DANISH TECHNOLOGICAL INSTITUTE

A Welch Power Spectrum Implementation Using Simulink and Xilinx Sysgen

Carsten Siggaard
(Senior Consultant, Danish Technological Institute, Aarhus Denmark)

Dan Anov
(Senior Consultant, Danish Technological Institute, Kolding Denmark)

Abstract
Passive acoustic emission spectroscopy is a technique used in industrial environments where noise and vibrations carry information about the state of the machinery and the product being processed by the machinery. The vibrations can be measured by laser, microphones or accelerometers. In the Data Interpretation and Model Management System (DIMMS) project, the processing of the data is done in two steps; first step is a pre-processing from the time domain into the frequency domain using a FFT and the 2-norm squared of the (complex) result. The result is a real-valued power spectrum. The second step is to match the result of the first step with a chemometric model. The second step will not be described in detail here. The calculations can be performed in software; however the introduction of DSP slices in modern FPGA's makes the calculation of power spectrum using FPGA's feasible. We will present an algorithm implemented in an FPGA using Simulink and Sysgen from MathWorks and Xilinx respectively. The result is an implementation which is capable of calculating power spectra on 4 channels at a sampling rate of 250Ks/s.

Keywords
PSD, FFT, Acoustics, Signal Processing, Xilinx

INTRODUCTION

A common problem when measuring physical phenomena is that your measurements are done in the time-domain (successive measurements making a time-series), whereas the description of the phenomena is most conveniently done in the frequency-domain. To go back and forth between the two representations a Fourier transform is applied. For a general discussion of this subject we refer to ([Press et. al., 1992]).

Passive Acoustic Emission Spectroscopy

In the present case the physical phenomenon under observation is the characteristic noise emitted by a product being processed in a piece of machinery. It may be possible to relate this noise to different characteristics of the product, using chemometric modelling of the power-spectrum. Examples are particle size distribution in a fluid-bed dryer or concentrations of fuel and de-ice in runoff water from an airfield.
The objective of Danish Technological Institute (DTI) is to make results from global research practical to trade and industry. For a number of years DTI has demonstrated the feasibility of

PAES in several applications, but an inhibiting factor is the cost of the equipment required. A large fraction of this cost is the unit doing the digital signal processing, and the most computational intensive part of this is the calculation of the Power Spectral Density (PSD), which is the topic of this article.

The Principle

The work described in this document is based on the algorithm described in ([Press et. al., 1992]). A brief demonstration will be outlined in the following. First we have a signal which contains some noise and two frequencies f_1 = 12.8 Hz and f_2 = 25.467 Hz. The signal is sampled with a sample frequency of f_s = 128 Hz. The signal is generated with the following MatLab script (excerpt):

```
fs    = 128; siglen=6;
t     = 1:1/fs:(siglen+1);
f1    = 12.8;f2=25.467;
sig   = sin(f1*2*pi*t)+sin(f2*2*pi*t);
noise = 7*(rand(1,size(t,2))-0.5);
sig   = sig+noise;
subplot(2,2,1);
plot(sig,'Color',[0 0 0]);
subplot(2,2,2);
stem(abs(1/128*fft(sig(1:128))),…
'Color',[0 0 0])
subplot(2,2,3);
plot(abs(1/128*fft(sig(1:128))).^2,…
     'Color',[0 0 0])
wdw=repmat(window('triang',128),1,10);
t = reshape(sig(1:(siglen)*(fs)),…
    fs/2,(siglen)*(2));
t = [ t(:,1:10) ; t(:,2:11)];
t = wdw .* t;
p = abs(fft(t)).^2;
subplot(2,2,4);
plot(10*log(sum(p')/sum(wdw(:))),…
'Color',[0 0 0]);
```

In this example a triangular window is used, other windows can be used. A discussion about the windows and their characteristics can be found in [Press et. Al., 1992].

