NTNU

Norwegian University of
Science and Technology

Tone mapping in video conference systems

Erik Strømme

Master of Science in Electronics
Submission date: June 2011
Supervisor: Per Gunnar Kjeldsberg, IET
Co-supervisor: Lars Aurdal, Cisco

Norwegian University of Science and Technology
Department of Electronics and Telecommunications

Problem Description

Student: Erik Strømme

Title: Tone mapping in video conference systems

Problem: Problem description
Supervisor: Lars Aurdal, Cisco Systems Norway
Subject teacher: Per Gunnar Kjeldsberg, Professor NTNU

Video signals from digital video cameras have a limited bit depth and a thus a limited dynamic range available. However, a particular scene might require a much higher dynamic range. This causes either dark regions in the image (if not too many pixels are to be saturated), or saturation in high-light regions in case the dark regions are exposed at a higher level.

One way of improving this is to introduce a global tone-mapping function, i.e. a mapping that takes input signal values and maps them to new output values. Typically, this mapping would boost dark areas in case the scene shows a lot of contrast. So-called local tone-mapping takes this one step further by using different mapping functions in different regions of the image depending on what the image looks like locally. Local tone-mapping will require some analysis functionality to support this local adaptation.

The task is a continuation of an autumn project and includes a literature research phase to explore different solutions, then simulations of the selected approach followed by an implementation on real time hardware (FPGA). The implementation is likely to involve the use of some kind of histogram function in the FPGA, implementation of some robust real-time logic to determine a suitable global tone-mapping curve and finally implementation of the actual tone-mapping function in FPGA. Global tone-mapping should be considered before local tone-mapping.

Abstract

Normal sensors are able to only capture a limited dynamic range. In scenes with large dynamic range, such as situations with both dark indoor and bright outdoor parts, the image will get either over- or under exposed if the exposure is not perfect.

Producing high dynamic range (HDR) images will capture the full dynamic range of the scene. There are two main ways of producing HDR images. One combines multiple exposures with a low dynamic range (LDR) sensor. Another is to use a sensors which are able to capture a higher dynamic range, so called wide dynamic range sensors.

Multiple exposures with a single low dynamic range sensor, is not suitable for real time video because this technique have large problems with movement. Wide dynamic range sensors only require one exposure, but these have difficulties in normal situations were LDR sensors are sufficient.

A type of algorithms called tone mapping are used to reduce the high dynamic range image to fit the limitations of normal monitors. Simulations show that using these algorithms on low dynamic range images will change the illumination of the scene, solving the problem.

Tone mapping algorithms presented in the literature are software algorithms. Two groups of algorithms exist; local and global tone mappers. Local algorithms are time consuming, and require large amounts of memory. They are not suitable for real time implementations since they rely on filtering operations for each pixel. Global algorithms, does not rely on filtering and are less time consuming. A precomputed curve is used to map the pixels to new values. This makes the global algorithms more suitable for video.

A reduced tone mapping system is presented. This reduction results in a segmented curve, which drastically reduces the memory required for defining the curve. It also makes it feasible to control temporal changes. The reduced system has been successfully implemented, achieving sufficient frequencies to be part of a real time system.

Preface

This master thesis is the final part of the Master of science degree in electronics at NTNU. The purpose of this thesis was to continue the project conducted in the autumn of 2010 regarding tone mapping.

I would like to thank my supervisors, Professor Per Gunnar Kjeldsberg at NTNU and Lars Aurdal from Cisco Systems Norway. I would also like to thank the rest of the camera group at Cisco Systems, especially Ronny Randen, for a very valuable discussion on the selected algorithm and the implementation. Finally, I would like to thank my colleague Thor Arne Brandsvoll for interesting discussions throughout the semester.

Trondheim, June 19, 2011

Erik Strømme

Contents

List of abbreviations

			Page
APS	-	Active pixel sensor	13
ASIC	-	Application Specific Integrated Circuit	27
CAM	-	Camera Appearance Model	27
FPGA	-	Field Programmable Gate Array	28
FPS	-	Frames Per Second	8
LDR	-	Low Dynamic Range	10
HDR	-	High Dynamic Range	10
HDRi	-	High Dynamic Range image	4, 10
HVS	-	Human Visual System	24
SNR	-	Signal to Noise Ratio	13
WDR	-	Wide Dynamic Range	13

1 Introduction and motivation

Humans are able to distinguish an incredible range and variety of colors in a natural scene. Since the earliest times it has been a quest to reproduce scenes for later. This reproduction was dependent on two main factors, where the first was the quality of the medium and the other the skill of the artist. One of the earliest examples is paintings in caves showing hunters and animals. The use of these can only be speculated on, but from a scientific point of view the paintings were done on a crude medium with only the most basic form of paint. This gave the painters few colors, or range of colors, to work with. As history progressed, painters increased the range of colors to work with and the quality of the medium. This gradually forced a shift from where the end result was dependent on the materials, to the skill of the painters.

With the invention of the photography the problem moved slightly towards the equipment. Before the digital era, photographers had to manually process each photo for the best result. This was more about choosing how the light should be represented. One way of doing it was with the zone system [1, 2, 3, 4]. Here the photographer selected the mid gray level, while the rest of the picture was mapped relative to this. This was based on the realization that the immediate surroundings of sections in the image influence how that section is experienced.

Perfectly reproducing a real world scene is a very difficult task which depends on many factors. The two most obvious factors today which prevent this, is the quality of the camera and the monitor. The quality of the camera is mainly depending on the image sensor and how many colors the image sensor can sense. The quality of the monitor depends on how many pixels the display has and how many colors each pixel can show.

In the modern digital world it has become possible to save pictures digitally. One important result of this is that it is possible to store more color variations than it is possible to show. The amount depends on the image sensor used to make the image. Algorithms to compress these images to a viewable range can be called image reproduction or tone mapping algorithms.

The earliest tone mapping algorithms were made for compressing high dynamic range (HDR) computer generated images to a viewable range [5]. Common for this type of image processing is that all details in the scene, such as the reflectance and transparency, are known.

With developments in sensors, production of HDR images of natural scenes became possible. The problem with this case compared to artificial HDR is that there is no information about the physical properties of the scene. This reduces the amount of information that the tone mapping algorithms have to work with.

Spivak et. al [6] summarized the overall high dynamic range imaging task to two stages. The first is to capture the scene in high dynamic range which would prevent loss of details. This would then be followed by post processing with tone mapping algorithms to be viewable on a normal computer screen.

Algorithms designed for video is not common. The reason for this is that video recording in high dynamic range is almost nonexistent. Also, if a high dynamic range video is to be compressed to a lower range, simply using algorithms for image processing is possible in a post production phase.

Video conferences have some unique challenges compared to normal video. The biggest difference is that there are requirements on how much delay there can be. If the delay is to large, the experience will degrade substantially.

One specific problem for video is scenes with challenging light conditions. Typical examples of this are scenes with both indoor and outdoor parts. The normal sensors are not able to capture the full range of light, and will choose an exposure. The exposure might be technically correct, but might results in a focus on parts of the scene which are not interesting. Typically, the indoor part of the scene might be casted very dark compared to the outdoor part. This will greatly reduce the subjective quality of the scene.

Using high dynamic range video would solve this since the video would then be able to capture both the indoor and outdoor part. The problem then would be how to view the video. HDR monitors are gradually becoming available [7], and these are able to display the HDR video and images. However, most monitors have a very limited dynamic range, and a processing step is necessary to reduce the HDR content to fir the limitations of this. This is known as tone mapping. Another use of tone mapping algorithms is to change the brightness of the scene. The problem with video conferences in this regard, is that it should be real time. This limits how much post processing can be done.

This report will first look at how an image is created in chapter 2. This is a natural start, since a video is basically a series of images taken in rapid succession. Here, it is natural to start looking at how the basic image sensor works, but also on some state of the art image sensors. Aspects such as noise removal and demosaicing will be briefly discussed as they are vital parts of any image pipeline. This will be followed by a discussion on what high dynamic range images are and how these differ from low dynamic range images. Different solutions to generating high dynamic range images will be presented. Special interest will be taken in how this can be implemented in a real time video system.

Chapter 3 is about tone mapping algorithms. There are two broad categories of tone mapping algorithms, global and local algorithms. From the first group, global tone mappers, several algorithms will be presented. These algorithms are more suitable for a real time implementation (see chapter 4) than the local algorithms. A summary of some of the most important and representative local algorithms will be presented. A more detailed discussion of these can be found in [8].

Chapter 4 contains a more in depth look into the constraints and limitations of using a tone mapper in real time on an FPGA. Here a selected algorithm will be discussed. A system for reducing the algorithm to better fit the constraints of the FPGA will be presented here.

Simulation results is presented in chapter 5. Here a number of practical situations will be compared, with tone mapping both on and off. How the reduction influences the results will be discussed here. The goal of this chapter will be to justify tone mapping visually.

The next chapter, chapter 6, will present the implementation of the reduced tone mapper. Various choices related to the implementation will be discussed here.

The report will be concluded in chapter 7.

The last chapter will outline directions for future work with tone mapping in real time video conferences. The main step here will be to test the tone mapping on a video or video system.

Main contributions of this thesis:

* Evaluation of HDR for video.

* System for reducing global algorithms to fit FPGA constraints.

* Implementation of a real time reduced tone mapping system.

2 The image

Tone mapping is specifically designed for processing High Dynamic Range images (HDRi). However, it is important to have a basic understanding of how normal low dynamic range images are made, and the limitations of these. This chapter will therefore be dedicated to how the image is created.

First a general image sensor will be discussed. This is to show the overall design which is shared by most image sensors. This will include a brief overview of noise sources and how color is captured. Both aspects are very important in how the image is perceived.

This will be followed by a discussion on dynamic range. The dynamic range of a sensor is one of many different characteristics for image sensors. Some common challenges in traditional photography will be discussed here, since these are relevant for both dynamic range and HDRi.

The next section will concern HDRi. This is a technique which is designed to solve some of the challenges of traditional photography. There are a number of different ways to produce HDRi, and some of these will be discussed here. The drawbacks of the techniques and how well they could work in a real time video will also be discussed.

2.1 The sensor

The first step of creating an image is to record the scene. This is done by measuring the visible light reflected by the scene.

Digital circuits capable of measuring the light of a scene arrived with the digital era. Many different photo detector architectures exist, but they all have in common that they are able to transform incident photo flux to photo current [9]. An example of a photo detector design can be seen in Figure 1.

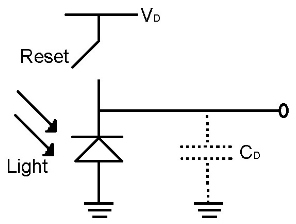

Figure 1: *This is a photo detector operating in direct integration. Charge collected is gathered across the diode capacitance C_D. At the end of the integration time, the charge is either directly read out or directly converted to voltage and then read out. Figure from [9].*

The total sensor is basically a matrix of photo detectors. During exposure, charge will be accumulated by each photo detector (Across C_D in Figure 1) based on how much light the photo detector is exposed by. The result is an electronic representation of the scene in black and white. This is then combined with analog to digital converters to produce a digital representation of the scene.

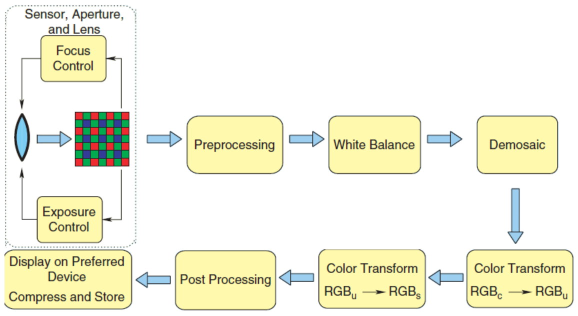
Sensor, Aperture, and Lens
Focus Control
Exposure Control
Preprocessing
White Balance
Demosaic
Color Transform $RGB_c \longrightarrow RGB_u$
Color Transform $RGB_u \longrightarrow RGB_s$
Post Processing
Display on Preferred Device
Compress and Store

Figure 2: *A general image pipeline. The pipeline sequence varies between manufacturers. Figure from [10]*

Figure 2 shows all the processing steps which are necessary to produce an image. The sequence of the processing steps varies from manufacturer to manufacturer. The mosaic pattern seen in the sensor, aperture and lens block, is a bayer filter. This is one way of producing color images, and will be discussed in the section about *Color capturing*. The demosaic block is related to the bayer filter. This step will use the raw sensor data, or bayer data to produce a color image.

Humans are able to understand white objects, even if the object emits light with wavelengths technically not white. White balance is a step which tries to emulate this, by changing the values produced. One way of doing this is to assume that the maximum value in the image represents white. This value should then be scaled up so the digital representation is maxed. All other values in the image should be scaled with an equal amount [10].

2.1.1 Noise

An image sensor pipeline with noise sources is shown in Figure 3. How much each of these sources contribute to the end sum of noise vary between sensors. The read out rate and how the pixels are designed are just some of the aspects which affect the amount of noise. Noise will be visible in the image, and too much will reduce the quality of the image. The goal is therefore to reduce the noise as much as possible.

Dark current is considered to be one of the defining characteristics of an image sensor [9]. This indicates how much current flows through each photo detector when they are not exposed to any illumination. The amount of dark current greatly affect the quality of the image in low light conditions [12, 11].

There is a series of additional noise sources, as seen in Figure 3. Temperature, exposure time and read out rate will all affect how much noise each source contributes. This makes it very difficult to remove all noise.

There have been a great deal of research done in how to remove noise, known as denoising [13]. Denoising algorithms can be done both in pre and post processing of the image.

Denoising in video sequences is closely related to denoising for images. The

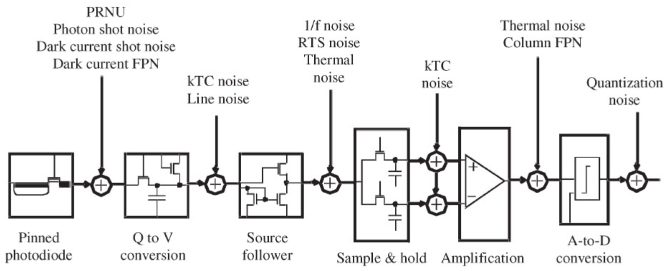

Figure 3: *Figure showing the noise sources in an image pipeline. These sources are commonly summarized when discussing and working with noise removal. Figure from [11].*

difference is that video sequences will have an additional temporal dimension as well. This dimension can be used in many ways to remove some of the random noise. One very simple algorithm is to take the average of two consecutive frames. This has many downsides, so more advanced techniques specific for video have been developed [14].

2.1.2 Color capturing

Humans perceive light with different wavelengths as color. The eyes contains cones which allow us to do this [15]. There are three types of cones, short, medium and long. These differ in which wavelengths they sense, and correspond to wavelengths which we experience as blue, green and red.

An image sensor can be said to capture only different amounts of light. The most common way of measuring color in digital photometry is with a bayer filter [16] that can be seen in Figure 4. There are other types of color filters used, but the bayer filter is the most common. Different types differ either in the distribution of filters or different filters altogether, like CYGM (cyan, yellow, green and magenta). The principles behind are the same, but the properties change somewhat with different filters. In a bayer filter, a red, green and blue filter is used to sample the light with different wavelengths.

A photo detector covered in a red filter will only let red light pass through, sensing how much red light is received by that photo detector. Green will only let green light through, and blue only blue light. Here red, green and blue are understood as light with wavelengths corresponding to what we humans perceive as red, blue and green. Figure 5 shows the spectral response of one sensor using a bayer filter.

After capturing an image with a sensor using a bayer filter, a series of steps are necessary to compute the color image.

The most important step, which is directly related to the bayer filter, is called demosaicing. Here the raw bayer data from the image is transformed to a proper image. Each raw bayer pixel only represent one color, so one way of doing this is to interpolate between groups of four Bayer pixels. The resulting average would then

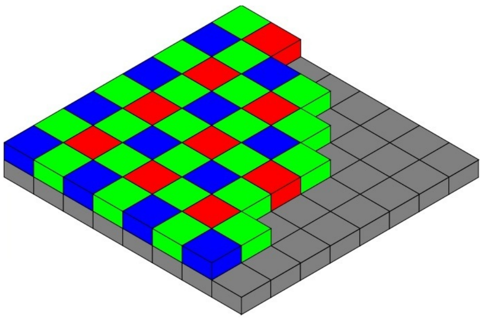

Figure 4: *A bayer filter. Here the gray blocks are the photo sensing pixels, while the colored blocks are filters. The photo sensing pixels are only able to sense light, not distinguish colors. Using a bayer filter will thus enable it to sample color, although the total resolution will be reduced. Figure from wikimedia, File:Bayer _pattern_on_sensor.svg*

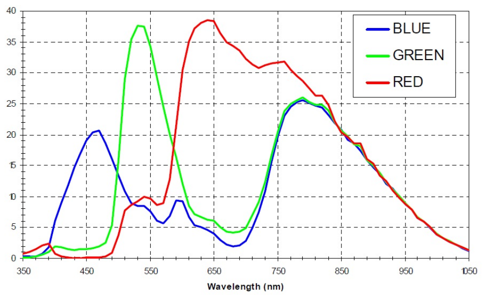

Figure 5: *Spectral response of a CMOS sensor using an RGB bayer filter. This response is specific for this particular sensor. Other sensors have different responses. Figure from [17]*

represent the pixel for that section in the image. There are many more advanced ways of doing this, some which are discussed by Xiu et. al [18].

Another step is color correction. This step is specific to a given type of image sensor. It will counter the variations of the image sensor and return an independent version of the image. In the color correction step, the colors are modified to counter variations of the sensor type and the filters.

2.2 Dynamic range

Dynamic range is a ratio between the largest and smallest measurable quantity of a varying quantity. Typically, the dynamic range is either represented as a ratio, as seen in Figure 6, or a decibel value.

The dynamic range of a camera is the ratio of the highest detectable non saturating photo current to its smallest detectable photo current. The smallest detectable photo current is limited by the dark current and the fixed pattern noise [19, 20, 9].

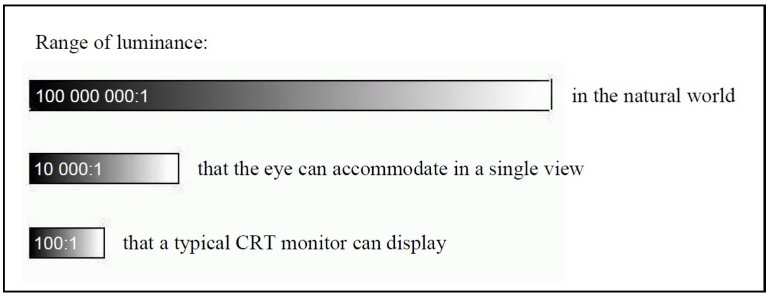
Range of luminance:
100 000 000:1
in the natural world
10 000:1
that the eye can accommodate in a single view
100:1
that a typical CRT monitor can display

Figure 6: *Illustration of how large dynamic range humans can perceive compared with a CRT monitor and the natural world. The numbers here are just for illustrating the differences. The human eye are capable of an extremely large dynamic range, but not at the same time. Figure from [21]*

Examples of dynamic range in decibel can be found in data sheets of sensors. One example is found in the data sheet to Kodak's KAF-09000 image sensor [22]. Here the dynamic range is given to be 84 dB. Other sensors have different dynamic ranges, for example the older KAC-3100 image sensor [23] which has a dynamic range of 48 dB. Higher dynamic range indicate that the sensor is capable of capturing a larger variety of light in the scene compared to a sensor with lower dynamic range.

These sensors are good examples on how different sensors have different uses. The KAF-09000 sensor only has a frame rate of 0.4 frame per second (fps), while the KAC-3100 sensor has 10 fps. This difference will greatly affect how much noise there is in the sensor. In general, higher frame rate will increase the noise [11], affecting the lower bound of the dynamic range.

It is also interesting to look at the difference between the KAC-3100 and KAC-5000 [24] sensors. Here the KAC-5000 model has a higher dynamic range, despite a bigger sensor. The higher dynamic range can be traced to the smaller read noise reported. This read noise is defined as the total temporal noise in dark, and it is the quantity Kodak use when calculating the dynamic range.

Both Yang et. al [19] and Gammal et. al [9] pointed out that dynamic range alone does not correspond well to the image quality. Another important factor is the signal to noise ratio (SNR). This quantity indicates how good the quality of the signal is. Extending the dynamic range at a cost of decreased SNR is possible as discussed by Yang et. al, but it will reduce the quality of the image.

Exposure

One parameter often seen in photography is exposure time. Varying this will change how long the sensor will measure the light of the scene.

The practical aspect of this in regards to the dynamic range can be seen in Figure 7. The dynamic range is unchanged, but the darkest and brightest points are changed equally by changing the exposure time.

By increasing the exposure time, the integration time of the sensor will be

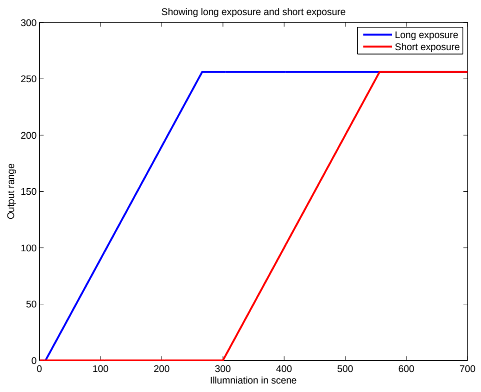

Figure 7: *This illustration shows how changing the exposure time will change the focus of a scene. The horizontal axis indicates how much light there is, while the vertical axis indicates how the corresponding light will be represented in the image. The horizontal part of the long exposure around 250 indicates that the sensor has gone into saturation. This means that light above this will be indistinguishable. However, details in the range of 0 to 300 will not be visible for the image taken with short exposure.*

increased. This means that the sensor will be given longer time to gather light before the sensor is read and reset. Typically, for dark scenes the exposure time should be long to capture as much light as possible. For a bright scene on the other hand, the exposure time should be low to prevent saturation of the sensor.

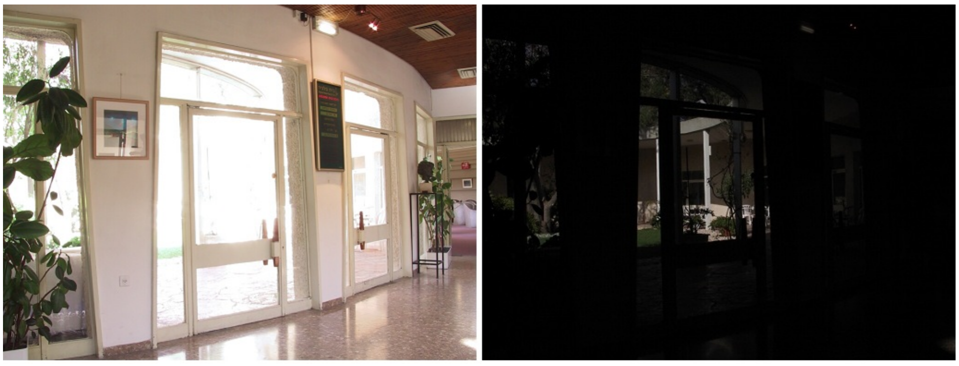

Figure 8: *This figure illustrates the difference between an overexposed image (left) to an underexposed image (right). Images from Fattals image library.*

Setting the exposure time too low will result in an underexposed image, as can be seen on the right part of Figure 8. Setting it to high on the other hand will result in an overexposed image. Here details in bright regions will be saturated.

2.3 High Dynamic Range images and video

The difference between high dynamic range (HDR) images and low dynamic range (LDR) images is a bit diffuse. A common misconception is that the bit depth alone defines the difference. In the book by Reinhard et. al [25] about High dynamic range imaging, one definition is proposed. The main difference given here is that LDR images are images which are output-referred, while HDR images are scene referred. Therefore, HDR images will be able to capture all the light variations of a scene [26].

This definition implies that LDR images can be viewed without any processing, while HDR image will have to be processed in some way. This is a result of normal monitors having a low dynamic range, and algorithms used for this are called tone mapping algorithms [7].

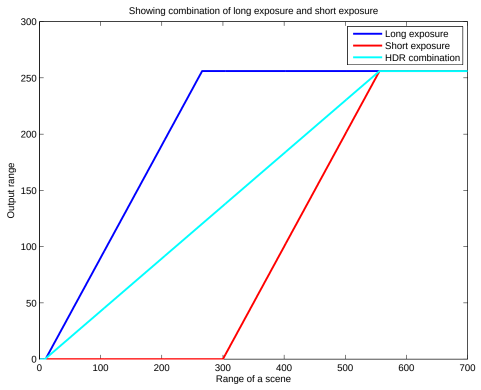

Figure 9: *This illustration shows how an HDR image will have a larger dynamic range than images with short and long exposure. The larger dynamic range for HDRi will allow it to store more information about the scene. A note here is that HDR images usually is normalized to 0,1 and represented in a floating point format to cover all the small variations.*

HDR images have two main sources. The first and most common are computer generated images. This technique is a result of powerful graphical processing units which are able to render artificial scenes with clearly defined details about light and reflection in a scene.

The other source is real world HDR images. There are many techniques used to produce these. One is to capture a series of images of a scene with different exposures. These can later be combined, giving an HDR image.

2.3.1 Multiple exposure

Images

The first, publicly available, method for generating high dynamic range images (HDRi) was presented by Debevec and Malik [27] in 1997.

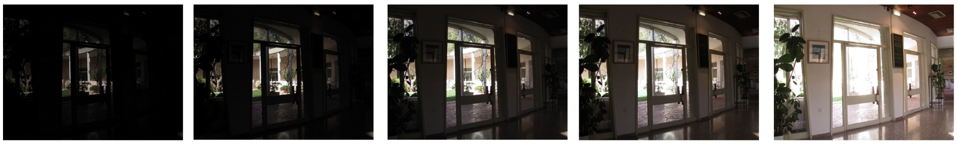

Figure 10: *This figure illustrates how images of a scene with different exposures will look. Combining these images will give an HDR image which will have a bigger dynamic range than a single exposure. Images from Fattals image library.*

In their paper they assume that the camera response curve is unknown. This means that they first have to develop an algorithm that is able to find this curve. The assumption made is that the system is an analog film camera, so the response curve will be non-linear. They present a solution to find the curve by taking a number of different pictures of the same scene while varying the exposure. These pictures should be aligned under the constant light conditions, so that a given pixel will represent the same section in all the exposures. An example of this can be seen in Figure 10. It is possible to find the camera response curve with as few as two exposures if this constraint is maintained. The accuracy of the camera curve however, will increase with the number of exposures.

$$lnE_i = \frac{\sum w(Z_{ij})(g(Z_{ij}) - ln\triangle t_i)}{\sum w(Z_{ij})} \tag{1}$$

In their paper they use the camera curve to produce an HDRi image. This can be seen in equation 1. Here g is the camera response function and Z_{ij} is pixel value number j in exposure i. $\triangle t_i$ is the exposure of image i. The result of this will be a linear HDRi image.

In the algorithm by Debevec and Malik more weight is given to a pixel with values closer to the middle. This is realized by $w(Z_{ij})$ in the equation. This results in more focus on the middle part of the available dynamic range, where most of the information is assumed to be.

Video and movement

In principle, making a video sequence in HDR is just the same as making an HDR image. The problem is that by nature, video is made for capturing movement. Simply using the basic HDR production will therefore introduce an effect known as ghosting.

Ghosting is an effect which will arise if there is movement between the exposures. For example, if a person moves between the exposures, a ghostly outline will be present. This effect can be seen in Figure 11. Larger movement between two exposures will result in a more visible ghosting effect, which drastically reduce the subjective experience of the scene. Moving the camera between two frames will also produce this effect

Kang et al. [29] presented a system for capturing HDR video. The first step is to film a sequence while varying between long and short exposure for every second

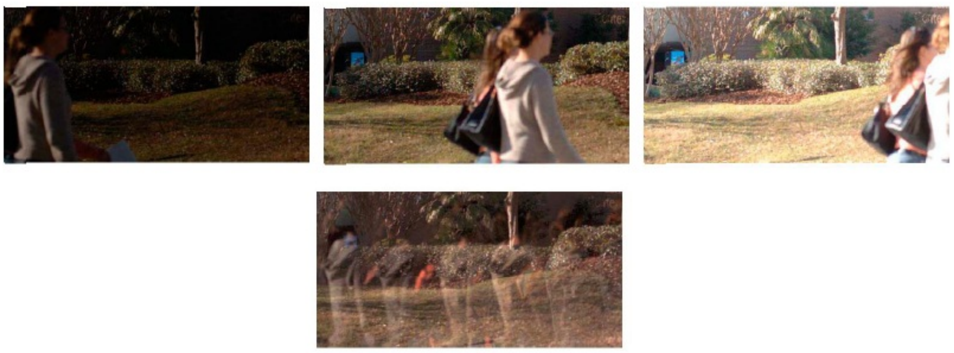

Figure 11: *The ghosting effect. The top three images are from a sequence of ten images which were used to make the HDR image shown at the bottom. (Figure from [28])*

frame. Essentially, this rely on the same concept as presented by Debevec and Malik [27], but the camera response curve is assumed to be known. An implementation of this would therefore be concerning how the different exposures are combined. The problem with this is the ghosting effect, so the authors propose a system to prevent this. In the post process production of the HDR sequence they incorporate both backward and forward temporal filtering for a given frame while relaxing of the transformation of the radiance map compared with the algorithms of Debevec and Malik. The central frame is then mixed with the two modified temporal adjacent frames.

