FPGA-Accelerated Dehazing by Visible and Near-infrared Image Fusion

Jonas Karlsson

May 20, 2015
Abstract

Fog and haze can have a dramatic impact on vision systems for land and sea vehicles. The impact of such conditions on infrared images is not as severe as for standard images. By fusing images from two cameras, one ordinary and one near-infrared camera, a complete dehazing system with colour preservation can be achieved. Applying several different algorithms to an image set and evaluating the results, the most suitable image fusion algorithm has been identified. Using an FPGA, a programmable integrated circuit, a crucial part of the algorithm has been implemented. It is capable of producing processed images 30 times faster than a laptop computer. This implementation lays the foundation of a real-time dehazing system and provides a significant part of the full solution. The results show that such a system can be accomplished with an FPGA.
Contents

1 Introduction ........................................... 6
  1.1 Scope ............................................. 7
  1.2 Outline ........................................... 7

2 Background ............................................ 7
  2.1 Dehazing ........................................... 7
  2.2 Image Fusion ....................................... 8
    2.2.1 Component substitution ........................... 8
    2.2.2 Multiresolution methods ......................... 8
    2.2.3 Hybrid methods ................................ 9
  2.3 Image Processing on FPGA ........................... 9
    2.3.1 Colour space conversion ......................... 9
    2.3.2 Implemented multiresolution algorithms ......... 9
    2.3.3 Utilise parallelism .............................. 9
  2.4 Image Quality Evaluation ........................... 10

3 Tools ................................................. 11

4 Fusion Algorithm Evaluations ........................ 11
  4.1 Example Images ...................................... 11
  4.2 Evaluation of Fusion Algorithms ................. 13
    4.2.1 HSI ............................................ 14
    4.2.2 DWT ............................................ 14
    4.2.3 Schaul et al. .................................. 15
    4.2.4 Result evaluation ............................... 16

5 Adapting an image fusion algorithm for FPGA ...... 16
  5.1 Adapting the Decomposition Algorithm ............ 17
  5.2 FPGA Design ....................................... 18
    5.2.1 Division ....................................... 19
    5.2.2 Solving the precision problem .................. 19
    5.2.3 Design ......................................... 21
  5.3 Design Implementation .............................. 25

6 Results ............................................... 26
  6.1 Image Fusion Algorithm ............................ 26
    6.1.1 Choice of algorithm ............................. 30
    6.1.2 Adaptation of the algorithm .................... 30
  6.2 Speedups .......................................... 31
  6.3 Verification of design .............................. 32

7 Discussion ........................................... 32
  7.1 He .................................................. 32
  7.2 Night vision ........................................ 33
  7.3 Bilateral filter for decomposition .................. 33
  7.4 Modified DWT ....................................... 34
  7.5 Future work ........................................ 34

8 Conclusion ........................................... 34

9 Acknowledgments ..................................... 34

References ............................................ 35
### Appendix

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.1 Composite part of Schaul implementation in verilog</td>
<td>37</td>
</tr>
<tr>
<td>A.2 Lattice HDR-60 Product Brochure</td>
<td>48</td>
</tr>
</tbody>
</table>
Acronyms

CCD  Charge-Coupled Device
DSP  Digital Signal Processors
DWT  Discrete Wavelet Transform
EPFL École Polytechnique Fédérale de Lausanne
FPGA  Field Programmable Gate Array
GPU  Graphics Processing Unit
HDL  Hardware Description Language
HSI  Hue, Saturation, Intensity
LED  Light-Emitting Diode
MS-SSIM  Multi-Scale Structural Similarity Index
NIR  Near-Infrared
SSIM  Structural Similarity Index
VIS  Visible Spectrum
WLS  Weighted Least Squares
1 Introduction

The quality of images from a vision system can drastically vary with weather conditions such as blizzards, storms, rain, snow, haze or fog. A vision system in a car built to warn the driver of obstacles in the road must be robust to the impact of such conditions but must also be very fast in analysing the sensor information. There exist several image processing algorithms to reduce the impact of weather conditions[1] but they are not directly suited for real-time systems where it is important to produce a stable result within a limited amount of time. A vision system that is able to reduce the impact of harsh weather conditions with only a few milliseconds latency would be of great value.

One method for reducing the impact of haze and fog is to use cameras that can record images using light in the infrared spectrum (the infrared spectrum spans the wavelengths of 700 nm to 1 mm). Rayleigh’s scattering law states that longer wavelength are less scattered by particles in the air[2] which means that infrared light is less scattered than visible light. Consequently, infrared images will be less degraded by haze and fog than ordinary images. Standard digital camera Charge-Coupled Device (CCD)s are sensitive to light wavelengths up to 1100 nm. They are usually combined with a filter that blocks the Near-Infrared (NIR) wavelengths (700 nm to 1100 nm), but if instead equipped with a filter that blocks the visible light, they are capable of recording NIR images. As a result, NIR images are less costly to acquire than infrared images with longer wavelength.

For a well functioning traffic system, vehicles need to be able to communicate with each other, and a key part of this communication is achieved through light signals. For car or ship applications, colour information is very important. A traffic light, a luminous sea mark or other light coding often plays a significant part in traffic interaction. In particular, light signals may have different meaning depending on their colour. You can, for instance, determine if a car is approaching or moving away considering the colour of the light signal you receive. Colour is a concept related to how the human brain interprets the response of the eye to visible light. Since infrared light is invisible to us it has no ‘true’ colour representation.

For a complete vision system built for land and sea vehicles it is therefore important not only to acquire infrared images but also to complement these with colour information, even if degraded, contained in ordinary images of the same scene. For a human observer as well as a vision system it would be favorable if vital parts of such a set of images could be fused and presented as an intuitive but informative single image.

Ordinary single or few-processor computers are unsuitable for computationally intense image processing for safety critical systems. By using dedicated computation platforms such as Graphics Processing Unit (GPU)s or Digital Signal Processors (DSP)s, it is possible to speed up the calculations and get a result in a much shorter and more predictable time than on an ordinary computer. Another example of such a platform is the Field Programmable Gate Array (FPGA) and this is the platform that will be considered in this report.

How can an image fusion vision system that reduces the impact of haze and fog, while preserving colour information, be implemented on an FPGA with only a couple of milliseconds latency? The purpose of this report is to answer this question.
1.1 Scope

This report is part of a thesis project and intends to give a comprehensive account of the work done in the project. The following bullet list describe the methodology of the project.

- **Evaluation of algorithms**
  Evaluation of existing image fusion methods and algorithms with respect to the chosen input data.

- **Choice of algorithm**
  With the application mentioned above in mind, an algorithm was chosen.

- **Adaptation of the algorithm**
  In order to be able to implement the algorithm on an FPGA, it needs to be adapted.

- **FPGA design and implementation**
  A design suitable for an FPGA was made to ensure an efficient and robust implementation.

- **Simulation**
  For debugging and verification purposes, the implementation was simulated.

A series of questions arise that this report will address. Is it possible to develop an image fusion algorithm for Visible Spectrum (VIS) and NIR images with only a couple of milliseconds latency? What types of algorithms are suited for the fusion of VIS and NIR images? Has the most suitable algorithm been fully or partly implemented on an FPGA?

1.2 Outline

In the section Background the theoretical background necessary to fully comprehend the content of this report is presented. The section Fusion Algorithm Evaluation presents the image data used and an evaluation of existing image fusion algorithms. The criteria for selecting one of these algorithms are also specified.

The section Adapting an image fusion algorithm for FPGA describes how the chosen algorithm was adapted for an FPGA implementation as well as the challenges of the FPGA design, the implementation and the simulation procedures.

In the section Results examples of fused images are presented, together with a measurement of the improved execution time for the algorithm. Alternative fusion methods and their possibilities are touched upon in the discussion section and the report ends with the conclusions that can be drawn from the present study.

2 Background

2.1 Dehazing

Most work in computer vision is based on the premise that all light rays from the objects reach the observer without attenuation or alteration [3]. Most sensors have been created to function on clear days. In reality, particles in the atmosphere cause light to be scattered resulting in reduced visibility. The amount of scattering depends mainly on the type and size of the particles and their concentration in the atmosphere [3]. The visual result of the scattering is loss of contrast and blurring of distant objects. Contrast enhancement can be used for image dehazing but as the effect of haze and fog is not constant over an image, this technique cannot be applied globally. There are techniques to perform dehazing on a single image based on a physical model of how haze deteriorates an image, but this algorithm provides poor result in images without fog or haze [4].

In 2008 a new approach for dehazing was proposed by Schaul et al. [5]. Instead of using elaborate dehazing algorithms on a single image based on physical models, they consider the same scene with two images. One image is taken in the VIS, and the other in the NIR part of the spectrum. Due to the relatively large wavelengths small particles do not scatter NIR light much. This physical
phenomenon makes haze and to some degree also mist more transparent to NIR light than to VIS light. With this solution a new problem arises: how to fuse two images into one so that maximum contrast and colour information is preserved?

2.2 Image Fusion

Image fusion is the procedure of generating a fused image in which the information carried by each pixel is determined from a set of images [6]. Image fusion is a wide research area. It has applications in fields such as remote sensing (space and earth observation), computer vision and medical image analysis [7].

Pixel-level image fusion is a method where a fusion rule is applied to every two corresponding pixels in a pair of images. Even though this is not the only existing image fusion method it will be the one considered to narrow the scope of this report.

An important prerequisite for image fusion is for the image pair to be aligned to one another, usually referred to as image registration [6]. Misalignment of the edges between the two images produces artifacts in the resulting fused image [8]. Image registration is difficult to achieve and there are extensive studies on the subject. In most image fusion research it is assumed that the images in the set are perfectly aligned [6], an assumption that is also made in this report.

2.2.1 Component substitution

Component substitution is an image fusion method. Apart from the 2D coordinates, colour models provide each pixel with additional components like, in the RGB model, the intensity of red, green and blue. By replacing one or more of these components with the corresponding components from another image, the resulting image will be a combination of the two images. The method is often used because it is fast and easy to implement but may yield poor results in terms of spectral fidelity [9].

2.2.2 Multiresolution methods

Multiresolution is a common technique in pixel-level fusion. Unique detail information can be observed at different resolutions of an image [10]. By decomposition of the input image set in a number of resolutions, these details can be gathered into contrast detail images. Every resolution level of an image is also stored as an approximate of the original image in that resolution. The fusion process is then performed pixel-wise according to some more or less elaborate criteria. A fused image is acquired by inverting the decomposition process [3].

In 2006 the IEEE Geoscience and Remote Sensing Society held a data fusion contest on pansharpening algorithms, a technique used in remote sensing of multispectral images. From the contributions Alparone et al [9] came to the conclusion that algorithms based on multiresolution methods generally performed better than those based solely on component substitution. Two algorithms outperformed the others in the contest and they share the same philosophy in taking into account the instrumental-related physical model.

A multitude of decomposition techniques have been proposed such as different pyramid decompositions, wavelet decompositions, bilateral filters or weighted least squares optimisations.

There exist several wavelet-based fusion algorithms and each has its own advantages and limitations [11]. Comprehensive testing is required in the choice of an appropriate wavelet-based fusion for any given condition. In a survey article by Irshad et al. [12] the authors conclude that it is impossible to design a universal method for image fusion. Not only the specific purpose but also the characteristics of the sensors and the application-dependent data properties should be taken into account. The authors also call for a multiresolution decomposition method that is edge-preserving.

