A Hardware/Software CoDesign Case: Part 2

By Senior Consultant Carsten Siggaard

Danish Technological Institute, January 2009

ABSTRACT

This article is the second in a series of applications using Model Based Design (MBD) to implement algorithms.

Many image processing applications do require integer calculations performed at high speed, especially for feed-back control systems and surveillance systems. Many DSP's are fast and might be able to do the job, but there are other alternatives: Field Programmable Gate Arrays. Standard FPGAs can perform 64 16-bit multiplications per clock cycle; where the clock can run up to several hundred MHz – for integer multiplications this is very fast. But it can get even better: If large FPGAs are used, the number of calculations can be as fast as 2000 16-bit multiplications at 500 MHz! With such a large number of calculations one of the remaining problems to be solved is memory bandwidth, both internally and externally

Using noise reduction and filtering algorithms as an example the requirements which the algorithms put on the bandwidth will be outlined. This article will end with a small discussion of the Altera Video and Image Processing suite.

Table of Contents

INTENDED ROADMAP

The intention of this document is to be a part in a series which describes a set of applications using a variety of signal processing algorithms. The scope of this document is limited to the application and its impact on requirements on the architecture. Note that this roadmap may change in the future.

Document Name	Subtitle	Status
A Framework for the Fast Fourier Transform targeted towards Field Programmable Gate Arrays.	Part 1	Written
Noise Reduction in Image Processing Using FPGA's	Part 2	This

		Document
Modulation and channel Modeling in Simulink and Matlab, targeted for FPGAs	Part 3	Planned
An OFDM implementation on FPGAs	Part 4	Planned
Geometric Distortion in image processing	Part 5	Planned
Position Detection Algorithms	Part 6	Planned

Table 1: Roadmap

BACKGROUND

The background for this document is to serve as a description of the requirements on the architecture imposed on the application(s) on FPGA by some commonly used image processing algorithms.

NOISE AND NOISE MODELING

Noise can be consequence of both the environment (such as poor lightning and dust) and also a natural consequence of the of the image acquisition process. Most image sensors are either of the Charge Coupled Device (CCD) type or the Complementary Metal Oxide (CMOS), none of these are noise free.

There are mainly three classes of noise mentioned in most literature and that is noise in the spatial domain, Noise in the frequency domain, and finally geometric distortions. The two first will be modeled in this article.

Noise Modeled in the Spatial Domain

Shot Noise

This kind of noise originates from Image Sensors where single pixels errors are likely to occur – this kind of errors is also known from LCD panels where single pixels errors occur, and sometimes also accepted. The reason is that CMOS transistors can have “stuck at” errors, i.e. errors where the transistor is either constantly turned on or turned off. The result is a lightened, discolored or dark pixel. In gray tone images the effect looks like salt and pepper has been distributed over image giving name to the alternative name “salt and pepper noise” [1][2][4].

Flake Noise

Flake noise is modeled because there are noise which is not limited to one pixel – in fact any kind of disturbance due to dust or particles caused by the processing can have a size which image projected on the sensor is larger than one pixel. In this case another model is used by us: The Flake noise model.

Flake noise differs from the Shot Noise by the fact that a pixel error at one pixel also results in errors in the surrounding pixels. One of the simple ways to model flake noise is to start with Shot Noise and let the erroneous pixels grow, like modeling crystal or snowflake growths. This gives name to the term “Flake Noise” or snowflake noise.

Randomly Distributed Noise

There are several distributions which can be used for modeling noise such as Gaussian, Erlang, and exponential. All of these distributions can be modeled as described in [4].

Noise Modeled in the Frequency Domain

One of the noise causes can be due to motion of the sensor, caused by vibrations in the production plant or due to movements of cameras mounted on shaking farming equipment. Note the presence of noise in the frequency domain does not exclude noise in the spatial domain.

I Model-Based design the use of the Matlab function `fspecial` is used to model motion. The result is depicted in Figure 1, where a cropping field is blurred due to motion in vertical direction.

