByte	Software	4	2.9 kHz	23%
5 GS/s	2 MSamples	1000	2 MByte	Software	-	0.7 kHz	14%
5 GS/s	2 MSamples	1000	2 MByte	Software	4	1.3 kHz	40%

Samplerate	Segmentsize	Averages	Notifysize	Mode	Threads	Max Triggerrate	CPU Load
5 GS/s	4 MSamples	1000	4 MByte	Software	-	340 Hz	15%
5 GS/s	4 MSamples	1000	4 MByte	Software	2	410 Hz	24%
5 GS/s	4 MSamples	1000	4 MByte	Software	4	390 Hz	50%
5 GS/s	8 MSamples	1000	8 MByte	Software	-	160 Hz	14%
5 GS/s	8 MSamples	1000	8 MByte	Software	2	190 Hz	35%

New - Using a CUDA-based GPU for averaging

In November 2018 Spectrum released some examples for block averaging using the SCAPP (Spectrum 's CUDA Access for Parallel Processing) option for very fast data processing. The basic concept is the same as above where the acquisition is done by the Digitizer and the data is transferred continuously over the PCIe bus. However, instead of calculating the average inside the PC a GPU is used. A big advantage of the GPU-solution is that it is designed for parallel calculations. This makes it a perfect fit for any kind of block average process.

SCAPP gives users the ability to port data directly to the GPU, using RDMA (Remote Direct Memory Access) transfers, where high-speed time and frequency domain signal averaging can be performed without the length or calculation power limitations typically found in other averaging products.

For example, an M4i.2220-x8 Spectrum digitizer can sample signals at 2.5 GS/s and average them continuously without missing an event, even when the signals being averaged are several seconds in length. Similarly, an M4i.4451-x8 digitizer with 14-bit resolution can perform the same function while sampling four signals simultaneously at 450 MS/s! The digitizer cards also include flexible trigger, acquisition and readout modes, which allows them to average signals even when the trigger rates are extremely high. In contrast to the FPGA-based solution, which needs the highest performance FPGAs, the GPU-based averaging already runs at full-speed, even with entry-level GPU cards!

The following table shows some of the test results with the GPU using a similar setup as before:

Samplerate	Segmentsize	Averages	Notifysize	Mode	Threads	Max Triggerrate	CPU Load
5 GS/s	32 kSamples	1000	1 MByte	GPU	-	103 kHz	<5%
5 GS/s	256 kSamples	1000	1 MByte	GPU	-	12.9 kHz	<5%
5 GS/s	1 MSamples	1000	1 MByte	GPU	-	3.2 kHz	<5%
5 GS/s	4 MSamples	1000	4 MByte	GPU	-	810 Hz	<5%
5 GS/s	16 MSamples	1000	16 MByte	GPU	-	203 Hz	<5%
5 GS/s	64 MSamples	1000	64 MByte	GPU	-	51 Hz	<5%
5 GS/s	256 MSamples	1000	64 MByte	GPU	-	13 Hz	<5%
5 GS/s	1 GSamples	1000	64 MByte	GPU	-	3 Hz	<5%

These results have been achieved using a simple Quadro P2000 GPU. As seen in the table the segment size or block size is not limiting the performance. The only limiting factor here is the memory of the GPU.

Frequency domain averaging with GPU

In cases where the frequency domain should be averaged the GPU can also be used allowing very large block sizes compared to an FPGA solution. A frequency domain average consists of an FFT of the block followed by a summation of the FFT result. The processing then consists of two steps where the FFT calculation is very demanding in terms of processing power. For frequency domain averaging the GPU is the only solution, besides an FPGA, as the PC is simply not suitable for FFT conversion at high speeds.

The following table shows some test results using an M4i.4451-x8 4 channel 14 bit digitizer with a maximum sampling rate of 500 MS/s. The acquisition is effectively gap-less with each block being converted into voltage levels, transformed into the frequency domain and then averaged.

Samplerate	Channels	Resolution	Blocksize	Notifysize	Mode	Max Triggerrate	CPU Load
500 MS/s	1 channel	14 Bit	512 kSamples	1 MByte	GPU	1 kHz	<5%
500 MS/s	1 channel	14 Bit	1 MSample	2 MByte	GPU	500 Hz	<5%
500 MS/s	1 channel	14 Bit	2 MSamples	4 MByte	GPU	250 Hz	<5%
500 MS/s	1 channel	14 Bit	8 MSamples	4 MByte	GPU	62 Hz	<5%
500 MS/s	1 channel	14 Bit	32 MSamples	4 MByte	GPU	16 Hz	<5%
500 MS/s	1 channel	14 Bit	128 MSamples	4 MByte	GPU	4 Hz	<5%
500 MS/s	1 channel	14 Bit	256 MSamples	4 MByte	GPU	2 Hz	<5%
500 MS/s	2 channels	14 Bit	256 kSampless	1 MByte	GPU	2 kHz	<5%
500 MS/s	2 channels	14 Bit	512 kSamples	2 MByte	GPU	1 kHz	<5%
500 MS/s	2 channels	14 Bit	1 MSample	4 MByte	GPU	500 Hz	<5%
400 MS/s	4 channels	14 Bit	128 kSampless	1 MByte	GPU	3 kHz	<5%
400 MS/s	4 channels	14 Bit	256 kSamples	2 MByte	GPU	1.5 kHz	<5%
400 MS/s	4 channels	14 Bit	512 MSample	4 MByte	GPU	750 Hz	<5%

Summary

As the above results show block averaging in PC-based software can be used to improve the overall segment size as long as the repetition rate doesn't get to high. When using a GPU one can even get to the limit of the bus transfer speed. Thanks to the high-speed data transfer rates of the PCIe bus much longer acquisitions can be averaged overcoming one of the main limitations of FPGA based averaging processes. For situations where high repetition rates need to be managed, that extend the transfer speed, hardware block averaging will still be the best choice.

The above test program is free to use for your own tests or as a base for implementation in other software programs. The GPU examples are part of the SCAPP option and are available to Spectrum customers against an NDA.

The best performance is reached when using a notifysize of 1 MByte. The number of averages that is performed does not have any visible impact on the test results. The time used to copy the result segment and to clear the result buffer is irrelevant compared to the

sample summation.

As the complete data handling and summation process does not differ when acquiring multiple channels the result can simply be re-calculated for other channel combinations. The following settings will all result in exactly the same maximum trigger rate:

- 1 channel 5 GS/s @ segmentsize
- 2 channels 2.5 GS/s @ segmentsize/2
- 4 channels 1.25 GS/s @ segmentsize/4

Reducing the sampling speed for one channel down to 2.5 GS/s allows one channel to run with the maximum theoretical software averaging speed. For a 1 MSample segmentsize, including the 160 samples dead time between triggers, the theoretical maximum trigger rate is at

```
(2.5 GS/s) / (1 MSample + 160) = 2.38 kHz
```

This is far below the measured maximum of 2.9 kHz @ 5 GS/s.