Bilateral Filter is a smoothing algorithm that was introduced by Tomasi et al. [13]. By means of nonlinear combination of neighboring pixel values it removes high frequency components of an image without blurring its edges. The algorithm is non-iterative and is considered to be simple and intuitive [14, 15].
In a recent article, Schaul et al. [5] propose a multiscale image fusion algorithm that uses two images, one visual and one NIR. Instead of the standard multiscale decomposition solutions with pyramid decomposition or wavelet-transformation, Schaul et al. use a weighted least squares (WLS) optimisation framework originally introduced by Farbman et al. [16]. One important feature of Farbman et al. is that their filter algorithm is edge-preserving. Halo artifacts, a problem that occurs in the fusion of VIS and NIR images with the most popular decomposition methods [17, 18], are for the most part avoided. Schaul et al. motivate their choice of WLS over the wavelet approach by pointing out that WLS is overcomplete, resulting in a more robust solution.

2.2.3 Hybrid methods

There are several implementations of image fusion that mix different algorithms. A hybrid solution for image fusion is to combine standard fusion schemes, such as component substitution, with multiresolution decomposition methods, for instance using wavelets [11]. This is often done by performing multiresolution decomposition on a single component e.g. on the intensity value of the hue-saturation-intensity (HSI) colour model.

2.3 Image Processing on FPGA

Most image processing algorithms are developed for PC platforms which usually cannot handle the massive amount of data that is produced by a real-time image processing system [19]. Image processing algorithms are rather convenient to parallelise since an image is not an entity but a collection of pixels that can be processed in parallel. Even though some algorithms consider a region of support pixels they are still parallelisable [20].

A flexible and powerful tool for parallel computing is an FPGA [21, 22]. FPGA technology provides a fast, compact, and low-power solution for image fusion [23].

2.3.1 Colour space conversion

Colour space conversion is a common image processing tool used for multiple applications and for which FPGA designs already exists. YCbCr is a colour model used in television transmissions and it is also often used in image processing. In an article by Ahirwal et al. [24] an implementation of the conversion between the colour spaces RGB and YCbCr on an FPGA succeeds in processing one pixel in 15 ns on a Xilinx 2s200pQ208-5.

2.3.2 Implemented multiresolution algorithms

There are a few examples of multiresolution algorithms that have been implemented on an FPGA. In 2006, Sims et al. [21] presented what they call the first hardware implementation of pattern-selective pyramidal image fusion. Their fusion algorithm uses pyramids and edge-filters which they implement on an Virtex-2 FPGA to fuse two gray-scale VGA videos in real time. On the FPGA, their system performs the image fusion over 100 times as fast as on a 2.8 GHz Pentium-4.

Bartys et al. [25] have developed a real-time hardware system called UFO to register and fuse VIS and long wavelength infrared images. Their intention is for the system to be mobile which means that there are constraints on power consumption and size and yet they need massive processing throughput. Using an FPGA operating at 150 MHz, their system was capable of producing 25 frames per second with the resolution 640x480.

2.3.3 Utilise parallelism

In a recent paper Li et al. [22] propose a double parallel scheme for edge detection in an image. The double parallel scheme involves two abstraction layers. The top layer, called the image parallel part, divides the image into two pieces which are processed in parallel. Each piece is processed by the lower layer which realises another parallel scheme, performing a median filter and edge detection. Taking advantage of the double parallel scheme Li et al. performed the same calculations 3.8 times...
faster on a 50 Mhz Cyclone II EP2 FPGA than on a 2.66 Ghz Intel core 2 quad processor. At the end of the paper, the limitations of the FPGA implementation is discussed, and its is pointed out that the image parallel part can be divided into more than two parallel pieces to increase the performance. It is also mentioned that the double parallel architecture can be applied to other image processing methods.

A bilateral filter is a widespread edge-preserving noise-reducing smoothing filter that can be used to generate a multiresolution representation of an image [3]. Gabiger et al. [20] present an FPGA design of the original bilateral filter and realise it as a highly parallelised pipeline. For a filter window of 5x5 pixels the bilateral filter needs to compute 24 weights for each pixel. In order to keep up with the reading of the image data there has to be one output pixel for each input pixel, i.e. it is necessary to calculate all weights during this clock cycle. This is made possible by using a secondary clock with quadruple speed and by dividing the window pixels into six groups which are processed simultaneously. With a Virtex-5 FPGA platform with a maximum clock frequency of 220 Mhz, the implementation of the design was capable of generating 30 frames per second at a full resolution of 1024x1024 pixels and 40 Mhz data rate.

2.4 Image Quality Evaluation

According to Wang et al. [26] the most used image quality evaluation algorithms are the following:

- Peak Signal-to-Noise Ratio (PSNR)
- Mean Square Error (MSE)

The problem with these algorithms is that they do not correlate well with how the image quality actually is perceived. Plenty of research has been done in an attempt to simulate the human visual system components in order to evaluate visual perception, and can briefly be summarised as:

- Pre-processing
  Can involve image adjustment, colour space transformation etc.

- Channel decomposition
  Transformation of the image into different spatial frequencies as well as into orientation selective sub bands.

- Error Normalization
  Weighting of error signals in each sub band.

- Error Pooling
  Combining the error signals in each sub band into one single quality/distortion value.

Unfortunately these kinds of methods (such as PSNR, MSE, etc.) have their limitations. The human visual system is a complex and non-linear system. Most early models for simulation of the visual system are based on linear and quasi-linear operations that make use of restricted and simplistic stimuli. They are dependent on strong assumptions and generalisations[27].

Structural Similarity provides an optional and complementing approach to the image quality assessment problem. It is based on the assumption that the human visual system is strongly conformed to extract structural information from the receiving image. The measurement of structural differences between images is for that reason considered as a good approximation of perceived image quality. It has been shown that Structural Similarity Index (SSIM), which is a simple implementation of this methodology, performed better than previous state-of-the-art measuring instruments for perceived image quality.

Borse et al. [28] have made a comparative analysis of 10 existing image quality assessment methods. The analysis is based on subjective as well as objective measurements. One image is distorted with 11 different techniques and then compared to the original image with all the image quality assessment methods. The authors came to the conclusion in the article that SSIM is the algorithm
that best correlates with the subjective results. They call for further research for the improvement of the SSIM algorithm.

Zaric et al. [29] have also made a subjective and objective analysis. This analysis compares three algorithms. Among these SSIM generally produced results with the greatest correlation with the subjective results even though the algorithm performed poorly when the sample images had been put through minor distortions.

It should be noted that SSIM is an approach based on a single-scale method. This means that the index is not dependent on the display conditions of the image such as screen resolution and distance to the viewer. By using a multi-scale approach, Multi-Scale Structural Similarity Index (MS-SSIM), it is possible to calibrate the parameters that weight the relative significance of different scales of the image [26].

3 Tools

The development, adjustment and evaluation of the image fusion algorithms was done with Mathwork's software MATLAB, and in particular with the Image Processing Toolbox which provides competent and intuitive routines for such purposes. The MATLAB script Colorspace by Pascal Getreuer [30] was used to perform the conversions between different colour spaces.

The implementation of the VIS and NIR image fusion algorithm was developed with a Lattice HDR-60 video camera development board in mind but an equivalent board should work just as well. The development board specifications can be found in the appendix.

For the setup of the environment for the HDR-60 video development board the Lattice software tool Diamond was used. The compilation, development of the test bench and debugging were carried out using the Active HDL tool by Aldec in the Hardware Description Language (HDL) Verilog.

4 Fusion Algorithm Evaluations

4.1 Example Images

For the development and evaluation of the fusion algorithms, suitable data samples are needed. The ideal sample would be two video streams taken in 25 fps, one with VIS images and one with NIR images. The ideal scene would have plenty of haze or fog and both light bulbs and Light-Emitting Diode (LED) lights in different colours. Acquiring such data samples turned out harder than expected. A NIR camera was not available, and neither video nor still images of the desired kind could be found on the Internet. In particular, not even a single pair of images could be found that contains haze and coloured light bulbs or LEDs at the same time. There were however some image pairs that fulfilled some of these requirements separately.

For the benefit of research in image fusion, a group at École Polytechnique Fédérale de Lausanne (EPFL) have published a large set of VIS and NIR images for different scenes [31]. Presented in this section is a selection of image pairs that together represent the ideal scene to perform the image fusion upon. The fusion algorithms in this thesis are developed and tested considering five images from the EPFL data set.

Two image pairs (figure 2 and 3) contain fog and haze in their VIS images, something that the algorithm must be able to reduce using the corresponding NIR image. In some of the images (figure 2, 4 and 6) the feature of NIR images to highlight vegetation may affect the fusion of the image pairs in a negative manner. Three of the image pairs (figure 4, 5 and 6) have thin spectral light sources, not visible in the NIR images. The fusion algorithm must be able to preserve these light sources.
Figure 2: Image set “0001”. The first image pair has a VIS image that has rich vegetation in the foreground and is very hazy. The corresponding NIR image has a very high contrast background and the foreground is characterised by highlighted vegetation.

Figure 3: Image set “0016”. The second image pair is similar to the first one in the sense that in the VIS image the background is obscured by haze. Note that a distant boat is visible only in the NIR image.

Figure 4: Image set “0038”. The traffic light is likely a LED light, and is invisible in the NIR image.
Figure 5: Image set “0051”. The fusion algorithm must work for both normal and hazy weather conditions. This scene also features a sign which is red in the visual, but invisible in the NIR image.

Figure 6: Image set “0098”. Neon lights also have a narrow spectral band. In this scene these appear as lines illustrating that a fusion algorithm for traffic applications must not only work for spot light sources.

4.2 Evaluation of Fusion Algorithms

With the sample images as reference the best image fusion algorithm can be found. The general objective is to attain a single image that contains as much information as possible from a pair of VIS and NIR images. More specifically, the goal is to find the image fusion algorithm that retains most contrast information from the pair of images at the same time as preserving the colours of the VIS image. There are some additional challenges that the image fusion algorithm needs to be able to handle. Light sources such as LEDs have a narrow frequency range in the visual spectrum and no spectral leakage to the NIR spectrum. Such light sources are therefore not registered in NIR images. Since LEDs are often used in traffic lights and vehicle lamps this problem must be addressed.

In addition, the image fusion algorithm should also be able to handle the problem with highlighted vegetation in NIR images. The resulting image from the fusion should be as alike the original VIS image and as free from discoloured vegetation as possible.

The following three image fusion algorithms have been identified as particularly promising,

- Hue, Saturation, Intensity (HSI)
- Discreet Wavelet Transform (DWT)
- Schaul et al

An additional advantage is that for all three an implementation in terms of a MATLAB script is available.
4.2.1 HSI

A common image fusion algorithm frequently used in remote sensing is the HSV or Hue, Saturation, Intensity (HSI) image fusion algorithm. It is a component substitution algorithm where the VIS image is first transformed into the HSI colour model. The intensity part of the image is then replaced by the NIR image and the merged image is transformed back into the original representation. Since there is no need for any pixel processing this technique performs image fusion very fast. Unfortunately two significant problems were encountered when using component substitution for image fusion of VIS and NIR images among the sample images. Since many lamps only emit light in the visual region they appear black in NIR images. Dark pixels have low intensity and since the fusion algorithm uses the intensity from the NIR image and the colours from the VIS image, it will try to colour a pixel that is already black which results in yet another black pixel. With component substitution methods such as the HSI algorithm, all lamps that are invisible in the infrared will be either darkly coloured or simply black.

By changing the substituted intensity image into a mixture of the VIS and NIR image the lamp problem was partly solved.

\[
\text{newIntensity} = \text{saturation}_{RGB} \cdot \text{intensity}_{RGB} + (1 - \text{saturation}_{RGB} \cdot \text{intensity}_{NIR})
\]

Overexposed lamps however still end up black since HSI conversion of completely white pixels gives low saturation values and with the mixture this affects the substituted image.

Moreover HSI image fusion of images with much vegetation results in an unnatural looking image where plants and trees appear illuminated. Vegetation reflects much NIR light and taking the intensity from the NIR image will transfer this effect to the final fused image.

Figure 7: The HSI image fusion with vegetation results in an unnatural highlight on plants and trees.