Ways to remove the ghosting effect have been proposed by many authors [28, 30, 31]. These typically make use of extensive edge detection and temporal filtering to remove the ghost effects. As a result of this, HDR video requires extensive post processing. This makes it virtually useless for a real time application, unless frame delays and frame buffers are acceptable.

Multiple sensors

This subgroup is based on the same principle as multiple exposures HDRi. The goal of this approach is to remove ghosting effects by making the capturing of the scene more efficient and consistent. In this type of solution, two (or possibly more) image sensors are used to capture the scene with different exposure times. Since two sensors are used, one can capture the scene with long exposure, while the other capture with a short exposure. By overlapping the timings of these, so that the short exposure is timed to be in the last part of the long exposure, ghost free HDRi can be made.

Systems based on this approach were discussed as early as 1993 by Ginosar et. al [32] in their patent application. Here a number of sensors were to be used instead of varying the exposure of one sensor. The actual image processing would then use the same principles as in the multiple exposures with one sensor technique.

2.3.2 Wide dynamic range sensors

There are many ways of extending the dynamic range, and one is to produce better circuits with lower noise levels. This will extend the lower bound of the dynamic range. Another is to expand the maximum measurable light which the sensor is able to detect. These sensors are called wide dynamic range (WDR) sensors. They have limitations to the dynamic range, so they can not produce images which are perfectly scene referred (HDRi). The dynamic range they are able to capture is larger than normal sensors, so they are called WDR instead to distinguish them from normal LDR sensors and HDR images.

Extending the dynamic range of a CMOS sensor

Much research has been done in extending the dynamic range of sensors. Spivak et. al [6] evaluated different solutions of increasing the dynamic range of a sensor by changing the response function. These are solutions which extend the maximum brightness which the sensor will be able to detect, thus extending the upper bound of the dynamic range. The problem is that if the signal to noise ratio (SNR) increase to much, the quality of the image will decrease despite the increase in dynamic range. For all the solutions presented, dynamic range, complexity and SNR were evaluated.

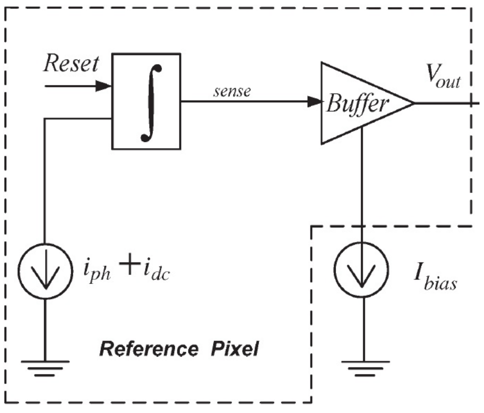

Figure 12: *The reference active pixel sensor (APS) used to compare the solutions.* i_{ph} *is the photocurrent generated during exposure and* i_{dc} *is the dark current. Figure from [6]*

The solutions presented can be divided into three different categories; piecewise - linear, non - linear and frequency based.

The non-linear methods include logarithmic, multi mode and time to saturation solutions. These solutions had the highest potential increase in the dynamic range of all the solutions. However, the cost of these was a drastic reduction in sensitivity, reducing the image quality.

Piecewise linear methods include sensors which use clipping and global and automatic resets of pixels. These were able to reach high dynamic ranges, as well as maintaining high sensitivity. Solutions of this type included multiple captures on a per pixel level. This method required an increase in number of readouts to maintain the frame rate, and this would again decrease the SNR.

Frequency based sensors are sensors which measure the time to first saturation or similar schemes. The dynamic ranges of these were high, but there was a drastic reduction in sensitivity for low brightness situations. The number of transistors required for these were much higher than the other methods. A much larger amount of post processing would also be required.

Common for all the methods is that there will be an increase in the number of transistors for each pixel. How many varies. This will lead to less effective pixels for a given area.

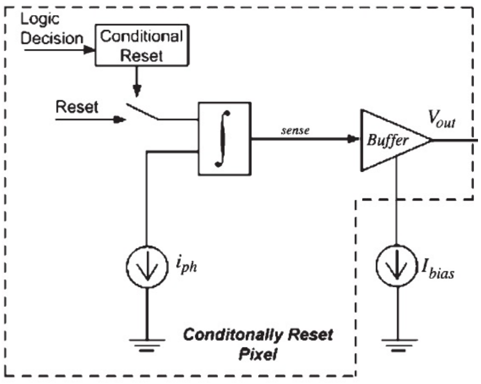

Figure 13: *The modified pixel used for autonomous reset.* i_{ph} *is the photocurrent generated during exposure. Figure from [6].*

The most interesting method presented is one of the piecewise linear methods which utilize a local conditional reset. The modification to the reference photo detector in Figure 12 can be seen in Figure 13. Each photo detector which is saturated during exposure will be locally reset. The number of resets for each pixel is stored and reported together with the final pixel value. This solution solves some of the problems with lack of sensitivity in both bright and dark regions. The amount of post processing will increase, but not as much as the frequency based solutions.

2.3.3 Piecewise linear sensor

One available WDR sensor is Kodaks KAC-9681 [33]. This sensor has an optional non-linear mode which increases the dynamic range. In normal mode the dynamic range is 62 dB, while the non-linear extends the dynamic range up too 110 dB. The non-linear mode enable a piecewise linear mode on the sensor which enables the sensor to measure a higher maximum illumination. The principle can be seen in Figure 14.

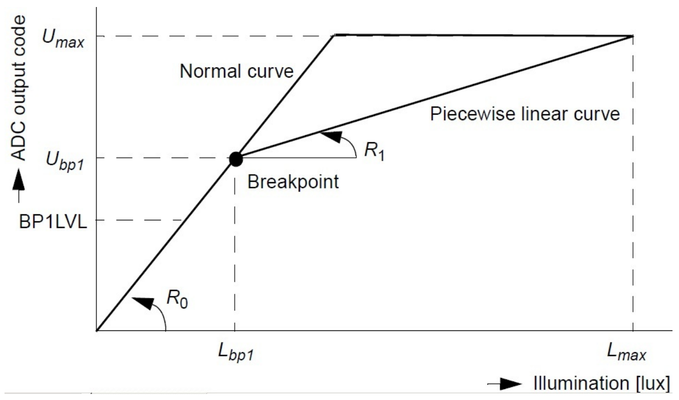

Figure 14: *This figure show in principle the difference between the two modes of KAC-9681. Figure from application note [33].*

The problem of using this scheme is that the number of possible encodings will not increase. Parts of the scene with illumination between L_{bp1} and L_max will have a limited amount of possible encodings. It will therefore be a smaller number of encodings relative to the extended range, compared with the normal curve.

2.4 Displaying HDR images and video

Displaying HDRi on LDR monitors require a reduction in the amount of data. This is done with tone mapping algorithms. These algorithms will reduce the high dynamic range content to a lower dynamic range to fit the limitations of the monitors. If the monitor were able to display the full range of the HDR image, tone mapping would not be required.

This is shown by Akyüz et. al [7] when they used used the DR37-P HDR display developed by BrightSide Technology [1] to test if HDR images is preferred over tone mapped images.

Akyüz et. al tested the preference of three pipeline structures. The most advanced was the full HDR pipeline with HDR capture and HDR display. The second pipeline used tone mapping to reduce the HDR image and display it on an LDR monitor. The third was to simply use a normal pipeline with LDR capture and LDR display.

The study showed that the full HDR pipeline was preferred over the tone mapping. The best LDR image from the sequence used to make the HDR image was also preferred above the tone mapping. The authors speculated that this result came because the participants were not used to tone mapped images and preferred images which appeared more familiar.

[1]BrightSide Technology was acquired by Dolby in 2007 and renamed Dolby vision

3 Tone mapping algorithms

Using tone mapping on an image is an interesting technique which will enhance the image. Careful application of this might reveal more details or enhance the brightness in a scene. Algorithms developed for tone mapping have been designed to process a high dynamic range (HDR) image to be viewable on a low dynamic range (LDR) medium. This medium can in principle be anything from a computer screen to a piece of paper.

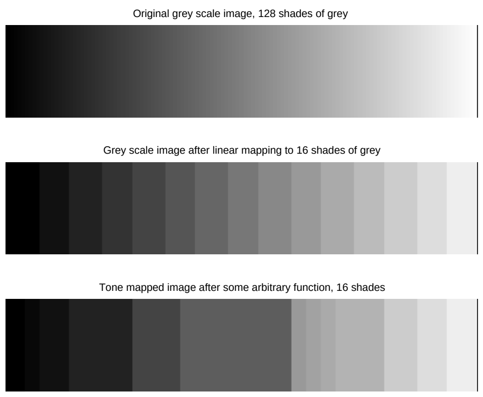
Original grey scale image, 128 shades of grey
Grey scale image after linear mapping to 16 shades of grey
Tone mapped image after some arbitrary function, 16 shades

Figure 15: *Figure showing how a gray scale image will be tone mapped in two different ways. Original image on top. Shades goes from 0 to 127 with one column of black pixels at end to limit the image. In the middle, the scale from the first are simply linearly transformed. This means that all the new values cover the same number of old values. The bottom part of the figure show tone mapping with an arbitrary operator. Here the new values does not cover the same amount of old values. In this particular example, more weight are given to one of the shades in the middle.*

Tone mapping algorithms can also be used as a form of dynamic compression, going from a high range to a lower range. Because of this, images with a high bit depth per color can be reduced to a lower bit depth by the same algorithms.

Tone mapping functions can be divided into two large groups, global (spatially uniform) and local (spatially varying). In the local type, pixels are mapped according to local image characteristics. This form can also be called tone reproduction operators, because in practice an operator is run on each pixel. In the global type, all the pixels are mapped according to a global curve. Another name for this type is therefore tone reproduction curve. An example on how different tone mappers will behave can be seen in Figure 15. Here the top part represent part of an imaginary image. Tone mapping this image with two different algorithms might return two different images, as seen in the middle and at the bottom.

Using different algorithms on an image will give different images. There is no objective measure, except performance, that distinguishes the results. Because of this, authors tend to use the same set of HDR images to compare algorithms.

Cadik [34] did an extensive review on the different tone mapping techniques.

One very interesting conclusion was that the most important aspects to an observer is how good the overall image is. Overall quality is a subjective measure that is shown to be correlated to the quality of color, contrast and brightness.

Ledda et al. [35] tested a number of algorithms on humans. Among the two most interesting tests were a test where the test subject were asked to rate the same picture given with different tone mappers next to each other. In the other test were the subjects were asked to rate pictures without any references. The result showed that algorithms that performed badly when compared to other algorithms performed a lot better without any image to refer directly to.

The gamma function will be presented first in this chapter. This is a simple example of a tone reproduction operator.

A selection of global tone mapping algorithms will be presented in the next section. This class of algorithms are most relevant for an hardware implementation (see chapter 4.1.1), so only a short summary of the most important local tone mapping algorithms be presented next.

Problems related to color treatment in tone mapping will be discussed after this. Most of the tone mapping algorithms presented here work in the intensity domain. Intensity is dependent on all three color channels, and changes in this domain are therefore reflected in all three channels.

3.1 Gamma correction function and sRGB

Gamma correction is not strictly a tone mapping function, but a well known tone reproduction function to code and decode an image. Originally it was used as a way to correct the non-linear characteristic between the input electron signal and the output luminance of CRT monitors [36]. The relation is approximated by a power function seen in equation 2.

$$\Psi = \xi^{\gamma} \tag{2}$$

In this equation, Ψ is the output intensity and ξ is the input signal of either color channel.

Figure 16 shows the gamma correction function recommended by ITU [37].

sRGB

The gamma function of CRT monitors form parts of the definition of sRGB which Microsoft and Hewlett Packard presented in 1996 [38]. The idea was to standardize a set of equations and characteristics, which would make an image to look the same on all monitors. Before this, manufacturers of both monitors and printers did not have any standard to follow on how a digital value should be represented.

3.2 Global tone mapping algorithms

All tone mapping functions that can be characterized as global will make a curve which depends on the image. Each algorithm introduces a certain trade off between

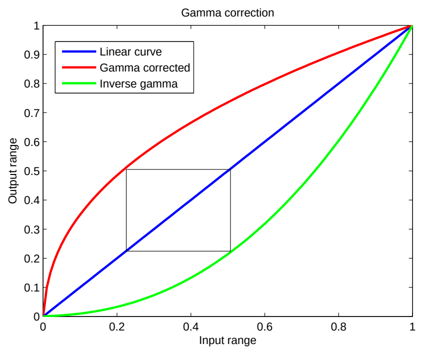

Figure 16: *Showing how the gamma function (red) will correct to non-linearities of the monitor (green). Starting at the lower left of the rectangle and following this clockwise show how the gamma function (red) correct the non-linearity of the monitor (green) to return the original value (blue).*

details in dark areas and bright areas. All the pixels are mapped according to the specific curve. This makes the algorithms very fast if the whole picture is available.

The most computationally expensive part in these algorithms is to calculate the curve. This type of algorithms will preserve the image characteristics; dark areas stay dark while bright areas stay bright after tone mapping. Unfortunately, local differences in contrast might disappear because of the compression.

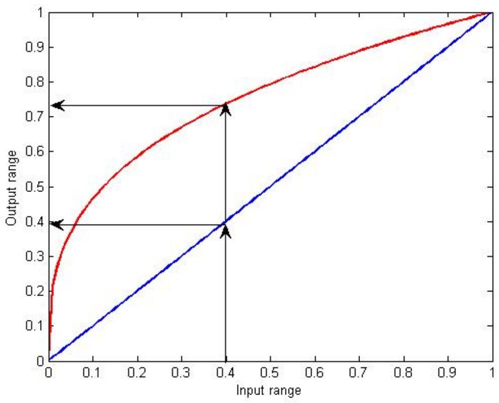

Figure 17: *Two arbitrary examples of tone mapping curves. The vertical axis is the new dynamic range while the horizontal axis is the original range. The arrows show how a value in the original image will be mapped too two different values, depending on the tone mapping curve used.*

Figure 17 shows two different curves, a linear curve and an arbitrary curve. These curves represent different tone mapping algorithms and are examples of how two global tone mapping curves will result in different images.

3.2.1 Tone reproduction for realistic images

An early tone reproduction algorithm was presented by Tumblin and Rushmeiers [5] in 1993. In this article they gave an algorithm that could tone map a gray scale picture to better reflect the proper brightness levels for film and television. The algorithm they propose is not incorporating color, nor spatial effects. Also, they have to use clipping for bright images, as the mathematical model for the algorithm is flawed for these values.

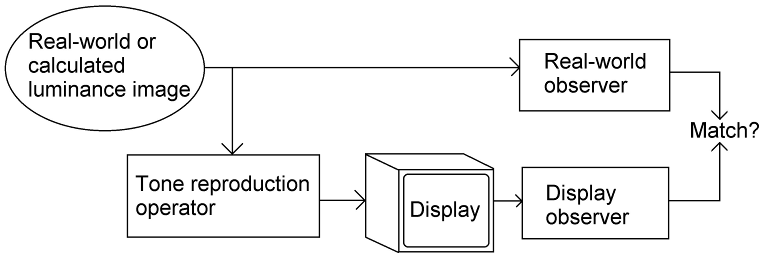
Real-world or calculated luminance image
Real-world observer
Match?
Tone reproduction operator
Display
Display observer

Figure 18: *This figure shows one way to define the tone reproduction problem. The goal of this is to replicate the exact same image of a scene on the display (in principle any medium) as it looks in the real world. Tone mapping is part of the tone reproduction operator. (Figure based on figure 1 in [5]).*

An illustration of the problem they define can be seen in Figure 18. To be able to solve this, the authors require that the tone reproduction operator must incorporate a real world model, an inverse model of the observer and an inverse model of the display. To fulfill this, they presented three different steps for an algorithm. The first step is an observer model, which incorporates a basic understanding of the visual system. An inverse display observer model and an inverse display device model is then approximated and concenated with the observer model to make the final tone reproduction operator.

3.2.2 Histogram equalization

The most basic form of tone mapping is linear mapping. This will typically result in images where the overall brightness and look of the image is conserved.

The opposite is histogram equalized mapping. In histogram equalization, a histogram over a quantity is made. In tone mapping, the intensity values are used for the histogram. The intensity is calculated by all three color channels, and changes here will therefore affect all the three color channels. A mapping is then made, which aim at making the histogram flat. This mapping is a curve which will show which new values the old values should be changed too (see Figure 19, bottom part).

Images tone mapped with histogram equalization will reveal more details, compared with images which are tone mapped with linear mapping. The reason for this is that in linear mapping, a large amount of pixels might be covered by a small output range (see Figure 19, top part). This might potentially remove much information from the image. Also, a large amount of output intensities might be dedicated to a range which contains few pixels.

Both these problems are solved in histogram equalized mapping. Here the amount of output intensities for a given range is depending on how many pixels have values in the range.

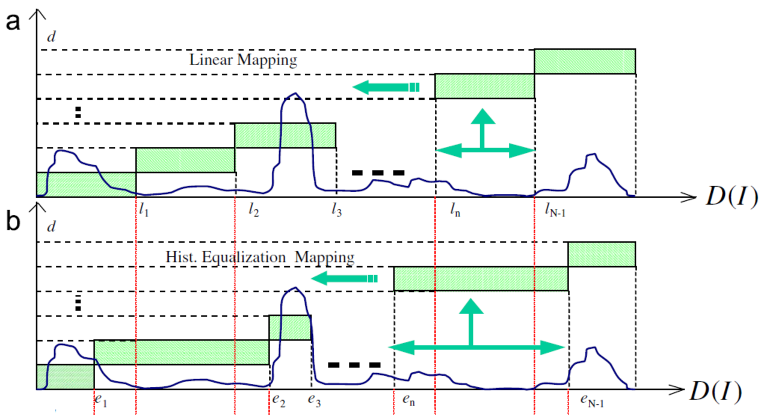

Figure 19: *Graphs showing the relationship between (a) linear and (b) histogram equalized mapping. Blue line is the pixel distribution. Horizontal axis is the range of the original image, while the vertical axis indicates the new range. Each rectangular green block indicate one new level, and all horizontal values covered by one green block is mapped to the same new level. (Figure from [39]).*

Fast tone mapping for HDR images

Jiang Duan and Guoping Qiu present in their paper [40] a fast global tone mapping function. This algorithm is a good example of how global tone mapping functions are dependent on histograms. Here they combine linear mapping with histogram equalized mapping to get the best from both worlds.

A histogram over all the luminance values of the image is first made. This histogram is then used to divide the possible new levels. Here, a combination of histogram and linear mapping will be done, depending on one parameter. This parameter can vary between 0 and 1, where the extreme values are linear mapping and histogram equalized mapping. Values in between will mix the two mappings. The authors present 0,5 as a good value for a variety of pictures, but this can be changed depending on the scene.

Qiu et al. [41] present a slightly different solution to this algorithm. The most interesting part here is how they divide the histogram. Here they assume that with linear mapping of the histogram each interval contain a known number of pixels. By exploiting this they get an equation, which indicates how many pixels each intervall contain. An iterative process to count through the histogram from each direction to find the final curve could then be done.

Multicurve

In a later paper, Qiu et al.[42], presented a way to modify the algorithm to incorporate local effects. Here the parameter that can be changed is transformed to a parameter depending on the the average luminance of a given block. They also propose a further improvement to their original luminance function. By varying the base of the logarithmic operation they manage to get high luminance values to be mapped to a wider display range. This will give the image similar contrasts regardless of the brightness of the scene.

Image Characteristic oriented

Chung et al.[43] removed the parameter all together. They present an algorithm which by nature is quite similar to Duan and Qius[40] algorithm. They note that after tone mapping with Duan and Qius algorithm and similar algorithms, the shape of the histogram will be slightly changed. The image will therefore appear different.

$$F_{std} = \sqrt{log10(8 + (Lmax_g - Lmin_g)/Diff_{mm}} \qquad (3)$$

To counter this, the authors propose an algorithm which will divide the output values based on the distribution of pixels in the histogram. Equation 3 is used to calculate what the authors call the population deflection. This indicates if the populating between $Lmax_g$ and $Lmin_g$ is concentrated.

This is combined with the standard deviation to see if the population is either concentrated or not. If it is, the range is not divided any more. The part will be divided into three new parts if the population of the part is not concentrated. Each part is given the same amount of output codings, so simple linear division is used inside each part to divide these evenly.

The last algorithm by Chung et al. is interesting because of the lack of parameters. The calculations of the standard deviation make it more computationally expensive than the algorithm presented by Duan and Qiu, but the authors claim better results when comparing the two.

3.2.3 Adaptive logarithmic tone mapping

Drago et al. [44] present a different global algorithm, called adaptive tone mapping for HDR images. The motivation is autonomous interactive use which demands a fast tone mapper.

The main part of this algorithm is its varying logarithm. If an image is mapped with a low base logarithm, like base 2, it will in general appear bright. The same image mapped with a higher base logarithm will appear darker, and will therefore reveal more details in very bright areas.

$$L_d = \frac{L_{dmax} * 0.01}{\log_{10}(L_{wmax}+1)} \frac{log(L_w + 1)}{log(2 + ((\frac{L_w}{L_{wmax}})^{\frac{log(b)}{log(0.5)}}) * 8)} \tag{4}$$

Equation 4 shows the main equation in the algorithm. B is the varying parameter and L_{dmax} is a scalefactor which is used to adapt the output to a given display. L_w is the average luminance, known as the world luminance for that scene.

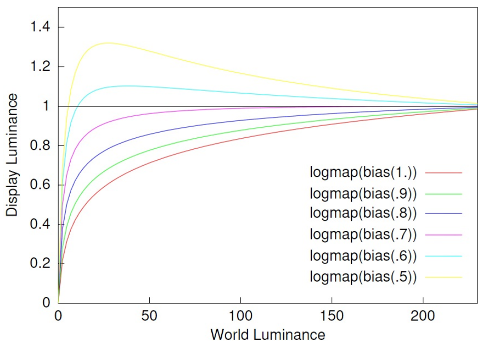

Figure 20: *Graph showing the resulting tone mapping curve from equation 4 while varying the bias value. The maximum displayable value is 1 so any values above this is clamped. Figure from [44]*

Figure 20 show how the varying bias will influence the tone mapping function. An interesting point is how similar it is to the gamma function. This tone mapper can therefore be seen as an adaptive gamma function. The problem with this algorithm is its complexity, since it includes divisions, multiplications and exponentials with varying decimal values as seen in equation 4.

3.2.4 Adaptive tone mapping

Cvetkovic et al. [45] developed a tone mapper for high dynamic range video recorders. Their goal was to design an adaptive gradation correction, which would enhance image details in dark regions while not compressing the bright regions. They base this method on a combination of the gamma function and a knee function. The knee function, as seen in figure 21, will expand the dark area relative to the bright area. This will reveal more details in the dark area.

The algorithm is a user controlled algorithm for HDR cameras. In these, the user sets a wanted video level. By using feedback, as seen in Figure 22, the parameters of the curve are gradually changed until the wanted video level is reached. This is measured by the measure and control unit which makes an histogram of the tone mapped values. The authors define the metric which the wanted video level is to be measured against as the median of the histogram.

To only enhance interesting details (decided by the wanted video level mentioned) neighboring pixels is only changed if they differ above a certain threshold. This gives the system certain local characteristics.

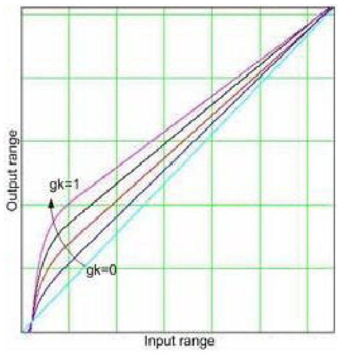

Figure 21: *Different knee functions. Figure 2 from [45].*

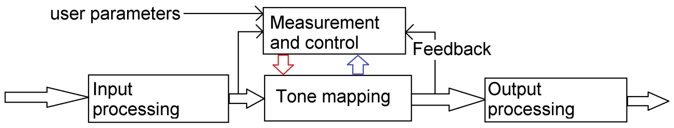

Figure 22: *System overview of how the feedback works. The mapping will be dependent on the the parameters from the measurement unit. The parameters will be changed until the wanted video level is reached. Based on figure 4 from [45].*

3.3 Local tone mapping algorithms

Using a local tone mapping function on an image will result in a different image compared with the image using a global function. The reason for this is that the local algorithms take into account the neighborhood of each pixel when tone mapping it, thus the name local. Two pixels with equal values placed in different parts of the image may therefore be enhanced in slightly different ways, depending on the surrounding pixels. This may reveal, or preserve, more details compared with global algorithms.

Land and Mccann were some of the first people to study how humans perceive colors. They developed the Retinex theory [46] which describes how the appearance of a point is affected by the surroundings. All local algorithms can be said to descend from the retinex theory.

An example can be seen in in Figure 23. Here, the colors of the gray squares in the middle will for some people appear different. This is because the eye use the surroundings of each square to decide how the gray square appears. The actual color values are identical.

Many types of local tone mapping algorithms exist. Common is that they use filtering for each pixel to make it dependent on the surrounding pixels. A thorough presentation of some of these can be found in [8].

Figure 23: *Optical illusion. The two gray boxes have the exact same color, but for many people they appear different. This is because of how the eye adapts to the surroundings of a point.*

HVS matching

One type of algorithms try to emulate how the human visual system (HVS) behave. If the algorithm emulates the eye correctly when reducing the image, the LDR image should appear the same as if the original scene was observed. This will theoretically work if the HDR image contains all the information of the scene. The problem is that all the details on how the eye work is not known. Also, showing the resulting LDR image is dependent on a monitor which is not able to display as much light as the natural scene.

The Visibility matching tone mapper proposed by Ward Larson et. al [47] and the perceptual tone mapping by Krawczyk [48] are examples of this. Effects such as glare and how the eye adapt to changing light conditions are incorporated here. Using these algorithms will produce LDR images which should be close to how we would see the scene.

Photographic techniques

Other approaches to tone mapping use photographic techniques. Photographic Tone mapper [4] by Reinhardt et. al does this, by basing it on the zone system by Adams. The zone system was used by photographers to produce visually pleasing images. The photographers also used a dodge and burn technique to vary how bright different parts of the image were. Reinhardt used Gaussian kernels (Figure 24) of different sizes to do an automatic dodge and burn operation. This algorithm is known to produce images with a very high quality and performs consistently well when compared to other algorithms.

Filter kernels

Another approach is more technical. Tumblins LCIS [49], Duan and Dorseys fast bilateral filtering [50] and gradient domain by Fattal et. al [51] base their algorithms on filter operations. Instead of including HVS they do edge detection to maintain edges in the image before compressing the dynamic range. This allows them to bring out a great deal of details, but not necessarily a correct scene as we would see it. Indeed, Tumblin even mention that the number of details revealed in the

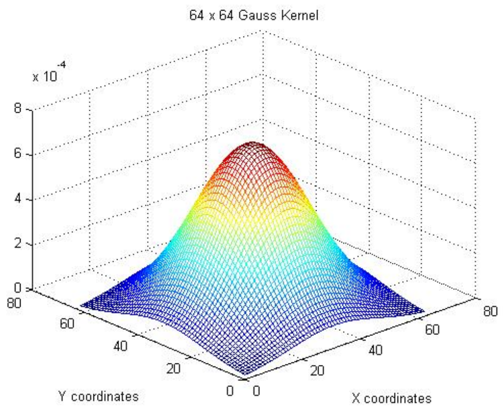

Figure 24: *The gauss kernel used by Reinhardt et. al in the Photographic Tone mapper [4]. This tone mapper uses eight kernels for each pixel with sizes up too eight by eight.*

image by the algorithm might make it more suitable for applications where the number of details are more important.

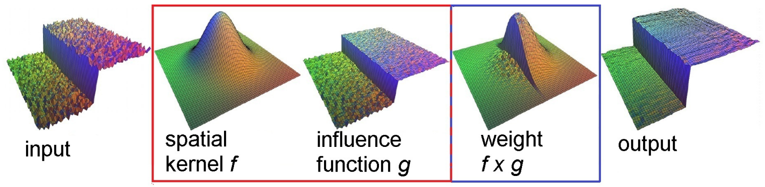

Figure 25: *Illustration on the filtering process done in the Fast bilateral filtering for HDR images. Figure from [50].*

Figure 25 shows the filtering process in the fast bilateral filtering algorithm. This is done on every pixel, making it a very computationally expensive filtering process.

Square kernels using global functions

A sub group of local tone mappers are based on running global operators on different segments of an image, giving them local characteristics. Duan et. al [39] use a simple histogram based tone mapper to tone map each pixels, but limit the histogram to just cover an area around the pixel instead of the complete image. Figure 26 shows how this will result in different curves for different pixel. This requires the global operator to be run on every single pixel, increasing the computational requirements a lot. Kim et. al [52] and suggested ways to reduce this, without increasing blocking effects.