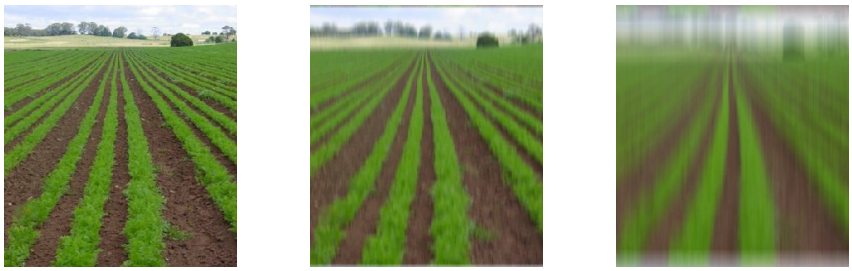

Figure 1: Un-blurred image (a) and 2 blurred images due do modeled motion 10 pixels (picture in the middle) and 50 pixels (rightmost picture).

A clean up of this kind of noise is not impossible, especially if you know how the noise was generated.

NOISE REMOVAL METHODS

Filtering in the Spatial Domain

A well-known noise removal technique is to use low-pass filtering, which is a well-known technique for noise removal technique also used in consumer grade cameras.

Another well-known technique is median filtering – median filtering gives an impressive result with respect to noise removal; the visible result is fine for photography and television as it can be seen in Figure 2. The images have been converted to grayscale; however the same methods apply also for color images in the RGB color space.

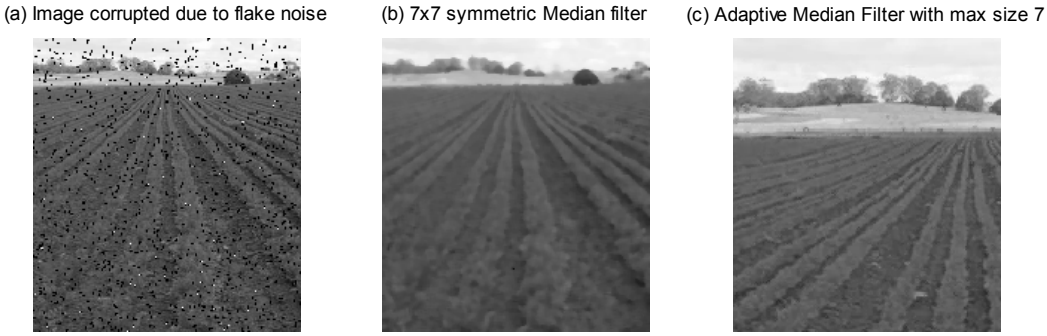

Figure 2: The result of median filtering.

Still the results are not good for neither scientific processing or for production control purposes, nor for control processing; sub-image c in Figure 4, which is enlarged a bit reveals artifacts which will degrade the performance of any edge detect or blob detect algorithm.

The calculations used in median filtering and also for other filters require knowledge of more than one pixel at the time – for the example above a 7x7 matrix is required.

In Figure 3 below the operation of a 3x3 filter is depicted; a small squared matrix moves over the entire image and for each pixel located at the center of the smaller matrix within the image, a calculation is performed and the result is stored in the corresponding position in the resulting image: Usually all pixels in the smaller matrix is used in the calculation, but it might not always be the case.

The exact calculation performed depends upon the filter type; for example a median filter takes the median of each 3x3 sub-matrix. For a detailed description of all the filters please see [2] and for implementation examples please see [4].