4.2.2 DWT

In the field of image fusion Discrete Wavelet Transform (DWT)\text{s} are commonly used and there are several slightly different implementations. The MATLAB Wavelet Toolbox has a built-in function \texttt{wfusimg} that performs image fusion using wavelets. The wavelet type, the amount of decomposition layers and the fusion rule are with this function user-defined. According to Amolins \textit{et al.} [11], a component substitution hybrid method is better than a plain wavelet image fusion. Performing the wavelet image fusion on the intensity part of the image the best results were found by using the wavelet \texttt{db2} (a Daubechies wavelet) [32] and linear fusion rules for both the approximation images and the detail images.

Compared to the result from the HSI algorithm, the DWT from the toolbox preserved more details in the presence of vegetation in the images. The result shows however a loss in colour diversity.
Figure 8: Comparison of HSI and DWT image fusion. The DWT image fusion is a hybrid image fusion algorithm of DWT and component substitution. Even though the result is much better than the HSI substitution there is a loss in colour diversity.

4.2.3 Schaul et al.

Schaul et al. [5] introduce a different multiresolution image fusion algorithm. As already mentioned, with the help of a novel edge preserving decomposition method, the Weighted Least Squares (WLS) optimisation framework by Farbman et al. [16], they were able to remove edge residing artifacts present in images fused with common decomposition methods [5]. For short, this algorithm will be labeled Schaul in this report.

The Schaul algorithm can, like other multiresolution methods, be split into three steps: analysis, criterion and synthesis.

The Schaul approach takes the VIS and NIR input images and generates a sequence of more and more smoothed images, referred to as coarse or coarsend images, using the WLS framework seen in equation 1 described by Farbman et al[16].

$$I_{k+1}^c = W_{\lambda_0^c}(I_0)$$ (1)

At the first level, there are four images, the two input images which will be entitled detailed images, and the two corresponding coarse images. From these four images two contrast images, one for VIS and one for NIR, can according to the equations 2 and 3 be derived. This procedure is repeated \( k \) times, but for each iteration it is done on input images with increased smoothness, or coarseness, and reduced size as compared to the previous iteration. In this way details can be extracted from multiple resolutions.

$$visContrast_k = \frac{visDetail_k - visCoarse_k}{visCoarse_k}$$ (2)

$$nirContrast_k = \frac{nirDetail_k - nirCoarse_k}{nirCoarse_k}$$ (3)

$$composite_k = composite_{k-1} \cdot [\max(visContrast_k, nirContrast_k) + 1]$$ (4)

According to a maximum pixel value criterion at each resolution level a composite image is constructed, according to the equation 4. The composite contrast image is merged, in equation 5, with the coarsend image of the lowest resolution synthesizing a fused image.

$$fusedImage = composite_n \cdot visCoarse_n$$ (5)

where \( n \) is the number of resolution levels.

The algorithm contain parameters for both the multiresolution method and the WLS filter. The WLS filter is controlled by two parameters, \( \lambda \) and \( \alpha \) as seen in equation 1. \( \lambda \) balances two terms, the data term and the smoothness term. Increasing \( \lambda \) will result in a smoother output image. The
parameter $\alpha$ determines the sensitivity to the gradients of the input image. Increasing the parameter will result in sharper preserved edges.

The multiresolution part adds two parameters, the number of multiresolution layers $n$ and a number, $c$, which determines the magnitude of the coarsening between the layers.

In the Schaul algorithm, $\lambda = \lambda_0 c^k$ from equation 1, controls the approximate image at the layer $k+1$ while $\lambda_0$ is the amount of coarseness of the first image. The authors chose the parameters as: $\lambda_0 = 0.1$, $c = 2$ and $n = 6$.

The presented algorithm has been implemented in MATLAB by Petykiewicz and Buckley [33]. In their solution the WLS filter is included as well. For the modification of the intensity image they use colour substitution with the colour model $YCbCr$.

Figure 9: Image fusion performed with Petykiewicz-Buckley’s implementation of the algorithm proposed by Schaul et al. with the parameters $\lambda_0 = 0.1$, $c = 2$, $n = 6$ and $\alpha = 0.9$.

4.2.4 Result evaluation

The evaluation of the three algorithms was made considering four different properties of the fused images.

- Colours preservation
- Spatial resolution for details inherited by VIS
- Spatial resolution for details inherited by NIR
- Light source preservation

In order to get a good objective measurement of these aspects the image quality evaluation algorithm MS-SSIM was used. As detailed in the Results section below, the HSI method performed best for the spatial resolution inherited by VIS. For the properties colour preservation, spatial resolution inherited by NIR and light source preservation, the Schaul algorithm performed best. Considering that Schaul et al. have a similar application in mind as the one presented in this report, the fusion of VIS and NIR images, these results are not surprising.

As a result of the image quality analysis in the Result section, Schaul was chosen as the image fusion algorithm that is best suited for the applications focused at in this report.

5 Adapting an image fusion algorithm for FPGA

Due to the fact that the Schaul algorithm is developed for the fusion of a VIS and NIR image pair, a less common application of image fusion, no implementation of that algorithm for FPGA could be found. For the part of the algorithm that deals with the conversion between the RGB and $YCbCr$ colour representation, several implementations were found (such as [34] and [35]) but the rest of the Schaul algorithm seems to lack any implementation for FPGA. Implementing the full
solution is an extended project, and the contribution of this thesis is to present an FPGA design and implementation for the actual image fusion step, i.e. for the construction of the composite image, and the multiplication with the coarsend VIS image. It should be noted that this part together with the image decomposition part, WLS, make out the non-trivial and non-previously surveyed parts of the Schaul algorithm. Even though the acceleration of the WLS part has not been carried out here (to keep the efforts within the limits of a thesis project), the presented design of the composite part brings a fully accelerated Schaul algorithm much closer to reality.

5.1 Adapting the Decomposition Algorithm

The Schaul algorithm achieves a multiresolution representation of the images by performing decomposition in several layers. Using the algorithm with only one layer, $n = 1$, provides a result that does not differ drastically from the result using more layers. It does however speed up the calculations a lot. Considering the applications mentioned in the introduction, the aim is as much for speed as for image quality. Using one layer would also greatly reduce the magnitude and complexity of the implementation since only one composite image needs to be calculated.

Figure 10: Changing the settings of Schaul to only one layer, $n=1$, does not drastically affect the result using more layers.

In figure 10 it can be observed that for $n = 6$ the details in the foreground are preserved better than for $n = 1$. However the details inherited by NIR are less apparent in the $n = 6$ example image. This can also be seen if applying the image quality evaluation algorithm MS-SSIM (see the section Results).

In order to gain some extra control of the resulting image, the Schaul algorithm is supplemented by five offset parameters. Four of these offset the pixel intensity of the input images and their coarsend counterparts, and the last offsets the composite image pixel intensity.

Figure 11 shows the flow chart of Schaul after the adaptation.
5.2 FPGA Design

In order to take full advantage of the possibilities that an FPGA solution can provide, and exploiting the fact that much image processing can be done pixel by pixel without inter-pixel interference, the design was made in a pipelined parallel manner. The pixels of an image are fed into the pipe one after the other, in such a manner that a new pixel is entered while several previous pixels are still at various stages of processing. Additionally, this is done for all four input signals which makes it even more parallel. For an easy view of the solution and to keep track of the timing in the parallel pipeline, the design was developed to perform only one operation for each step to allow one clock cycle per step. Therefore the three stages marked in figure 11 as green were divided into nine steps.

The pseudo-code for the three stages is presented in equation 6.

\[
\begin{align*}
\hat{vis}_{\text{Detail}} &= vis_{\text{Detail}} - VIS_{\_}\_DETAIL_{\_}\_BRIGHTNESS \\
\hat{vis}_{\text{Coarse}} &= vis_{\text{Coarse}} + VIS_{\_}\_COARSE_{\_}\_BRIGHTNESS \\
\hat{nir}_{\text{Detail}} &= nir_{\text{Detail}} - NIR_{\_}\_DETAIL_{\_}\_BRIGHTNESS \\
\hat{nir}_{\text{Coarse}} &= nir_{\text{Coarse}} + NIR_{\_}\_COARSE_{\_}\_BRIGHTNESS \\
\hat{vis}_{\text{Contrast}} &= vis_{\text{Contrast}} + VIS_{\_}\_CONTRAST_{\_}\_INTENSITY
\end{align*}
\]
\[ \text{nirContrast} = \text{nirContrast} + \text{NIR\_CONTRAST\_INTENSITY} \]

\[ \text{visContrast} = \frac{\text{visDetail} - \text{visCoarse}}{\text{visCoarse} + \text{VIS\_COARSE\_BRIGHTNESS}} \]

\[ \text{nirContrast} = \frac{\text{nirDetail} - \text{nirCoarse}}{\text{nirCoarse} + \text{NIR\_COARSE\_BRIGHTNESS}} \]

\[ \text{composite} = \max(\text{visContrast}, \text{nirContrast}) \cdot \text{visCoarse} + \text{COMPOSITE\_BRIGHTNESS} \]

\[ (6) \]

The upper case entities are offsets, determined before the execution. In order to better illustrate the problems that arise in the designing procedure these constants are taken to be zero, so that

\[ \text{visContrast} = (\text{visDetail} - \text{visCoarse}) \cdot \frac{1}{\text{visCoarse}} \]

\[ \text{nirContrast} = (\text{nirDetail} - \text{nirCoarse}) \cdot \frac{1}{\text{nirCoarse}} \]

\[ \text{composite} = \max(\text{visContrast}, \text{nirContrast}) \cdot \text{visCoarse} \]

where all quantities have pixel values that can assume any value between 0 and 255. In the full FPGA implementation reported in the Appendix, the constants have been included. Since an FPGA cannot directly perform division, except through bit-shifting, in the above expressions the divisions by visCoarse and nirCoarse require special attention, and will be discussed in some detail.

5.2.1 Division

Since the pixel intensity ranges from 0 to 255 there are 256 possible results to the division with an input pixel value. A fast and resource efficient way to perform division for a limited number of values is to place the result in a look up table. In that way one can avoid the introduction of an extensive float representation that will drain resources on the FPGA and increase the computation time. However, for the FPGA application the look up table should only contain integer approximations to the result of the division. It will therefore be built not for \( \frac{1}{\text{pix}} \) but for \( 2^n/\text{pix} \), with \( \text{pix} = 1, 2, \ldots, 255 \), and for some \( n \) which is large enough for the round off errors to be small, and it has to be accompanied by a subsequent division by \( 2^n \).

The usefulness of the factor \( 2^n \) is the following. A logical shift is a bitwise operation that shifts all bits to either right or left a given number of bit positions and fills the vacant bit positions with zeros. It can be used as an efficient tool for performing multiplications and divisions by \( 2^n \) due to the fact that shift operations are easily performed on an FPGA. Two things should be kept in mind though. The first thing is that performing a logic right shift will result in a division by two and round off the value, not to the nearest integer value but always to the lower value (the least significant bit is shifted away). Secondly, performing logical shifts on a signed negative integer will ruin the concept that the two-complement representation of these integers is based upon and will produce incorrect results.