Using a filter can lead to blocking effects, also known as halo effects. These effects arise from the sampled region used for the local property. If there is a sharp

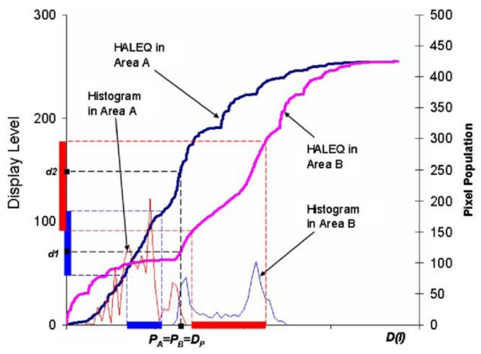

Figure 26: *Illustration on how different segments of the image will result in different tone mapping curves. Figure from [39].*

border between two regions and a large difference in values from one side to the other, the effect will arise. This can typically be seen around bright objects in the scene.

3.4 Color handling in tone mapping

Linear and non linear transformation

Tone mapping can be done on all three color channels separate [53], but these rely on mathematical equations which is difficult to control. Modern tone mappers work on the luminance, or intensity range, for the image. The reason for this is that changes in the intensity domain is reflected on all three color channels

The first step for a tone mapper is to calculate the gray scale image, also known as the intensity image. This is done with equation 5.

$$L_{in}(x,y) = 0.299 * R(x,y) + 0.587 * G(x,y) + 0.114 * B(x,y) \tag{5}$$

The original factors shown here were first used by Schlick [54]. They can be traced back too the CIE 1931 colorimetric standard [55]. The factors vary slightly between authors [40, 4, 56].

The problem which arises from working directly on the luminance is how to go back to the RGB space while preserving the colors. The most common way is by the equation 6 introduced by Schlick [54] in 1994. The proof of this can be found in appendix A.

$$C_{out}(x,y) = \frac{C_{in}(x,y)}{L_{in}(x,y)} * L_{out}(x,y) \tag{6}$$

$$\begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} = \begin{bmatrix} R \\ G \\ B \end{bmatrix} * \frac{Y'}{Y} \tag{7}$$

Another way of writing equation 6 is equation 7. Here Y' indicate the modified luminance, while Y is the original luminance.

The quantity Y'/Y is problematic, but impossible to circumvent with this color transform. Using other color spaces is also possible, but these will in general be even more complex. Akyuz and Reinhard [57] discussed color appearance in high dynamic range images. They discussed how using camera appearance models (CAM) will improve the appearance of HDR images. A problem is that modern CAMs are complex and require many parameters. The complexity in both parameters and transforms make these unrealistic for a real time implementation.

3.5 Exploiting temporal properties

Limiting the real time video to video conferences gives one very important advantage; the video stream can be assumed to have a very high measure of temporal correlation. This means that consecutive frames will be very similar. A general way of exploiting this characteristic is pointed out by Coria et al.[58]. In their paper, the authors introduce a concept they call Group Of Pictures, GOP.

Local tone mapping is only done on the first frame. This frame will act as the reference for the rest of the group. Motion estimation techniques are used between the next frames to estimate the movement of the blocks. If an exact or similar block is found, the information from the corresponding tone mapped area in the reference frame is used. If no similar block is found, a scaled down version of the image algorithm could be run. This is a way of reducing how many times the local tone mapper will need to be computed.

Using this approach to tone map a video stream gives the designer possibilities of using separate algorithms for the frames.

4 Evaluation and choice of tone mapper

Real time video conferences have some special challenges. To get a natural flow in a meeting, the delay has to be minimal. The natural stops in discussions where different people talk is difficult to maintain if the delay gets large. This is because people are used to one amount of silence before talking. This will change in a video conference. The main sources of delay are transmission delay, but processing delay for the video stream will also be present. Careful selection of processing algorithms will minimize this delay.

Another problem is control of parameters. In a meeting with many people the meeting should be in focus, not tuning the parameters to get a good image. This would also break the natural rhythm of a meeting if changing light conditions would require that the parameters would change. Ideally, the system itself would be fully automatic requiring no parameters.

Demosaicing and denoising are just some of the low level processing, as discussed in chapter 2, which is done in the camera. There are many ways of implementing algorithms close to the sensor. One way is to make an Application Specific Integrated Circuit (ASIC) which contains all the processing elements. Another way is to use a Field Programmable Gate Array (FPGA). The choices between these are a question of cost and development time. If the camera is going into large scale production, ASIC might be a good choice since the cost of development will be small when spread out on a large number of units. The problem with ASIC is that it is not possible to change the implemented algorithms later. FPGAs on the other hand, are very suitable for changing or improving the algorithms.

The evaluation and choice of tone mapper will be made with an implementation on an FPGA in mind. This allows great flexibility, but limits how much memory the algorithm can use. HDR video have been discussed (chapter 2.3, but is outside of the scope of this report to implement. The video stream will therefore be assumed to be a normal LDR video stream. The difference from HDR is that LDR might have problems with very bright or very dark scenes, due to problematic exposure. Using tone mapping on LDR content will change the brightness of the image, solving some of these problems.

This chapter will evaluate and make a decision on which tone mappers to continue to work with. The tone mappers chosen here will be based on the feasibility of implementation and the constraints discussed here.

A detailed discussion of the selected algorithm will follow this. A system to reduce the algorithm and other similar algorithms will be presented. This will result in a reduced system which better fits the limitations of an FPGA. This system is implemented in chapter 6.

4.1 Algorithm decision

HDR images are scene referred, and contain much more information than it is possible to display on a normal LDR monitor. Just linearly scaling the dynamic range of HDR images to a lower range is possible, but much information will be lost. Typically, photographers making HDR images have done much post processing to get a very special look on the images. This can either be done photography editing software which has implemented a tone mapping algorithm.

Tone mapping algorithms are designed for reducing the HDR images to LDR. This allows the images to be displayed on a normal LDR monitor. By using tone mapping on HDR images the resulting image should be closer to what we would have seen if we were looking at the same scene. The algorithms can, as discussed in chapter 3.5, also be used on video.

4.1.1 Local Algorithms

The main problem with local algorithms is how complex they are. These rely on a series of mathematical operations for each pixel together with a spatial filter which makes them local. Real time implementations of both Reinhardts photographic [59] and Fattals Gradient domain [60] tone mappers have been done, but the implementation require large amounts of both memory and hardware.

Figure 27 show a 5 by 5 filter kernel. The problem with this is that when the lines gets large, the amount of buffering for filtering a single pixel also get very large. For the filter in the figure, a total of 4 lines and 5 pixels need to be buffered

(i-2, j-2) 1	(i-1, j-2) 2	(i, j-2) 3	(i+1, j-2) 4	(i+2, j-2) 5
(i-2, j-1) k + 1	(i-1, j-1) k + 2	(i, j-1) k + 3	(i+1,j-1) k + 4	(i+2,j-1) k + 5
(i-2, j) 2k + 1	(i-1,j) 2k + 2	(i,j) 2k + 3	(i+1, j) 2k + 4	(i+2, j) 2k + 5
(i-2, j+1) 3k + 1	(i-1, j+1) 3k + 2	(i, j+1) 3k + 3	(i+1, j+1) 3k + 4	(i+2, j+1) 3k + 5
(i-2, j+2) 4k + 1	(i-1, j+2) 4k + 2	(i, j+2) 4k + 3	(i+1, j+2) 4k + 4	(i+2, j+2) 4k + 5

Figure 27: *A 5 by 5 filter kernel. The black indices indicate positions relative to the center pixel, while the red letters indicate the number of the pixel relative to the first pixel in the filter kernel. K is the length of one line.*

for one kernel. For the next pixel, pixel (i+1, j) most of the buffer can be reused. If the filter kernels required become larger, even larger buffers are required.

This problem comes from the fact that the algorithms are designed for post processing in software. The time used for calculating a complex mathematical function for each pixel is therefore not a concern. The buffering seen in figure 27 is not a problem either, since the whole image is already available.

Because of this, local algorithms will not be used for implementation. However, Reinhardts Photographic tone mapper will be used for simulation purposes to compare with the other algorithms. A description of the algorithm can be found in appendix B.

4.1.2 Global algorithms

Global algorithms are different from local algorithms in that they do not use any spatial filters. By not using this they loose the local properties and become global instead. This makes them more feasible for a realistic real time implementation, since the amount of memory required is greatly reduced.

$$Y_{out} = TM(Y_{in}) \tag{8}$$

Equation 8 show the fundamental equation for global tone mappers. The algorithms use a mathematical function on each pixel. The new values can in principle be pre calculated, since a mathematical operation is used on all pixels. Pixels with the same value will be mapped to the same new value.

Some algorithms, like adaptive logarithmic tone mapping (chapter 3.2.3), only rely on one equation. This equation is controlled by a series of parameters, which makes it very advanced.

Another series of algorithms are based on histograms (chapter 3.2.2). The algorithms use a histogram over the image to form a curve which the values should be mapped with. The different algorithms vary in how they do this and how many parameters they require. One algorithm by Chung (chapter 3.2.2) relies on statistical properties of the histogram. No parameters are required, but the complexity behind the calculation rise.

Other algorithms also work on the histogram, but use a parameter to control how the algorithm calculate the curve. The fast tone mapping by Duan and Qiu (chapter 3.2.2) is a good example of this. This algorithm relies on simple operations, but requires one parameter. Variants of this which require less operations have also been presented (chapter 3.2.2).

Delay

To calculate a curve for a frame, the whole frame is required. This is problematic for a real time implementation, since it will incur a frame of delay. This can be circumvented by using an older, precalculated, curve on the new frame. This is similar to the temporal filtering described in chapter 3.5. By using a precalculated curve, the tone mapping of each pixel can be done very fast. The idea can be seen in Figure 28.

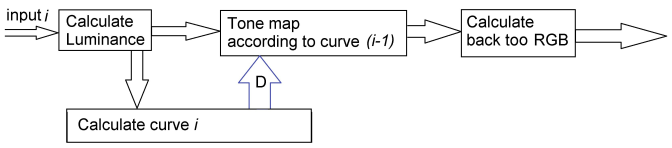

Figure 28: *Showing how an old curve can be used to tone map a frame, while the frame is used to calculate a curve which is to be used later.*

Ideally, the quantity which the tone mapper are relying on will have thresholds which prevent it from changing too much between each frame. Krawczyk [48] discussed how the human eye uses temporal adaption to adapt to changing light condition. Based on this, two different thresholds can be used. One will prevent the image from getting bright to fast, while the other will prevent the image from getting dark to fast. Changes in the tone mapping should then appear more natural.

The chosen algorithm will be the one proposed by Duan and Qiu, Fast Tone mapping. This rely on simple equations which has some interesting properties by mixing histogram equalized mapping and linear mapping.

$$[L_{min}, C_{1,0}], [C_{1,0}, C_0], [C_0, C_{0,1}], [C_{0,1}, L_{max}] \tag{9}$$

Equation 9 is a small example on how one curve, or set of values, is defined. Here the original range between L_{min} and L_{max} is remapped to four distinct values. Values between L_{min} and $C_{1,0}$ are mapped to the same value. Values in range $C_{1,0}$ too C_0 are mapped to another value and so on. This can also be an example on how a set of precomputedd values are used.

There is one problem with precalculating a curve and that is that it has to be stored. Equation 9 together with Figure 29 is an example of this. Here there are only four available ranges, so it require three distinct points to store how the ranges are divided, $C_{1,0}$, C_0 and $C_{0,1}$. How much memory it requires is depending

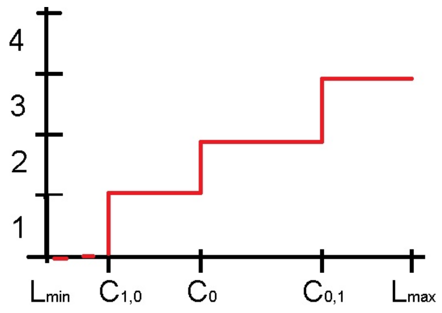

Figure 29: *Equation 9 can define this curve. Horizontal axis indicates incoming value, while vertical axis indicates outgoing values. Value between for example $C_{1,0}$ and C_0 are mapped to the same value.*

on how many points which are used and how the implementation is done. All the algorithms which use a histogram discussed in chapter 3.2.2 divide the available output codings into 256 different values, requiring 255 distinct points to be defined.

4.2 Selected algorithm

The algorithm proposed by Duan and Qiu (chapter 3.2.2) is based on a histogram over the intensities in the image. The goal of this algorithm is to divide the histogram into 256 different bins which each indicate one output level. An example of this can be seen Figure 29. In this figure, only four output intensities are defined. The size of each bin is defined by a parameter, α which vary from 0 to 1. When it is 0, the mapping is linear, while it is 1 it result in a histogram equalized mapping (see chapter 3.2.2).

$$C_0 = \frac{L_{max} + L_{min}}{2} + \alpha(\beta_0 - \frac{L_{max} + L_{min}}{2}) \tag{10}$$

Equation 10 shows the equation with parameters for the first iteration of the image. Here β_0 indicate the value which divide the population in the sub histogram between L_{max} and L_{min} in two equal parts. L_{max} and L_{min} is the maximum and minimum values of the sub histogram. For the first iteration, these correspond to the maximum possible and minimum possible value of the histogram.

$$[L_{min}, C_0], [C_0, L_{max}] \tag{11}$$

Equation 11 shows how the curve is defined after the first iteration. Now, two new sub histograms can be defined. One is between L_{min} and C_0 and the other is between C_0 and L_{max}. If the algorithm were to just divide the output range into two, values which fall inside one of these sub histograms would be mapped to the same value.

$$C_{1,0} = \frac{C_0 + L_{min}}{2} + \alpha(\beta_{1,0} - \frac{C_0 + L_{min}}{2}) \tag{12}$$

$$C_{0,1} = \frac{L_{max} + C_0}{2} + \alpha(\beta_{0,1} - \frac{L_{max} + C_0}{2}) \qquad (13)$$

Equation 12 and 13 show the next iteration of the algorithm. Here equation 12 is dividing the sub histogram between L_{min} and C_0. $\beta_{1,0}$ is now defining the value which split this sub histogram in two equal parts. Equation 13 is doing the same thing for the sub histogram from C_0 and L_{max}.

$$[L_{min}, C_{1,0}], [C_{1,0}, C_0], [C_0, C_{0,1}], [C_{0,1}, L_{max}] \qquad (14)$$

Equation 14 shows how the curve is stored after the first three iterations. The algorithm continues until it has divided the histogram into 256 different parts. Each range will then correspond to one output intensity.

The algorithm has some interesting properties. It is a mix between histogram equalization and linear mapping. Both of these are interesting. Histogram equalization will reveal details which might get lost in linear mapping.

The problem with this algorithm is that it is quite complex to calculate the new curve. In their paper, they run equation 12 255 times to find all the values which form the curve. β will vary for each iteration depending on the result from the previous iteration and this is the biggest problem with this algorithm.

Two more problems are related to the curve itself. Equation 14 shows how it is defined with four distinct values. For each incoming pixel, a search would be required to find which bin it is related too. One way would be to do implement a binary search. Another would be to realize the curve as a look up table. Using the incoming values as indexes, the associate new output value could be stored for very fast retrieval. The size of the incoming values and the number of possible output value would decide how large this table would be required to be.

The second problem with using a curve is how to implement limits to temporal changes. The curve should only gradually change from one frame to another. If the curve changes too much from one frame to another, the appearance of the image would also change. If the curve first increases before decreasing, flickering (large intensity changes) will appear. Therefore, it is essential that a real time tone mapper have some sort of limitations on how much the curve can change on a per frame basis.

4.3 Proposed system for implementation of algorithm

One possible solution is to combine aspects from the different algorithms presented in chapter 3.2. The goal is to design a tone mapper which would be more suitable for a real time hardware implementation. The problems require solutions are:

- Few parameters
- Versatile for different situations
- Suitable for limiting temporal variations
- Suitable for an hardware implementation

In chapter 3.2.2, an automatic algorithm proposed by Liu et. al were discussed. The algorithm itself is quite similar to the one described by Duan and Qiu (discussed in chapter 4.2 and 3.2.2). To remove the parameters, they use advanced mathematical operations to decide how large each segment can be. If one segment covers less than a given amount of the population, the curve in this segment will be linear. On the other hand, if it contains more than this amount, the segment will be divided into smaller segments.

Qiu et. al (chapter 3.2.2) also suggested mathematical equations which were used to divide the histogram. By using these, they reduced the time it would take for the curve to be defined. Assumptions on how much population each range would cover gave them the equations.

The two ideas, to set a minimum amount of population for a given range and use linear mapping for this range together with equations to find the range, is very interesting. This can be used as a foundation for a tone mapping system, which will use a limited number of lines, or segments to define the tone mapping curve. Each segment can be reduced to equation 15, the equation for a linear graph.

$$Y = aX + b \tag{15}$$

To define the curve, a set of coefficients for each segment will be enough. In addition, each segment would require a minimum and maximum value to limit the extent of the segment. This greatly reduces the memory required for storing the curve if the number of segments is kept low. This makes it more realistic and practical to implement. The equation itself is suitable for an FPGA implementation.

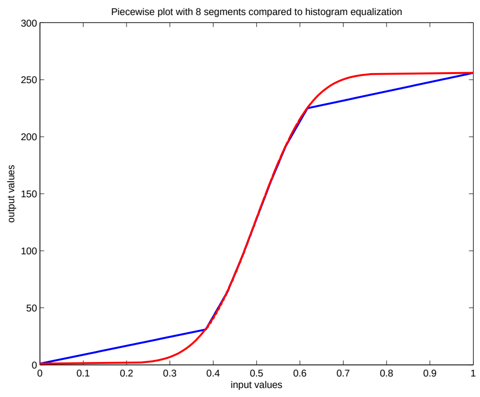

Figure 30: *Graph comparing piecewise curve (red) with full histogram equalization (blue) for a population with normal distribution. Blue curve practically use 255 segments, while red curve use three.*

Figure 30 shows a comparison of two different curves. Blue curve shows a full curve for histogram equalization. This curve use 255 different bins, or segments, which correspond to possible output intensities for an arbitrary situation. The red curve only uses 8 segments. Each of these segments has been defined to cover 12%

each of the population. This is the same as histogram equalization, since each segment in histogram equalization also cover a given amount of population. In this case, each output value from the histogram equalized curve will cover 1/255 % of the population. Because of this, each segment of the reduced curve will start and end on the histogram equalized curve. For example, the first segment, which covers 12% of the population, will start at 0 and end at the point on the curve of the histogram equalized which also covers 12%. This is at output value 30.6 in this curve.

This can be also be demonstrated by extending the number of pieces and comparing the resulting curves to a curve from histogram equalization, as seen in Figure 31. Each segment of the proposed algorithm will start and stop on the proposed curve if the reduced curve is based on the same algorithm.

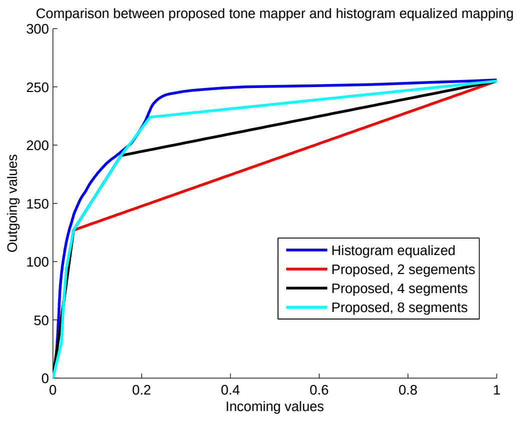

Figure 31: *Graph showing a comparison between histogram equalization and reduced versions with varying number of segments. Each segment covers the same amount of population. Based on histogram of sofa image discussed in section 5.3.*

Figure 31 shows how reducing the curve will lead to a less detailed curve. This is to be expected, since the information about the curve is drastically reduced. Using another algorithm than histogram equalization is also possible. In practice, any algorithm which defines a tone mapping curve can be used instead. The algorithm would then first be run, and then reduced to a segmented version.

4.3.1 Reducing the curve

The general way of producing the reduced curve would be to first run the main algorithm, which would produce the full tone mapping curve. A defined subset of the points would be picked from the algorithm. Figure 32 shows a section of the complete curve and how it is defined by a series of points. Incoming values will be mapped to the closest point.

From the full curve, a set of seven points is required to define each segment. Figure 33 shows the seven points which will be used in the reduction. The last

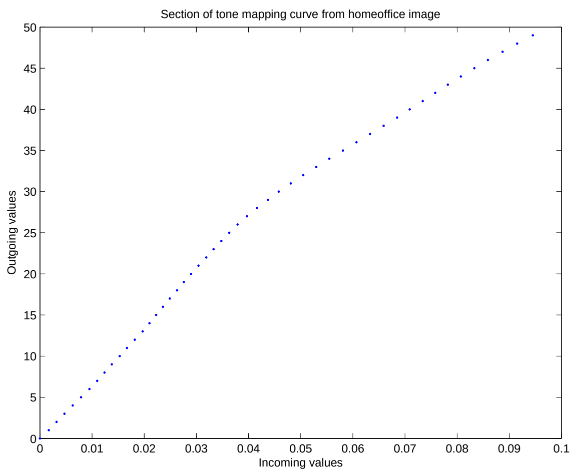

Figure 32: *Graph showing a small section of the curve used to tone map the home office image. Notice that it is essentially built of a series of points. Points located between two points will either be mapped to the higher or lower value, based on how it is implemented.*

point is simply the highest possible value. These are based on how each segment should start and end. Here, dividing the output intensities into eight parts which correspond to eight segments is used. Each segment will thus cover one eighth of the output intensities.

The next step, after receiving the seven points is to interpolate between them. Linear interpolation is the easiest way, and will result in linear segments between the points. This will also result in a curve defined by linear segments, which is advantageous for an implementation.

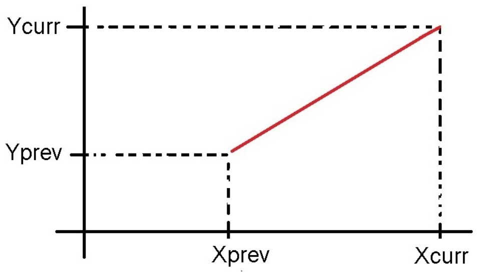

Figure 34: *Relations between the values deduced from the histogram for an arbitrary segment and settings of the tone mapper. X values are deduced from the full curve. Y values are based on settings of the reducer. Xprev is defining the end of the previous segment, while Xcurr define the limit of the current segment. Both Yprev and Ycurr are values which can be controlled. The current coefficients will define the red line.*

Figure 34 shows the basic situation before interpolating between two points. Yprev and Ycurr are values which indicate the extent of the current segment.

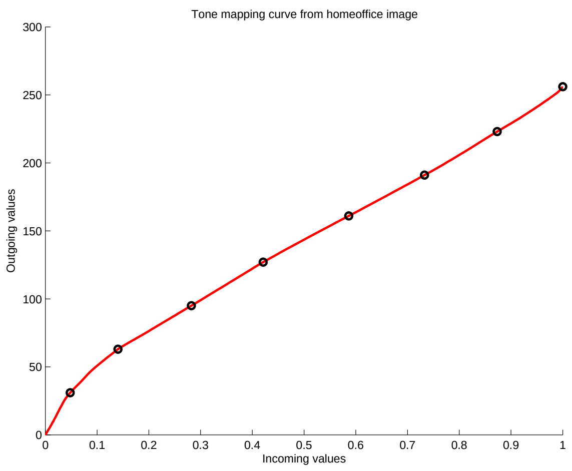

Figure 33: *Reducing a full tone mapping curve to eight segments require seven points, shown here as black circles. The points correspond to output intensities which divide the output intensities into 8 segments.*

These are defined by the settings of the tone mapper. Xprev and Xcurr are points deduced from the full curve. These indicate where the segment should start and end.

$$Y = aX + b \tag{16}$$

The values in Figure 34 are required to produce the coefficients for the current line (red in the figure). Interpolating with linear interpolation will result in equation 16 which defines the line. Yprev is b, so only a is missing from the equation. Equation 17 shows the equation which will be required to find the a paramter for each segment.

$$a = \frac{Ycurr - Yprev}{Xcurr - Xprev} \tag{17}$$

The a, b, and Xprev would then be sent to the tone mapper. Xprev will be used to denote the lowest value which each segment cover.

Using more advanced interpolating technique will produce curves which are closer to the full tone mapping curve. This would make each segment be defined by more parameters and more advanced mathematical operations. Simulations show that using linear interpolating is sufficient to produce results on par with the full curve.

4.3.2 Temporal properties

Each segment which form the tone mapping curve is defined by its relative maximum value, as seen in Figure 35. This makes it very suitable for extending the algorithm to consider temporal changes. By limiting how much the maximum of a segment can change, large changes of the curve can be prevented. Figure 35 shows two curves for different frames. Blue is the current curve, while red/purple are the next curve. Purple is the true curve, but this might result in large changes from frame to frame. Limiting the increase would return the red curve instead.

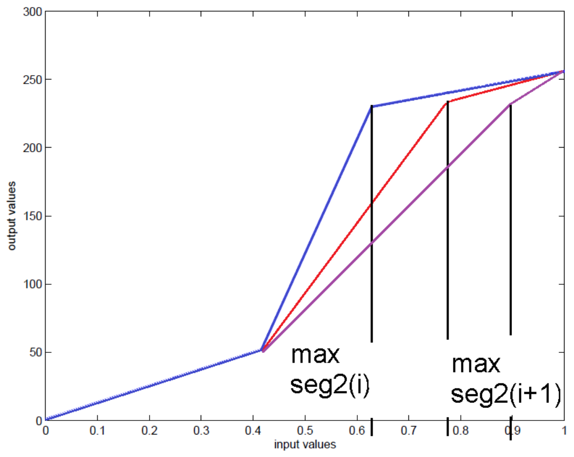

Figure 35: *Graph demonstrating how the blue curve from one frame changes to the red curve for another frame. The maximum value of segment 2 has moved to the right, resulting in a different curve. Purple shows the change which would happen if the maximum of segment 2 increase more.*

4.3.3 Extent of histogram

The histogram which this is based on will only use the middle 98% of the input intensities. As with all the histogram based algorithms, it is assumed that the histogram reflect which parts of the scene are interesting. Exluding the lowest and highest 1% will remove outliers from affecting the histogram. More of the output intensities will therefore be dedicated to the assumed interesting part of the image. If a scene has appear very bright (see section 5.1 for an example) excluding the top 1% will in remove the influence of burnt out regions. It is assumed that there will be no relevant information for this range, indicating that any interesting information will be located in other parts of the histogram. The algorithm will then aim to distribute the output intensities in such a way that the parts of the histogram with more population will get more output intensities. The same can be said for very dark images (section 5.2), where it is assumed that information below 1% of the intensities is of no interest.

5 Results and discussion

This chapter will discuss how tone mapping can improve images with difficult light conditions. Each section use one image to illustrate how changes in the tone mapper affect the result. The algorithm uses is the selected algorithm discussed in section 4.2 and the reduced version of this discussed in section 4.3.

A comparison between the global tone mapper and Reinhardt's photographic tone mapper is done in section 5.4.4. Here the best result of the tone mapping with the global algorithm will be compared to a local tone mapper.

A summary of the most important considerations when selecting and implementing a global tone mapper can be found in section 5.5. More implementation specific considerations are discussed in chapter 6.

The matlab code for generating these images can be found on the accompanying DVD.

5.1 Home office

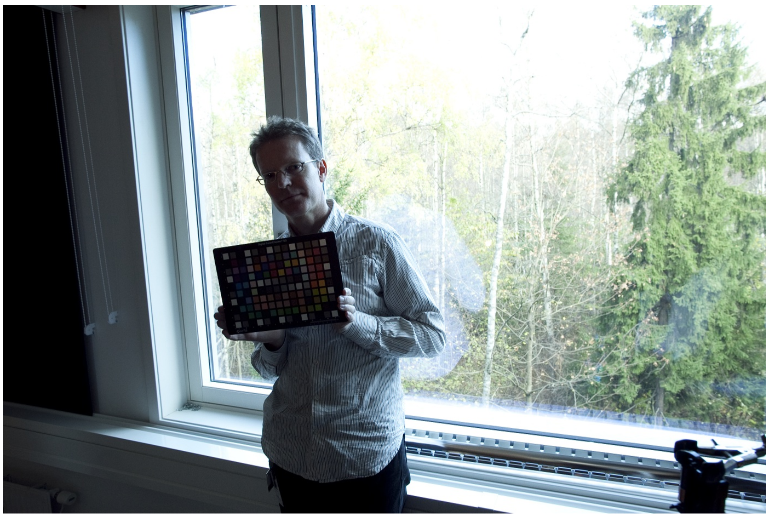

Figure 36: *Home office: Person holding up a Macbeth Color chart. This scene has a large window which is common for some offices. On a sunny day like this, the indoor part of the scene will be very dark relative to the outdoor part of the scene.*

Figure 36 shows an image called home office. This scene is showing a person holding up a Macbeth color chart in front of a large window on a sunny day. Because of the window, the person and the indoor environment appear dark. This is a typical problem which might happen in meeting rooms or offices with large windows. The goal of tone mapping in this case will be to make the indoor area appear brighter.