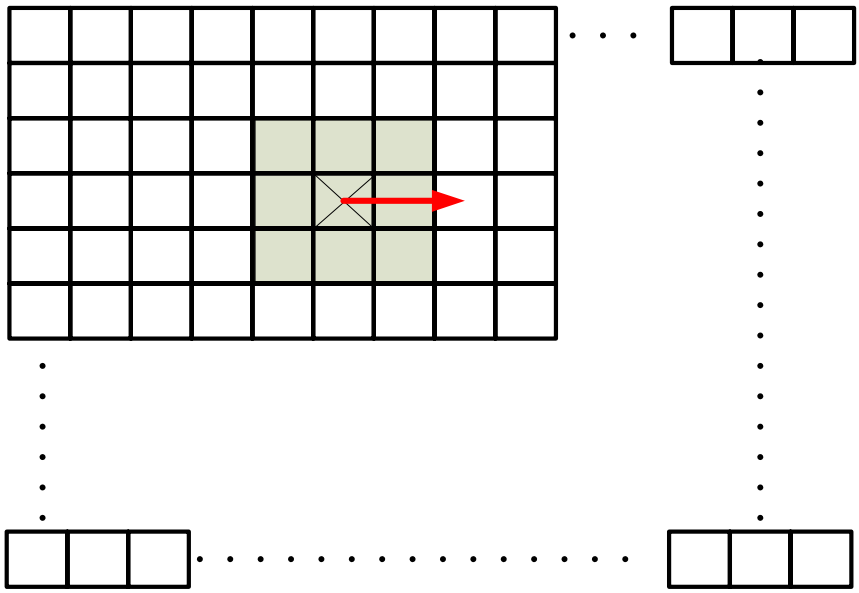

Figure 3: Operation of a 3x3 filter.

These filtering schemes put some requirements for implementations in hardware: first of all the operands needed for each new pixel calculated is located on different locations in the memory, which might be far away from each other. Furthermore if the calculations must run at high speed, the output bandwidth from the image memory must be up to 9 times the calculation speed. Furthermore the calculations of addresses can be quite time-consuming, especially if generic image sizes are to be supported.

Inter-Frame Filtering

Inter Frame filtering is a simple solution which have large advantages over other kinds of filtering; however it does require more images to be generated. The technique is quite simple; just average the pixel values over say 5 frames and divide the result by 5. The number of frames used in for example a video will be divided by 5, so a high speed camera might be necessary. The results of using both the inter-frame filter and the median filter depicted in Figure 4. An alternative version using low-pass filters in the time domain instead of the spatial domain can also be used; in this case the number of frames which are lost is lowered.

There is one problem which should be taken into account; the result will get “colored” by the noise, so post processing the contrast and light might be necessary.

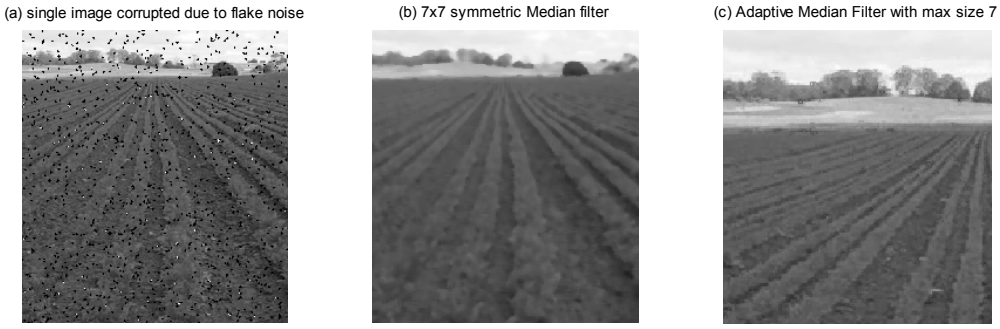

Figure 4: The results of median filtering on inter-frame filtered images.

The inter-frame filter in this version has one advantage compared with the spatial filters: information about the neighboring pixels is not needed; the algorithm only has to sum the pixel values, and divide the result with the number of frames over which the images are averaged.

Filtering in the Frequency Domain

This chapter describes the removal of noise due to filtering in the frequency domain; it includes two major groups of filtering: Filtering with known degradation functions and with unknown degradation functions a.k.a. Blind Deconvolution.