5.2.2 Solving the precision problem

There are two conditions that affect the choice of exponent \( n \) used in the construction of the division look up table. First, the round-off errors must be kept under control. Ideally, \( n \) should be chosen so large that the round-off error in \( 2^n/\text{pix} \) is negligible for all values of \( \text{pix} \). If for example \( n = 16 \), in the worst case of \( \text{pix} = 255 \), the round-off error is of the order of 1, or \( \sim 1/2^8 \sim 0.4\% \), while for \( \text{pix} = 16 \) the error is \( \sim 1/2^{12} \), a much smaller number. For the applications at hand, a large value like \( n = 16 \) seems uncalled for. However, choosing instead \( n = 8 \) leads to a worst case error
of $\sim 100\%$. For a more typical case, consider the following example where the factor $2^8$ is used to multiply and divide:

$$\text{composite} = [(\text{nirDetail} - \text{nirCoarse}) \cdot \text{oneOver}(\text{nirCoarse}) \cdot 2^8] \cdot \text{visCoarse} \cdot 2^{-8} \quad (7)$$

and consider a situation where $\text{nirDetail} = 232$, $\text{nirCoarse} = 225$ and $\text{visCoarse} = 255$

$$\text{composite} = [(232 - 225) \cdot \text{oneOver}(225) \cdot 2^8] \cdot 255 \cdot 2^{-8}$$
$$= \left[7 \cdot \frac{1}{225} \cdot 2^8\right] \cdot 255 \cdot 2^{-8}$$
$$\approx [7.96] \cdot 255 \cdot 2^{-8}$$

Due to the logic shift all real values are rounded off to the lower value.

$$\approx [7] \cdot 255 \cdot 2^{-8}$$
$$= 1785 \cdot 2^{-8}$$
$$\approx 6.97$$
$$\approx 6$$

The actual value of the algorithm without approximations is 7.93 i.e. the error is about 25%.

To further illustrate this point, consider the sample image "0001". Let the maximum approximation error be the largest difference between the actual result and that of the algorithm, considering all pixels. Table 1 shows this error, as well as the corresponding mean of the differences, for a range of $n$-values.

Table 1: Approximation error of sample image “0001”. $2^n$ is the factor used in the calculations to maintain precision without the need of introducing a float representation.

<table>
<thead>
<tr>
<th>$2^n$ factor</th>
<th>Maximum approximation error</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^8$</td>
<td>18</td>
<td>3.98</td>
</tr>
<tr>
<td>$2^9$</td>
<td>11</td>
<td>1.95</td>
</tr>
<tr>
<td>$2^{10}$</td>
<td>4</td>
<td>0.80</td>
</tr>
<tr>
<td>$2^{11}$</td>
<td>3</td>
<td>0.39</td>
</tr>
<tr>
<td>$2^{12}$</td>
<td>2</td>
<td>0.20</td>
</tr>
<tr>
<td>$2^{13}$</td>
<td>1</td>
<td>0.10</td>
</tr>
<tr>
<td>$2^{14}$</td>
<td>1</td>
<td>0.06</td>
</tr>
</tbody>
</table>

The second condition on the exponent $n$ comes from how multiplication is implemented in an FPGA. The intensity value of a gray-scale pixel requires 8 bits. When considering signed 8 bit integers an extra bit is needed for the sign information. The HDR-60 development board is equipped with a DSP system called sysDSP that have a number of dedicated multipliers with widths of 9x9, 18x18, 36x18 and 36x36 bits.

Table 2: Available multipliers and their operating frequency on the HDR-60 development board.

<table>
<thead>
<tr>
<th>Multiplier</th>
<th>Available</th>
<th>Speed (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>9x9</td>
<td>256</td>
<td>420</td>
</tr>
<tr>
<td>18x18</td>
<td>128</td>
<td>420</td>
</tr>
<tr>
<td>36x36</td>
<td>32</td>
<td>281</td>
</tr>
</tbody>
</table>

In order to efficiently use the available resources and optimise speed it is preferable to use small multipliers and try to avoid using the 36x36 multiplier. The first multiplication in the design
is between the difference of two pixel values and the \( \text{oneOver()} \cdot 2^n \) factor, \( (\text{Detail} - \text{Coarse}) \cdot \text{oneOver(\text{Coarse})} \cdot 2^n \). The difference of the pixel values can assume negative values. In practice the difference is very seldom larger than 128. In order to fit the multiplication in the 18x18 multiplier, thresholds were placed at -128 and 127 so that the value can fit within 7 bits plus an extra sign bit for the sign representation. To process this product with the next product in the algorithm, \( \text{composite} = \max(\text{visContrast}, \text{nirContrast}) \cdot \text{visCoarse} \), the result of this product need to be at most 18 bits to fit within the 18x18 bit multiplier. In order to do so the \( \text{oneOver()} \cdot 2^n \) factor requires at most 10 bits since combined with the 8 bit difference it results in 18 bits. \( \text{oneOver()} \) only provides values between \( \frac{1}{255} \) to 1. Thus \( n \leq 10 \) for the product to be computed with that multiplier.

Summarising, a value \( n = 10 \) is both resource efficient, and keeps the round-off errors small. Furthermore, the visual quality of the resulting image is not noticeably affected, and with a fast video stream this is more important than a perfect fusion result. In figure 12 obvious quality loss can be observed between the \( n = 9 \) and \( n = 10 \) images. Stepping up the quantisation level from \( n = 10 \) to \( n = 11 \) however does not cause apparent artifacts.

Figure 12: The image fusion results for \( 2^9 \), \( 2^{10} \) and \( 2^{11} \) with magnification of a specific region.

5.2.3 Design

The FPGA design, as developed considering the precision and division problems just discussed, is shown in figure 13.
The numbers within the brackets represent the number of bits used for a particular register. The steps are separated with a line and a flip-flop for synchronisation.

With the assumption that the input images are perfectly aligned (an assumption made in the beginning of the report) every pixel in the input images should correspond to the same place in both images and thus in all four image inputs. Every corresponding pixel in these images are processed at the same time in one step in the design and then forwarded to the next like a stream. The lines will be referred to as pixel streams.

Following is a more detailed description of each step with illustrating code snippets. The full implementation can be found in the appendix.

Step 1
WLS is performed in the FPGA module (not implemented here) just before the composite module. In WLS both the VIS and the NIR input images are used to generate two coarsend images. These four image streams are the input for this first step in the composite module. In this step four offset parameters vis_coarse_brightness, vis_detail_brightness, nir_coarse_brightness and nir_detail_brightness are used to adjust the image brightness for the four images. This is necessary
for calibrating the spatial detail in intensity.

Here is a code snippet for one of the four parameter additions:

```verilog
always @(posedge clk) begin /*visCoarse pixel stream*/
    if(reset==1'b1)
        visCoarseAdjS2<=0;
    else begin
        if(vis_coarse_brightness>=0) begin
            if(visCoarse+vis_coarse_brightness <= 255) //Checking so that the upper limit isn't breached
                visCoarseAdjS2<=visCoarse+vis_coarse_brightness;
            else
                visCoarseAdjS2<=255;
        end
        else begin
            if(visCoarse>=-vis_coarse_brightness) //Checking so that the lower limit isn't breached
                visCoarseAdjS2<=visCoarse+vis_coarse_brightness;
            else
                visCoarseAdjS2<=0;
        end
    end
end
```

**Step 2**

In this step the difference between the pixel streams for the detail image and the coarse image is evaluated for both the VIS and the NIR image streams.

There is one detail that must be taken into consideration here. In the next step the difference from this step will be multiplied with a 10 bit value from a lookup table. If the difference in this step is more than 8 bits the result of the subsequent multiplication will be more than 18 bits and later in the pipeline, in step 7, instead of a 18x18 multiplier there will be a need of a 36x36 bit multiplier (see section Solving the Precision Problem). In practice the difference of two pixels in an image seldom gets higher than 127. To save resources a threshold has been implemented so that the difference only can assume values between -128 and 127. In this manner, all 8 bits are required, 7 for the difference in intensity and one for the sign.

```verilog
if(visDiff>127) begin
    visDiff<=127;
end
else if(visDiff<-128) begin
    visDiff<=-128;
end
```

The output from this step is the two difference streams as well as the coarse image stream for the VIS and NIR images. These are forwarded in parallel with the rest of the pipeline to the step 7 as can be seen in figure 13.

**Step 3**

In this step, two contrast streams are generated. The step actually has two operations but the first one is just an asynchronous fetch from a lookup table. No processing is needed for this operation and there is time to spare for another operation.

The lookup table is contained in the sub-module one_over. This sub-module also have an additional role. In order to avoid using extensive float representations yet keeping the precision in the upcoming calculations, every value in the lookup table have been pre-multiplied by $2^{10}$. The
reason for this is explained in detail in the section Solving the Precision Problem. In step 8 the precision is not longer critical and at that step the pixel stream will be divided by $2^{10}$.

```
module one_over(dividor, out_frac);
    input [7:0] dividor;
    output [9:0] out_frac;
    reg [9:0] out_frac;

    always @ (dividor) begin
        case(dividor)
            10'd0 : out_frac=10'd1023;
            10'd1 : out_frac=10'd1023;
            10'd2 : out_frac=10'd512;
            10'd3 : out_frac=10'd341;
            //...
        endcase
    end
```

The output from the sub-module one_over are 10 bit values, and these are multiplied with visDiff or nirDiff that each are represented by an 8 bit value. This operation demands an 18x18 bit multiplier and will for that reason provide an output of 36 bits. However since the highest values we can multiply are $2^8 \cdot 2^{10}$ the result can at most be $2^{18}$ which means that the output from these multiplications can be contained within 18 bits.

**Step 4**

The two contrast image streams of the previous step are adjusted so that no image is dominating the other in the upcoming calculations. This is done by means of weight parameters.

**Step 5**

The Schaul algorithm serves to select the parts of the VIS and NIR images that contain most details. The algorithm compares the VIS and NIR pixel stream with each other and forwards the pixels with a higher contrast value.

```
if(visContrastAdj>nirContrastAdj)
    MaxValue<=visContrastAdj;
else
    MaxValue<=nirContrastAdj;
end
```

**Step 6**

In this step another parameter controls the intensity of the composite contrast image. If the intensity is too low the details for the end result will not be as distinct as they could have been.

**Step 7**

The remaining pixel stream containing the details of the image fusion is in this step multiplied with the coarsend VIS pixel stream. The details will be carved into the coarsend image. The detail pixel stream is 18 bit while the VIS coarsend pixel stream is still 8 bit thus extended using an 18x18 bit multiplier. This step results in one single pixel stream of 36 bits.

```
composite<=MaxValueAdj*visCoarseS7;
```
Step 8

The 36 bit pixel stream is bit shifted to the right 10 times to perform a rough but effortless division. This is done to compensate for the pre-processed multiplication of $2^{10}$ in the one_over sub-module in step 3. Multiplying with $2^{10}$ was done to maintain precision and the operations made since have the limit of $2^{18} - 1$ instead of $2^{8} - 1$. Even though the container of the value at this point is 36 bit the limit is $2^{18} - 1$ and thus the resulting output after the bit shift of this step can be mapped to an 8 bit value.

Step 9

In this step a final parameter is used to adjust the image brightness of the end result. As in step 1 the pixel values are restricted to values in the interval 0 - 255.

5.3 Design Implementation

A major aspect of this work is that it is a subset of a potential full video image fusion solution. For this reason the HDR-60 camera development board was chosen for implementation. For the present study, however, no images will be aquired from the camera module. The Lattice Software Project Environment [36] by Lattice Semiconductor Corporation was included with the board and was written in Verilog so for simplicity the development is done in that language. The implementation has been made with consideration to, but otherwise independent of, the example code. The composite module and the testbench was designed solely by the author of this report.

With a detailed FPGA design the actual implementation is mostly a matter of syntax and testbench development. A testbench is a module that can be placed on top of the module-in-development to debug the code in a simulation. The testbench for this module reads an image from a file and pushes the pixels into the composite module, see figure 14. Following is a code snippet from the testbench.

```vhdl
while(!$feof(fileVD)) begin //Continue until the end of the file
    #2 //Wait one clock cycle (it takes time units for one clock cycle)
    $fscanf(fileVD, "%d", visDetail[7:0]); //Read pixel from VIS image
    $fscanf(fileVC, "%d", visCoarse[7:0]); //Read pixel from VIS coarse image
    $fscanf(fileND, "%d", nirDetail[7:0]); //Read pixel from NIR image
    $fscanf(fileNC, "%d", nirCoarse[7:0]); //Read pixel from NIR coarse image
end
```

The output of the module is passed from the testbench back to MATLAB where it is compared with the help of MS-SSIM to the composite part performed in MATLAB.
6 Results

6.1 Image Fusion Algorithm

The results provided in this part of the report are based upon image quality measurements of four properties of the fused images.