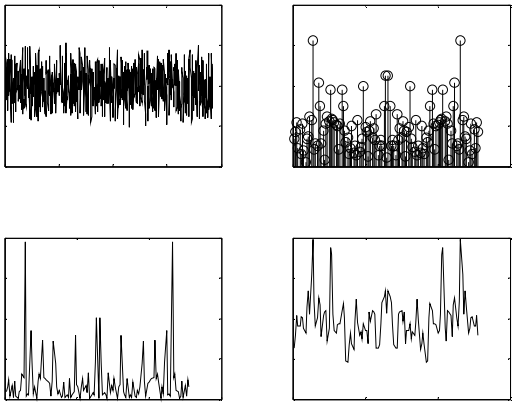

Figure 1: signal with two frequencies f1 = 12.8 Hz and f2 = 25.467 Hz and some white noise.

The result is depicted in Figure 1, where Figure 1(a) is the signal, Figure 1(b) is the FFT of the signal, Figure 1(c) depicts the absolute value of the signal and Figure 1(d) is the absolute values of the signals squared and averaged over 10 windows. The window used is a triangular window. There are several items to note here; first note that the signal is very noisy, still a simple FFT (subfigure b) can be used to identify f_1 but f_2 disappears and can hardly be distinguished from the noise. The reason why f_1 is easy to identify in subfigure b is that f_s is divisible by f_1 so there is no *power leakage* or *spectral leakage* from the 12.8 frequency "bin" to the neighbouring frequency bins. Taking the absolute value squared helps, but there is still a lot of noise (subfigure c). Both frequencies become much clearer when the result is averaged over several windows, furthermore there is less power leakage due to the use of a triangular window.

Note that the signal is real-valued; hence the Fourier transform satisfies the relation $H(-f) = |H(f)|^*$, where the asterisk indicates complex conjugation, which implies that the frequency spectrum is complex conjugate symmetric, There is no additional information revealed in the frequency spectrum above the Nyquist frequency $f_n = f_s/2$

Design goals

The Accelerated Digital Design group at DTI wanted to demonstrate that a modern FPGA with DSP slices is capable of calculating PSD-spectra at the rate required by the experimental setup used in the PAES equipment.

The fundamental requirements were:

- Speed 250 kS/s
- 18 bit per channel.
- min. 4 channels upgradeable to more.
- Low HW Cost/price
- Easy integration with Linux.

RELATED WORK

Most modern oscilloscopes contain functions for spectral analysis e.g. by using the FFT functionality, however there is no simple interface and implementing an oscilloscope into a harsh industrial environment is not desirable, furthermore Spectrum Analyzers tend to be expensive.

A C/C++ implementation of a PSD implementation is described in [Press et. a., 1992], which forms the basis of the implementation described in this document. It is possible to use general purpose processors (GPP) in a solution calculating power spectra, this has been done in previous solutions of the PAES equipment used at DTI. The GPP implementations have been expensive both with regard to hardware and with regard to software licenses.

Another description can be found in the documentation for the signal processing toolbox [MathWorks spt 2008], where the concepts of periodogram and the Welch method are described. The limiting factor here is that MathWorks does not provide a VHDL implementation for their FFT block.

There are vendors providing systems with PSD capabilities. But systems with the combination of 4-31 channels with 18 bit accuracy and a rate of 250 kS/s is difficult to find in the low-budget market.

DETAILS

The process used in this project is based upon a development flow used at DTI; the process is an iterative development flow starting with Ivar Jacobsons Use Cases [Douglass 2004]. The rationale using the use-case technique is to identify representative subtasks and critical parts of the system. The next step is to implement or prototype the identified use cases. Using this approach the developer can verify the feasibility of the chosen platform, by verifying that the platform chosen is capable of performing the identified use cases. Furthermore the use case technique is a very usable means for communication during the development process. Two use cases which plays an important role for the PAES equipment is depicted in Figure 2**.** The rationale for choosing these two use cases is that they play a vital/crucial role for the system with regard to the functionality ("*Calculate PSD*"), resource requirement ("*Calculate PSD*") and remote control of the system ("*Download System Image*"). The remote control is not a topic for the rest of this paper and will not be described further. The use case "*Calculate PSD*" is implemented using both MatLab and Xilinx Sysgen (a Simulink Block set). The detailed requirement specifying the algorithm originates from a Fortran90 procedure, which was used as an input for the implementation. The development process was divided into two phases, where the first was to verify a MatLab procedure against a Fortran90 procedure and the second phase was to verify a Simulink model with the MatLab model.