Filtering with known degradation functions

If the degradation function known one efficient function is the Wiener Filtering, Wiener filtering can both remove noise in the spatial domain and in the frequency domain. Other filters are the Inverse-Filter and the Lucy-Richardson Algorithm.

Figure 5: Blurred image (a) and the same image after noise removal (b) using Wiener Filtering.

All filtering in the frequency domain uses the Discrete Fourier transform in 2 dimensions.

Blind Deconvolution

Blind deconvolution differs from the previous function by the fact that the degradation function must be estimated instead of being known in advance. A sample result using the blind deconvolution function is depicted in Figure 6.

Figure 6: Blurred image and the results of the blind deconvolution.

The blind convolution does not provide an impressive result for the noise reduction (Compared with the previous results), but it can be used to provide the input for the Wiener filtering.

Filtering in the frequency domain does require the 2-dimensional FFT to be implemented. This might seem complicated; however as written in [6], the calculations is both simple and easy to understand; take the FFT on each row and store it in an intermediate image, and on this image take the FFT on each column.

This can also be verified from the definition of 2 dimensional DFT:

$$F(u,v) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y) \cdot e^{-j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}$$

Which can be rearranged so that:

$$= \sum_{x=0}^{M-1}\left(e^{-j2\pi(ux/M)}\sum_{y=0}^{N-1} f(x,y)e^{-j2\pi(vy/N)}\right)$$

The result using the above formula on the farming image and on using Matlab's built-in 2-dimensional FFT is depicted in . Note that the `fftshift` function has been used to center the point of zero to the middle of the image.

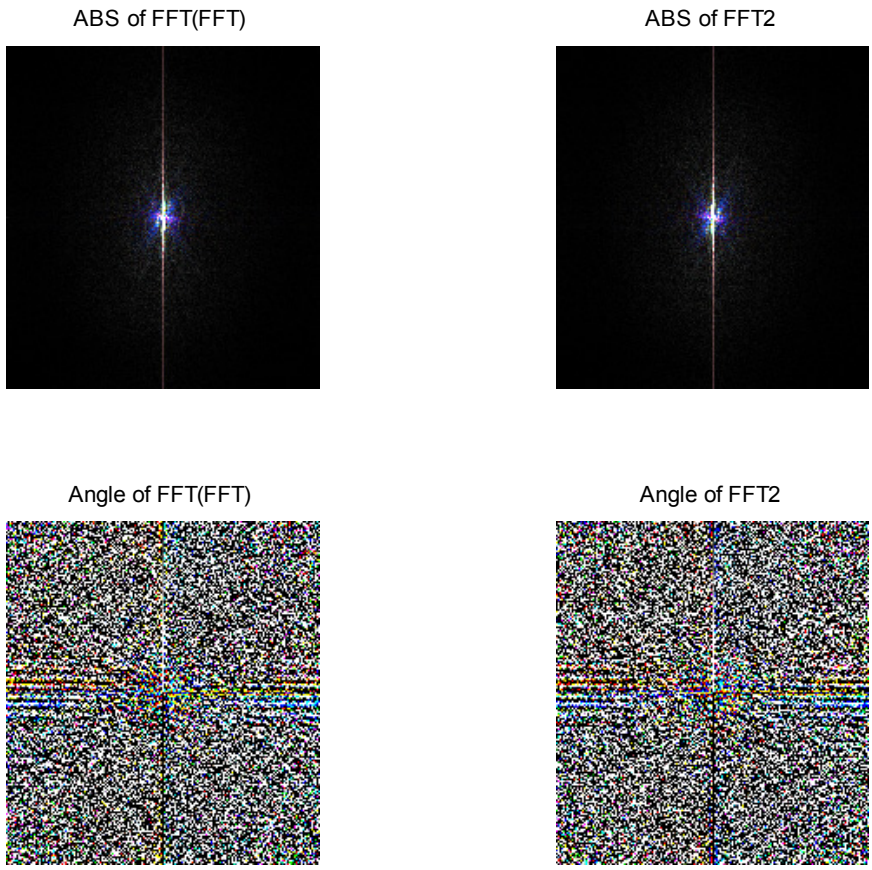

Figure 7: Result from two different implementations of the 2-dimensional FFT.