- Colour preservation
- Spatial resolution for details inherited by VIS
- Spatial resolution for details inherited by NIR
- Light source preservation

For this purpose the image quality evaluation algorithm MS-SSIM has been used.

**Colour preservation**

An important aim of the fusion algorithm is to produce an image that preserves the colours of the VIS image while adding the contrast information of the VIS and the NIR images. As is clearly seen in figure 15, vegetation is highlighted with both HSI and DWT image fusion.
Figure 15: Resulting images that illustrates the amount of colour preservation.

This can also be seen when using the image quality evaluation algorithm MS-SSIM. From the images in the set that is presented in the beginning of the report, the regions that contain vegetation have been selected for more specific analysis. Since vegetation has high reflection in NIR these corresponding regions in VIS and NIR are very dissimilar pixel-intensity-wise. The aspiration in this thesis is to get resulting images that are as alike the original VIS image as possible but still being enhanced with details from NIR. This means that the image fusion algorithm of final choice must not let the NIR high intensity regions affect the colours of the resulting images. A good measurement of the colour realism of the result can therefore be obtained by focusing on regions with vegetation. A part of the sample image 0001 was selected for this measurement, see figure 16.

Figure 16: With almost only vegetation in this sample image measurements will be focused on the amount of colour deviations.

It was processed with the three image fusion algorithms and then compared with the help of MS-SSIM with the original VIS image. The results can be found in table 3. A high value indicates better agreement.

Table 3: MS-SSIM performed on the three fusion algorithms for the sample image in figure 16.

<table>
<thead>
<tr>
<th></th>
<th>Schaul</th>
<th>DWT</th>
<th>HSI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Score</td>
<td>0.964</td>
<td>0.821</td>
<td>0.877</td>
</tr>
</tbody>
</table>

**Spatial resolution**

Considering spatial resolution, the algorithms perform differently for the NIR dominant part and the rest of the image. In the HSI algorithm the NIR dominant part is preserved with high spatial resolution while the rest of the image has low spatial quality. The DWT and the Schaul algorithms both perform well for both parts, see figure 17.
The algorithms need to be able to preserve details from both the VIS and NIR images. By performing an MS-SSIM test on a region with a lot of details from the VIS images and relevant details from the NIR images, measurements of the degree of spatial resolution preservation can be collected from that sample.

The performance of the image fusion algorithms differs for the spatial resolution inherited from the VIS images and that of the NIR images. For this reason two sample images have been selected, one with details inherited from VIS and one from NIR.

Figure 18: A region selected from the sample image 0051 where details are inherited from VIS.

The results of the MS-SSIM test for this sample image can be found in table 4.

Table 4: MS-SSIM performed on the three fusion algorithms for the sample image in figure 18.

<table>
<thead>
<tr>
<th></th>
<th>Schaul</th>
<th>DWT</th>
<th>HSI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schaul</td>
<td>0.929</td>
<td>0.826</td>
<td>0.783</td>
</tr>
</tbody>
</table>

Measurements where also taken with MS-SSIM for the fusion algorithms processing the sample region found in figure 19 and the results can be found in table 5.
Figure 19: A region selected from the sample image 0016 where details are inherited from NIR.

Table 5: MS-SSIM performed on the three fusion algorithms for the sample image in figure 19.

<table>
<thead>
<tr>
<th></th>
<th>Schaul</th>
<th>DWT</th>
<th>HSI</th>
</tr>
</thead>
<tbody>
<tr>
<td>MS-SSIM</td>
<td>0.977</td>
<td>0.981</td>
<td>0.993</td>
</tr>
</tbody>
</table>

**Light source preservation**

For traffic scenes, it is very important to preserve light source information, in particular when the sources appear non-existent in NIR. From the figure 20 it can be observed that all the image fusion algorithms succeed in preserving light sources.

Figure 20: Comparison of light sources in “0098”

By selecting regions with narrow-band light, objective measurements can be collected showing how much the processed images differ from the original image. In that way the algorithm with the best capability to preserve narrow-band light can be found.
Figure 21: A region with narrow-band light is selected from the sample image 0098.

The region in figure 21 has been processed with the three image fusion algorithms. The MS-SSIM results can be found in table 6.

Table 6: MS-SSIM performed on the three fusion algorithms for the sample image in figure 21.

<table>
<thead>
<tr>
<th></th>
<th>Schaul</th>
<th>DWT</th>
<th>HSI</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.886</td>
<td>0.660</td>
<td>0.845</td>
</tr>
</tbody>
</table>

6.1.1 Choice of algorithm

The results of the previous section are summarised in table 7. Overall, the Schaul algorithm performed best according to the MS-SSIM image quality evaluation algorithm. Only for the amount of inherited details from the NIR image was it beaten by the other two image fusion algorithms. This was expected, since the intensity part of the HSI algorithm is directly transferred from the NIR image. For the same reason the HSI algorithm performed worst regarding the inherited details from the VIS image.

Table 7: Results of MS-SSIM performed on the three fusion algorithms for the sample images in figures 16, 18, 19 and 21.

<table>
<thead>
<tr>
<th></th>
<th>Schaul</th>
<th>DWT</th>
<th>HSI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Colour preservation</td>
<td>0.964</td>
<td>0.821</td>
<td>0.877</td>
</tr>
<tr>
<td>Spatial resolution VIS</td>
<td>0.929</td>
<td>0.826</td>
<td>0.783</td>
</tr>
<tr>
<td>Spatial resolution NIR</td>
<td>0.977</td>
<td>0.981</td>
<td>0.993</td>
</tr>
<tr>
<td>Light source preservation</td>
<td>0.886</td>
<td>0.660</td>
<td>0.845</td>
</tr>
</tbody>
</table>

As a result of the analysis, the Schaul algorithm was chosen as the image fusion algorithm that is best suited for the applications focused on here.

6.1.2 Adaptation of the algorithm

The Schaul algorithm was adapted so that it would be more suitable for a FPGA implementation by reducing the number of layers used. Considering figure 10 it can be observed that the spatial resolution in the VIS image is somewhat reduced for Schaul using WLS with \( n = 1 \) level in comparison to Schaul with \( n = 6 \).

With the same technique as used in the previous section, MS-SSIM can be used to measure the cost of reducing the number of layers used. Using the region from the sample image 0051, see figure 18, the MS-SSIM algorithm compared Schaul running WLS for \( n = 6 \) and \( n = 1 \) levels with the VIS version of the sample region. This provides a measurement of the amount of details inherited from VIS. The results can be found in table 8.

Table 8: MS-SSIM measurements for Schaul using different number of layers for the sample region found in figure 18. The measurements show the spatial resolution reduction for details inherited from VIS.

<table>
<thead>
<tr>
<th>Schaul n=6</th>
<th>Schaul n=1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.972</td>
<td>0.934</td>
</tr>
</tbody>
</table>

30
In the same way the amount of details inherited from NIR can be measured by performing MS-SSIM on the region from the sample image 0016, see figure 19 for Schaul using WLS for \( n = 6 \) and \( n = 1 \) levels. The results can be found in table 9.

Table 9: MS-SSIM measurements for Schaul using different layers for the sample region found in figure 19. The measurements show the spatial resolution reduction for details inherited from NIR.

<table>
<thead>
<tr>
<th></th>
<th>Schaul ( n = 6 )</th>
<th>Schaul ( n = 1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Details</td>
<td>0.874</td>
<td>0.816</td>
</tr>
<tr>
<td>inherited</td>
<td></td>
<td></td>
</tr>
<tr>
<td>from NIR</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

From the measurement data it can be seen that reducing the amount of levels of WLS lessen somewhat the image quality for details inherited from both VIS and NIR. Introducing parameters weighting the VIS and NIR image contributions to the composite image however, enables more flexible tuning of the fused image, see figure 22. These parameters are discussed further in the section FPGA Design.

Figure 22: Adding parameters to the algorithm makes it more flexible. In this example the NIR contribution is enhanced.

The results of the comparisons performed with MS-SSIM made with Schaul with one level of WLS and Schaul with one level of WLS and with additional offset parameters can be found in table 10.

Table 10: MS-SSIM measurements for Schaul using one level of WLS for the sample region found in figures 18 and 19. The measurement show the spatial resolution difference for details inherited by VIS and NIR.

<table>
<thead>
<tr>
<th></th>
<th>Schaul ( n = 1 )</th>
<th>Schaul ( n = 1 ) with offset parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Details</td>
<td></td>
<td></td>
</tr>
<tr>
<td>inherited</td>
<td>0.934</td>
<td>0.929</td>
</tr>
<tr>
<td>from VIS</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.816</td>
<td>0.977</td>
</tr>
<tr>
<td>Details</td>
<td></td>
<td></td>
</tr>
<tr>
<td>inherited</td>
<td></td>
<td></td>
</tr>
<tr>
<td>from NIR</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For details inherited by NIR, Schaul with \( n = 1 \) level of WLS and with additional offset parameters performs better than Schaul with \( n = 6 \) levels of WLS without offset parameters. For details inherited by VIS, however, Schaul with \( n = 6 \) levels of WLS provides better results.

Compared to Schaul with \( 6 \) levels of WLS, Schaul with \( n = 1 \) and with offset parameters offers a faster algorithm that is more suitable for FPGA parallelism. All things considered, the latter algorithm therefore outweighs the \( n = 6 \) version.

6.2 Speedups

To determine the magnitude of the speedup the average processing time for an image in MATLAB will be compared with that for an image processed by the Verilog implementation.

The part of the fusion algorithm selected for acceleration, the image composite part, was processed 100 times for all five images presented in this report. The MATLAB script was run on an Acer
Aspire laptop equipped with an Intel i5 CPU running at 2.4 GHz and a 4 GB physical memory. The time measurement was performed with the built-in stopwatch functions TIC and TOC surrounding the selected part of the algorithm, see table 11. The MATLAB script was stripped of any code not necessary for the output result.

Table 11: The minimum, maximum and average time it took to process the composite part for one image pair in MATLAB. The measurement was performed 100 times for all six image-pairs presented in this report.

<table>
<thead>
<tr>
<th>Minimum value (s)</th>
<th>Average value (s)</th>
<th>Maximum value (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1069</td>
<td>0.1154</td>
<td>0.1381</td>
</tr>
</tbody>
</table>

The design implemented in Verilog was synthesised, mapped, placed and routed alone without any premade surrounding modules. The software used, Lattice Diamond, calculated the maximum frequency for the module on this type of FPGA board to 129.955 MHz and mapped it to a clock running at 109.938 MHz. Since the design is pipelined and every step in the pipeline takes one clock cycle the data rate will be the same as the clock frequency. Since the pipeline has 9 stages, it generates the output pixels 9 clock cycles after receiving the input pixels. The images have the resolution 800x533 corresponding to 426400 pixels. To fill the pipeline and process one full image pair 426409 clock cycles are required. With the mapped clock one image pair will be processed by the composite module in:

\[
\frac{426409}{109.938 \cdot 10^6} \approx 3.9 \cdot 10^{-3} \text{ seconds}
\]

For one image pair the part of the algorithm selected for FPGA implementation will have a computation time reduced by a factor of 30 on the HDR-60 as compared to the MATLAB script.

6.3 Verification of design

In the verification of the design it is necessary to take into account that some errors are bound to occur due to the round-off and precision issues described in section FPGA Design.

For the verification the sample image 0001 was used. The VIS and the NIR images and their coarse counterparts where exported from MATLAB to a format that the FPGA test bench could handle. The test bench put these images pixel by pixel through the FPGA design and saved the results. The results was imported into MATLAB and the MS-SSIM algorithm was used to compare it with the MATLAB version of the composite part. MS-SSIM provided the result 0.9995 and the maximum difference was the value 2.