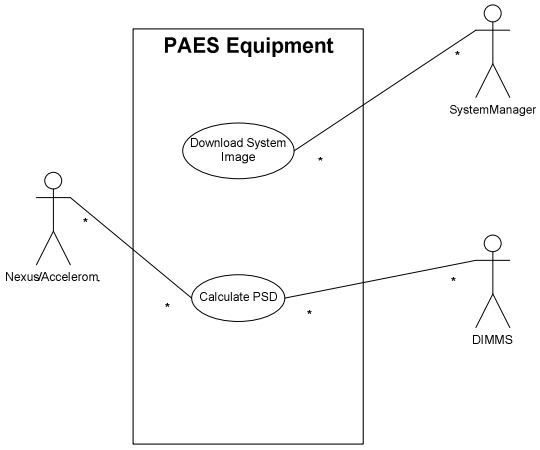

Figure 2: A use case diagram depicting the requirements for the PAES system.

MatLab

The detailed requirement for the functionality is a Fortran90 procedure, the rationale for using this approach is to create a MatLab function which is reviewable by Fortran90 programmers and capable of providing results which can be used to compare with the results from the Simulink simulation.
The MatLab function is a simple translation from Fortran90 into 'm', resulting in a correct but non-optimal calculation of the Power Spectrum. The result is a procedure which calculates the PSD using a Welch window, as described in (Press et. al., 1992]), with the exception that the size of the spectrum covers the range from 0 to $f_n = f_s/2$ - this is due to the fact that the FFT of a real signal is complex conjugate symmetric.

The head of the M-function is:

```
[k F] = ps1d(T, NT, MF)
```

Where:

- **k** is the resulting number of windows contains the Power spectrum (of size MF)
- **F** is a row vector containing the frequency spectrum from DC to the Nyquist Frequency.
- **T** contains a row vector containing the samples.
- **NT** is the number of samples
- **MF** is the number of samples per window (half the full spectrum size)

In addition to the algorithmic topics of the PSD the quantization from analogue values into discrete values can be investigated, these considerations are important because they determine important parts of the detailed design and are also limited by the capabilities of the building blocks provided by Xilinx Inc.

The FFT block from Xilinx Inc. has amongst others the configuration parameters listed in Table 1.

Parameter	Value
Architecture	Continuous, Radix-4 and Radix-2 (Radix-2)
Twiddle factor Size	8-24(18)
Input Size	8-24(18)
NFT	64-65536 (1024)
Scaling	On/Off (on)

Table 1: Parameters controlling the FFT core (The chosen parameters in parenthesis).

The architecture is set to Radix-2 (Butterfly Arithmetic) because the performance of this architecture is sufficiently fast and small. The twiddle factors are set to 18-bit resolution, which is found experimentally using MatLab and the bit accurate C-model [XilbacFFT 2008]. Due to the fact that the outputs from the ADCs are 18 bits, the input size is predetermined to 18. NFT is set to 1024. Scaling is applied to prevent overflow.

Simulink

All hardware specific modelling is done by using Xilinx Sysgen running in the Simulink environment. Xilinx Sysgen does not use the standard Simulink blocks, instead it relies on blocks generated by the Xilinx CoreGen tool and some standard blocks such as registers and Gateway blocks. Using Xilinx Sysgen it is

possible to perform Hardware in the Loop (HIL) testing, increasing the confidence of the implemented solution.

It was decided early in the development process to use Xilinx Sysgen blocks instead of modelling with Simulink Standard blocks.

Important for making this decision was the free FFT (Cooley-Tukey) from Xilinx Inc. This block can calculate 64 to 65536 point FFT's with a precision range of 8 to 24 bits for both input and for twiddle factors independently. Also there are no standard HDL-coder implementations for the FFT.