For sake of implementation on FPGAs on the basic implementation please see [7]. Note that the result from this implementation is to be stored in intermediate storage and reloaded afterwards. If external memory must be used, e.g. dynamic memory the storage scheme must be carefully thought off: an entire image is unlikely to be stored in a single memory page.

OTHER FILTERS

There are other filters which do have an impact on the platform architecture; e.g. the sobel, canny and Prewitt edge detectors. Many filters are convolution filters, and they impact the architecture of the processing platform much like the median filter, i.e. they process each pixel using the neighboring pixels and the pixel itself as input.

ALTERA'S IMAGE PROCESSING INTERFACE

Altera uses its own interface denoted the Altera ST streaming interface. A block using this interface is depicted in Figure 8 **Fejl! Henvisningskilde ikke fundet.**. The interface can handle one pixel per clock cycle, requiring a minimum clock frequency of 30 Hz per pixel at a frame rate of 30 fps. For example a video with a resolution of 640 time 400 pixels requires a clock frequency of 7.68 MHz (progressive scan). For video broadcasting using full HD resolution the requirement is 62 MHz using progressive scan. For most image and video processing applications the Altera Interface will be sufficient.

There are more advantages using the Avalon ST protocol:

- Unrecognized packets must be passed through unchanged, i.e. it is possible to insert control packets into the video stream.
- Avalon ST does also support packets using the YCbCr digital television color model.
- Both interlacing and progressive scan are supported.

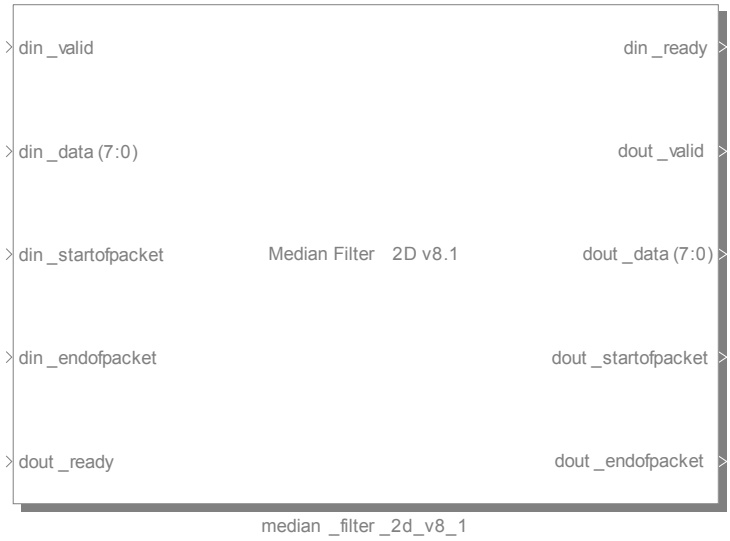

Figure 8: Sample Altera image processing block the Median Filter.

The interface consists of the following signals

Valid data is indicated by setting the value of the control signal ***din_valid*** to 1.
Data is contained signaled over the ***din_data*** signal bus - if used with ***din_startofpacket*** and ***din_endofpacket*** data can also contain control information.
Start of data is indicated by asserting the ***din_startofpacket***; when ***din_startofpacket*** is asserted ***din_data*** must either be (X's means don't care):

- X:X:0, i.e. the lower eight bits must be zero, to indicate start of frame (RGB).
- X:X:1-8 User packet types
- X:X:9-14 Reserved for future Altera use.
- X:X:15 Control Packets

End of data is indicated by asserting the ***din_endofpacket.***
Flow control is used by the ***din_ready*** signal; flow control can be used per input sample, i.e. the data stream can be back pressured within a frame.