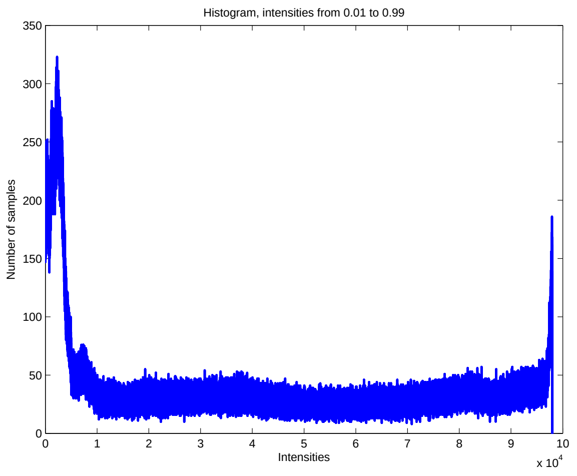

Figure 37: *Histogram for home office. This histogram disregards 1 % of the smallest and 1 % of the biggest values in the image. Horizontal axis indicate intensities, which correspond to values between 0.01 and 0.99*

5.1.1 Histogram

Figure 37 shows the histogram of the intensities for the home office image. 15 % of the image have intensities above 0.99 and therefore falls outsidede the histogram. Intensities above 0.99 % is assumed to be burnt out. 12 % of the total number of pixels falls below 0.01 which is the smallest value that will be counted in the histogram. Both groups are outliers, and it is assumed that there is no interesting information in this area.

By looking at the histogram, the most interesting parts of the scene appear to be in the low range where most of the remaining pixels are clustered. In the image, this corresponds to the indoor area in shadow.

5.1.2 Tone mapping curves

Figure 38 shows ten curves (red, blue and eight cyan) generated with Duan and Qius tone mapper and a black curve which is a reduced version of histogram equalization.

When alpha is increased, Duan and Qius tone mapper resemble more histogram equalization. The blue curve has a very steep increase for the low values. This will result in many possible output values given to a small range of incoming input values. This is not surprising, since a large amount of the pixels is located in the lower end of the intensity range (Figure 37). The blue curve corresponds to alpha 1 with Duan and Qius tone mapper. Decreasing alpha with steps of 0.1 will give the cyan curves. When alpha is 0 the resulting curve is equivalent to the red linear curve.

The black curve is the resulting curve from reducing histogram equalization to segments covering 10 %, 80 % and 10 % of the population in the histogram. The

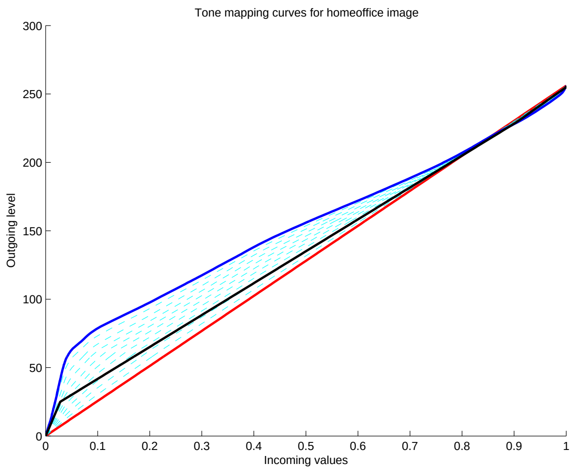

Figure 38: *Tone mapping curves for home office. The red lines indicate linear mapping, while the blue line indicate histogram equalized mapping. Horizontal axis indicate incoming values, while the vertical axis indicate outgoing values. The black curve is generated by reducing the histogram equalized curve, while the others are from Duan and Qius fast tone mapper with varying parameters.*

first segment is quite steep, approximating Duan and Qius tone mapper, while the rest of the curve is linear. The segment covering 80% ends at outgoing level 200. From here, all the curves from Duan and Qius tone mapper are almost equal.

5.1.3 Resulting images

Figure 39 shows how changing the parameter of Duan and Qius algorithm will result in different focus of the scene. The lower right is tone mapped with the proposed algorithm, and it can be placed somewhere between the alpha 0 and alpha 1 image in focus.

Varying alpha more will result in images which can be placed between the two images at the top when comparing the distribution of intensities in the scene. A set of images with alpha varying from 0 to 1 in increments of 0.1 can be found on the accompanying DVD.

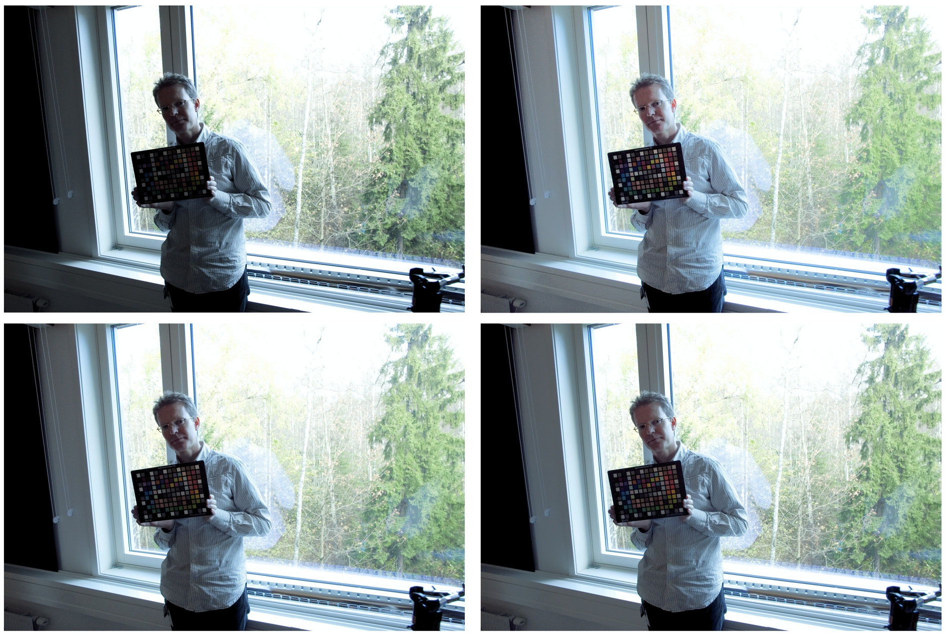

Figure 39: *Showing four different tone mapped versions. Top left, alpha 0 (equal to original). Top right, alpha 1. Lower left, alpha 0.5. Lower right, proposed algorithm.*

5.2 Student room

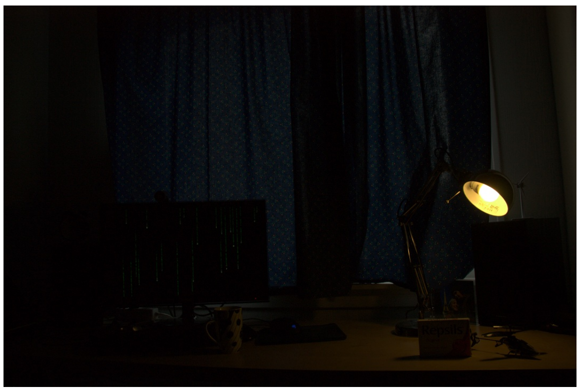

Figure 40: *Figure showing an example of a dark room.*

Figure 40 shows a very dark scene with a single light source. This is an interesting example, since it is much darker than the home office image.

5.2.1 Histogram

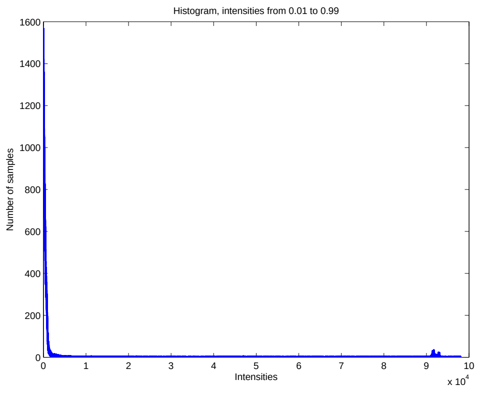

Figure 41: *Histogram for student room.*

Figure 41 shows the histogram for the student room. The histogram show how dark the image actually is. With outliers defined as 1% of the lowest and highest values, only 6% of the total pixels will be contained in the histogram.

5.2.2 Tone mapping curves

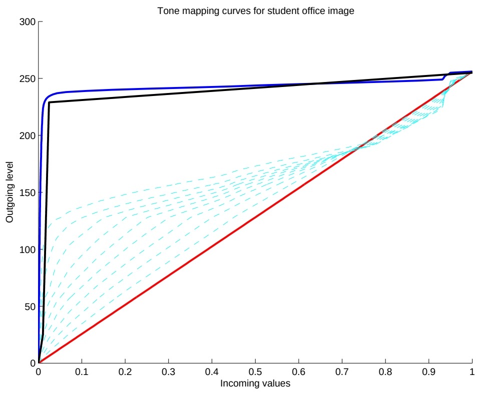

Figure 42: *Tone mapping curves for student room. The red curve is linear mapping, while the blue curve is histogram equalized mapping. Horizontal axis indicates incoming values, while the vertical axis indicates outgoing values. The black line is the curve from the reduced tone mapper, while the others are from Duan and Qius fast tone mapper with varying alpha.*

Figure 42 shows the tone mapping curves for the student room image. The spread for Duan and Qius tone mapper with different parameters is very large here. Alpha equal 1 will have a very steep curve at the start. This is not surprising when the histogram is studied. The reduced tone mapper show how each segment will start and end on the curve which it is reduced from.

5.2.3 Resulting images

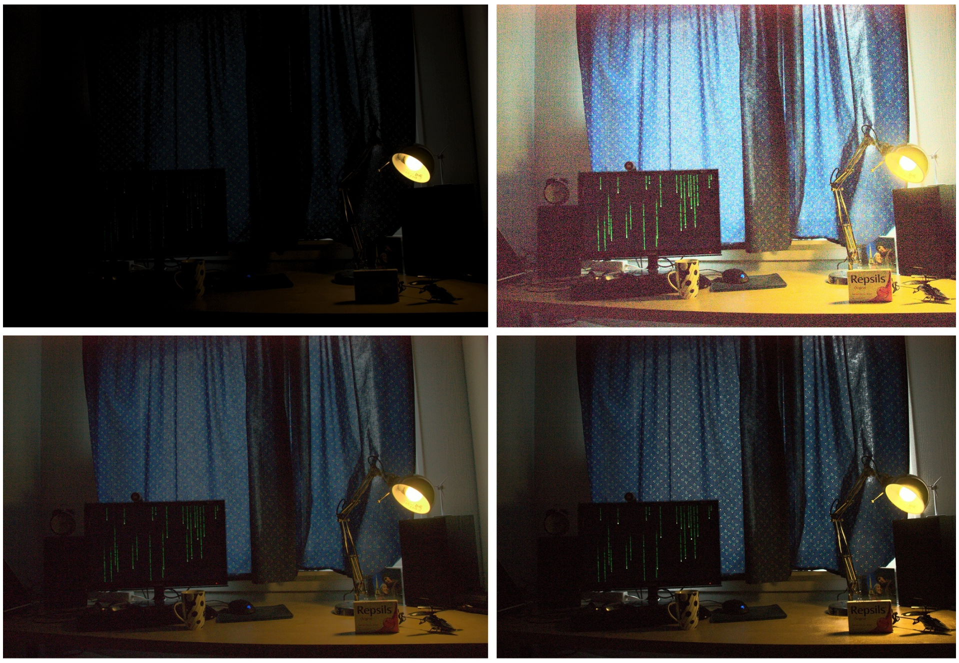

Figure 43: *Top left, alpha 0 (equal to original). Top right, alpha 1. Lower left, alpha 0.5. Lower right, reduced alpha 1 curve with three segments.*

Figure 43 shows four results from tone mapping. The raw image which was used has not been denoised. This is mainly a problem when going back to RGB and can be seen in the lower left part of the scene. The image was taken with very short exposure, which prevents the image sensor from collecting a lot of photons, resulting in a very dark image. In the dark area, there will be a lot of small variations in how many photons are collected.

In the original image the noise is not visible. This is because the noise is indistinguishable from the surroundings and disappears when the image is linearly reduced. When the image is tone mapped, the pixels are essentially multiplied by a factor. This will lead to noise being multiplied. Small variations from noise will essentially be multiplied by a factor. This results in colored dots across the image since the noise is varying across the image.

This can be countered by increasing the value which limits the influence of outliers. If this value is increased to 4%, the noise will disappear as seen in the right version of Figure 44. The reason for this however, is that the areas filled with noise are almost not changed. Figure 45 shows how changing the lower bound

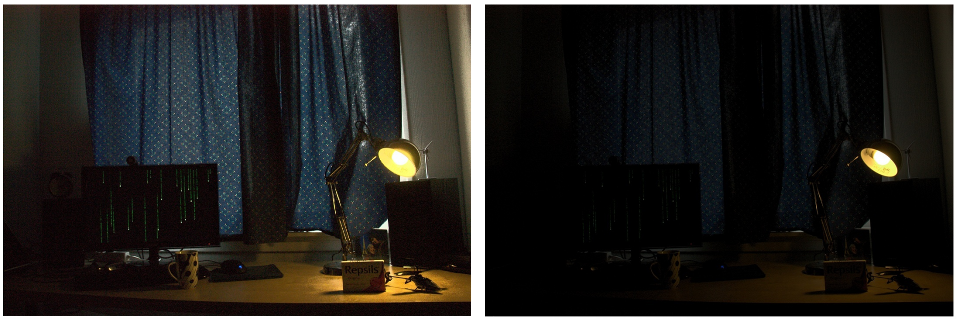

Figure 44: *Showing how changing the lower bound on outliers affect the result when tone mapping with alpha 0.5. Left image, lower limit is 0.01% of the intensities. Right image, lower limit is 0.04 % of the intensities.*

outlier affects the result of Duan and Qius with alpha 1. Noise is still more evident here, and this follows from the new tone mapping curve seen in Figure 46.

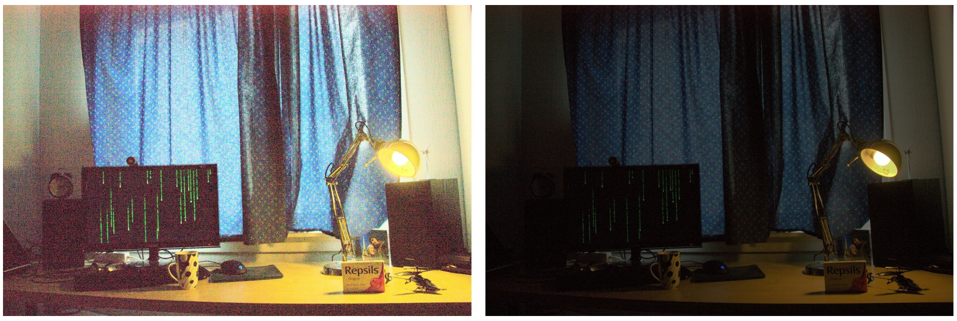

Figure 45: *Showing how changing the lower bound on outliers affect the result when using Duan and Qius tone mapper with alpha 1. Left image has lower bound 1% while right image has lower bound of 4%*

A large amount of low values in the histogram seen in Figure 41 will be removed. As a result of this, the dark areas will not have any influence on the histogram, resulting in drastic changes of the curves as seen in Figure 46.

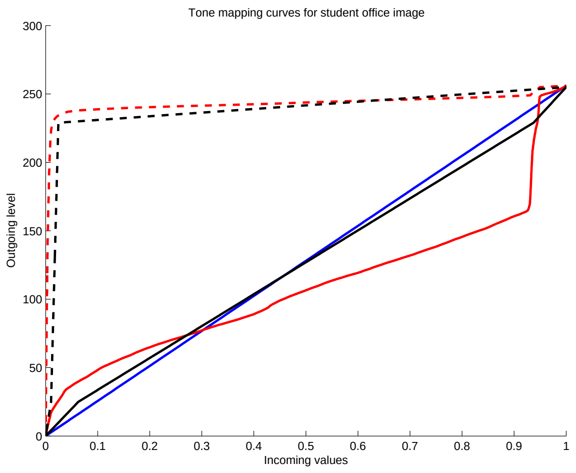

Figure 46: *Curves after limiting the histogram from 4% too 99% of input intensities. Stippled curves are curves with 1% lower bound. Red curve is for Duan and Qius tone mapper with alpha 1. Black curve is the reduced tone mapper. Blue line is linear mapping.*

5.3 Sofa

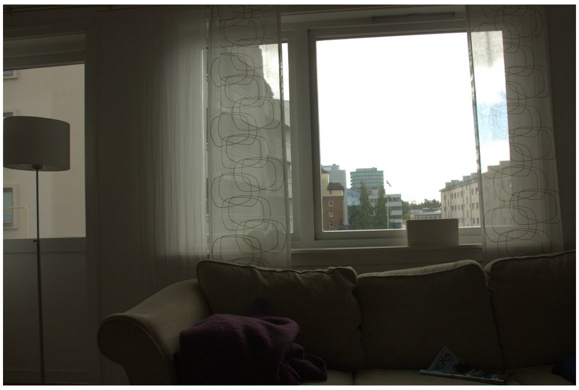

Figure 47: *Image dominated by windows on a sunny day. This makes the indoor, and the presumed interesting area, very dark.*

Figure 47 shows a scene which appear very dark. The reason for this is that it is dominated by the large outdoor window on a very bright day.

5.3.1 Histogram

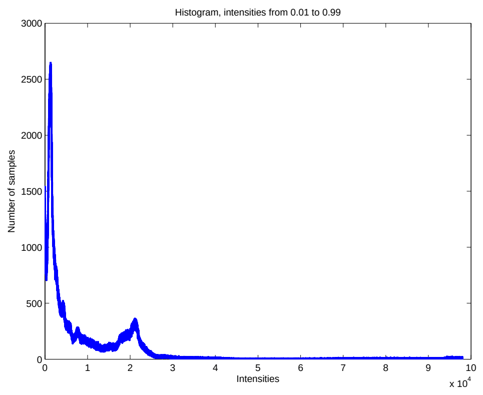

Figure 48: *Histogram for sofa image.*

Figure 48 shows the histogram for the sofa image. The bright outdoor area have no influence on the histogram, as can be seen by the very flat curve for higher values. This again will result in a curve defined by the darker area which is contained in the histogram.

5.3.2 Tone mapping curves

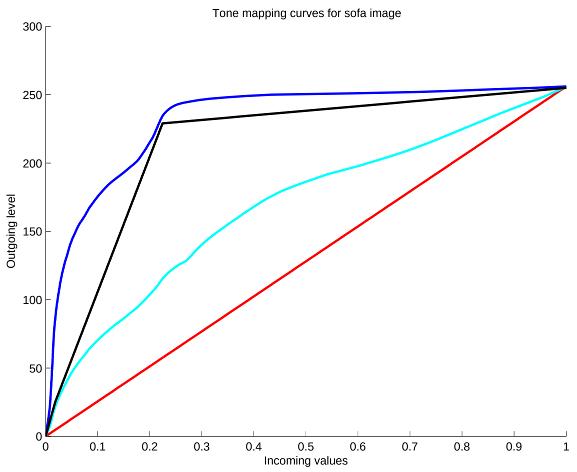

Figure 49: *Comparing linear curve (red), histogram equalized curve (blue), 3 segmented histogram equalized curve (black) and alpha 0.5 from Duan and Qius algorithm.*

Figure 49 shows a comparison of four different curves. The black curve is a reduced version of histogram equalization. It is not surprising, similar to the histogram equalization. Varying the parameter of Duan and Qius algorithm will move the cyan curve towards the linear mapping, red, or the blue histogram equalized mapping. The one shown here has alpha equal to 0.5.

5.3.3 Resulting images

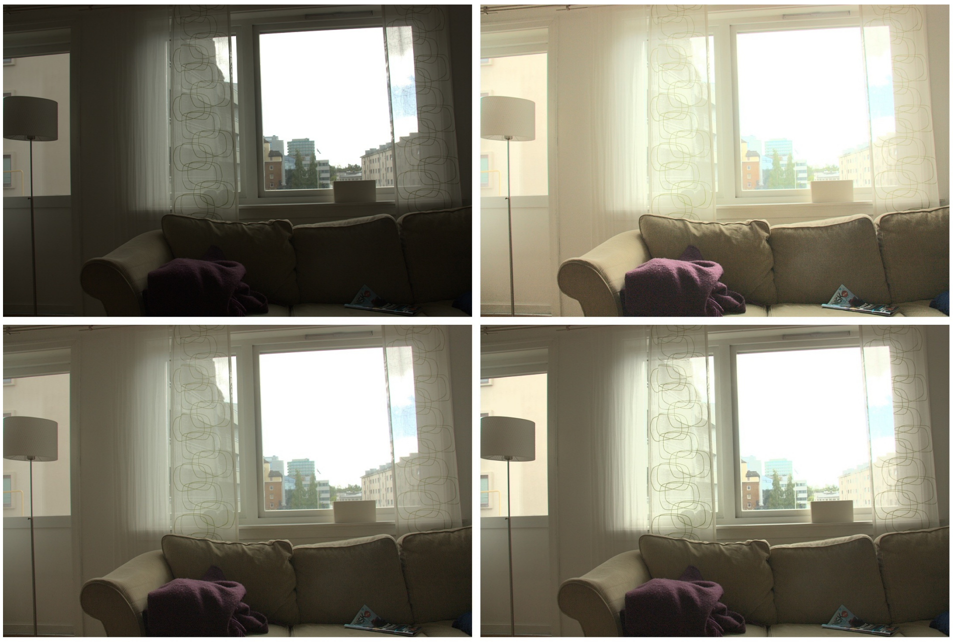

Figure 50: *Top left, alpha 0 (linear mapping equal to original). Top right, alpha 1 (histogram equalization). Lower left, alpha 0.5. Lower right, reduced histogram curve.*

Figure 50 shows a comparison of tone mapping the image with varying curves. The top left image is the original. With histogram equalization, at top right, the indoor area is much more visible. Unfortunately, the outdoor area gets washed out here. In this situation, Duan and Qius tone mapper with 0.5 will be the best solution, since the buildings in the outdoor does not get washed out. This image is a good trade off between indoor and outdoor visibility.

Changing the number of segments which the histogram equalized curve should use will not affect the outdoor result. The problem can be seen in Figure 51. Despite increasing the number of segments from three which was used in the bottom right part of Figure 50, the curve stays quite similar. This can be seen in Figure 52.

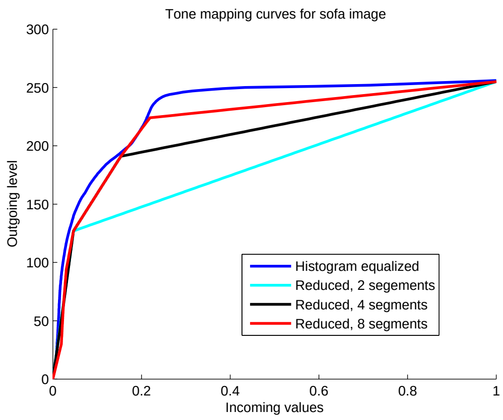

Figure 51: *Top left, alpha 0 (linear mapping equal to original). Top right, alpha 1 (histogram equalization). Lower left, alpha 0.5. Lower right, reduced histogram curve.*

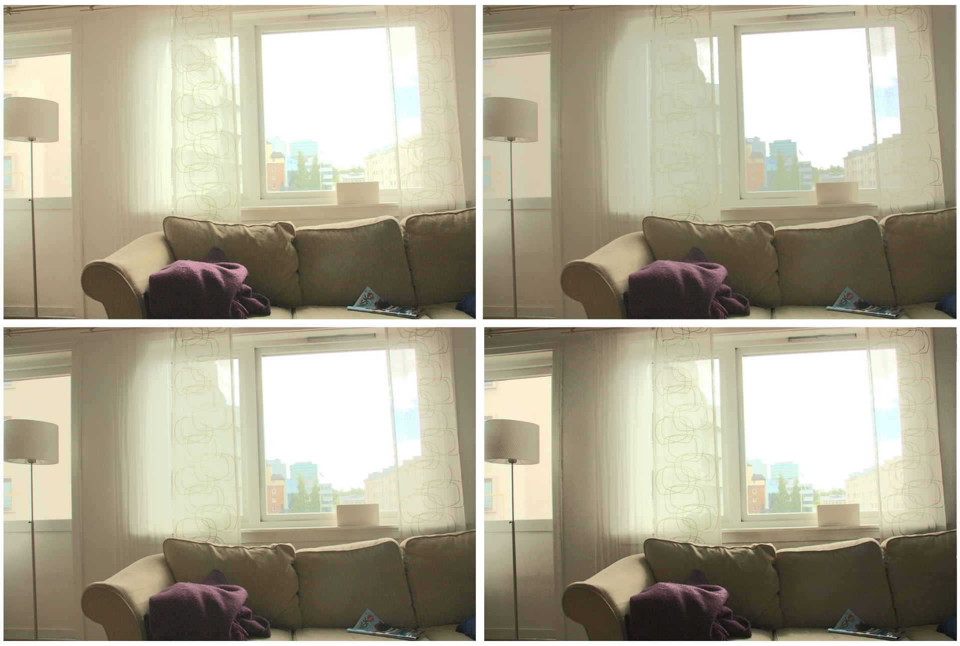

Figure 52: *Top left, alpha 1 (full histogram equalization). Top right, two segments. Lower left, four segments. Lower right, 8 segments.*

The images in Figure 52 are tone mapped with the curves from Figure 51. They are very similar, which is not surprising when examining the curves. The curves are all reduced versions of one curve.

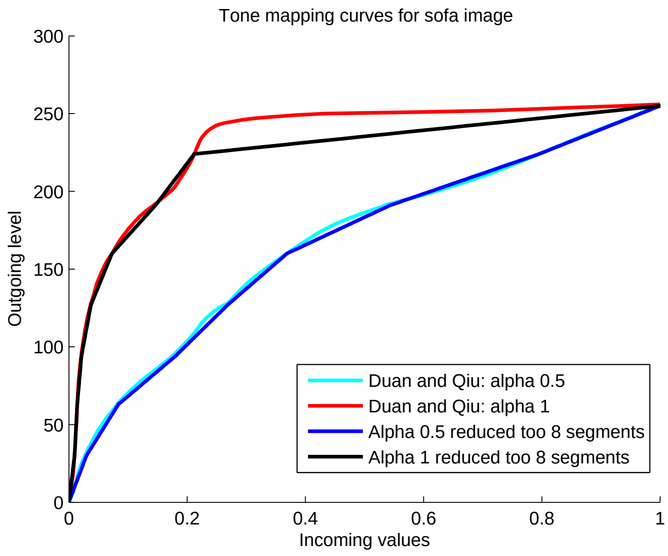

Figure 53: *Graph showing two different tone mapping curves from Duan and Qiu, and the reduced versions. Notice that the reduced alpha 1 curve (black) differ more from the alpha 1 curve (red) than the reduced alpha 0.5 curve (blue) compared to the alpha 0.5 curve (cyan). The reason for this is that alpha 0.5 is a more linear curve than the alpha 1 curve.*

Figure 53 shows a comparison of the full curves from Duan and Qius algorithm and the reduced curves with eight segments. The full alpha 0.5 curve has a linear component and reducing this curve will result in a curve which is very similar. The alpha 1 curve (histogram equalization) differs much more compared to its reduced curve. This is because the histogram equalized is not influences by any linear components.

Figure 54 shows tone mapping with the curves in Figure 53. The difference between the top images is very small, despite the large difference of the reduced and full curve when alpha is 1. This can be traced to the histogram of the image, Figure 48, where there are very few pixels with high values. The biggest difference of the curves is also in this area. Pixels will be mapped differently, but there are simply not enough to be noticeable.

The bottom images in Figure 54 is very similar as well. This is not surprising, when looking at the curves in Figure 53. The full curve and reduced curve for alpha 0.5 is almost identical, and the result will then be almost identical as well.