7 Discussion

7.1 He

Much interest has recently been directed towards removing haze in a single image, and a dehazing algorithm proposed by He et al. [4] has received much attention. Ambient light reflected into the line of sight by atmospheric particles, referred to as airlight, can be used to attain a depth map from a single image. With this depth map He et al. recover an image with less haze. Even though this algorithm works well for most hazy images it has some limitations. Objects similar to airlight will disturb the process and the algorithm is not fit for haze-free or indoor environments [4].

If applied after image fusion with Schaul, depending on what scene it is applied to, the result can be better with more details and less haze, see figure 23. A problem is that with the He et al.’s algorithm the resulting image become unrealistically dramatic with amplified clouds, see figure 24. In the applications intended for this project, emphasizing the background and clouds could be disturbing and taking the focus off more important aspects of the fused image.
7.2 Night vision

Schaul extracts the contrast details from the pair of images and use it on the VIS image. This causes a problem for images taken in darkness. The NIR image may be brighter and contain more contrast than the VIS image but if the VIS image is dark, that contrast information will be merged with that darkness. The contrast information will in this case hardly improve the VIS image at all. A possible solution to this problem is to brighten the VIS image before the algorithm is applied. Another solution can be to transfer the brightness from the NIR image to the fused image as well as the contrast information, even if this causes highlighted vegetation in the fused image. In any case, the techniques developed in this project might need some modifications in case of images collected under poor light conditions.

7.3 Bilateral filter for decomposition

Schaul fusion makes use of the WLS decomposition algorithm which according to Schaul et al. [5] and Farbman et al. [16] is more capable of preserving edges than other decomposition methods. The WLS decomposition is however a novel algorithm and is not as widely used as for instance the bilateral filter. Gabiger et al. [20] has published an FPGA design for the original bilateral filter introduced by Tomasi et al. [13]. Replacing the WLS decomposition in Schaul with the bilateral filter results in fused images with more artifacts along edges but would offer an algorithm were all the designs needed for a full solution are available, saving many hours in design development time.
7.4 Modified DWT

Many DWT image fusion algorithms have been proposed and in this project only a few implementations have been tested. A more extensive search for a more suitable DWT algorithm may result in an image fusion with higher quality. It seems however unlikely that it would outperform Schaul.

The DWT algorithm that was presented in this report can be finetuned. For instance, it may give better results if a contrast boost and colour enhancement is applied.

7.5 Future work

This report suggests an image fusion algorithm to be used for VIS and NIR images in vehicles. A full design and an implementation in HDL code has been made and simulated for a part of the algorithm. Yet to be designed and implemented is the colour model transformation and the WLS decomposition. In order to achieve a full solution with two video streams, one VIS and one NIR camera image stream should be fed to an FPGA containing the fully implemented algorithm. The output should be a real time image fusion of streamed images from the two cameras.

8 Conclusion

This report deals with the problem of how to design an image vision system that reduces the impact of haze and fog, while preserving colour information. In addition, the solution should be fast enough to be applicable for a real-time stream of images, like the one obtained from cameras in a vehicle. The suggested solution is based on the fusion of visual and near-infrared images, processed using an FPGA device. Combining visible and near-infrared images of the same scene, it is indeed possible to reduce the impact of haze and fog. Simpler colour model component substitution algorithms like HSI fusion are also very fast but there exist more complex multiresolution algorithms that are far superior when it comes to image quality. A comparison of two multiresolution algorithms, DWT and Schaul, was performed and it was shown that the Schaul algorithm, whose initial purpose was precisely to fuse a visible and a near-infrared image, performed slightly better in colour diversity.

FPGAs provide a fast and highly configurable platform for acceleration of image fusion algorithms. The full Schaul algorithm has three parts, a colour transformation part, an image decomposition part and an image composition part, and in this report the FPGA implementation of the last part is studied in detail. Two challenges arose in this context. An FPGA does not have a simple way of performing division. The problem was solved by using a lookup table and this proved to be a fast and simple solution. Maintaining precision under division introduced a second challenge. Instead of implementing a float representation, precision was achieved by multiplying the input value by $2^{10}$, thereby converting decimal numbers to integers, and dividing the result with the same value. This method proved to be more resource efficient.

The implementation of the composite part was simulated and the result showed that one image could be processed within 4 milliseconds. Weighted least squares (WLS), the image decomposition method used by Schaul, has never been implemented on an FPGA but a similar and simpler decomposition method, the bilateral filter, has previously been successfully implemented, generating fused images at a rate of 30 frames per second. It is the author's opinion that the WLS filter also can be implemented with a similar frame rate.

9 Acknowledgments

I have had the privilege to be able to do my thesis project at the company Cybercom and I am grateful for the support that I have received. Especially my tutor Karl Lundén has been of great support and a valuable source of knowledge in the development of the FPGA design and implementation.

The theme of the thesis was first proposed by my former advisor Graham McCarthy who also has been of great support in my work.
My advisor at Mälardalens Högskola, Fredrik Elstrand, has been of great help structuring the thesis project and ensuring the quality of the report.

I would also like to direct a note of gratitude to the Image and Visual Representation Group at the École Polytechnique Fédérale de Lausanne that has made available the set of visible and near-infrared images which have allowed me to focus on the actual task of image fusion rather than the acquisition of image-pairs.

Thanks to the support and encouragement from Emelie Enström, my girlfriend, and Bengt Karlsson, my father, I can now gladly say that I am proud of what I have accomplished.

References


A Appendix

A.1 Composite part of Schaul implementation in verilog

/*
This module implements the last part of the Schaul algorithm in a pipelined parallel manner.
In stage 1 the design adds constant values to the pixels. In the stages 2-4 the two contrast pixels are calculated according to: contrast=(detail-coarse)/coarse+const. Next the composite map is calculated by choosing the maximum contrast pixels and adjusting them by a constant. This is done in stage 5-6.
Next step is to merge the composite map with the coarse vis map so that the resulting image is a black and white version of the fused image. This is done in stage 7-8.

All steps take one clock cycle, thus the module takes 8 clock cycles to complete one calculation.
Based on the work of Schaul et al. Schaul implementation in matlab by Petykiewicz-Buckley and Gabiger et al.

Articles and links:
http://vistalab.stanford.edu/teach/index.php/PetykiewiczBuckley
*/

module Composite(
    visDetail,
    visCoarse,
    nirDetail,
    nirCoarse,
    compositeOut,
    clk,
    reset
);

input clk;
input reset;
input [7:0] visDetail; //One 8 bit pixel of the detail image of the image taken in visible light.
input [7:0] visCoarse; //One 8 bit pixel of the coarse image of the image taken in visible light.
input [7:0] nirDetail; //One 8 bit pixel of the detail image of the image taken in near-infrared light.
input [7:0] nirCoarse; //One 8 bit pixel of the coarse image of the image taken in near-infrared light.
output [7:0] compositeOut;

//Stage 1 : Adjusting initial values
reg [7:0] visCoarseAdjS2;
reg [7:0] visDetailAdjS2;
reg [7:0] nirCoarseAdjS2;
reg [7:0] nirDetailAdjS2;

//Constants
reg [7:0] vis_coarse_brightness;
reg [7:0] vis_detail_brightness;
reg [7:0] nir_coarse_brightness;
reg [7:0] nir_detail_brightness;
reg [17:0] vis_contrast_intensity;
reg [17:0] nir_contrast_intensity;
reg [17:0] composite_contrast_intensity;
reg [7:0] composite_brightness;

initial begin
    vis_coarse_brightness=17;
    vis_detail_brightness=-17;
    nir_coarse_brightness=10;
    nir_detail_brightness=-10;
    vis_contrast_intensity=-102;
    nir_contrast_intensity=20;
    composite_contrast_intensity=819;
    composite_brightness=10;
end

always @(posedge clk) begin //visCoarse pixel stream
    if(reset==1'b1)
        visCoarseAdjS2<=0;
    else begin
        if(vis_coarse_brightness>=0) begin
            if(visCoarse+vis_coarse_brightness <= 255) //Checking so that the upper limit isn't breached
                visCoarseAdjS2<=visCoarse+vis_coarse_brightness;
            else
                visCoarseAdjS2<=255;
        end
        else begin
            if(visCoarse>=-vis_coarse_brightness) //Checking so that the lower limit isn't breached
                visCoarseAdjS2<=visCoarse+vis_coarse_brightness;
            else
                visCoarseAdjS2<=0;
        end
    end
end

always @(posedge clk) begin //visDetail pixel stream
    if(reset==1'b1)
        visDetailAdjS2<=0;
    else begin
        if(visDetail+vis_detail_brightness <= 255) //Checking so that the upper limit isn't breached
            visDetailAdjS2<=visDetail+vis_detail_brightness;
        else
            visDetailAdjS2<=255;
        end
        else begin
            if(visDetail>=-vis_detail_brightness) //Checking so that the lower limit isn't breached
                visDetailAdjS2<=visDetail+vis_detail_brightness;
            else
                visDetailAdjS2<=0;
        end
    end
end
visDetailAdjS2<=0;
else begin
  if(vis_detail_brightness>=0) begin
    if(visDetail+vis_detail_brightness <= 255) //Checking so that the upper
      limit isn't breached
      visDetailAdjS2<=visDetail+vis_detail_brightness;
    else
      visDetailAdjS2<=255;
  end
  else begin
    if(visDetail>=-vis_detail_brightness) //Checking so that the lower limit
      isn't breached
      visDetailAdjS2<=visDetail+vis_detail_brightness;
    else
      visDetailAdjS2<=0;
  end
end
end

always @ (posedge clk) begin //nirCoarse pixel stream
  if(reset==1'b1)
    nirCoarseAdjS2<=0;
  else begin
    if(nir_coarse_brightness>=0) begin
      if(nirCoarse+nir_coarse_brightness <= 255) //Checking so that the upper
        limit isn't breached
        nirCoarseAdjS2<=nirCoarse+nir_coarse_brightness;
      else
        nirCoarseAdjS2<=255;
    end
    else begin
      if(nirCoarse>=-nir_coarse_brightness) //Checking so that the lower limit
        isn't breached
        nirCoarseAdjS2<=nirCoarse+nir_coarse_brightness;
      else
        nirCoarseAdjS2<=0;
    end
  end
end

always @ (posedge clk) begin //nirDetail pixel stream
  if(reset==1'b1)
    nirDetailAdjS2<=0;
  else begin
    if(nir_detail_brightness>=0) begin
      if(nirDetail+nir_detail_brightness <= 255) //Checking so that the upper
        limit isn't breached
        nirDetailAdjS2<=nirDetail+nir_detail_brightness;
      else
        nirDetailAdjS2<=255;
    end
    else begin
      if(nirDetail>=-nir_detail_brightness) //Checking so that the lower limit
        isn't breached
        nirDetailAdjS2<=nirDetail+nir_detail_brightness;
      else
        nirDetailAdjS2<=0;
    end
  end
end

39
end

//Stage 2 : Calculating the difference between detail and coarse
reg signed [7:0] visDiff; //visDiff and nirDiff is defined with 18 bits since the
multipliers later on can handle 9x9 bits, 18x18 bits or 36x36 bits. Its is not
necessary in the calculations in step 1 and 2, but in order to perform
reg signed [7:0] nirDiff; // the product in step 3 at least one of the multiplier or the
multiplicand needs to be 18 bits so that the result can become 18 bit.
// visDiff and nirDiff will in reality never exceed the limits -128
too 127. If they would the 36x36 bits multiplier would need to
used not once for every pixel, but twice. In that case
2^9*2^10=2^19 which is too much for the 18x18 multiplier.