Furthermore the implementation can be verified using the Xilinx JTAG HIL block. One disadvantage is the fact that Xilinx Sysgen has its own typeset for fixed numbers and that the compatibility with the Fixed-Point Toolbox is varying.

The implementation requires no more resources than available on a Virtex- 4 FX20, which makes the integration with embedded Linux possible. The final system can be integrated on a standard evaluation board from Xilinx with one FPGA, containing the PSD core and a microprocessor taking care of communicating the calculated Power Density Spectra over the Internet.

System Architecture

The entire system is located within one single device a Virtex 4FX-20 FPGA. This device contains (in addition to the basic FPGA functionality) both DSP48 (programmable multipliers and accumulators) and one PPC405 core (a GPP without FPU).

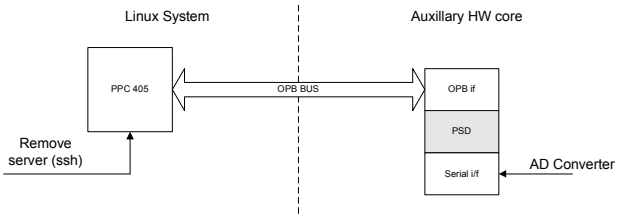

Figure 3: The system architecture of the "PAES" Equipment.

The system architecture is depicted Figure 3. The PSD core is controlled by the PPC, which is running Linux. Linux is used because it is capable of taking care of the transmission of the Power Spectra over network using SSH.

A description of the entire PAES system with Linux in which this core operates can be found in [Bjerge 2008].

CORE ARCHITECTURE

The core is depicted in Figure 4. The core consists of 4 main blocks and a small helper block.

The blocks are:

- **Inputhandler** The input handler takes care of storing the data in intermediate Block RAM. In this implementation 4 input lines can be handled. The handler also buffers the incoming samples into frames of length 1024 samples with an overlap of 50 %.
- **PowerSpectrumCalc** This block calculates the power spectrums. The calculation is done by taking the FFT of the 1024-sample frames and calculating the squared norm: $PS = |FFT(s)|^2$, where s is the frame containing the samples.
- **Accumulator** The accumulator adds the calculated power spectrum with the contents of one of the 4 accumulators. Each time n spectrums have been summed, the results are forwarded to the register interface, and the accumulator is emptied (set to 0).
- **Register Interface** The register interface contains a circular buffer used to store the averaged power spectrums from the accumulator. Each time an averaged power spectrum is stored, a done flag is set for one sample period, signalling to

the controlling operating system that a new frame is ready, to be forwarded.

All the blocks were implemented using the Xilinx Sysgen Block set. The PSD uses a standard Xilinx Block (LogicCore FFT 4.1) for the FFT calculation: The implementation is a Radix-2 FFT using the Cooley-Tukey algorithm. For more details please see [XilFFT 2007].

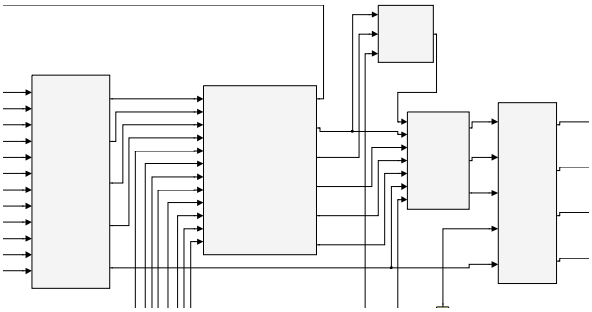

Figure 4: The architecture of the Welch PSD core (excerpt).

Inside the Input handler there are 4 input blocks which are used to buffer-up the incoming samples. The buffers are also used to generate the 50% overlaps. Whenever a buffer is ready to forward a frame it signals an interrupt to an arbiter, which acknowledges the request back to the input.

The arbiter is crucial to the operation of this multichannel analyzer. The arbiter was implemented using a tokenized scheme where each channel is allowed to use the shared resource once, before passing the token to the next channel; this scheme ensures fairness under all circumstances also under heavy load.

The main engine of the core is depicted in Figure 5. The central core marked with a ´X' is the FFT core from Xilinx. The remaining blocks are the Window handler (bottom left), the normalization block (at the top to the right) and 1 housekeeper block for each of the 4 channel (bottom right).