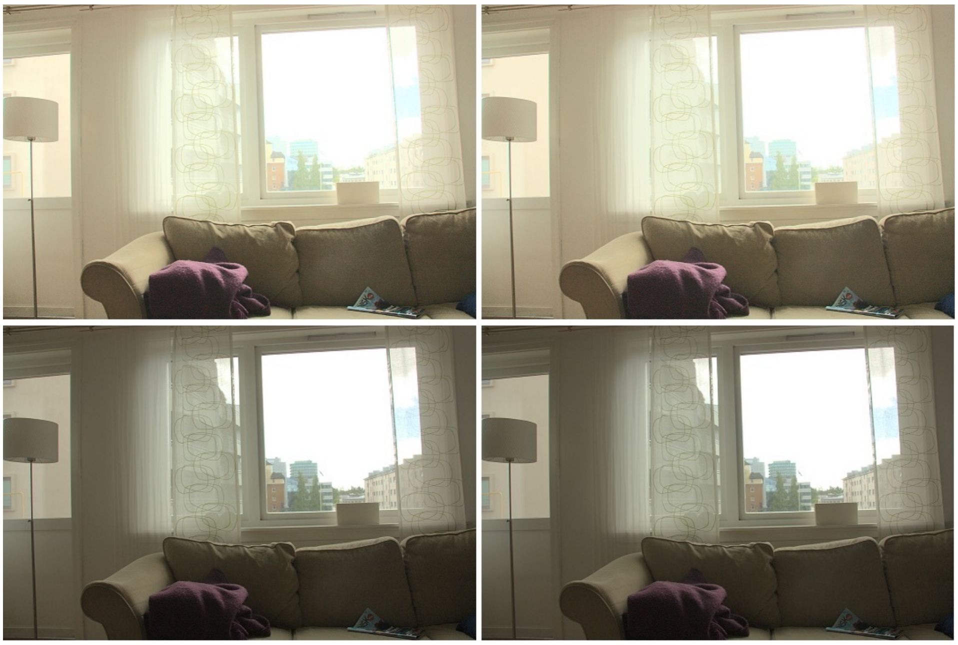

Figure 54: *Comparison of tone mapping with full curves from Duan and Qius algorithm and reduced curves. Top left: Alpha 1, full curve. Top right: Alpha 1, reduced curve. Bottom left: Alpha 0.5, full curve. Bottom right: Alpha 0.5, reduced curve. These examples show that it is very difficult to distinguish between tone mapping with the full curves and the reduced curves.*

5.4 Reading place

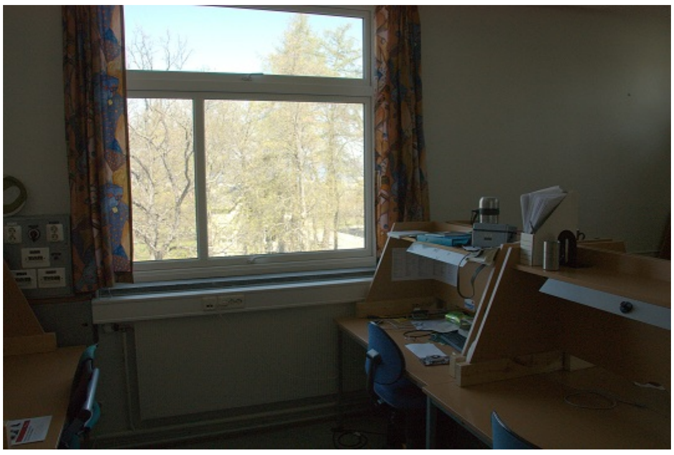

Figure 55: *Reading place*

Reading place, Figure 55 is another image which can be considered to have difficult light conditions.

5.4.1 Histogram

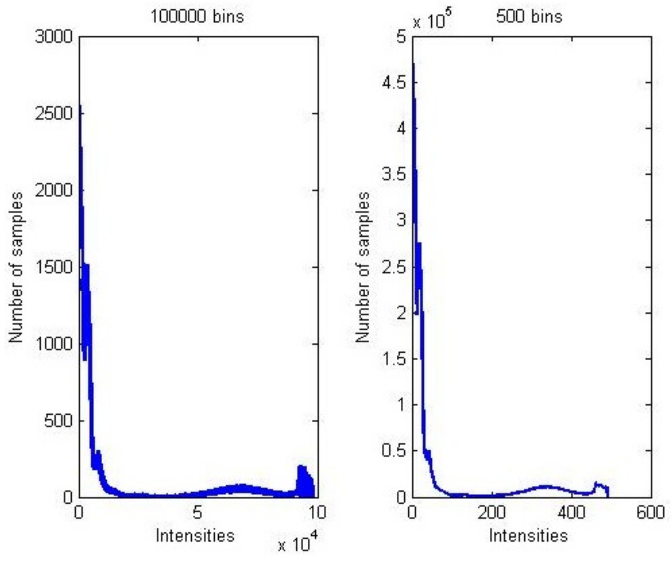

Figure 56: *Two histograms over the intensities between 0.01 and 0.99 in the reading place image. Left have 100000 bins, while the right have 500*

Figure 56 shows how changing the number of bins used to make the histogram affect the histogram. The histograms are made by using the histc function in matlab. If a value falls between the defined edges for each bin, it will be counted in that bin.

The first observation made in Figure 56 is that the shape of the histogram is maintained. Naturally, in the histogram which use only 500 bins each bin cover more values and will thus contain more samples when compared to the histogram with 100000 bins.

$$[L_{min}, \frac{L_{min}+1}{500}] \tag{18}$$

$$[L_{min}, \frac{L_{min}+1}{100000}] \tag{19}$$

Equation 18 shows how many values the first bin cover in the small histogram. Equation 19 shows how many values the first bin cover in the large histogram. This means that the first bin of the small histogram cover the same range as 200 bins in the large histogram.

5.4.2 Tone mapping curves

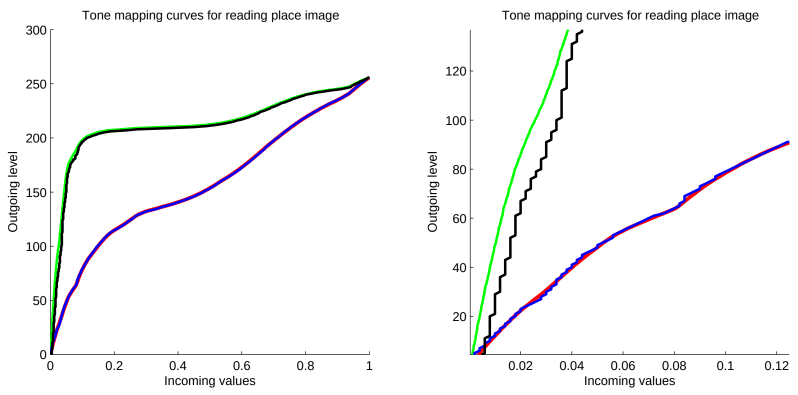

Figure 57: *Left: Showing how the number of bins in the histogram is reflected on the full tone mapping curve. Green is result from alpha 0.5 with large histogram. Black is how the curve ends up with the smaller histogram. Blue alpha 0.5 with the small histogram. Blue covers red which is the curve with alpha 0.5 with the large histogram. Right: Zoomed in on the low range*

Figure 57 shows how full tone mapping curves with alpha 1 and alpha 0.5 change when the number of bins in the histogram is reduced. The most noticeable difference is found for the alpha 1 curves, green and black. This can be seen in the right part of the figure, which is zoomed in on the low range.

Alpha 1 corresponds to histogram equalization, and this curve is therefore very dependent on the histogram. Essentially, increasing the number of bins in the histogram will result in a more accurate histogram. This will result in a more accurate curve, since there are more bins to choose from when limiting the curve. Each bin corresponds to one specific value of input intensities. This is seen with the smooth increase in values for the green curve in the figure.

That the green and black alpha 1 curves differ more than the red and blue curves, which are made with alpha 0.5, because the alpha 1 curves are more dependent on the histogram. They will be more affected by changing the size of the histogram.

Figure 58 shows how the number of bins in the histogram are reflected on the reduced curves. These are almost identical, and apparently not influenced by the changes in the histogram. These curves were made by a reduced implementation the Duan and Qiu algorithm. Instead of running it until all 255 points are found, only the seven first points are used to define the curve.

The right part of Figure 58 shows more clearly how the different histograms affect the curve. The green curve is made with the large histogram, while the black is made with the small histogram. The difference is small, because the curves are only defined by seven points. As discussed in chapter 4.2, the algorithm divides the whole histogram into smaller sub histograms. The divisions will be less accurate

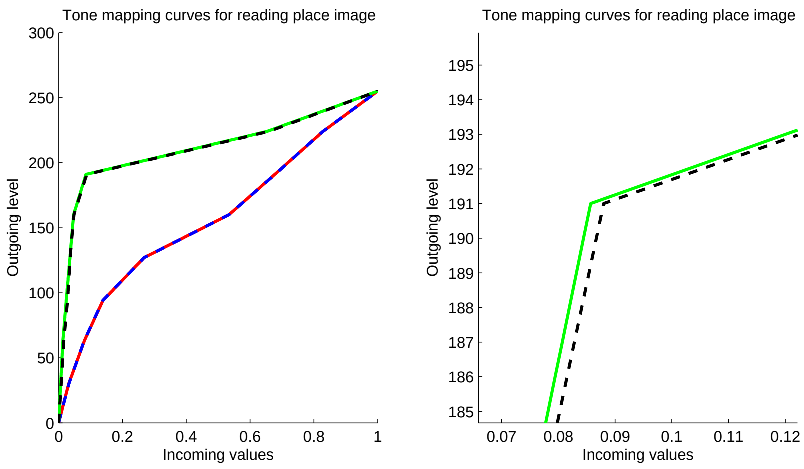

Figure 58: *Left: Reduced curves and how they are affected by reducing the number of bins in the histogram. Green and black are curves from alpha 1, while red and blue are curves from alpha 0.5. Right: Zoomed in on one of the points where a new segment start.*

when the number of bins are smaller, and for each division the difference will grow. This is because each bin in the histogram with a small number of bins cover a much larger range of values.

5.4.3 Resulting images

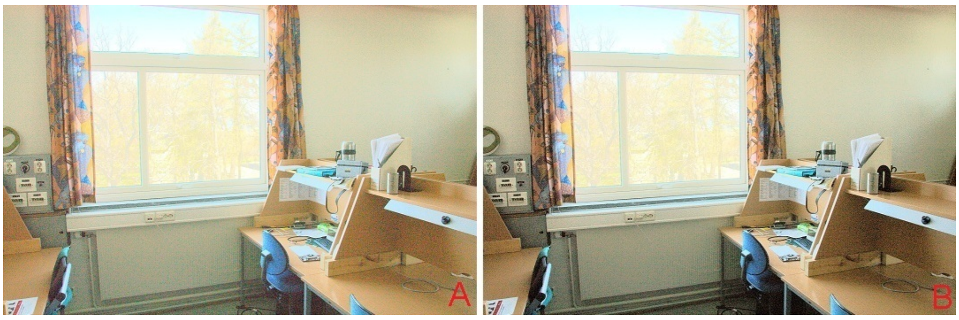

Figure 59: *Image A: Full curve with large histogram. Image B: Full curve with small histogram. Alpha 1*

Figure 59 and 60 shows how reducing the histogram affects the results when tone mapping with alpha 1. The results in Figure 59 differ the most. This is not surprising, considering the alpha 1 curves in Figure 57. Using the smaller histogram results in some portions of the image getting darker, compared to the one using the full histogram.

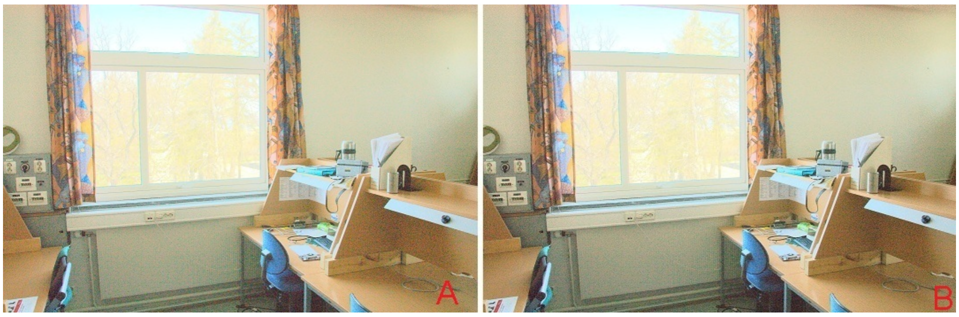

Figure 60: *Image A: Reduced curve with large histogram. Image B: Reduced curve with small histogram. Alpha 1*

Figure 61 shows how the changes of the histogram affect the images when they are tone mapped with alpha 0.5. The differences between using a small and large histogram is not noticeable. This is not surprising, since the tone mapping curves of alpha 0.5, shown in Figure 57, is very similar.

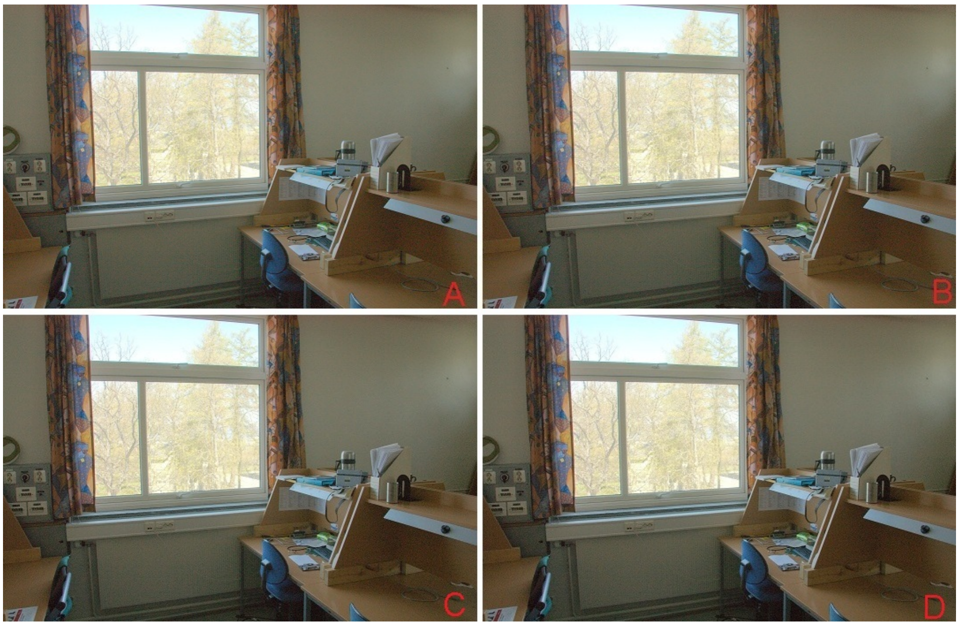

Figure 61: *Image A: Full curve with large histogram. Image B: Full curve with small histogram. Image C: Reduced curve with large histogram. Image D: Reduced curve with small histogram. Alpha 0.5 in all images.*

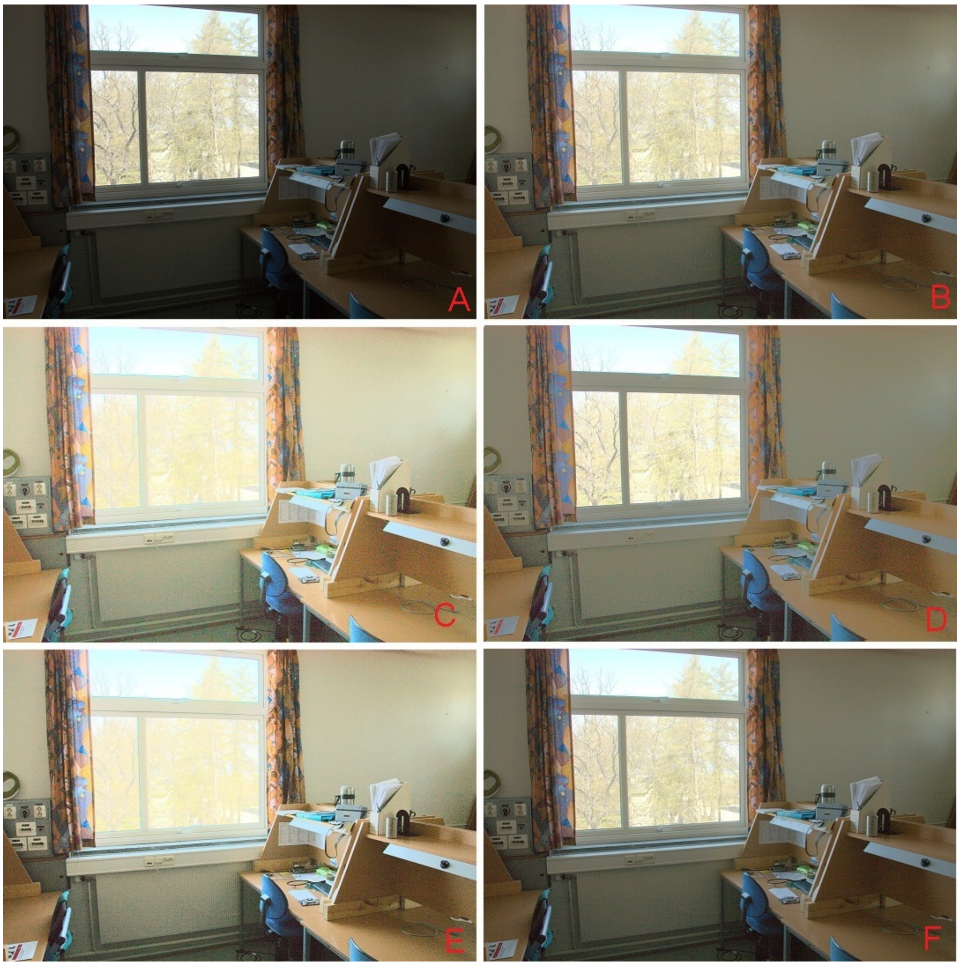

Figure 62: *A: Original. B: Duan and Qius mapper with alpha 0.5. C: alpha 1 (histogram equalized). D: three segments, limited by 20% and 80%. E: alpha 1 reduced to 8 even segments. F: Alpha 0.5 reduced to eight segments. The histograms for these images use 100000 bins.*

These images show that limiting the number of bins in the histogram is a possibility that will not reduce the qualitiy. The reduced tone mapping curve works very well with the reduced histogram. Reducing the size of the histogram will reduce the required memory for storing the histogram on the FPGA.

5.4.4 Comparison of local and global algorithm

This section show comparisons of results when tone mapping with Reinhardt's tone mapper (described in appendix B) and Duan and Qius tone tone mapper.

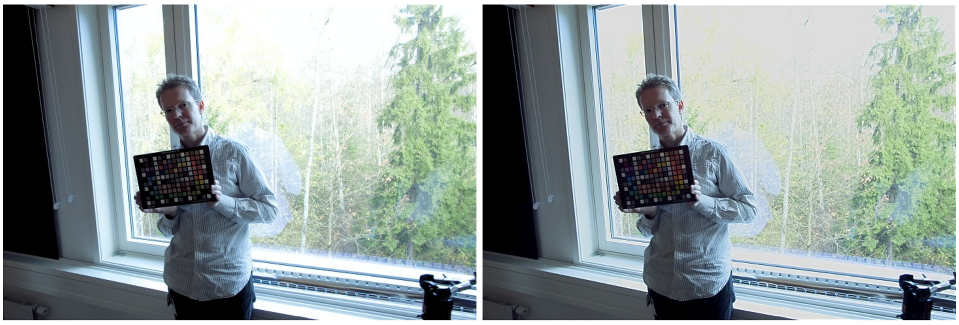

Figure 63: *Left image is tone mapped with Duan and Qius algorithm with alpha 1. Right image is tone mapped with Reinhardt's photographic tone mapper.*

Figure 63 shows how the home office image changes when it is tone mapped with a local tone mapper. The only main difference in this image is in the outdoor part, where there is a slight increase in details.

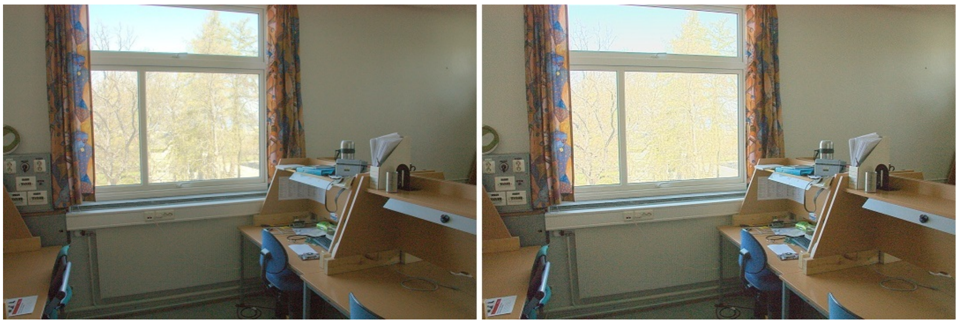

Figure 64: *Left image is tone mapped with Duan and Qius algorithm with alpha 0.5. Right image is tone mapped with Reinhardt's photographic tone mapper.*

Figure 64 shows a comparison between tone mapping with the selected global algorithm and the local algorithm. This is an example on how the increase in details might reduce the subjective experience. The right hand image is tone mapped with the local algorithm, and it has a small increase in details for the outdoor area. This increase shows more details than the global version.

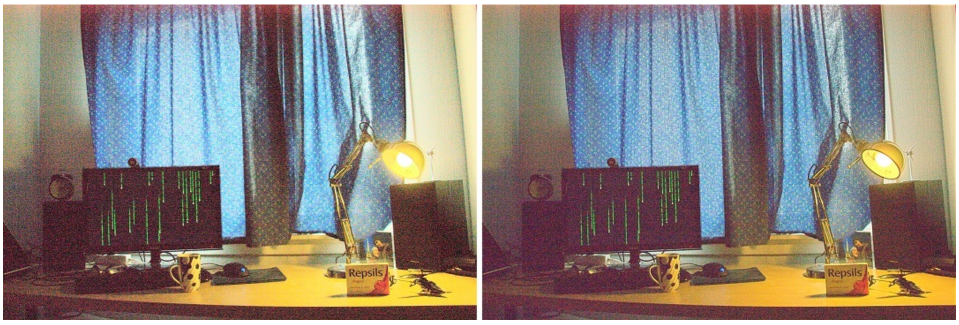

Figure 65: *Left image is tone mapped with Duan and Qius algorithm with alpha 1. Right image is tone mapped with Reinhardt's photographic tone mapper.*

Figure 65 shows how the very dark student room image is tone mapped. This is an example of how local algorithms might introduce halo effects. The lamp has a very pronounced dark ring along the edge. This is because this area is a sharp boundary between the dark surroundings and the very bright inside of the lamp. Tuning of the parameters might change this.

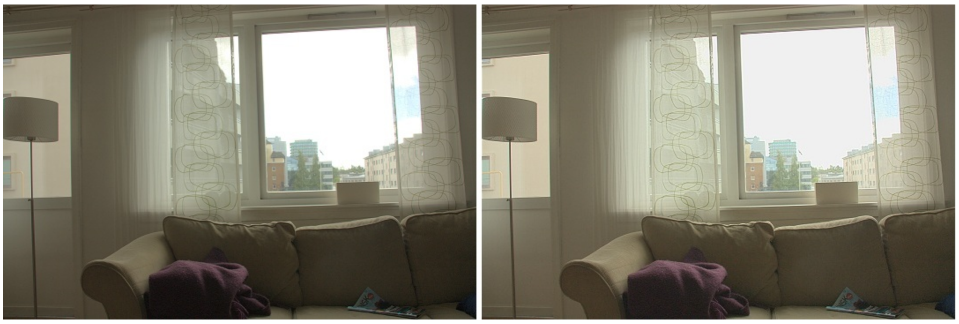

Figure 66: *Left image is tone mapped with Duan and Qius algorithm with alpha 0.5. Right image is tone mapped with Reinhardt's photographic tone mapper.*

Figure 66 shows how the outdoor area of the sofa image contains slightly more details. Other than that, the image looks very similar.

5.5 Summary of discussions

5.5.1 Tone mapping

This chapter shows some common situations where tone mapping could be used to change the intensities of the scenes. The tone mapping algorithms are designed for reducing high dynamic range image to fit the limitations of low dynamic range monitors. It would be part of the post processing if the sensors of the system were changed to sensors with extended range.

Another use of tone mapping algorithms is to change the brightness of a scene. This solves the main problem of the difficult light situation, by more evenly distributing the light across the scene. For the images shown here, this results in a brighter indoor area compared to the original images.

5.5.2 Global or local

The main problem of using a global tone mapping curve is that it can not distinguish between what is outside and inside. In a video conference situation with difficult light conditions, the indoor part where the participants are should be made more bright while not changing the outdoor part.

Using a local algorithm, as seen in section 5.4.4 will theoretically solve this, by including information on the surroundings of each pixel. The examples shown here show a slight increase in details in the outdoor area in all the images, but not enough to be very noticeable. The problem with halo effects can be seen in Figure 65.

Using a global algorithm, but incorporating local effects could be one solution to this. Different global curves could be generated for different parts of the image. One way would be to detect the pixels which correspond to the outdoor part. These pixels could then be used to generate one curve, while the indoor part could generate another curve. This requires detection of the outdoor part which is very difficult.

A more practical approach would be to divide the image into smaller parts, giving some locality to the pixel in each part. However, this will lead halo effects if pixels bordering each other are tone mapped with curves that differ to much. The complexity of implementing a global algorithm which varies based on the location in the image, will rise significantly compared to using a global curve for the complete image.

5.5.3 Tone mapping in video

As discussed in chapter 4.1.1, using local algorithms for tone mapping in a video situation is very difficult. The main problem is the delay incurred by the kernels and the memory required for buffering the lines. The other problem is related to maintaining temporal consistency. Changes of the scene should only be gradually propagated from one frame to another to prevent large changes between consecutive frames. If an area or pixel change to much, a flickering effect will appear. This reduces the subjective experience and is also very problematic for other parts of the system. To prevent the flickering the whole tone mapped framed will be buffered. Each new tone mapped pixel must then be compared to the previous pixel to limit how much it can change.

Global algorithms are much more suitable for real time video, as discussed in section 4.1.2. The delay incurred by using one of these is much smaller when compared to a local tone mapper. The problem of how to control temporal changes is reduced to limiting how much the curve can change each time it is updated. Another option is to keep the curve constant after the tone mapping is turned on.

In this solution, the curve would only be calculated once. The problem with this is that large changes in the scene will not be reflected.

The curve can be calculated in parallel with one frame and then used for the next frame. This leads to no additional delay for calculation the curve. The only delay then will be the actual tone mapping based on the curve. How to update the curve, and store curve is problematic, because the curve require much data to be clearly defined. this is further discussed in chapter 6.

6 Implementation details

A top level illustration of a global tone mapper can be seen in Figure 67. This figure shows the most important parts of a tone mapper, and will look the same despite changing the tone mapper.

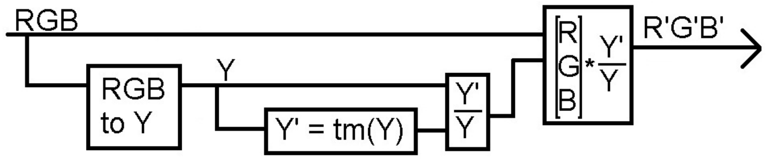

Figure 67: *System outline with the most important parts of the tone mapper implementation. The tone mapping operator is denoted Tm. No delays are shown here, but the R, G and B channels are required to be delayed as long as it takes for their relative Y value to be processed.*

Figure 67 shows a general system outline. Changing the tm operator will reflect changes of tone mapper. The other parts are necessary to maintain color consistency and are invariant to the actual tone mapper used.

The sequence of operations required to tone map an image is:

1. Calculate the intensity value of each pixel
2. Calculate the tone mapping curve:
   (a) Make a histogram of the luminance values
   (b) Calculate the curve
3. Tone map the image
4. Calculate the new colors

Tone mapping a video stream is essentially the same. As discussed in chapter 3.5, the curve of one frame can be used on the next frame. This will allow the tone mapper to calculate a new tone mapping curve in parallel to the actual tone mapping. Figure 68 shows an overview of this. The curve used for the current frame, number i, use the previous curve, number i - 1, for tone mapping.

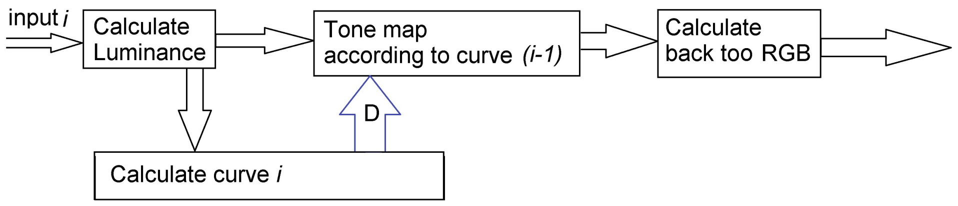

Figure 68: *System outline of operations for tone mapping a video stream. Notice that the previous curve is used to tone map the image, while the current curve is being calculated*

This chapter will discuss implementation specific choices related to the implementation of the proposed tone mapping system, discussed in chapter 4.3. The reduced system is very versatile and it is more suitable for an FPGA implementation than a full tone mapper.

The first section will discuss how and where the coefficients could be calculated. This is the most important decision, since it will affect the rest of the system.

Section 6.2 is used to discuss how the histogram can be implemented. This discussion is based on the results from section 5.4. It was shown that the calculations of the curve is very dependent on the histogram, so it is an important part of the system.

How the coefficients are used in the tone mapper is discussed in section 6.3. There are two distinct ways of doing this and they will be compared here.

Section 6.4 will summarize the implementation results and describe each component separately. The specific VHDL code for all the modules can be found in appendix E and on the DVD. This also include matlab code for generating test vectors for test benches to each module.

6.1 Calculating the coefficients

The tone mapping algorithm selected will calculate the full curve based on the histogram. The curve will then be reduced to a segmented version. Different algorithms will return different values. To reduce the curve, only a limited number of points are required. These will correspond to values which limit each segment. The procedure for this reduction is described in section4.3.

There are three ways of implementing this. The first is to implement the calculation of both the full curve and the reduction in dedicated hardware. This will lead to fast calculations of both the curve and the reduced curve.

The other way is to move the calculations to a coprocessor. This allows resource sharing with other parts of the system when the coefficients are not calculated. Most of the time, no calculations will be done because the coefficients are only required at the start of each frame, and will not be updated until the next frame.

A mix between these two is also an option. The problem here would be how to transfer the full curve to the reducer. A common way of communicating with the coprocessor is through a bus, but transferring the whole curve across the bus will severely clog it.