reg [7:0] visCoarseS3;
reg [7:0] nirCoarseS3;

always @ (posedge clk) begin
  if(reset==1'b1)
    visDiff<=0;
  else begin
    visDiff<=visDetailAdjS2-visCoarseAdjS2;
    if(visDiff>127) begin
      visDiff<=127;
    end else if(visDiff<-128) begin
      visDiff<=-128;
    end
  end
end

always @ (posedge clk) begin
  if(reset==1'b1)
    nirDiff<=0;
  else begin
    nirDiff<=nirDetailAdjS2-nirCoarseAdjS2;
    if(nirDiff>127) begin
      nirDiff<=127;
    end else if(nirDiff<-128) begin
      nirDiff<=-128;
    end
  end
end

always @ (posedge clk) begin
  if(reset==1'b1)
    visCoarseS3<=0;
  else
    visCoarseS3<=visCoarseAdjS2;
end

always @ (posedge clk) begin
  if(reset==1'b1)
    nirCoarseS3<=0;
  else
    nirCoarseS3<=nirCoarseAdjS2;
end

//Stage 3 : Performing division of the differences and the coarse images
reg signed [17:0] visContrast; //visDiff and nirDiff are 8 bits, vis_res_frac and
nir_res_frac are 10 bits. Multiplied they need 18 bits.
reg signed [17:0] nirContrast;
reg [7:0] visCoarseS4;
wire [9:0] vis_res_frac;
wire [9:0] nir_res_frac;

//One_over is a lookup-table module that asynchronously returns one over the 8 bit input
data multiplied by 2^10 for precision.
one_over one_overVis(visCoarseS3, vis_res_frac);
one_over one_overNir(nirCoarseS3, nir_res_frac);

always @(posedge clk) begin
  if(reset==1'b1)
    visContrast<=0;
  else begin
    if(visDiff<0)
      visContrast<=(~{16'h0000, ~visDiff}+1)*(vis_res_frac); //Filling visDiff to be
      //represented as a 18 bit value.
    else begin
      visContrast<=visDiff*(vis_res_frac);
    end
  end
end

always @(posedge clk) begin
  if(reset==1'b1)
    nirContrast<=0;
  else begin
    if(nirDiff<0)
      nirContrast<=visDiff*(nir_res_frac);
    else begin
      nirContrast<=visDiff*(nir_res_frac);
    end
  end
end

always @(posedge clk) begin
  if(reset==1'b1)
    visCoarseS4<=0;
  else
    visCoarseS4<=visCoarseS3;
end

//Stage 4 : Adjusting the contrast pixels
reg signed [17:0] visContrastAdj;
reg signed [17:0] nirContrastAdj;
reg [7:0] visCoarseS5;

always @(posedge clk) begin
  if(reset==1'b1)
    visContrastAdj<=0;
  else
    visContrastAdj<=visContrast+vis_contrast_intensity;
end

always @(posedge clk) begin
  if(reset==1'b1)
    nirContrastAdj<=0;
  else
    nirContrastAdj<=nirContrast+nir_contrast_intensity;
end
always @ (posedge clk) begin
    if(reset==1'b1)
        visCoarseS5<=0;
    else
        visCoarseS5<=visCoarseS4;
end

//Stage 5 : Choosing the maximum contrast pixel
reg signed [17:0] MaxValue;
reg [7:0] visCoarseS6;
always @ (posedge clk) begin
    if(reset==1'b1)
        MaxValue<=0;
    else begin
        if(visContrastAdj>nirContrastAdj)
            MaxValue<=visContrastAdj;
        else
            MaxValue<=nirContrastAdj;
    end
end
always @ (posedge clk) begin
    if(reset==1'b1)
        visCoarseS6<=0;
    else
        visCoarseS6<=visCoarseS5;
end

//Stage 6 : Adjusting the maximum contrast pixel
reg signed [17:0] MaxValueAdj;
reg [7:0] visCoarseS7;
always @ (posedge clk) begin
    if(reset==1'b1)
        MaxValueAdj<=0;
    else begin
        MaxValueAdj<=MaxValue+composite_contrast_intensity;
    end
end
always @ (posedge clk) begin
    if(reset==1'b1)
        visCoarseS7<=0;
    else
        visCoarseS7<=visCoarseS6;
end

//Stage 7 : Merging the contrast pixels with the visual coarse image.
reg signed [35:0] composite;
always @(posedge clk) begin
    if(reset==1'b1)
        composite<=0;
    else
        composite<=MaxValueAdj*visCoarseS7;
end

//Stage 8 : Bitshifting right by 10.
// Since the value returned from the module one_over was multiplied by 2^10, the
// composite value needs to be bitshifted 10 times to be returned to a 8 bit value.
reg [7:0] composite8bit;

always @(posedge clk) begin
  if(reset==1'b1)
    composite8bit<=0;
  else begin
    if(composite<262144) //If composite is more than 2^18 it cannot be bitshifted
      and then mapped to a 8 bit value
      composite8bit<=composite>>10;
    else
      composite8bit<=255;
  end
end

//Stage 9 : Adjusting brightness of pixel. At the moment I don’t need to adjust the
brightness but I might need to do that later on.
reg [7:0] compositeAdj;

always @(posedge clk) begin
  if(reset==1'b1)
    compositeAdj<=0;
  else begin
    if(composite_brightness>=0) begin
      if(composite8bit+composite_brightness <= 255) //Checking so that the upper
        limit isn’t breached
        compositeAdj<=composite8bit+composite_brightness;
      else
        compositeAdj<=255;
    end
    else begin
      if(composite8bit>=-composite_brightness) //Checking so that the lower limit
        isn’t breached
        compositeAdj<=composite8bit+composite_brightness;
      else
        compositeAdj<=0;
    end
  end
end

//Assign output value
assign compositeOut=compositeAdj;
endmodule

/*
This is an asynchronous module that upon an 8 bit input returns (1/input)*2^10.
*/
module one_over(dividor, out_frac);
  input [7:0] dividor;
  output [9:0] out_frac;
  reg [9:0] out_frac;