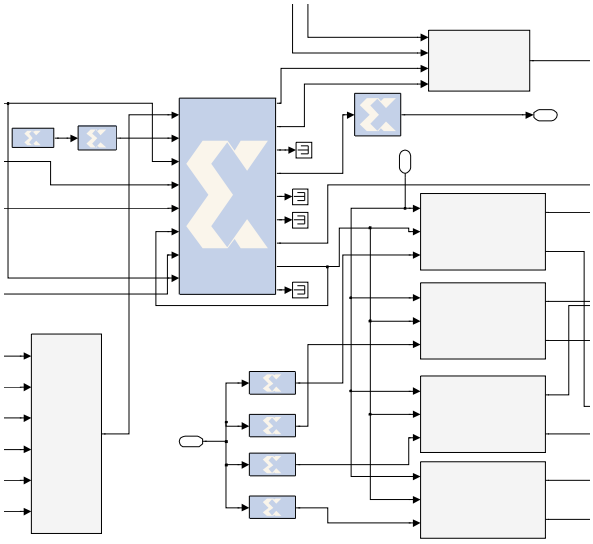

Figure 5: Main Engine of the PSD core (excerpt).

The windowing function is made so that any window can be used; in the experimental results described in the next section a Welch window is used.

EXPERIMENTAL RESULTS

Verification

In this implementation three main tests are interesting to perform:

- Identification of the correct frequencies.
- Channel Separation.
- Level of the resonance frequencies.
- Robustness

The first and the third test determine the quality of the calculation, and the second the implementation of the arbitration scheme and been done correctly.

Identification of the correct frequencies and the correct level.

The results of the test "Identification of the correct frequencies" are depicted in Figure 6. The input is a single sinusoidal signal with

frequency 30 kHz, which the PSD core had to detect correctly. In Figure 6 there are 4 subfigures. Figure 6(a) depicts the result after simulation of the PSD core using the block set from Xilinx. Figure 6(b) depicts the result after simulation using the JTAG HIL simulator from Xilinx. Figure 6(c) depicts the result from the MatLab simulation, i.e. calculation of the power spectrum using MatLab expressions and floating point numbers. Figure 6(b) depicts the difference from the curves in Figure 6 (c) and Figure 6(b).
The peak is located at bin 124 which equals a frequency of 124*125/512= 30.3 kHz. In this test neither noise nor DC-offset has been injected into the system.

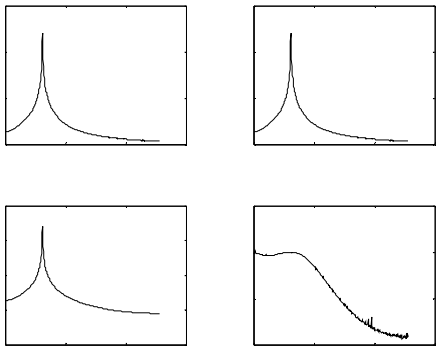

Figure 6 Result from verification of the correct frequencies.

The top level for the original algorithm was calculated to be 51.8358 and the top level for the HIL was found to be 51.8347s these figures gives an absolute difference of 0.0010.

Channel Separation and Noise suppression´

The Output from the PSD core with 4 sinusoidal signals (without noise) is depicted Figure 7. The figure shows no influence from one peak to another. Note that this test does not depict the channel separation from the analog circuit; only digital channel separation is depicted. This test is performed to make sure that the arbitration scheme works, and that the reservation of the shared resource (FFT and normalization works properly).

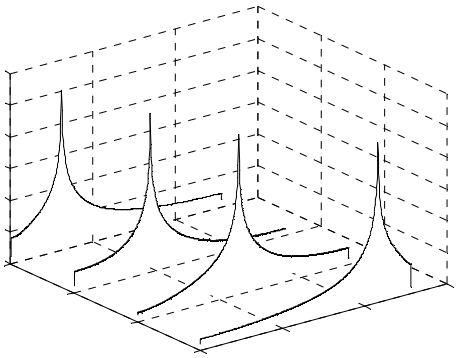

Figure 7: Output from the core when processing data from 4 channels simultaneously.