Implementing the calculation of both the full curve and the reduced curve will be done in a coprocessor. This allows for close interaction between calculation of the full curve and the reduced curve. The code for calculating this will be left for future work, since it has to be adapted to the final system. A memory interface will instead be implemented to make the tone mapper able to communicate with the coprocessor across a system bus.

6.2 Histogram

The tone mapper chosen is based on a histogram. When a frame is complete, the calculation of the coefficients can start. This practically means that the curve can only be updated at the beginning of a frame. The proposed tone mapper relies on a set of coefficients, so the constraint implies that the coefficients have to be constant for the duration of the frame. Updating the curve will therefore be required to happen between two frames, and the coefficients will need to be calculated before this. Calculations of the coefficients can start when the histogram is complete. More information can not be added to the histogram when the calculations have started.

Updating the histogram

Each calculated intensity pixel should be added to the histogram. The problem is when to stop updating the histogram, so that the calculations can start.

One way it to assume that the processor will be able to complete the calculations during the vertical synchronization. By doing this, the curve will only be delayed by one frame, since the histogram will be complete once the frame is finished.

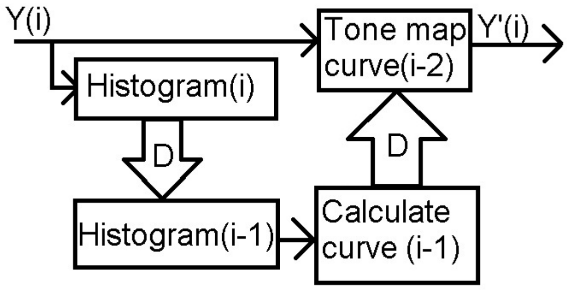

Figure 69: *Doing calculations in parallel to the frame will require a histogram buffer seen here. Arrows with D indicate operations done between frames. The curve will in this setup be delayed by two frames.*

Another way is to relax the requirements of how delayed the tone mapping curve can be. The calculations of the new curve can be done in parallel to the next frame, resulting in two frames of delay for the curve (one to store the histogram and one to calculate). The problem with this is that a buffer for the histogram would be required to prevent the histogram from being overwritten after the calculations have started as seen in Figure 69.

The third option is to only update the curve every third frame. This would allow the histogram to be based on one frame. The histogram would then be frozen for the next frame during the calculations. At the beginning of the third frame the curve would be updated. This is similar to the situation shown in Figure 69 except that the histogram buffer would not be required.

The fourth option is to start the calculations of the next curve during the last lines of the current frame. The histogram would then be frozen after the calculations start. In this setup, the last lines of a frame would not have any influence on the histogram. These lines would then be assumed to contain little to no information which would drastically influence the curve.

The last option would be to calculate the coefficients on a host system, and transfer the coefficients back to the tone mapper. It is assumed that the tone mapper is deep in the image pipeline, so the outgoing data will be tone mapped if the tone mapping is turned on. This makes it impossible for the host computer to make a new curve, unless the tone mapping is turned off for one frame. A new histogram made by the host will be based on data which is already tone mapped.

Size of histogram

How large to make the histogram is one very important decision. There are two problems with having a large histogram. The first is that it the algorithms will use more time to calculate the curve with a large histogram. The second is that it requires memory to store.

The size of the histogram was discussed in section 5.4. The size of the histogram will affect how the curve is calculated. However, it is shown that reducing the histogram to only 500 bins affect the reduced tone mapper less than it does for the full curve.

It can be assumed that the video will be full HD, 1920 pixels in each of the 1080 lines. This correspond to a total of 2 million pixels. Counting all pixels in one register will require 21 bit. Each bin of the histogram should therefore be 21 bits to cover for the worst case scenario. The other option is to use less bits, but then overflow detection would be required to prevent the bins from wrapping around. The transfer itself can be done in two ways. The most easy would be to give each bin in the histogram a separate address. The other would be to use on address for the histogram. Each clock cycle which this is enable would transfer the lowest bin, and shift the rest of the histogram. Reading out the complete histogram would take equal amount of time, but this approach would use less addresses to read it.

Transferring the histogram

Using a smaller histogram also makes it practical to transfer the whole histogram across the communication bus. Limiting the histogram to 500 bins, as done in section 5.4, would enable the histogram to be transferred in 500 packets. Each packet would then correspond to one bin. Another possibility would be to reduce

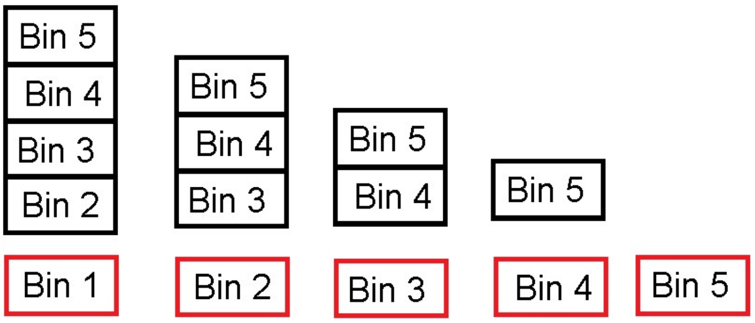

Figure 70: *Showing how the histogram is shifted each time it is accessed. The column correspond to accesses, and the red bins will be read out on each access. Only one of the original bins are remaining after four accesses.*

Bin number	Decimal value	Binary value
0	335298	0101 0001 1101 1100 0010
1	401016	0110 0001 1110 0111 1000
2	466734	0111 0001 1111 0010 1110
3	532452	1000 0001 1111 1110 0100

Table 1: *Showing the smallest values in the first four bins in the histogram.*

the size of each bin until two bins can be transferred with one packet. This would halve the number of packets sent across the bus.

How to access each bin can be done in two ways across the bus. One would be to give each bin a separate address. This would result in 500 addresses being reserved for the histogram. Another would be to use only one address and access the histogram as a stack, as seen in Figure 70. Each access to the histogram address would shift the histogram by one. This is more complex than using 500 addresses, but would reduce the number of addresses used.

Limiting the bins

In the simulations, only intensity values between 0.01 and 0.99 were included in the histogram. The intensities are calculated as 25 bit numbers in the implementation. The smallest Y will be 0, while the biggest will be 33529860. This means that the biggest value counted in the histogram should be 33194561, while the smallest should be 335298. Using 500 bins indicate that each bin should cover 65719 values.

These number are a bit unfortunate, because in terms of binary they are very irregular, as seen in table 1. Each incoming value have to be compared to all the bin delimiters to find where it should be placed.

This can be countered by choosing a bit more convenient values. In the original histogram, each bin covers 65718 values. Changing this to 65536 would make the limiting values more convenient, since increasing each bin by this will make the delimiting values more systematic. Table 2 compares the binary values of the two

Decimal value	Binary value
65718	1 0000 0000 1011 0110
65536	1 0000 0000 0000 0000

Table 2: *Binary values*

Bin number	Decimal value	Binary value
0	196608	011 0000 0000 0000 0000
1	327679	100 0000 0000 0000 0000
2	393215	101 0000 0000 0000 0000
3	458751	110 0000 0000 0000 0000

Table 3: *Showing the smallest value in the first four bins in the modified histogram.*

numbers.

By choosing the smallest and biggest value differently, a very systematic set of numbers to limit each bin can be made. This can be seen in table 3. Here, the last 16 bit of the 25 bit incoming values does not need to be compared. For example, if bits 19 too 17 of the incoming pixel is 101, the pixel will be counted in bin 1, since it is smaller than the minimum of bin 2 and equal or bigger than the smallest value in bin 1.

Using table 3 reduces the problem from comparing 25 bits to comparing 9 bits. 9 bits is to be expected, since 9 bits are required to access 500 positions. The 9 most significant bits can then be used to indicate which bin in the histogram to increase by one.

6.3 Tone mapper

The chosen tone mapper is based on a piecewise linear curve (see section 4.3). Figure 71 shows the essence of the algorithm.

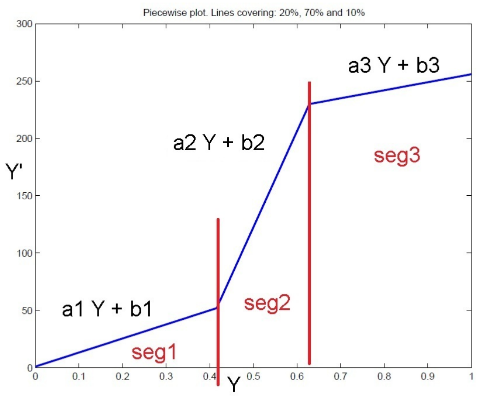

Figure 71: *Showing a piecewise linear curve with three segments.*

Each segment is based on equation 20, and the coefficients of this is calculated beforehand.

$$Y' = aY + b \tag{20}$$

There are two distinct ways of implementing equation 20. The first is to pre-computed all the possible output values and put them in a look up table. This is required for full size algorithms like Duan and Qius algorithm to define the curve. Using a look up table will lead to minimal delay, based on how large the table is. The two problems with this is the size of the table and how to update the table.

The other way is to use the fact that the curve is piecewise linear. Regardless of the number of lines, each line can be condensed to the simple equation as seen in equation 20. The incoming value will then need to be placed in the correct segment first, and then the correct coefficient can be used.

$$Y' = (Y - Y_{segx}) * a_x + b_x \tag{21}$$

Equation 21 shows the complete equation for line segment x. Here Y_{segx} is the maximum value of the previous segment. In Figure 71 this is the Y' value at the red lines, indicating different segments.

The pseudo code for the the tone mapping will be:

```
**for** Number of lines - 1 **do**
  Find segment which contain Y, segX
**end for**
Coefficients for segX: aX, bX, $Y_{seqX}$
Calculate: $(Y - Y_{seqX}) * aX + bX = Y'$
Return Y'
```

This is the strength of the proposed system from an implementation point of view. Other tone mappers which rely on a curve will essentially need to be implemented with a large look up table, while the segmented version can use resources on the FPGA to be calculated for each pixel.

The implementation will therefore be done with the mathematical operations. All the coefficients will be stored in the tone mapper. It will be pipelined in two parts, where the first will calculate delta Y (Y - Y_{seqX}) based on which segment it belong to. It will transfer this value together with the corresponding aX and bX to the next stage. The second and last pipeline stage will calculate the new Y value, (Y').

6.3.1 Tone mapping on or off

Simply bypassing the whole tone mapper when it off too reduce the delay will change the delay of the system. This is unfortunate for a video system, because the throughput and delay should be maintained.

One options is too turn off the calculations of the Y value. This would reduce the switching inside the tone mapping module. The modifying value would then have to be set to one, too prevent changes. The incoming RGB values could then be sent through the tone mapper with constant delay.

Another way would be to set the delimiting values should to maximum. This will force all the values to be tone mapped with the coefficients in the first segment. Setting this segment to one for alpha and zero for beta will not change the Y value. The modifier for each pixel will then be 1 and the pixels will then not be changed. This option will be used in the implementation.

6.4 Implementation results

The proposed tone mapper was implemented in VHDL for a general FPGA. Eight modules were implemented and tested. The buffers and the reducers are identical, and were thus simply replicated multiple times. The code for the top level design can be found in appendix E.2 and a figure showing the rtl view of the system can be found in appendix D.

The design was synthesized for the Cyclone III EP120 FPGA. This is the largest of the cyclone III FPGAs. It has over 200000 logic elements, and 396 embedded 18 x 18 multipliers.

	Without retiming	With retiming
Frequency	198.9 MHz	198.9 MHz
Logical elements:	32146	32166
Registers:	26383 (21 %)	26485 (21 %)

Table 4: *Results from synthesizer*

Table 4 show a summary of the results from the synthesizer. These were achieved by using Synplify Pro C-2009.06.

A significant amount of the registers used are because of the histogram. It was implemented with full buffering, so the 500 bins of 21 bit each account for 21000 of the registers. Implementing the histogram in another way, as discussed in section 6.2, would therefore reduce the number of registers significantly.

The maximum frequency achieved is more than enough for real time video. Critical paths were mainly removed by increasing the number of stages for the complete pipeline. The final delay will be 53 cycles from a set of pixels enter the system until they are tone mapped and returned. Most of the delay come from the divider which is implemented as a pipelined non restoring array divider. Changing this to a look up table instead would reduce the delay by 38 cycles. This would lead to a very interesting trade off, because using fewer cycles to find the luminance modifier will require less buffering of the original RGB values.

Table 4 show how the results change with and without retiming. Using retiming will in general improve the frequency by scheduling operations so that each stage has an equal size [2]. In this particular case, no improvement is made to the clock frequency, but the number of registers and logical elements are increased slightly. This suggest that some retiming has been done to make each stage similar in length.

The following sections contain brief descriptions of the implementation of each module. The specific VHDL code can be found in appendix E and on the accompanying DVD. Simulation waveforms are located in appendix F.

6.4.1 RGB to Y

The rgb2ychan module takes the incoming RGB triplet and transforms it to Y, or luminance, with equation 22. The multiplications and summarization is a potential critical path if it is done in one cycle. Therefore, it is pipelined so that the multiplications is done in parallel in one cycle before they are summarized the next. The code for this module can be seen in appendix E.3.

$$Ychan = red * 1742 + green * 5858 + blue * 591 \tag{22}$$

These coefficients are based on the CIE colorimetric for calculating the XYZ triplet. Only the coefficients for Y are used, and those are multiplied by 8192 and limited to 13 bit.

Y is also sent out of the top level entity. This is to make it available for adaption to the rest of the system for calculations of the tone mapper.

6.4.2 TMA

TMA is the first part of the tone mapper itself. This module receives the segmented curve from the memory interface and the delimiters which limit the extent of each segment. Using two cycles, each incoming value is placed in its proper segment. This is done by subtracting the incoming Y value from all the delimiter values. This is done in the first cycle. The next cycle checks the most significant bit of the results from the subtractions. The first subtraction which does not lead to a negative number indicates which segment the value belongs to. Calculation of delta Y is then conducted before this value and the coefficients for the segment is sent to the next module, TMB. Code for this can be found in appendix E.4.

6.4.3 TMB

$$Y' = deltaY * A + B \tag{23}$$

TMB is the second part of the tone mapper. This module is also pipelined, realizing equation 23. In the first cycle, the incoming value delta Y multiplied by the incoming alpha. Alpha is 13 bit large, where the last eight bit corresponds to the decimal part of the number. DeltaY is divided by 256 before the multiplication to limit how large the numbers can get. Beta is added in the next cycle. Values above $2^{26} - 1$ will be set to $2^{26} - 1$. This code can be found in E.5.

[2]Retiming will move registers so that there is a more even amount of operations in each cycle. Typically, a very long critical path will then be spread out on a number of cycles by moving the registers. This might lead to an increase in the number of registers required.

6.4.4 Luminance modifier

As discussed in chapter 3.4 equation 24 is required to find how much each pixel have been modified. This is by far the biggest operation done on every pixel. The division is implemented as an array divider (discussed in appendix C). This results in a large combinational divider. Using one cycle through it is not an option because the critical path would be to long. This divider is therefore pipelined into 40 stages, each corresponding to one coefficient of the result.

$$lummod = \frac{Y'}{Y} \tag{24}$$

The new luminance is multiplied by 12 bit (4096) to align it with the original luminance. This results in the large division. However, only the 17 last bit of the coefficient are interesting. This is to limit how much each pixel might be maximally changed with. 12 of these 17 bits correspond to decimal numbers.

An alternative to implementing the divider in hardware is to realize it as a look up table. This will reduce the delay to only two cycles, but requires memory to do. The code for the array divider can be found in appendix E.6.

6.4.5 Reducer

Equation 25 shows the equation which should be done on each color channel (C is either R, G or B) to get the new image. This is implemented with a multiplication. The module will then limit the output to eight bits. The code for this can be found in appendix E.7.

$$C' = C * lummod \tag{25}$$

6.4.6 Buffer

A FIFO buffer was implemented to buffer the original value of RGB and the original Y value until it is needed. This is done with a simple array structure which propagates the incoming values down in an array. The last array element will send the data too the output. The size of the array reflects how many clock cycles the elements should be delayed by. This is controlled by simple generic constructs. The code can be seen in appendix E.8

6.4.7 Histogram implementation

The histogram was implemented with buffering of the histogram. Other possibilities of doing this is discussed in section 6.2. Each bin is 21 bit large, which is enough to count the entire frame. The histogram is buffered in the bus interface. The code can be found in appendix E.9.

Address	Content
0-7	Alpha values for segment 0-7
8-15	Beta values for segment 0-7
16-22	7 delimiting values
23	Control register: d0 update curve, d1 store histogram
24 - 524	Histogram bins

Table 5: *Bus addresses used to load curve in top level, offset 0*

6.4.8 Bus interface

The calculations of the coefficients will be done either in a coprocessor or an embedded processor. One way for a coprocessor to communicate is on a bus, so a bus interface was implemented. This will handle loading of the coefficients from the coprocessor and transfer of the histogram back to the processor. Each bin of the histogram has one specific address. This results in 500 addresses being reserved for the histogram. There are alternatives to this, as discussed in section 6.2. Table 5 shows the addresses used to access different parts of the system when the ofset is equal to zero.

The sizes of the coefficients are 13 bit for alpha and 25 bit for beta. The delimiting values used for limiting each segment are 25 bit large, the same as Y. The data bus is assumed to be 32 bit large, so one packet is required for each beta and delimiting value. Two alpha values can be transported in one packet, but for ease of implementation each packet is constrained to one alpha. One address is also dedicated to controlling the tone mapper. The code can be found in appendix E.10.

After no more values are to be loaded into the histogram, the storehist signal goes high. This will switch the current histogram into the buffer too save it. The buffered histogram will be available until the next time the storehist signal goes high.

6.4.9 Verification

A matlab script for running histogram equalization was written. This was done using only integers, which makes it suitable for verification. An option was included, which allowed the results of each step to be written out too text files in binary form. The matlab can be seen in appendix G.

Test benches were written for each component. These read the values which were used in the calculation in matlab and the expected result. The values were used to excite the components, and the result was measured against the expected result.

The following messages are examples of what will be printed if there is an discrepancy between the result from the module and the expected result stored in the test vector:

```
# ** Warning: Calculation done wrong.
```

```
T_Y: 0 while expected value is: 95743 according to testfile
#    Time: 50 ns  Iteration: 0  Instance: /m00300_tb_rgb2ychan
# ** Warning: Calculation done wrong.
T_Y: 95743 while expected value is: 123096 according to testfile
#    Time: 90 ns  Iteration: 0  Instance: /m00300_tb_rgb2ychan
```

These specific messages are a result of testing the rgb2ychan module with only one cycle of delay, while it use two. The test bench should start checking the outputs after two cycles, not after one as it does in this case. Simply changing the test bench to start checking after two cycles solves this specific case.

The module only have one output. Other modules with more outputs will have similar checks on all the output signals.

Finding where the mistakes are if only the top level is tested is difficult and time consuming. Testing each module of the system separately is a way of to verify that each module function correctly. A test bench for the top level module was also written to verify that the system as a whole was working correctly. The test bench for the RGB to Y module can be found in appendix E.11. The other test benches follow the same structure as this one.

7 Conclusion

Sensor level solutions for extended dynamic range

Meeting rooms or offices often have windows. This is very difficult for cameras because of the large dynamic range of the scene on sunny days. One option is to extend the dynamic range of the sensors and use a Wide dynamic range sensor.

Another option is to produce high dynamic range images with multiple exposures. Unfortunately, multiple exposures will be required. Realizing this with one normal sensor is prone to effects like ghosting (section 2.3.1) which will drastically reduce the quality of the image. Using two sensors with overlapping exposure times is an option which will solve this, but the required hardware is doubled.

Using advanced sensors which are able to capture multiple exposures on a per pixel basis is interesting. The additional hardware compared to multiple sensors are very low. However, there will be a drastic increase in the complexity of the required post processing. This is problematic for real time video, and would require additional hardware.

Tone mapper

Another solution to the problems of difficult light conditions is to use tone mapping to redistribute the intensities in the scene. Simulations shown in chapter 5 that using a global algorithm is an effective way to solve difficult light conditions.

Global tone mappers are most suitable for real time video. These make a curve, which is then used to tone map each frame. The amount of data which define the whole tone mapping curve is too large to be practical for an FPGA implementation. A solution to this is presented here, in section 4.3. This system first reduces the curve to a limited number of segments, before transferring it to the tone mapper. Comparisons between tone mapping with a full global algorithm proposed by Duan and Qiu (discussed in section 4.2) and the reduced version, shows that the quality of the images with the reduced curve is on par with the full curve. The simulation results can be seen in chapter 5.

The number of parameters for controlling the curve should be kept low. Ideally, turning the tone mapper on or off should be enough to improve the image. The problem is that varying situations require different parameters, as seen in chapter 5.

Implementation

The amount of data requires to define the whole curve is reduced to just storing the coefficients of each segment. This reduction makes it feasible to implement a tone mapper on an FPGA without using all the available resources. The implementation is also independent to the actual algorithm used to find the full curve. This would be hidden to the tone mapper in the reduction system, as seen in Figure 72.

The curve will be calculated either in a coprocessor or an embedded processor. This requires adaption to the final system, and is left for future work. This solution allows close interaction with other parts of the system during the calculation of the

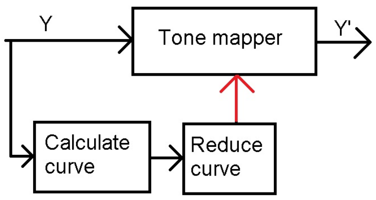

Figure 72: *Illustration on how the curve is hidden from the tone mapper. The tone mapper will only see the reduced information about the curve.*

curve. Important considerations here are how often to update the curve and by how much. The proposed reduction system is only defined by a limited number of points. This makes it very suitable for limiting how much the curve can change from frame to frame, as discussed in section 4.3.2.

This implementation allows the global algorithm to be easily changed later. Doing the calculations in a coprocessor or embedded processor will allow the tone mapping to be done in conjunction with other parts of the system. This will allow resource sharing, so that other calculations can be done when the coefficients are not calculated.

A bus interface was implemented to enable the module to communicate with a processor. This interface handles both the updating of the curve, and the transfer of the histogram to the processor.

How large the histogram is will affect the resource usage and how it can be transferred to the coprocessor. How reducing the size histogram was discussed in section 5.4.2. Here it was shown that the reduced tone mapping system works well with a smaller histogram. A histogram using 500 bins have been implemented based on this discussion.

The histogram is buffered too prevent it from being overwritten. This doubles the number of registers required for the histogram. Alternative ways of implementing this, discussed in section 6.2 will reduce the memory required.

The tone mapper was successfully implemented, reaching a maximum frequency of 198.1 MHz. This is sufficient for tone mapping each pixel real time. The delay through the tone mapper is 53 clock cycles, where 40 are incurred by the divider. Changing the divider to a look up table will reduce the delay substantially, but increase the memory required for the implementation.

The implementation makes it possible to do tone mapping in real time, as well as controlling the temporal changes of the curve.

8 Future work

Implementing a multiple sensor system to achieve a higher dynamic range would be very interesting. This will reduce some of the problems that exist with WDR sensors. There are two main uses for this, one would be to have the same exposure time for both sensors. Combining the two images would then reduce the noise. Setting the two sensors to different exposure times could be used to make an image with a higher dynamic range. This could either be always on or just turned on when the scene has difficult light conditions.

Changing from the global tone mapper to a global algorithm with local variations could be interesting. The difficulty would be to control the temporal changes between frames. The memory required will increase by how local each curve is. The extreme version is to use a separate curve for each pixel. This would require more memory than a frame buffer. Then a more advanced local algorithm could be used instead.

Using a graphical processing unit (GPU) in the pipeline might make a local algorithm more feasible. GPU's are designed with large frame buffers. This has been used in one graphical technology demo called Lost Coast by valve software [61]. They used computer generated HDR and local tone mapping in real time to simulate how light behave. The data used are much more detailed than what can be deduced from real images, but it is interesting and is show that local tone mapping could work for real time if enough memory and processing power is available.

Possible advances in sensors should be followed, especially sensors like the piecewise linear sensor. This type does not reduce the encodings used for dark areas. Other sensor types should also be evaluated. Sensors which are able to adjust the response curve on the fly are especially interesting.

Future work related to the tone mapper could also take into account the limitations of the monitor that will show the video. This is most interesting if the camera system is to be used with one specific type of monitors. The tone mapper could then be adapted to better fit the sensor data to the monitor. The maximum brightness of the monitor should influence how the maximum value of the sensor is processed.

References

[1] A. Adams. *The Camera.* Little, Brown and Company, 1980.

[2] A. Adams. *The Negative.* Little, Brown and Company, 1981.

[3] A. Adams. *The Print.* Little, Brown and Company, 1983.

[4] Erik Reinhard, Michael Stark, Peter Shirley, and James Ferwerda. Photographic tone reproduction for digital images. *ACM Trans. Graph.*, 21:267–276, July 2002.

[5] J. Tumblin and H. Rushmeier. Tone reproduction for realistic images. *Computer Graphics and Applications, IEEE*, 13(6):42 –48, November 1993.

[6] A. Spivak, A. Belenky, A. Fish, and O. Yadid-Pecht. Wide-dynamic-range cmos image sensors;comparative performance analysis. *Electron Devices, IEEE Transactions on*, 56(11):2446–2461, "November" 2009.

[7] Ahmet Oğuz Akyüz, Roland Fleming, Bernhard E. Riecke, Erik Reinhard, and Heinrich H. Bülthoff. Do hdr displays support ldr content?: a psychophysical evaluation. *ACM Trans. Graph.*, 26, July 2007.

[8] Erik Strømme. Tone mapping in video conferences. Project report, 2010.

[9] A. El Gamal and H. Eltoukhy. Cmos image sensors. *Circuits and Devices Magazine, IEEE*, 21(3):6 – 20, "May-June" 2005.

[10] R. Ramanath, W.E. Snyder, Y. Yoo, and M.S. Drew. Color image processing pipeline. *Signal Processing Magazine, IEEE*, 22(1):34 – 43, jan. 2005.

[11] R.D. Gow, D. Renshaw, K. Findlater, L. Grant, S.J. McLeod, J. Hart, and R.L. Nicol. A comprehensive tool for modeling cmos image-sensor-noise performance. *Electron Devices, IEEE Transactions on*, 54(6):1321 –1329, 2007.

[12] K. Irie, A.E. McKinnon, K. Unsworth, and I.M. Woodhead. A technique for evaluation of ccd video-camera noise. *Circuits and Systems for Video Technology, IEEE Transactions on*, 18(2):280 – 284, February 2008.

[13] A. Buades, B. Coll, and J. M. Morel. A review of image denoising algorithms, with a new one. *Simul*, 4:490–530, 2005.

[14] Thor Arne Brandsvoll. Fpga based noise reduction in video cameras. Project report, 2010.

[15] Gerald H. Jacobs. Evolution of colour vision in mammals. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 364(1531):2957–2967, 2009.

[16] Bryce E. Bayer. Color imaging array. Patent application, 1976. Number Color imaging array, Filling data 3 march 1975, issue date 20 july 1976.

[17] Kodak. Color correction for image sensors. Application note to image sensors from Kodak's webpage, 2008.

[18] X. Li, B. Gunturk, and L. Zhang. Image demosaicing: a systematic survey. In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 6822 of *Presented at the Society of Photo-Optical Instrumentation Engineers (SPIE) Conference*, January 2008.

[19] David X. D. Yang and Abbas El Gamal. Comparative analysis of snr for image sensors with enhanced dynamic range. In *in Proceedings of SPIE*, pages 197–211, 1999.

[20] Y. Bandoh, Guoping Qiu, M. Okuda, S. Daly, T. Aach, and O.C. Au. Recent advances in high dynamic range imaging technology. In *Image Processing (ICIP), 2010 17th IEEE International Conference on*, pages 3125 –3128, "September" 2010.

[21] Kate Devlin. A review of tone reproduction techniques. Technical Report CSTR-02-005, Department of Computer Science, University of Bristol, November 2002.

[22] Kodak. Kaf-09000 image sensor. kodak.com, 2010.

[23] Kodak. Kac-03100 image sensor. kodak.com, 2006.

[24] Kodak. Kaf-5000 cmos image sensor. kodak.com, 2006.

[25] Erik Reinhard, Greg Ward, Suanta Pattanik, Paul Debevec, Wolfgang Heidrich, and Karol Myszkowski. *High Dynamic Range Imaging: Acquisition, Display and Image-Based Lightning.* Morgan Kaufmann, 2 edition, 2010.

[26] The eye and high-dynamic-range vision. In Bernd Hoefflinger, editor, *High-Dynamic-Range (HDR) Vision*, volume 26 of *Springer Series in Advanced Microelectronics*, pages 1–12. Springer Berlin Heidelberg, 2007.

[27] Paul E. Debevec and Jitendra Malik. Recovering high dynamic range radiance maps from photographs. In *ACM SIGGRAPH 2008 classes*, SIGGRAPH '08, pages 31:1–31:10, New York, NY, USA, 2008. ACM.

[28] E.A. Khan, A.O. Akyuz, and E. Reinhard. Ghost removal in high dynamic range images. In *Image Processing, 2006 IEEE International Conference on*, pages 2005 –2008, October 2006.