always @(dividor) begin
  case(dividor)
    10’d0 : out_frac=10’d1023;
    10’d1 : out_frac=10’d1023;
    10’d2 : out_frac=10’d512;
    10’d3 : out_frac=10’d341;
    10’d4 : out_frac=10’d256;
    10’d5 : out_frac=10’d205;
    10’d6 : out_frac=10’d171;
  endcase
end
10'd7 : out_frac=10'd146;
10'd8 : out_frac=10'd128;
10'd9 : out_frac=10'd114;
10'd10 : out_frac=10'd102;
10'd11 : out_frac=10'd93;
10'd12 : out_frac=10'd85;
10'd13 : out_frac=10'd79;
10'd14 : out_frac=10'd73;
10'd15 : out_frac=10'd68;
10'd16 : out_frac=10'd64;
10'd17 : out_frac=10'd60;
10'd18 : out_frac=10'd57;
10'd19 : out_frac=10'd54;
10'd20 : out_frac=10'd51;
10'd21 : out_frac=10'd49;
10'd22 : out_frac=10'd47;
10'd23 : out_frac=10'd45;
10'd24 : out_frac=10'd43;
10'd25 : out_frac=10'd41;
10'd26 : out_frac=10'd39;
10'd27 : out_frac=10'd38;
10'd28 : out_frac=10'd37;
10'd29 : out_frac=10'd35;
10'd30 : out_frac=10'd34;
10'd31 : out_frac=10'd33;
10'd32 : out_frac=10'd32;
10'd33 : out_frac=10'd31;
10'd34 : out_frac=10'd30;
10'd35 : out_frac=10'd29;
10'd36 : out_frac=10'd28;
10'd37 : out_frac=10'd28;
10'd38 : out_frac=10'd27;
10'd39 : out_frac=10'd26;
10'd40 : out_frac=10'd26;
10'd41 : out_frac=10'd25;
10'd42 : out_frac=10'd24;
10'd43 : out_frac=10'd24;
10'd44 : out_frac=10'd23;
10'd45 : out_frac=10'd23;
10'd46 : out_frac=10'd22;
10'd47 : out_frac=10'd22;
10'd48 : out_frac=10'd21;
10'd49 : out_frac=10'd21;
10'd50 : out_frac=10'd20;
10'd51 : out_frac=10'd20;
10'd52 : out_frac=10'd20;
10'd53 : out_frac=10'd19;
10'd54 : out_frac=10'd19;
10'd55 : out_frac=10'd19;
10'd56 : out_frac=10'd18;
10'd57 : out_frac=10'd18;
10'd58 : out_frac=10'd18;
10'd59 : out_frac=10'd17;
10'd60 : out_frac=10'd17;
10'd61 : out_frac=10'd17;
10'd62 : out_frac=10'd17;
10'd63 : out_frac=10'd16;
10'd64 : out_frac=10'd16;
10'd65 : out_frac=10'd16;
<table>
<thead>
<tr>
<th>Decimal</th>
<th>Fractional Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>10'd66</td>
<td>out_frac=10'd16</td>
</tr>
<tr>
<td>10'd67</td>
<td>out_frac=10'd15</td>
</tr>
<tr>
<td>10'd68</td>
<td>out_frac=10'd15</td>
</tr>
<tr>
<td>10'd69</td>
<td>out_frac=10'd15</td>
</tr>
<tr>
<td>10'd70</td>
<td>out_frac=10'd15</td>
</tr>
<tr>
<td>10'd71</td>
<td>out_frac=10'd14</td>
</tr>
<tr>
<td>10'd72</td>
<td>out_frac=10'd14</td>
</tr>
<tr>
<td>10'd73</td>
<td>out_frac=10'd14</td>
</tr>
<tr>
<td>10'd74</td>
<td>out_frac=10'd14</td>
</tr>
<tr>
<td>10'd75</td>
<td>out_frac=10'd13</td>
</tr>
<tr>
<td>10'd76</td>
<td>out_frac=10'd13</td>
</tr>
<tr>
<td>10'd77</td>
<td>out_frac=10'd13</td>
</tr>
<tr>
<td>10'd78</td>
<td>out_frac=10'd13</td>
</tr>
<tr>
<td>10'd79</td>
<td>out_frac=10'd13</td>
</tr>
<tr>
<td>10'd80</td>
<td>out_frac=10'd13</td>
</tr>
<tr>
<td>10'd81</td>
<td>out_frac=10'd13</td>
</tr>
<tr>
<td>10'd82</td>
<td>out_frac=10'd12</td>
</tr>
<tr>
<td>10'd83</td>
<td>out_frac=10'd12</td>
</tr>
<tr>
<td>10'd84</td>
<td>out_frac=10'd12</td>
</tr>
<tr>
<td>10'd85</td>
<td>out_frac=10'd12</td>
</tr>
<tr>
<td>10'd86</td>
<td>out_frac=10'd12</td>
</tr>
<tr>
<td>10'd87</td>
<td>out_frac=10'd12</td>
</tr>
<tr>
<td>10'd88</td>
<td>out_frac=10'd12</td>
</tr>
<tr>
<td>10'd89</td>
<td>out_frac=10'd12</td>
</tr>
<tr>
<td>10'd90</td>
<td>out_frac=10'd11</td>
</tr>
<tr>
<td>10'd91</td>
<td>out_frac=10'd11</td>
</tr>
<tr>
<td>10'd92</td>
<td>out_frac=10'd11</td>
</tr>
<tr>
<td>10'd93</td>
<td>out_frac=10'd11</td>
</tr>
<tr>
<td>10'd94</td>
<td>out_frac=10'd11</td>
</tr>
<tr>
<td>10'd95</td>
<td>out_frac=10'd11</td>
</tr>
<tr>
<td>10'd96</td>
<td>out_frac=10'd11</td>
</tr>
<tr>
<td>10'd97</td>
<td>out_frac=10'd11</td>
</tr>
<tr>
<td>10'd98</td>
<td>out_frac=10'd10</td>
</tr>
<tr>
<td>10'd99</td>
<td>out_frac=10'd10</td>
</tr>
<tr>
<td>10'd100</td>
<td>out_frac=10'd10</td>
</tr>
<tr>
<td>10'd101</td>
<td>out_frac=10'd10</td>
</tr>
<tr>
<td>10'd102</td>
<td>out_frac=10'd10</td>
</tr>
<tr>
<td>10'd103</td>
<td>out_frac=10'd10</td>
</tr>
<tr>
<td>10'd104</td>
<td>out_frac=10'd10</td>
</tr>
<tr>
<td>10'd105</td>
<td>out_frac=10'd10</td>
</tr>
<tr>
<td>10'd106</td>
<td>out_frac=10'd10</td>
</tr>
<tr>
<td>10'd107</td>
<td>out_frac=10'd10</td>
</tr>
<tr>
<td>10'd108</td>
<td>out_frac=10'd9</td>
</tr>
<tr>
<td>10'd109</td>
<td>out_frac=10'd9</td>
</tr>
<tr>
<td>10'd110</td>
<td>out_frac=10'd9</td>
</tr>
<tr>
<td>10'd111</td>
<td>out_frac=10'd9</td>
</tr>
<tr>
<td>10'd112</td>
<td>out_frac=10'd9</td>
</tr>
<tr>
<td>10'd113</td>
<td>out_frac=10'd9</td>
</tr>
<tr>
<td>10'd114</td>
<td>out_frac=10'd9</td>
</tr>
<tr>
<td>10'd115</td>
<td>out_frac=10'd9</td>
</tr>
<tr>
<td>10'd116</td>
<td>out_frac=10'd9</td>
</tr>
<tr>
<td>10'd117</td>
<td>out_frac=10'd9</td>
</tr>
<tr>
<td>10'd118</td>
<td>out_frac=10'd9</td>
</tr>
<tr>
<td>10'd119</td>
<td>out_frac=10'd9</td>
</tr>
<tr>
<td>10'd120</td>
<td>out_frac=10'd9</td>
</tr>
<tr>
<td>10'd121</td>
<td>out_frac=10'd8</td>
</tr>
<tr>
<td>10'd122</td>
<td>out_frac=10'd8</td>
</tr>
<tr>
<td>10'd123</td>
<td>out_frac=10'd8</td>
</tr>
<tr>
<td>10'd124</td>
<td>out_frac=10'd8</td>
</tr>
</tbody>
</table>
10'd125 : out_frac=10'd8;
10'd126 : out_frac=10'd8;
10'd127 : out_frac=10'd8;
10'd128 : out_frac=10'd8;
10'd129 : out_frac=10'd8;
10'd130 : out_frac=10'd8;
10'd131 : out_frac=10'd8;
10'd132 : out_frac=10'd8;
10'd133 : out_frac=10'd8;
10'd134 : out_frac=10'd8;
10'd135 : out_frac=10'd8;
10'd136 : out_frac=10'd8;
10'd137 : out_frac=10'd7;
10'd138 : out_frac=10'd7;
10'd139 : out_frac=10'd7;
10'd140 : out_frac=10'd7;
10'd141 : out_frac=10'd7;
10'd142 : out_frac=10'd7;
10'd143 : out_frac=10'd7;
10'd144 : out_frac=10'd7;
10'd145 : out_frac=10'd7;
10'd146 : out_frac=10'd7;
10'd147 : out_frac=10'd7;
10'd148 : out_frac=10'd7;
10'd149 : out_frac=10'd7;
10'd150 : out_frac=10'd7;
10'd151 : out_frac=10'd7;
10'd152 : out_frac=10'd7;
10'd153 : out_frac=10'd7;
10'd154 : out_frac=10'd7;
10'd155 : out_frac=10'd7;
10'd156 : out_frac=10'd7;
10'd157 : out_frac=10'd7;
10'd158 : out_frac=10'd6;
10'd159 : out_frac=10'd6;
10'd160 : out_frac=10'd6;
10'd161 : out_frac=10'd6;
10'd162 : out_frac=10'd6;
10'd163 : out_frac=10'd6;
10'd164 : out_frac=10'd6;
10'd165 : out_frac=10'd6;
10'd166 : out_frac=10'd6;
10'd167 : out_frac=10'd6;
10'd168 : out_frac=10'd6;
10'd169 : out_frac=10'd6;
10'd170 : out_frac=10'd6;
10'd171 : out_frac=10'd6;
10'd172 : out_frac=10'd6;
10'd173 : out_frac=10'd6;
10'd174 : out_frac=10'd6;
10'd175 : out_frac=10'd6;
10'd176 : out_frac=10'd6;
10'd177 : out_frac=10'd6;
10'd178 : out_frac=10'd6;
10'd179 : out_frac=10'd6;
10'd180 : out_frac=10'd6;
10'd181 : out_frac=10'd6;
10'd182 : out_frac=10'd6;
10'd183 : out_frac=10'd6;
10'd184 : out_frac=10'd6;
10'd185 : out_frac=10'd6;
10'd186 : out_frac=10'd6;
10'd187 : out_frac=10'd5;
10'd188 : out_frac=10'd5;
10'd189 : out_frac=10'd5;
10'd190 : out_frac=10'd5;
10'd191 : out_frac=10'd5;
10'd192 : out_frac=10'd5;
10'd193 : out_frac=10'd5;
10'd194 : out_frac=10'd5;
10'd195 : out_frac=10'd5;
10'd196 : out_frac=10'd5;
10'd197 : out_frac=10'd5;
10'd198 : out_frac=10'd5;
10'd199 : out_frac=10'd5;
10'd200 : out_frac=10'd5;
10'd201 : out_frac=10'd5;
10'd202 : out_frac=10'd5;
10'd203 : out_frac=10'd5;
10'd204 : out_frac=10'd5;
10'd205 : out_frac=10'd5;
10'd206 : out_frac=10'd5;
10'd207 : out_frac=10'd5;
10'd208 : out_frac=10'd5;
10'd209 : out_frac=10'd5;
10'd210 : out_frac=10'd5;
10'd211 : out_frac=10'd5;
10'd212 : out_frac=10'd5;
10'd213 : out_frac=10'd5;
10'd214 : out_frac=10'd5;
10'd215 : out_frac=10'd5;
10'd216 : out_frac=10'd5;
10'd217 : out_frac=10'd5;
10'd218 : out_frac=10'd5;
10'd219 : out_frac=10'd5;
10'd220 : out_frac=10'd5;
10'd221 : out_frac=10'd5;
10'd222 : out_frac=10'd5;
10'd223 : out_frac=10'd5;
10'd224 : out_frac=10'd5;
10'd225 : out_frac=10'd5;
10'd226 : out_frac=10'd5;
10'd227 : out_frac=10'd5;
10'd228 : out_frac=10'd4;
10'd229 : out_frac=10'd4;
10'd230 : out_frac=10'd4;
10'd231 : out_frac=10'd4;
10'd232 : out_frac=10'd4;
10'd233 : out_frac=10'd4;
10'd234 : out_frac=10'd4;
10'd235 : out_frac=10'd4;
10'd236 : out_frac=10'd4;
10'd237 : out_frac=10'd4;
10'd238 : out_frac=10'd4;
10'd239 : out_frac=10'd4;
10'd240 : out_frac=10'd4;
10'd241 : out_frac=10'd4;
10'd242 : out_frac=10'd4;
A.2 Lattice HDR-60 Product Brochure
HDR-60 Video Camera Development Kit

Real-time HD Video Camera with Fully Integrated HDR Image Signal Processing

The HDR-60 Video Camera Development Kit is a production-ready High Definition (HD) video camera development system based on the LatticeECP3™ FPGA family. Pre-loaded with a plug-and-play evaluation Image Signal Processing (ISP) pipeline based on Intellectual Property (IP) cores from Lattice partner Helion GmbH, the kit works right out of the box. The IP is capable of delivering 1080p performance at 60 frames per second with 2D noise reduction and High Dynamic Range (HDR).

Designed to fit into commercially available camera housings and capable of supporting two sensors simultaneously, the kit enables rapid evaluation and prototyping of high-definition HDR video cameras for security and surveillance, traffic control, video conferencing and automotive applications. Schematics and layout files are available for download to purchasers of the kit.

The HDR-60 Video Camera Development Kit has been designed using a LatticeECP3-70 FPGA; however, the ISP IP pipeline needed to implement a complete 1080p60 HDR camera requires only a 33K LUT LatticeECP3-35 device. The available space offers designers plenty of room for experimentation and adding custom IP.

The kit offers camera manufacturers several unique benefits, including a fully integrated HDR image signal processing pipeline from sensor to HDMI/DVI display. Equipped with an Aptina 720p HDR sensor, the kit also offers the industry’s fastest auto-exposure, greater than 120dB system dynamic range, a highly-efficient auto white balance algorithm and 2D noise reduction – all in streaming mode through the FPGA without the need for an external frame buffer, enabling extremely low latency and further reducing system cost. On-board DDR2 memory also enables applications such as 3D noise reduction, image stitching from multiple sensors, image rotation and de-warping.

Key Features and Benefits
- FPGA-based Image Signal Processing
- Fully Production-ready HDR Camera Design
- IP is Capable of 1080p at 60fps
- IP Supports up to 16 Megapixel Sensors
- Full 60fps in Streaming Mode Needs No External Frame Buffer
- Fast Auto Exposure Quickly Adjusts to Changing Light
- Greater than 120dB High Dynamic Range (HDR) Performance
- Extremely Low Latency
- Comprehensive Image Processing IP Library
- On-board Broadcom® Broadreach™ PHY Enables Ethernet over Coaxial Cable for Distances up to 700m
- On-board FTDI Device provides Easy Programming via Standard USB Cable

Ordering Information

<table>
<thead>
<tr>
<th>Product</th>
<th>Description</th>
<th>Ordering Part Number</th>
</tr>
</thead>
</table>
| HDR-60 Video Camera Development Kit   | • HDR-60 Base Board with LatticeECP3 FPGA, pre-loaded with Image Signal Processing (ISP) Demo  
• NanoVesta Head Board with Aptina A-1000 720p HDR Sensor and Sunex lens  
• Two USB cables  
• HDMI cable with HDMI-to-DVI adapter  
• 12V AC adapter power supply  
• QuickSTART Guide | LFE3-70EA-HDR60-DKN      |
Image Signal Processing Demo

The pre-loaded Image Signal Processing pipeline demo is plug-and-play, and works directly with a compatible HDMI/DVI monitor. Simple instructions are provided to demonstrate Fast Auto Exposure, Auto White Balance and High Dynamic Range for evaluation of the pipeline’s capabilities.

HDR ISP Pipeline Reference Design

The pre-loaded demo bitstream contains a fully functional HDR ISP Pipeline with the following IP blocks:

- Sensor Port
- Linearization Block
- Defective Pixel Correction/2D Noise Reduction
- High-Quality 5 x 5 De-Bayer Block
- Color Correction Matrix
- Statistics Engine
- Fast Auto Exposure Block
- Auto White Balance Block
- Helion BLENDFEST™ HDR Block
- Gamma Correction
- Graphics and Text Overlay

Comprehensive IP Library

The kit is supported by a comprehensive intellectual property (IP) library. Containing more than 90 individual pieces of IP, the library is available for licensing either entirely or in parts. Developed by Lattice partner Helion GmbH, the library provides complete sensor to display ISP support including:

- Sensor and Display Interfaces
- Color Pipeline
- Image Enhancement
- Peripherals and Overlay

Applications Support
1-800-LATTICE (528-8423)
503-268-8001
techsupport@latticesemi.com

Copyright © 2012 Lattice Semiconductor Corporation. Lattice Semiconductor, L (stylized) Lattice Semiconductor Corp., Lattice (design), Lattice Diamond and LatticeECP3 are either registered trademarks or trademarks of Lattice Semiconductor Corporation in the United States and/or other countries. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

March 2012
Order #: I0213A