Most of the behavior of the block is done by setting the block configuration parameters: Each of the blocks configuration parameters determines how the interface looks in detail, i.e. once the parameters are determined, the operation of the block is fixed.

Altera provides a number of demonstrations for using the Video and Image Processing suite, and the MegaCores of this suite. One example is depicted in Figure 9~~Figure 10~~. Note that this harness is a modified version of an example provided for an elder version of the Altera ST interface.

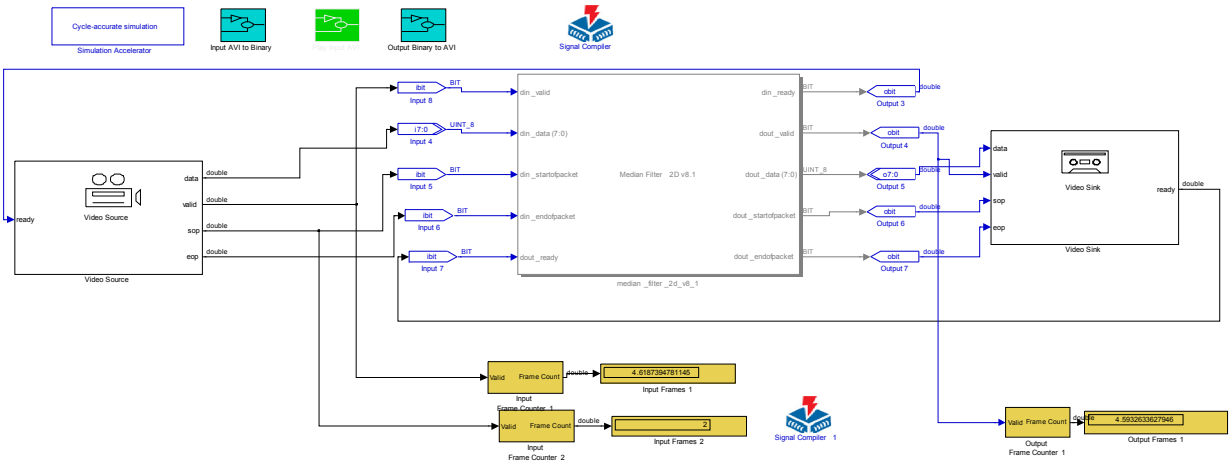

Figure 9~~10~~: Sample Altera Test harness.

Each MegaCore can be translated and tested as “Hardware In the Loop” (HIL) tests using the Altera DSP builder Blockset for Simulink. Note that the test harness even is not cycle-accurate is very slow, and that plain Matlab and Simulink blocks outperform these blocks significantly.

Still having sufficient time for the simulation the time spent pays of as it can be seen from Figure 10 and Figure 11**Fejl! Henvisningskilde ikke fundet.**; Figure 11 is significantly improved due to a simple 3x3 median filter:

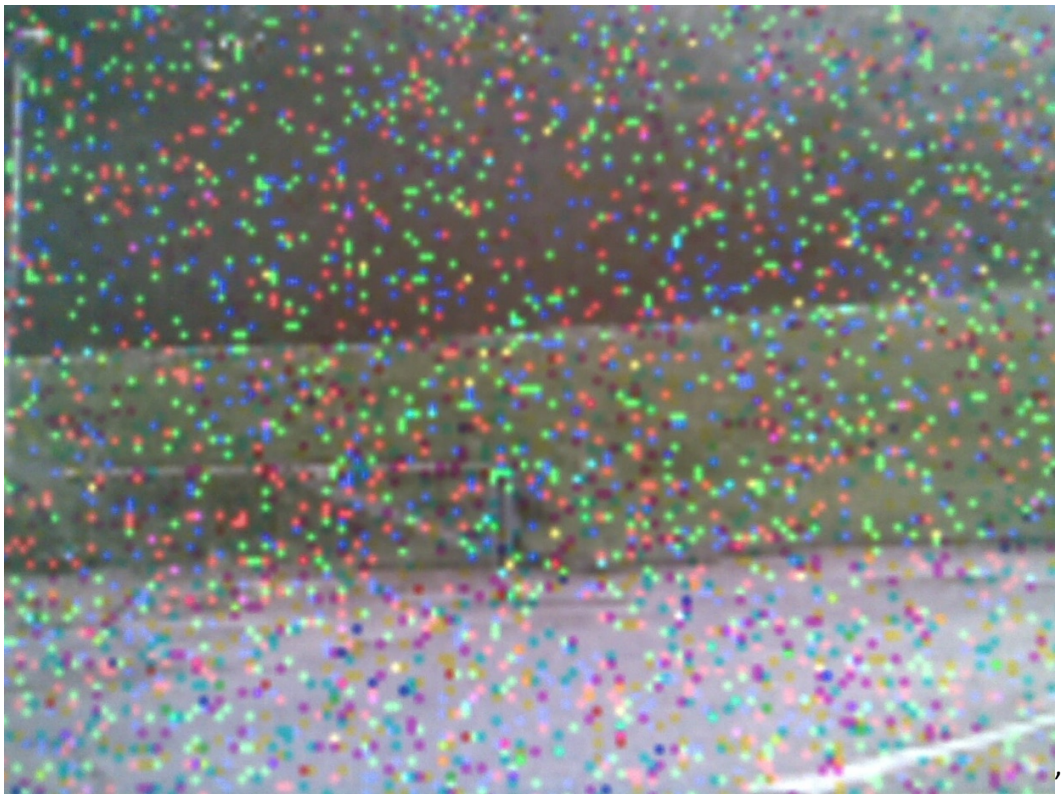

Figure 10: An image of a video movie corrupted by shot noise (excerpt).

Even though the image might not be have sufficient quality for edge detection; edge detection is very sensitive to noise, and although the noise have been reduced considerably, the image is not suited for edge detection, due to artifacts introduced by the median filter.

Figure 11: Another image of the same move after been processed by the median filter.

CONCLUSION

During this document 3 different kind of noise suppression schemes was outlined, and their effects on the bandwidth requirements was also outlined. It must be noted that we have only touched this image enhancement class of filters and that other algorithms and filters might behave slightly different. An outline for the implementation and usage of the FFT blocks was also mentioned.

There are two main reasons for improving images which do puts different requirements on the image enhancement algorithm: If the reason is for viewing an image, a simple median filter will help a lot, the human eye and brain will compensate for the blurring artifacts. On the other hand other means have to be used if edge detection and positioning is the application: the blurring caused by the median filter is a no-go for these applications.

At the very end a brief introduction to the Avalon ST interfaces was given, and the theoretical maximum bandwidth was calculated.

Finally a short demonstration of an Altera median filter was done; this was to indicate the similarities between image and video processing.

REFERENCES

[1] John C. Russ, The Image Processing Handbook, Fifth Edition CRC Taylor & Francis Group, 2006.

[2] Rafael C. Gonzales and Richard E. Woods. Digital Image Processing, Third Edition Pearson International Edition, 2008.

[3] John R. Parker, Algorithms for Image Processing And Computer Vision, Wiley Computer Processing 1997.

[4] Rafael C. Gonzales, Richard E. Woods and Steven L. Eddins. Digital Image Processing Using Matlab, Pearson/Prentice Hall 2004.

[5] Mathworks Inc., the Image Processing Toolbox, Users Guide, the Mathworks Inc.

[6] Steven W. Smith: Digital Signal Processing – A practical Guide for Engineers and Scientists. Newnes, Demystify Technology Series. 2003.

[7] Senior Consultant Carsten Siggaard: An Implementation of a Fast Fourier Transform on a Field Programmable Gate Array A Hardware/Software CoDesign Case: Part 1, Teknologisk Institut, December 2008.

[8] Altera, 101 Innovation Drive, San Jose, CA 95134. Video and Image Processing Suite – Users Guide. Software version 8.1 November 2008.