Robustness

To verify that the core is robust against overloading a testbench was developed which generated samples with a normalized rate of 1.3 versus the rate the core was dimensioned to handle. There were two success criteria; the most important was that the arbiter should not do a lock-up. A minor criterion was that it was ok to drop chunks of samples so that the result still was correct, albeit with loss of precision. The results from the run are depicted in Figure 8.

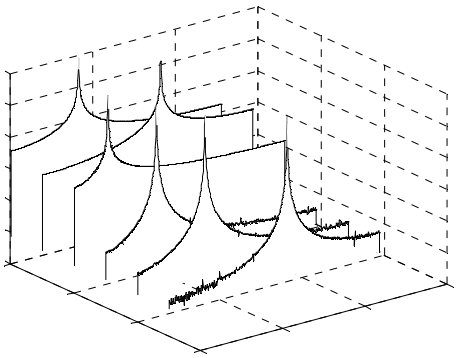

Figure 8: Output from the core when processing data from 4 channels simultaneously, with normalized load 1.3.

Implementation

The results from the implementation are depicted in Table 1. Note that these figures consists of an IBM CoreConnect architecture with a UART a 100 Mbit Ethernet core and the PSD core. The device used is a Xilinx Virtex4FX20 speedgrade -10. The synthesis result is for a system with a processor capable of running at 300 MHz and the bus system (CoreConnect) running at 100 MHz. Note that the Welch Core data path is underused: The max speed is determined by the FFT, which takes 6500 cycles per transformation:

IP Block	Used	Percentage
DSP48	34	87
JTAGPPC	1	100
RAMB16	61	89
Slices	8000	84

Table 2 Device usage

The number of transformations per second in one channel running at 250 kS/s is 490. The theoretical number of power spectrums for the FFT core is 100 MHz/6500 = 15384 transformations per second. That means that the core is capable of supporting 15384/490 = 31 channels. The Limiting factor for this implementation is the amount of block RAM available on this device, and the bandwidth requirements out of the system.
If a larger FPGA had been chosen then a continuous mode FFT could have been chosen. This would increase the theoretical amount of transformations to 88.000 transformations per second increasing the maximum number of channels to 88,000/490=181.

CONCLUSION

To implement an entire PSD application using the Xilinx block set is cumbersome, when comparing this with a standard Simulink implementation. However most signal processing blocks from Xilinx are free, easy to use and do not require any additional Simulink block set. Still Simulink is a usable environment to develop in, and when using the other capabilities of Simulink and MatLab for testing the productivity of the developer increases.

The developed system fulfils all the requirements, is robust and forms a base for further development and enhancements of the PAES system.

Future enhancements cover topics which have been considered less necessary in this implementation such as:

- More channels.
- Variable Spectrum Size.
- Another scheme than Cooley-Tukey (radix-2), e.g. mixed radix. Factorizations of N which are not restricted to $N = 2^d$. (such a scheme is described in [Van Loan 1992]).

REFERENCES

[Bjerge 2008] Kim Bjerge (2008) A HW/SW Co-design Methodology based on UML - How to apply a model based UML design for an embedded system.
[Douglass 2004] Bruce Powel Douglass – Real Time UML third Edition Advances in the UML for Real-Time Systems.
[MathWorks spt., 2008] The MathWorks (2008) Signal Processing Toolbox(TM) 6-User's Guide, version 6.9
[Press et. al., 1992] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. Numerical Recipes in C - The art of scientific Computing. Cambridge University Press, 1992.
[Van Loan 1992] Van Loan, C (1992) "Computational Frameworks for the Fast Fourier Transform. Siam Frontiers in applied Mathematics.
[XilbacFFT 2007] (2007) XILINX Inc. Xilinx LogicCore, Fast Fourier Transform 4.1.
[XilFFT 2007] (2007) XILINX Inc. LogicCore(TM) Fast Fourier Transform v5.0, Bit Accurate C-Model, October 2007.