[29] Sing Bing Kang, Matthew Uyttendaele, Simon Winder, and Richard Szeliski. High dynamic range video. *ACM Trans. Graph.*, 22:319–325, July 2003.

[30] K. Jacobs, C. Loscos, and G. Ward. Automatic high-dynamic range image generation for dynamic scenes. *Computer Graphics and Applications, IEEE*, 28(2):84 –93, March 2008.

[31] T. Jinno and M. Okuda. Motion blur free hdr image acquisition using multiple exposures. In *Image Processing, 2008. ICIP 2008. 15th IEEE International Conference on*, pages 1304 –1307, October 2008.

[32] Ran Ginosar, Oliver Hilsenrath, and Yehoshua Zeevi. Wide dynamic range camera. Patent application, 1993. Number 5144442, Filling data 21 nov 1991, issue date 1 sep 1992.

[33] Kodak. Kaf-9618 cmos image sensor. kodak.com, 2007.

[34] Martin Cadk. *Perceptually Based Image Quality Assessment and Image Transformations.* PhD thesis, Czech Technical University in Prague, 2008. Chapter 4.

[35] Patrick Ledda, Alan Chalmers, Tom Troscianko, and Helge Seetzen. Evaluation of tone mapping operators using a high dynamic range display. *ACM Trans. Graph.*, 24:640–648, July 2005.

[36] Naoya Katoh, Tatsuya Deguchi, and Roy S. Berns. An accurate characterization of crt monitor (i) verifications of past studies and clarifications of gamma. *Optical Review*, 8:305–314, 2001. 10.1007/s10043-001-0305-0.

[37] ITU-R. Recommendation itu-r bt.709-5. Technical report, International Telecommunications Union, 1993-2002. -5 is the current version published in 2002.

[38] Michael Stokes, Matthew Anderson, Srinivasan Chandasekar, and Ricardo Motta. A standard default color space for the internet - srgb. Technical report, Microsoft and Hewlett-Packard, November 1996.

[39] Jiang Duan, Marco Bressan, Chris Dance, and Guoping Qiu. Tone-mapping high dynamic range images by novel histogram adjustment. *Pattern Recognition*, 43(5):1847 – 1862, 2010.

[40] Jiang Duan and Guoping Qiu. Fast tone mapping for high dynamic range images. In *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, volume 2, pages 847 – 850, August 2004.

[41] Guoping Qiu, Jian Guan, Jian Duan, and Min Chen. Tone mapping for hdr image using optimization a new closed form solution. In *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, volume 1, pages 996 –999, 2006.

[42] Min Chen and Guoping Qiu. A multicurve tone mapping operator for the display of high dynamic range image and video. In *Visual Media Production, 2007. IETCVMP. 4th European Conference on*, pages 1 –7, November 2007.

[43] Chun Hung LIU, Oscar C. Au, P.H.W Wong, and M.C.Kung. Image characteristic oriented tone mapping for high dynamic range images. In *Multimedia and Expo, 2008 IEEE International Conference on*, June 2008.

[44] F. Drago, K. Myszkowski, T. Annen, and N. Chiba. Adaptive logarithmic mapping for displaying high contrast scenes. *Computer Graphics Forum*, 22(3):419–426, 2003.

[45] S.D. Cvetkovic and P.H.N. Klijn. Adaptive tone-mapping transfer functions for high dynamic range video cameras. In *Consumer Electronics, 2008. ICCE 2008. Digest of Technical Papers. International Conference on*, pages 1 –2, January 2008.

[46] Edwin H. Land and John J. McCann. Ligthness and retinex theroy. *Journal of the Optical Society of America*, 61(1), 1970.

[47] G.W. Larson, H. Rushmeier, and C. Piatko. A visibility matching tone reproduction operator for high dynamic range scenes. *Visualization and Computer Graphics, IEEE Transactions on*, 3(4):291 –306, October 1997.

[48] Grzegorz Krawczyk, Karol Myszkowski, and Hans-Peter Seidel. Perceptual effects in real-time tone mapping. In *Proceedings of the 21st spring conference on Computer graphics*, SCCG '05, pages 195–202, New York, NY, USA, 2005. ACM.

[49] Jack Tumblin and Greg Turk. Lcis: a boundary hierarchy for detail-preserving contrast reduction. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '99, pages 83–90, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[50] Frédo Durand and Julie Dorsey. Fast bilateral filtering for the display of high-dynamic-range images. *ACM Trans. Graph.*, 21:257–266, July 2002.

[51] Raanan Fattal, Dani Lischinski, and Michael Werman. Gradient domain high dynamic range compression, 2002.

[52] Joung-Youn Kim, Lee-Sup Kim, and Seung-Ho Hwang. An advanced contrast enhancement using partially overlapped sub-block histogram equalization. *Circuits and Systems for Video Technology, IEEE Transactions on*, 11(4):475 –484, April 2001.

[53] Jack John Erwin Tumblin. *Three methods of detail-preserving contrast reduction for displayed images.* PhD thesis, Georgia Institute of Technology, 1999.

[54] Christophe Schlick. Quantization techniques for visualization of high dynamic range pictures. pages 7–20. Springer-Verlag, 1994.

[55] T Smith and J Guild. The c.i.e. colorimetric standards and their use. *Transactions of the Optical Society*, 33(3):73, 1931.

[56] Wei-Ming Ke, Tsun-Hsien Wang, and Ching-Te Chiu. Hardware-efficient virtual high dynamic range image reproduction. In *Proceedings of the 16th IEEE international conference on Image processing*, ICIP'09, pages 2665–2668, Piscataway, NJ, USA, 2009. IEEE Press.

[57] Ahmet Oguz Akyuz and Erik Reinhard. Color appearance in high-dynamic-range imaging. *Journal of Electronic Imaging*, 15(3):033001, 2006.

[58] L. Coria and P. Nasiopoulos. Using temporal correlation for fast and highdetailed video tone mapping. In *Imaging Systems and Techniques (IST), 2010 IEEE International Conference on*, pages 329 –332, July 2010.

[59] Firas Hassan and Joan Carletta. An fpga-based architecture for a local tone-mapping operator. *Journal of Real-Time Image Processing*, 2:293–308, 2007.

[60] Lavanya Vytla. A real-time implementation of gradient domain high dynamic range compression using a local poisson solver. Master's thesis, University of Akron, USA, 2010.

[61] Gary McTaggart, Chris Green, and Jason Mitchell. High dynamic range rendering in valve's source engine. In *ACM SIGGRAPH 2006 Courses*, SIGGRAPH '06, New York, NY, USA, 2006. ACM.

[62] Hugh S. Fairman, Michael H. Brill, and Henry Hemmendinger. *Color Research & Application*, 22(1):11–23, 1997.

[63] Hugh S. Fairman, Michael H. Brill, and Henry Hemmendinger. How the cie 1931 color-matching functions were derived from wright-guild data. *Color Research & Application*, 22(1):11–23, 1997.

[64] M. Eitz. High dynamic range imaging and tonemapping. http://user.cs.tu-berlin.de/˜eitz/hdr/, 2007.

A XYZ and Yxy

Yxy is based on the CIE1931 color standard [55], and is very simple [62].

Y is here defined as the luminance of a point, while x and y contain the chromatic, or color, of the point. Increasing Y, while leaving x and y constant will make the point brighter while the colors are maintained.

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = [M] \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{26}$$

To go from RGB to Yxy, an intermediate step through XYZ needs to be done. XYZ were defined by CIE in 1931 together with RGB and Yxy. XYZ form the basis of all other color spaces [62, 63].

This first step is a simple matrix multiplication, as seen in equation 26. Big letters in these equations indicate values in XYZ space. Here M indicate a 3x3 matrix.

$$\begin{aligned} Y &= Y \\ x &= \frac{X}{X+Y+Z} \\ y &= \frac{Y}{X+Y+Z} \end{aligned} \tag{27}$$

Going back to RGB from Yxy first involve inverse transform of equation 27 which can be seen in equation 28.

$$\begin{aligned} X &= \frac{x}{y}Y \\ Y &= Y \\ Z &= \frac{1-x-y}{y}Y \end{aligned} \tag{28}$$

However, the only quantity that change during tone mapping is Y, denoted Y'. This simplifies equation 28 to equation 29.

$$\begin{aligned} X &= \frac{Y'}{Y}X \\ Y &= Y' \\ Z &= \frac{Y'}{Y}Z \end{aligned} \tag{29}$$

Equation 29 shows some interesting points. The complete transform to Yxy space is shown to be redundant when only Y is changed. The equation also show some similarities with equation 6 commonly used. The saturation constant has disappeared because it is no longer necessary.

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = [M]^{-1} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{30}$$

Transformation back to RGB is shown in equation 30. M^{-1} is the inverse matrix used in the transform from RGB to XYZ. Without any changes to XYZ, the colors should therefore be the same.

$$\begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} = [M]^{-1} \begin{bmatrix} X\frac{Y'}{Y} \\ Y\frac{Y'}{Y} \\ Z\frac{Y'}{Y} \end{bmatrix} = \begin{bmatrix} R \\ G \\ B \end{bmatrix} * \frac{Y'}{Y} \tag{31}$$

The quantity Y'/Y can be directly multiplied into RGB since the transformations from RGB to XYZ and back are linear. This can be seen in equation 31.

Using the full transformations will give negative numbers and divisions by 0. This is circumvented by setting the RGB triplet to 0 where the incoming Y is 0. If no changes were done to the Y channel, RGB will be multiplied with 1, so color consistency will be maintained if RGB is linear.

B Reinhardts photographic tone mapper

Reinhardts photographic tone mapper [4] is known to produce consistently good result in a variety of situations. It is based on the zone system [1, 2, 3] which were used by photographers before the digital era to change how images looked. Here, the photographers used an automatic dodge and burn operation to either increase or decrease the light of certain parts of the scene.

$$R_i(x,y,s) = \frac{1}{\pi(\alpha_i s)^2} exp(-\frac{x^2+y^2}{(\alpha_i s)^2}) \tag{32}$$

$$V_i(x,y,s) = L(x,y) \otimes R_i(x,y,s) \tag{33}$$

The tone mapper proposed by Reinhardt use eight different Gaussian filter kernels, equation 32 and 33, to simulate the dodge and burn operation. The smallest kernel is 1 by 1 pixel large. Each size increase by a factor of 1.6 and the largest kernel is therefore 42 by 42 large.

The goal is to find the biggest of these which have a fairly even distribution of luminance. The s parameter in both equation 32 and 33 denote the scale.

$$V(x,y,s) = \frac{V_1(x,y,s) - V_2(x,y,s}{2^{\phi}a/s^2 + V_1(x,y,s)} \tag{34}$$

All eight results, with varying s, of this equation is compared to find the biggest area around each pixel with equation 33. Here, V_1 is the center function, while V_2 is the surround function. Essentially, V_1 is set to V_2 when the scale is increased one.

$$|V(x,y,s_m)| < \varepsilon \tag{35}$$

Equation 35 is the used to iterate through all the different size to find the biggest scale which is still less than ε

$$L_d(x,y) = \frac{L(x,y)}{1 + V_1(x,y,s_m(x,y))} \tag{36}$$

Equation 36 is the final equation. This takes the biggest kernel relative to ε and tone map each pixel.

This algorithm was implemented at TU Berlin in 2007 [64]. The code from this source is changed slightly and modified to achieve better running times.

C Array divider

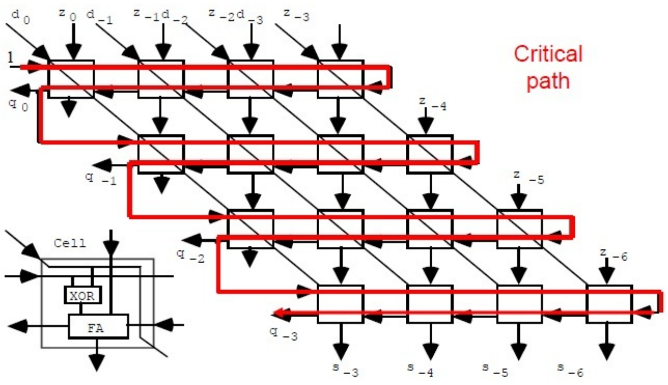

Figure 73: *Structure of a non restoring array divider. Each cell is similar to the one shown at the bottom left. Red path show the critical path.*

The array divider is one structure which is used for division in hardware. Figure 73 shows the structure together with the critical path. The most obvious way of speeding up the division is to pipeline it. This can be done by producing one quotient each clock cycle. The structure would then be divided into horizontal slices with registers between.

Implementing each cell by itself forces the synthesizer to realize the cells combinational. An observation that can be made here, is that each horizontal line of cells is actually a ripple carry adder. If the previous quotient was 1, the divisor is xored, and the carry in is set too 1. This is the same operation which is done when calculating the 2's complement of the divisor.

Each line of the array can therefore be implemented as an adder. If the previous quotient was 1, the divisor should be inverted before adding it. The carry in will then be 1, which complete the 2's complement calculation. The other input to the adder is simply the shifted version of the remainder from the previous line.

By implementing each line as an adder, the synthesizer will be given more room to optimize the additions. The synthesizer can then use FPGA specific functions to speed up the implementation. Tests shows that implementing each cell as a shown in Figure 73 results in a max frequency of about 50 MHz, slightly faster than solving the division combinational. The critical path will also be through the divider. Implementing the divider with adders instead, will move the critical path away from the divider.

Looking at Figure 73, an alternative way of pipelining it can be suggested. By using the principles of carry select adders, each line can be calculated with both situations of quotients. This allows the structure to be pipelined vertically instead of horizontally. The critical path will be drastically reduced, but it will require twice as much hardware.

D RTL view

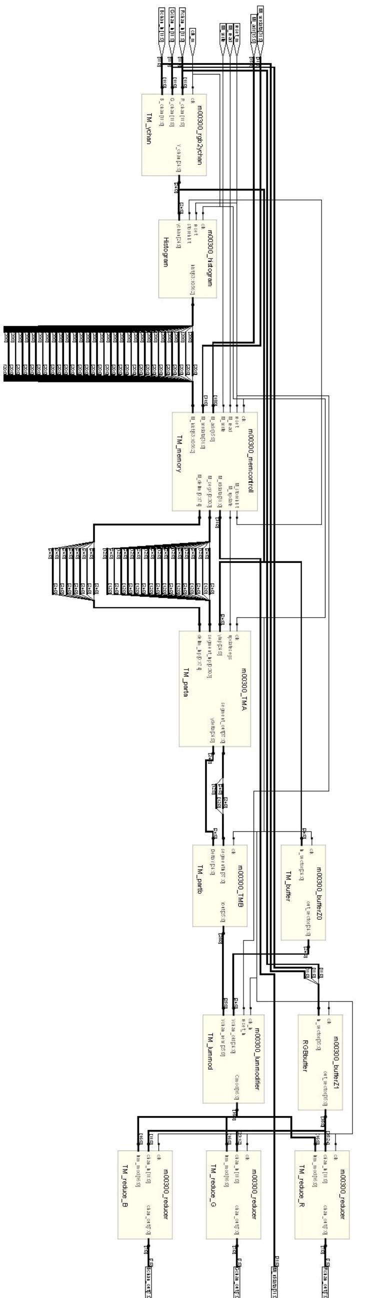

Figure 74: *Showing the RTL view of the implemented system.*

E VHDL code

E.1 Package

```
-------------------------
-- Thesis: Tone mapping in video conferences
--
-- Description:
-- Package containing component declarations and other declarations
-- used in m00300
--
-- Function:
-- void
--
-- Parameters:
-- void
--
--@author: Erik Str mme
-------------------------

LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.numeric_std.all;

package m00300_pkg is

----------
-- Register memory map
constant m00300_ofset  : integer := 0;

---------
-- Types:
--Hist size from 3 ("11") because first value is limited by 11.
type histogram is array (integer range <>) of std_logic_vector(20 downto 0);

type segmentX is record
  alpha : unsigned(12 downto 0); --Rise
  beta  : unsigned(24 downto 0); --Vert ofset
end record;

type segmentarray_type is array(INTEGER range <>) of segmentX;
type delimiter_array is array(INTEGER range <>) of unsigned(24 downto 0);
------------------------
-- Components:

component m00300_toplevel is
  port(
    clk_m     : in std_logic;
    reset_m   : in std_logic;

    Rchan_in  : in std_logic_vector(11 downto 0);
    Gchan_in  : in std_logic_vector(11 downto 0);
    Bchan_in  : in std_logic_vector(11 downto 0);
    --Memory interface
    Mi_write    : in std_logic;
    Mi_read     : in std_logic;
    Mi_adr      : in std_logic_vector(15 downto 0);
    Mi_wrdata   : in std_logic_vector(31 downto 0);
    Mi_rddata   : out std_logic_vector(31 downto 0);

    Rchan_out : out std_logic_vector(7 downto 0);
    Gchan_out : out std_logic_vector(7 downto 0);
    Bchan_out : out std_logic_vector(7 downto 0)
  );
end component;

component m00300_memcontroll is
  port(
    clk        : in std_logic;
    reset      : in std_logic;

    M_write    : in std_logic;
    M_read     : in std_logic;
    M_adr      : in std_logic_vector(15 downto 0);
    M_wrdata   : in std_logic_vector(31 downto 0);
    M_rddata   : out std_logic_vector(31 downto 0);
    --For histogram
    M_storehist : out std_logic;
    M_hist     : in histogram(3 to 502);

    --For curve:
    M_segX     : out segmentarray_type (0 to 7);
    M_delim    : out delimiter_array (0 to 6);
    M_update   : out std_logic


  );
end component;

component m00300_histogram is
  port(
    clk        : in std_logic;
    reset      : in std_logic;
    storehist  : in std_logic;
    ychan      : in std_logic_vector(24 downto 0);
    hist       : out histogram(3 to 502)
  );
end component;

component m00300_rgb2ychan is
  port(
    clk          : in std_logic;

    R_chan       : in std_logic_vector(11 downto 0);
    G_chan       : in std_logic_vector(11 downto 0);
    B_chan       : in std_logic_vector(11 downto 0);

    Y_chan        : out std_logic_vector(24 downto 0)
  );
end component;
```

```vhdl
component m00300_TMA is
  port(
    clk           : in std_logic;
    yinp          : in std_logic_vector(24 downto 0);

    updatesegs    : in std_logic; --High when loading
    segment_inp   : in segmentarray_type (0 to 7); --Segment data (alpha
         beta)
    delim_inp     : in delimiter_array (0 to 6);

    segment_out   : out segmentX; --Alpha and beta for deltay
    ydelta        : out std_logic_vector(24 downto 0) --Deltay
  );
end component;

component m00300_TMB is
  port(
    clk           : in std_logic;
    segmentin     : in segmentX;
    DeltaY        : in std_logic_vector(24 downto 0);

    Yout          : out std_logic_vector(25 downto 0)
  );
end component;

component m00300_lummodifier is
  port(
    clk_in        : in std_logic;
    reset_in      : in std_logic;

    Ychan_old     : in std_logic_vector(24 downto 0);
    Ychan_new     : in std_logic_vector(25 downto 0);

    Cmod          : out std_logic_vector(16 downto 0)
  );
end component;

component m00300_reducer is
  port(
    clk           : in std_logic;

    chan_in       : in std_logic_vector(11 downto 0);
    lum_mod       : in std_logic_vector(16 downto 0);

    chan_out      : out std_logic_vector(7 downto 0)
  );
end component;

component m00300_buffer is
  generic(inpwidth : natural :=24; --Size of input/output
          delayed : natural :=1);  --Delayed: How many clks the output
               should be delayed with
  port(
    clk          : in std_logic;

    in_vector    : in std_logic_vector(inpwidth - 1 downto 0);
    out_vector : out std_logic_vector(inpwidth - 1 downto 0)
  );
end component;

component m00300_arraydivider
generic(size : integer :=8);
port(     clk        : in std_logic;
          reset      : in std_logic;
          dividend   : in std_logic_vector(size - 1 downto 0);
          divisor    : in std_logic_vector(size - 1 downto 0);
          quotient   : out std_logic_vector(size - 1 downto 0)
   );
end component;
end package m00300_pkg;
```

code/m00300_pkg.vhd

E.2 Top level

```
-------------------------
-- Thesis: Tone mapping in video conferences
--
-- Description:
-- Toplevel interface for Tone mapper
--
-- Function:
-- Connecting all the components
--
-- Parameters:
-- none
--
--@author: Erik Str mme
-------------------------

library ieee;
use ieee.std_logic_1164.all;
use IEEE.numeric_std.all;
use work.m00300_pkg.all;

entity m00300_toplevel is
  port(
    clk_m      : in std_logic;
    reset_m    : in std_logic;

    Rchan_in   : in std_logic_vector(11 downto 0);
    Gchan_in   : in std_logic_vector(11 downto 0);
    Bchan_in   : in std_logic_vector(11 downto 0);

    --------Memory interface
    Mi_write   : in std_logic;
    Mi_read    : in std_logic;
    Mi_adr     : in std_logic_vector(15 downto 0);
    Mi_wrdata  : in std_logic_vector(31 downto 0);
    Mi_rddata  : out std_logic_vector(31 downto 0);
    --------------------------------------
    Rchan_out : out std_logic_vector(7 downto 0);
    Gchan_out : out std_logic_vector(7 downto 0);
    Bchan_out : out std_logic_vector(7 downto 0)
  );
end entity m00300_toplevel;

architecture struct of m00300_toplevel is
------------------------------------
--------Interconnecting signals

signal C_segX : segmentarray_type(0 to 7);
signal C_delim : delimiter_array(0 to 6);
signal C_update : std_logic;

signal TMA_segment  : segmentX;
signal TMA_deltay   : std_logic_vector(24 downto 0);
signal TMB_ychan    : std_logic_vector(25 downto 0);
signal TM_mod       : std_logic_vector(16 downto 0);

signal storehist_i : std_logic;
signal hist_i : histogram(3 to 502);
------------------------------------
--------Delay signals:
signal Ychan_org, Ychan_org_delayed : std_logic_vector(24 downto 0);
signal RGB_input, RGB_delayed : std_logic_vector(35 downto 0);
signal Rchan_delayed, Gchan_delayed, Bchan_delayed : std_logic_vector
    (11 downto 0);
------------------------------------
begin

RGB_input <= (Rchan_in & Gchan_in & Bchan_in);
Rchan_delayed <= RGB_delayed(35 downto 24);
Gchan_delayed <= RGB_delayed(23 downto 12);
Bchan_delayed <= RGB_delayed(11 downto 0);

RGBbuffer : component m00300_buffer
generic map( 36, 48)
port map(      clk => clk_m,
               in_vector => RGB_input,
               out_vector => RGB_delayed);

Histogram : component m00300_histogram
port map(    clk => clk_m,
             reset => reset_m,
             storehist => storehist_i,
             ychan => Ychan_org,
             hist => hist_i);


TM_ychan :   component m00300_rgb2ychan
port map(    clk => clk_m,
             R_chan => Rchan_in,
             G_chan => Gchan_in,
             B_chan => Bchan_in,
             Y_chan => Ychan_org);

TM_buffer : component m00300_buffer
```

```vhdl
generic map( 25, 6)
port map(       clk => clk_m,
                in_vector => Ychan_org,
                out_vector => Ychan_org_delayed);

TM_parta : component m00300_TMA
port map( clk => clk_m,
          yinp => Ychan_org,

          updatesegs => C_update,
          segment_inp => C_segX,
          delim_inp => C_delim,

          segment_out => TMA_segment,
          ydelta => TMA_deltay);

TM_partb : component m00300_TMB
port map(clk => clk_m,
         segmentin => TMA_segment,
         DeltaY => TMA_deltay,
         Yout => TMB_ychan);

TM_lummod : component m00300_lummodifier
port map(clk_in     => clk_m,
         reset_in   => reset_m,
         Ychan_old  => Ychan_org_delayed,
         Ychan_new  => TMB_ychan,
         Cmod => TM_mod);

TM_reduce_R : m00300_reducer
port map( clk => clk_m,
          chan_in => Rchan_delayed,
          lum_mod => TM_mod,
          chan_out => Rchan_out);

TM_reduce_G : m00300_reducer
port map( clk => clk_m,
          chan_in => Gchan_delayed,
          lum_mod => TM_mod,
          chan_out => Gchan_out);

TM_reduce_B : m00300_reducer
port map( clk => clk_m,
          chan_in => Bchan_delayed,
          lum_mod => TM_mod,
          chan_out => Bchan_out);

TM_memory : m00300_memcontroll
port map( clk => clk_m,
          reset => reset_m,
          M_write => Mi_write,
          M_read => Mi_read,
          M_adr => Mi_adr,
          M_wrdata => Mi_wrdata,
          M_rddata => Mi_rddata,
          M_storehist => storehist_i,
          M_hist => hist_i,
          M_segX => C_segX,
          M_delim => C_delim,
          M_update => C_update);
end architecture;
```

code/m00300_toplevel.vhd

E.3 rgb2ychan

```
-------------------------
-- Thesis: Tone mapping in video conferences
--
-- Description:
-- Component for finding luminance from RGB triplet.
--
--
-- Function:
-- Transform RGB to Lum
--
-- Parameters:
-- None
--
--@author: Erik Str mme
-------------------------
library ieee;
use ieee.std_logic_1164.all;
use IEEE.numeric_std.all;
use work.m00300_pkg.all;

entity m00300_rgb2ychan is
  port(
    clk         : in std_logic;

    R_chan      : in std_logic_vector(11 downto 0);
    G_chan      : in std_logic_vector(11 downto 0);
    B_chan      : in std_logic_vector(11 downto 0);

    Y_chan       : out std_logic_vector(24 downto 0)
  );
end entity m00300_rgb2ychan;

architecture struct of m00300_rgb2ychan is
constant coeff1   : natural := 1742;
constant coeff2   : natural := 5858;
constant coeff3   : natural := 591;

signal Ychan_calcRed   : unsigned(25 downto 0);
signal Ychan_calcGreen : unsigned(25 downto 0);
signal Ychan_calcBlue  : unsigned(25 downto 0);
signal Ychan_calcsum : unsigned(25 downto 0);

begin

Ychan_calcsum <= (Ychan_calcRed) + (Ychan_calcGreen) + (Ychan_calcBlue
    );
```

```vhdl
clocked : process(clk)
  begin
    if(rising_edge(clk)) then
      Ychan_calcRed <= (resize(unsigned(R_chan),13) * coeff1);
     Ychan_calcGreen <= (resize(unsigned(G_chan),13) * coeff2);
      Ychan_calcBlue <= (resize(unsigned(B_chan),13) * coeff3);

      Y_chan <= std_logic_vector(Ychan_calcsum(24 downto 0));
  end if;
end process;
end architecture;
```

code/m00300_rgb2ychan.vhd

E.4 TMA

```vhdl
-------------------------
-- Thesis: Tone mapping in video conferences
--
-- Description:
-- Module for tone mapping luminance.
-- See my thesis
--
-- Function:
-- First tonemapping module (out of two)
-- This will only find which segment it should use.
-- Also, this component will update the segments into registers
--
-- Parameters:
-- none
--
--@author: Erik Str mme
-------------------------

library ieee;
use ieee.std_logic_1164.all;
use IEEE.numeric_std.all;
use work.m00300_pkg.all;

entity m00300_TMA is
  port(
    clk         : in std_logic;
    yinp        : in std_logic_vector(24 downto 0);

    ----Loading new segments:
    updatesegs  : in std_logic;                    --High when loading
         new segments
    segment_inp : in segmentarray_type (0 to 7);   --Segment data (
        alpha beta)
    delim_inp   : in delimiter_array (0 to 6);     --Delimiter (YsegX)
    ------------------------
    segment_out : out segmentX;                     --Alpha and beta
        for Deltay sent to part 2 of TM
    ydelta      : out std_logic_vector(24 downto 0) --Deltay
  );
end entity m00300_TMA;


architecture struct of m00300_TMA is

--Clocked inputs:
signal delim_inp_reg : delimiter_array (0 to 6);
signal segment_inp_reg : segmentarray_type (0 to 7);
signal updatesegs_reg : std_logic;
signal yinp_reg : std_logic_vector(24 downto 0);

signal delimiter_in, delimiter_out : delimiter_array (0 to 6);
signal segmentarray_in, segmentarray_out : segmentarray_type (0 to 7);

signal segment_out_del : segmentX;
signal ydelta_l : signed(25 downto 0);
signal yinp_reg_d : std_logic_vector(24 downto 0);


type comp_arr is array (integer range <>) of signed(25 downto 0);
signal comparevector,comparevector_d : comp_arr(0 to 6);
signal comparevector_MSBs : std_logic_vector(6 downto 0);
begin

with comparevector_MSBs select
   ydelta_l <= signed('0' & yinp_reg_d) when "1111111",
            comparevector_d(0) when "1111110",
            comparevector_d(1) when "1111100",
            comparevector_d(2) when "1111000",
            comparevector_d(3) when "1110000",
            comparevector_d(4) when "1100000",
            comparevector_d(5) when "1000000",
            comparevector_d(6) when others;

with comparevector_MSBs select
  segment_out_del <= segmentarray_out(0) when "1111111",
                     segmentarray_out(1) when "1111110",
                     segmentarray_out(2) when "1111100",
                     segmentarray_out(3) when "1111000",
                     segmentarray_out(4) when "1110000",
                     segmentarray_out(5) when "1100000",
                     segmentarray_out(6) when "1000000",
                     segmentarray_out(7) when others; --"000000";

segmentarray_in <= segment_inp_reg when updatesegs_reg = '1' else
     segmentarray_out;
delimiter_in <= delim_inp_reg when updatesegs_reg = '1' else
     delimiter_out;

compare : for i in 0 to 6 generate
  comparevector(i) <= signed('0' & yinp_reg) - signed('0' &
      delimiter_out(i));
end generate compare;

msbs : for i in 0 to 6 generate
  comparevector_MSBs(i) <= comparevector_d(i)(25);
```

```vhdl
end generate msbs;

clocked : process(clk, delimiter_in, segmentarray_in, yinp)
begin
if(rising_edge(clk)) then
    --Clocked inputs:
    updatesegs_reg <= updatesegs;
    segment_inp_reg <= segment_inp;
    delim_inp_reg <= delim_inp;

    yinp_reg <= yinp;
    yinp_reg_d <= yinp_reg;

    --Registers:
    segmentarray_out <= segmentarray_in;
    delimiter_out <= delimiter_in;

    comparevector_d <= comparevector;

    --Clocked outputs:
    ydelta <= std_logic_vector(ydelta_l(24 downto 0));
    segment_out <= segment_out_del;
end if;
end process;
end architecture;
```

code/m00300_TMA.vhd

E.5 TMB

```vhdl
-------------------------
-- Thesis: Tone mapping in video conferences
--
-- Description:
-- Module for tone mapping luminance.
--
-- Function:
-- Second tonemapping module (out of two)
-- This will calculate the output value depending on which segment
    input
-- it get from part 1 of tone mapper.
--
-- Parameters:
-- None
--
--@author: Erik Str mme
-------------------------

library ieee;
use ieee.std_logic_1164.all;
use IEEE.numeric_std.all;
use work.m00300_pkg.all;

entity m00300_TMB is
```

```
  port(
    clk           : in  std_logic;
    segmentin     : in  segmentX;
    DeltaY        : in  std_logic_vector(24 downto 0);

    Yout          : out  std_logic_vector(25 downto 0)
  );
end entity m00300_TMB;


architecture struct of m00300_TMB is

signal Ycalc1,Ycalc2   : unsigned(30 downto 0);
signal Y_ugs_inp : unsigned(16 downto 0);
signal beta_delayed : unsigned(24 downto 0);

begin

Y_ugs_inp <= unsigned(DeltaY(24 downto 8));

clocked : process(clk)
begin
  if(rising_edge(clk)) then
    beta_delayed <= segmentin.beta;
    Ycalc1 <= resize((segmentin.alpha * Y_ugs_inp), 31);
    Ycalc2 <= Ycalc1 + beta_delayed;

    --Clock outputs
    if (std_logic_vector(Ycalc2(30 downto 26)) = "00000") then
       Yout <= std_logic_vector(Ycalc2(25 downto 0));
    else
       Yout <= (others=>'1');
    end if;
  end if;
end process;
end architecture;
```

code/m00300_TMB.vhd

E.6 Lum modifier

```
--------------------------
-- Thesis: Tone mapping in video conferences
--
-- Description:
-- Component for finding how much each pixel should be modified with.
-- Probably going to change the divider here
--
-- Function:
--  Ynew / Yold = Cmod
--
-- Parameters:
-- None
--
--@author: Erik Str mme
```

```vhdl
--------------------------

LIBRARY ieee;
library ieee;
use ieee.std_logic_1164.all;
use IEEE.numeric_std.all;
use work.m00300_pkg.all;

entity m00300_lummodifier is
  port(
    clk_in      : in std_logic;
    reset_in    : in std_logic;
    Ychan_old   : in std_logic_vector(24 downto 0);
    Ychan_new   : in std_logic_vector(25 downto 0);

    Cmod        : out std_logic_vector(16 downto 0)
  );
end entity;

architecture struct of m00300_lummodifier is

signal Cmod_max  : std_logic_vector(20 downto 0);
signal Cmod_full : std_logic_vector(37 downto 0);

signal Ychan_new_extended : std_logic_vector(37 downto 0);
signal Ychan_old_extended : std_logic_vector(37 downto 0);
begin

arraydivider : component m00300_arraydivider
generic map(38)
port map(     clk => clk_in,
              reset => reset_in,
              dividend => Ychan_new_extended,
              divisor => Ychan_old_extended,
              quotient => Cmod_full);


Ychan_new_extended(37 downto 12) <= (Ychan_new);
Ychan_new_extended(11 downto 0)  <= (others=>'0');
Ychan_old_extended(37 downto 25) <= (others=>'0'); --padding
Ychan_old_extended(24 downto 0)  <= Ychan_old;

Cmod_max <= (others=>'0');
Cmod <= Cmod_full(16 downto 0) when Cmod_max = Cmod_full(37 downto 17)
     else (others=>'1');

--
--
--process (clk)
--begin
--  if(rising_edge(clk)) then
--    --Cmod_full <= unsigned(Ychan_extended) / unsigned(
    Ychan_old_extended);
--  end if;
--end process;

end architecture;
```

code/m00300_lummodifier.vhd

```vhdl
-------------------------
-- Thesis: Tone mapping in video conferences
--
-- Description:
-- Pipelined Array divider. Delay is the same as size plus one.
--
-- Reference
-- Computer Architecture: Algorithms and hardware designs
-- by Behrooz Parhami
-- http://www.ece.ucsb.edu/~parhami/text_comp_arch.htm
--
-- Function:
-- Dividend / Divisor = Quotient
--
-- Parameters:
-- size : how many bits dividend and divisor is.
--
--@author: Erik Str mme
-------------------------

LIBRARY ieee;
library ieee;
use ieee.std_logic_1164.all;
use IEEE.numeric_std.all;
use work.m00300_pkg.all;
use work.m00300_divider_pkg.all;

entity m00300_arraydivider is
  generic(size : natural := 8);
  port(
    clk      : in std_logic;
    reset    : in std_logic;
    dividend : in std_logic_vector(size - 1 downto 0);
    divisor  : in std_logic_vector(size - 1 downto 0);

    quotient : out std_logic_vector(size - 1 downto 0)
  );
end entity m00300_arraydivider;

architecture struct of m00300_arraydivider is

type rem_array is array (integer range <>) of std_logic_vector(size -
    1 downto 0);
signal rem_out : rem_array(size - 1 downto 0);

signal quotient_slices_out : std_logic_vector( size - 1 downto 0);
--signal quotient_slices_in  : std_logic_vector( size - 1 downto 0);


type datapacket_t is record
  divisor : std_logic_vector(size - 1 downto 0);
  dividend : std_logic_vector(size - 1 downto 0);
  rem_in : std_logic_vector(size - 1 downto 0);
```

```vhdl
--  rem_out: std_logic_vector(size - 1 downto 0);
--  quotient_slices_out : std_logic;
  quotient_slices_in : std_logic;
  quotients : std_logic_vector(size − 1 downto 0);

end record datapacket_t;
type datapacket_t_array is array (integer range <>) of datapacket_t;


signal packet_in, packet_out : datapacket_t_array(0 to size − 1);


signal divisor_input, dividend_input, quotient_output :
    std_logic_vector(size − 1 downto 0);



begin


---Load pipe:
packet_in(0).divisor <= divisor_input;
packet_in(0).dividend <= dividend_input;
packet_in(0).rem_in(size − 2 downto 0) <= (others =>'0');
packet_in(0).rem_in(size − 1)   <= dividend_input(size − 1);
packet_in(0).quotient_slices_in <= '1';
packet_in(0).quotients <= (others=>'0');
--This results in one load stage.. Can be done better probably.

generate_slices : for i in 0 to size−1 generate
  slicex : component m00300_divider_slice
  generic map(size)
  port map(
    divisor_in => packet_out(i).divisor,
    quot_in    => packet_out(i).quotient_slices_in,
    remainder_slice_in => packet_out(i).rem_in,
    remainder_slice_out => rem_out(i),
    quot_out => quotient_slices_out(i)
  );
end generate generate_slices;


load_quotients : for i in 0 to size − 2 generate
  packet_in(i+1).quotients <= packet_out(i).quotients(size−2 downto 0)
       & quotient_slices_out(i); --Shift in new quotients.
  packet_in(i+1).quotient_slices_in <= quotient_slices_out(i); --
      Propogation of quotients  --Same same.
end generate load_quotients;

propogate_rems : for i in 1 to size − 1 generate
  packet_in(i).rem_in(size−2 downto 0) <= rem_out(i−1)(size − 1 downto
       1);
  packet_in(i).rem_in(size−1) <= packet_out(i−1).dividend(size−1−i);
end generate propogate_rems;

propogate_packet_constants : for i in 1 to size − 1 generate
  packet_in(i).dividend <= packet_out(i − 1).dividend;
  packet_in(i).divisor <= packet_out(i - 1).divisor;
end generate propogate_packet_constants;



quotient_output <= packet_out(size-1).quotients(size - 2 downto 0) &
    quotient_slices_out(size-1);


clocking : process(clk, reset) is
begin
  if(rising_edge(clk)) then
    if (reset = '1') then

       reset_loop : for i in 0 to size-1 loop
          packet_out(i).dividend <= (others=>'0');
          packet_out(i).divisor <= (others=>'0');
          packet_out(i).rem_in <= (others=>'0');
          packet_out(i).quotient_slices_in <= '0';
          packet_out(i).quotients <= (others=>'0');
      end loop reset_loop;
      else
      --Registers:
      packet_out <= packet_in;

      --Clocking input:
      divisor_input <= divisor;
      dividend_input <= dividend;
      quotient <= quotient_output;
      end if;
  end if;
end process;
end architecture;
```

code/m00300_arraydivider.vhd

```
-------------------------
-- Thesis: Tone mapping in video conferences
--
-- Description:
-- One line of array divider.
-- This returns the remainder and quout of current line.
-- Both should be propogated.
--
-- Function:
--
-- Parameters:
-- size : how many bits dividend and divisor is.
--
--@author: Erik Str mme
-------------------------


library ieee;
use ieee.std_logic_1164.all;
use IEEE.numeric_std.all;
use work.m00300_pkg.all;
use work.m00300_divider_pkg.all;

entity m00300_divider_slice is
  generic(size : natural := 8);
  port(
    divisor_in            : in std_logic_vector(size - 1 downto 0);
    quot_in               : in std_logic;
    remainder_slice_in    : in std_logic_vector(size - 1 downto 0);
    remainder_slice_out   : out std_logic_vector(size - 1 downto 0);
    quot_out              : out std_logic
  );
end entity m00300_divider_slice;

architecture struct of m00300_divider_slice is


signal add_result   : unsigned(size downto 0);
signal divisor_calc, quotient_in_calc : unsigned(size-1 downto 0);
signal remainder_slice_in_calc : unsigned(size - 1 downto 0);
begin


--Flip remainder in:
flip : for i in 0 to size-1 generate
  remainder_slice_in_calc(i) <= remainder_slice_in(size-1-i);
end generate flip;

fliprem : for i in 0 to size-1 generate
  remainder_slice_out(i) <= add_result(size-1-i);
end generate fliprem;


--    remainder_slice_out <= std_logic_vector(add_result(size - 1
    downto 0));


    divisor_calc <= unsigned(not(divisor_in)) when quot_in = '1' else
        unsigned(divisor_in);
    quotient_in_calc(size-1 downto 1) <= (others=>'0');
    quotient_in_calc(0) <= quot_in;
    add_result <= (divisor_calc) + ('0'& unsigned(
        remainder_slice_in_calc)) + quotient_in_calc;
    quot_out <= add_result(size);

end architecture;

-- Under this is original design with controlled FA blocks
--architecture struct of m00300_divider_slice is
--signal quotput_slice : std_logic_vector(size downto 0);
--
--
--procedure onecell
--(signal co, s : out std_logic;
--signal r, d, ci, q : in std_logic) is
--begin
--  co <= (r and (d xor q)) or (r and ci) or ((d xor q) and ci);
--  s <= r xor d xor q xor ci;
```

```
--end onecell;
--
--
--
--begin
--quot_out <= quotput_slice(0);
--quotput_slice(size) <= quot_in;
--
--oneslice : for i in 0 to size - 1 generate
--     onecell(
--     quotput_slice(i),
--     remainder_slice_out(i),
--
--     remainder_slice_in(i),
--     divisor_in(size - 1 - i),
--     quotput_slice(i+1),
--     quot_in);
--end generate oneslice;
--end architecture;
```

code/m00300_divider_slice.vhd

E.7 Reducer

```
-------------------------
-- Thesis: Tone mapping in video conferences
--
-- Description:
-- Component for multiplying one of the RGB channels with the lum
    modifier.
-- One component per channel
--
-- Function:
-- Multiply channel by lum mod.
-- Set to max if it is out of bounds.
--
-- Parameters:
-- None
--
--@author: Erik Str mme
-------------------------
library ieee;
use ieee.std_logic_1164.all;
use IEEE.numeric_std.all;
use work.m00300_pkg.all;

entity m00300_reducer is
  port(
    clk          : in std_logic;
    chan_in      : in std_logic_vector(11 downto 0);
    lum_mod      : in std_logic_vector(16 downto 0);

    chan_out     : out std_logic_vector(7 downto 0)
  );
end entity m00300_reducer;
```

```

architecture struct of m00300_reducer is
signal chan_full : unsigned(28 downto 0);
signal chan_sel : std_logic_vector(7 downto 0);

signal mula : unsigned(11 downto 0);
signal mulb : unsigned(16 downto 0);
begin

chan_sel <= std_logic_vector(chan_full(23 downto 16)) when (chan_full
    (27 downto 24) = "0000") else (others=>'1');
chan_full <= mula * mulb;

clocked : process(clk)
begin
  if(rising_edge(clk)) then
    mula <= unsigned(chan_in);
    mulb <= unsigned(lum_mod);

    --Register output:
    chan_out <= chan_sel;
  end if;
end process;
end architecture;
```

code/m00300_reducer.vhd

E.8 Buffer

```
-------------------------
-- Thesis: Tone mapping in video conferences
--
-- Description:
-- Pipeline element with variable delay and variable size
--
-- Function:
-- Delay the input by DELAYED clk'events
--
-- Parameters:
-- inpwidth = size of inputs
-- delayed  = depth of buffer
--
--@author: Erik Str mme
-------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
USE work.m00300_pkg.all;

entity m00300_buffer is
  generic(inpwidth : natural :=24; --Size of input/output
          delayed : natural :=1);  --Delayed: How many clks the output
               should be delayed with
  port(
```

```vhdl
    clk         : in std_logic;

    in_vector   : in std_logic_vector(inpwidth - 1 downto 0);
    out_vector  : out std_logic_vector(inpwidth - 1 downto 0)
  );
end entity m00300_buffer;

architecture struct of m00300_buffer is

type delayarray is array(0 to delayed - 1) of std_logic_vector(
    inpwidth - 1 downto 0);

signal arr_in, arr_out : delayarray;
begin


gen_arrays : for i in 1 to delayed-1 generate
arr_in(i) <= arr_out(i-1);
end generate gen_arrays;
arr_in(0) <= in_vector;
out_vector <= arr_out(delayed-1);

clocked : process(clk)
begin
  if(rising_edge(clk)) then
     arr_out <= arr_in;
  end if;
end process;

end architecture;
```

code/m00300_buffer.vhd

E.9 Histogram

```vhdl
-------------------------
-- Thesis: Tone mapping in video conferences
--
-- Description:
-- Component for collecting Y in a histogram.
--
-- Function:
-- Count Y in proper index
--
-- Parameters:
-- None
--
--@author: Erik Str mme
-------------------------

library ieee;
use ieee.std_logic_1164.all;
use IEEE.numeric_std.all;
use work.m00300_pkg.all;
```

```vhdl
entity m00300_histogram is
  port(
    clk       : in std_logic;
    reset     : in std_logic;
    storehist : in std_logic;
    ychan     : in std_logic_vector(24 downto 0);

    hist      : out histogram(3 to 502)
  );
end entity m00300_histogram;

architecture behavior of m00300_histogram is
signal ychan_r : std_logic_vector(8 downto 0);
signal storehist_r : std_logic;
signal hist_in_curr, hist_out_curr : histogram(3 to 502);

begin

--Add registered Y to hist:
--Smallest value to count is 3 ("11"), so hist is shifted.
addinc : for i in 3 to 502 generate
  hist_in_curr(i) <= std_logic_vector(unsigned(hist_out_curr(i)) + 1)
  when ychan_r = std_logic_vector(to_unsigned(i, 9))
  else hist_out_curr(i);
end generate addinc;

hist <= hist_out_curr;

clocked : process(clk) is
begin
  if(rising_edge(clk)) then
    if(reset = '1') then
      --Reset everything
      resetloop : for i in 0 to 499 loop
        hist_out_curr(i+3) <= (others=>'0');
      end loop;
    else
      --Register inputs:
      ychan_r <= ychan(24 downto 16);
      storehist_r <= storehist;
      --Register outputs:
      if(storehist_r = '1') then
        local_reset : for i in 0 to 499 loop
          hist_out_curr(i+3) <= (others=>'0');
        end loop;
      else
          hist_out_curr <= hist_in_curr;
      end if;
    end if;
  end if;
end process;
end architecture;
```

code/m00300_histogram.vhd

E.10 Memory interface

```vhdl
-------------------------
-- Thesis: Tone mapping in video conferences
--
-- Description:
-- Bus interface
--
-- Function:
-- Recieving data from the bus if the adress is apropriate.
--
-- Parameters:
-- None
--
--@author: Erik Str mme
-------------------------

library ieee;
use ieee.std_logic_1164.all;
use IEEE.numeric_std.all;
use work.m00300_pkg.all;

entity m00300_memcontroll is
  port(
    clk            : in std_logic;
    reset          : in std_logic;
    --External Interface:
    M_read         : in std_logic;
    M_write        : in std_logic;
    M_adr          : in std_logic_vector(15 downto 0);
    M_wrdata       : in std_logic_vector(31 downto 0);
    M_rddata       : out std_logic_vector(31 downto 0);
    M_storehist    : out std_logic;

    --Internal interface:
    M_hist         : in histogram(3 to 502);
    M_segX         : out segmentarray_type (0 to 7);
    M_delim        : out delimiter_array (0 to 6);
    M_update       : out std_logic
  );
end entity m00300_memcontroll;

architecture struct of m00300_memcontroll is

---Register signals:
signal M_update_in, M_storehist_in : std_logic;
signal segX_arr_in, segX_arr_out : segmentarray_type (0 to 7);
signal del_arr_in, del_arr_out : delimiter_array (0 to 6);

---Signals:
signal M_write_d, M_read_d : std_logic;
signal M_adr_d     : std_logic_vector(15 downto 0);
signal M_wrdata_d  : std_logic_vector(31 downto 0);
signal adress_in_range : std_logic;
signal M_hist_d, M_hist_din : histogram(0 to 499);

constant m00300_last_adress : integer := m00300_ofset + 523;
```

```vhdl
begin
adress_in_range <= '0' when to_integer(unsigned(M_adr_d)) >
    m00300_last_adress or to_integer(unsigned(M_adr_d)) < m00300_ofset
     else '1';
-------Load alpha and beta:
load_logic : for i in 0 to 7 generate
    segX_arr_in(i).alpha <= unsigned(M_wrdata_d(12 downto 0)) when
        adress_in_range = '1' and to_integer(unsigned(M_adr_d)) = (
        m00300_ofset + i) and M_write_d = '1'
                                                    else segX_arr_out(i).
                                                        alpha;
    segX_arr_in(i).beta <= unsigned(M_wrdata_d(24 downto 0))  when
        adress_in_range = '1' and to_integer(unsigned(M_adr_d)) = (
        m00300_ofset + 8 + i) and M_write_d = '1'
                                                    else segX_arr_out(i).
                                                        beta;
end generate load_logic;


--------Load delimiting values:
load_dels : for i in 0 to 6 generate
    del_arr_in(i) <= unsigned(M_wrdata_d(24 downto 0)) when
        adress_in_range = '1' and to_integer(unsigned(M_adr_d)) = (
        m00300_ofset + 16 + i) and M_write_d = '1'
                                             else del_arr_out(i);
end generate load_dels;

M_update_in <= '1' when adress_in_range = '1' and to_integer(unsigned(
    M_adr_d)) = (m00300_ofset + 23) and to_integer(unsigned(M_wrdata_d
    )) = 0 else '0';
M_storehist_in <= '1' when adress_in_range = '1' and to_integer(
    unsigned(M_adr_d)) = (m00300_ofset + 23) and to_integer(unsigned(
    M_wrdata_d)) = 1 else '0';
M_hist_din <= M_hist_d;

clocked : process(clk, reset)
begin
  --Clocking registers:
  if(rising_edge(clk)) then
    M_write_d <= M_write;
    M_read_d <= M_read;
    M_adr_d <= M_adr;
    M_wrdata_d <= M_wrdata;

    if(reset = '1') then

    resetloop : for i in 0 to 7 loop
      segX_arr_out(i).alpha <= (others=>'0');
      segX_arr_out(i).beta <= (others=>'0');
      M_segX(i).alpha <= (others=>'0');
      M_segX(i).beta <= (others=>'0');
      M_update <= '0';
    end loop resetloop;

    resetloop2: for i in 0 to 6 loop
      M_delim(i) <= (others=>'1');
      del_arr_out(i) <= (others=>'1');
    end loop resetloop2;

    histloop : for i in 0 to 499 loop
      M_hist_d(i) <= (others=>'0');
    end loop histloop;

    else
      segX_arr_out <= segX_arr_in;
      del_arr_out <= del_arr_in;

      M_storehist <= M_storehist_in;
      M_update <= M_update_in;
      M_segX <= segX_arr_in;
      M_delim <= del_arr_in;

      --Buffer histogram:
      if(M_storehist_in = '1') then
        normalize : for i in 0 to 499 loop
          M_hist_d(i) <= M_hist(i + 3);
        end loop normalize;
      else
        M_hist_d <= M_hist_din;
      end if;

      -------Write histogram:

      if(adress_in_range = '1' and M_read_d = '1') then
          M_rddata(20 downto 0) <= M_hist_d(to_integer(unsigned(
              M_adr_d))-24);
          M_rddata(31 downto 21) <= (others=>'0');
        else
          M_rddata <= (others=>'0');
        end if;
    end if;
  end if;
end process;
end architecture;
```

code/m00300_memcontroll.vhd

E.11 Test bench

All the test benches follow the same structure seen here.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
USE work.m00300_pkg.all;

use ieee.std_logic_textio.all;
use std.textio.all;

ENTITY m00300_TB_rgb2ychan IS
END m00300_TB_rgb2ychan;

ARCHITECTURE behavior OF m00300_TB_rgb2ychan IS
```

```vhdl
signal T_clk    :   std_logic := '0';
signal T_R      :   std_logic_vector(11 downto 0):=(others=>'0');
signal T_G      :   std_logic_vector(11 downto 0):=(others=>'0');
signal T_B      :   std_logic_vector(11 downto 0):=(others=>'0');
signal T_Y      :   std_logic_vector(24 downto 0);

constant clk_period : time := 40 ns;
constant num_delay : integer := 2;
BEGIN
    dut: component m00300_rgb2ychan
    PORT MAP(
      clk => T_clk,

      R_chan => T_R,
      G_chan => T_G,
      B_chan => T_B,

      Y_chan => T_Y
    );

clk_proc : process
begin
  T_clk <= not T_clk;
  wait for clk_period/2;
end process;

excite_proc : process
file file_red   : text open read_mode is "vectors\m00300_red.txt"; --
    12 bit
file file_green : text open read_mode is "vectors\m00300_green.txt";
file file_blue  : text open read_mode is "vectors\m00300_blue.txt";

variable line_red, line_green, line_blue : line;
variable red, green, blue : std_logic_vector(11 downto 0);

begin
  wait until ( T_clk'event and T_clk = '1');
  while not endfile(file_red) loop
    READLINE(file_red, line_red); --Dump line into L
    READLINE(file_green, line_green);
    READLINE(file_blue, line_blue);

    READ(line_red, Red); --Next argument into red
    READ(line_green, Green);
    READ(line_blue, Blue);

    T_R <= Red(11 downto 0); --Assign value to interface
    T_G <= Green(11 downto 0);
    T_B <= Blue(11 downto 0);

  wait until ( T_clk'event and T_clk = '1');
  end loop;
 wait;
end process;

```

```vhdl
verify_dut : process
file file_ychan  : text open read_mode is "vectors\m00300_ychan.txt";
    --25 bit
variable line_ychan : line;
variable ychan_ver : std_logic_vector(24 downto 0);

begin
  wait until ( T_clk'event and T_clk = '1');
  delayloop : for i in 1 to num_delay loop
    wait until ( T_clk'event and T_clk = '1');
  end loop delayloop;

  while not endfile(file_ychan) loop
    wait for clk_period/4;
    READLINE(file_ychan, line_ychan);
    READ(line_ychan, ychan_ver);

    assert T_Y = ychan_ver
      report "Calculation done wrong. T_Y: " & Integer'image(
          to_integer(unsigned(T_Y))) & " while expected value is: "
      & Integer'image(to_integer(unsigned(ychan_ver))) & " according
          to testfile"
      severity  warning;

    wait until ( T_clk'event and T_clk = '1');
  end loop;
  wait;

end process;
end architecture;
```

code/m00300_TB_rgb2ychan.vhd

E.11.1 Test bench for top level

```vhdl
-------------------------
-- Thesis: Tone mapping in video conferences
--
-- Description:
-- Testbench for m00300_topleve
--
-- Function:
-- Testing the module to see if it works.
-- Do visual comparison test to see if T_Xchan_outs are
-- according to test file. This will be changed if the
-- divider is changed.
--
--
-- Parameters:
-- none
--
--@author: Erik Str mme
-------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
USE work.m00300_pkg.all;

use ieee.std_logic_textio.all;
use std.textio.all;

ENTITY m00300_TB_TOPLEVEL IS
END m00300_TB_TOPLEVEL;

ARCHITECTURE behavior OF m00300_TB_TOPLEVEL IS

signal T_clk          :   std_logic := '0';
signal T_reset        :   std_logic := '0';
signal T_Rchan_in     :   std_logic_vector(11 downto 0);
signal T_Gchan_in     :   std_logic_vector(11 downto 0);
signal T_Bchan_in     :   std_logic_vector(11 downto 0);

signal T_Rchan_out    :   std_logic_vector(7 downto 0);
signal T_Gchan_out    :   std_logic_vector(7 downto 0);
signal T_Bchan_out    :   std_logic_vector(7 downto 0);

signal T_m_read     :   std_logic;
signal T_m_write    :   std_logic := '0';
signal T_m_adr      :   unsigned(15 downto 0);
signal T_m_data     :   unsigned(31 downto 0);
signal T_m_rddata   :   std_logic_vector(31 downto 0);

signal TB_segmentarray : segmentarray_type(0 to 7);
signal TB_delimter_array : delimiter_array(0 to 6);
signal TB_setupcomplete : std_logic :='0';
signal TB_histogram : histogram(0 to 499);
signal TB_starthistogramcheck : std_logic :='0';


constant clk_period : time := 40 ns;
constant num_delay : integer :=50;

BEGIN
    dut: component m00300_toplevel
    port map(
    clk_m      => T_clk,
    reset_m    => T_reset,

    Rchan_in   => T_Rchan_in,
    Gchan_in   => T_Gchan_in,
    Bchan_in   => T_Bchan_in,

    Mi_write   => T_m_write,
    Mi_read    => T_m_read,
    Mi_adr     => std_logic_vector(T_m_adr),
    Mi_wrdata  => std_logic_vector(T_m_data),
    Mi_rddata  => T_m_rddata,

    Rchan_out  => T_Rchan_out,
    Gchan_out  => T_Gchan_out,
    Bchan_out  => T_Bchan_out
  );
```

```vhdl


clk_proc : process
begin
  T_clk <= not T_clk;
  wait for clk_period/2;
end process;

excite_proc : process
file file_alpha     : text open read_mode is "vectors\m00300_alpha.txt
    ";--13
file file_beta      : text open read_mode is "vectors\m00300_beta.txt"
    ; --25
file file_delim     : text open read_mode is "vectors\
    m00300_segdelimter.txt";--25

file file_red      : text open read_mode is "vectors\m00300_red.txt";
    --12
file file_green    : text open read_mode is "vectors\m00300_green.txt"
    ;--12
file file_blue     : text open read_mode is "vectors\m00300_blue.txt";
     --12


variable line_alpha, line_beta, line_delim, line_yinp : line;

variable alpha     : std_logic_vector(12 downto 0);
variable beta      : std_logic_vector(24 downto 0);
variable delimiter : std_logic_vector(24 downto 0);

variable counter   : unsigned(2 downto 0) := "000";

variable line_red, line_green, line_blue : line;
variable v_red, v_green, v_blue : std_logic_vector(11 downto 0);

variable v_adress  : integer :=0;
begin

  T_Rchan_in <= (others=>'0');
  T_Gchan_in <= (others=>'0');
  T_Bchan_in <= (others=>'0');
  T_m_read <= '0';
  T_m_write <= '0';
  T_m_adr <= (others=>'0');
  T_m_data <= (others=>'0');


  resetloop1 : for i in 0 to 6 loop
    TB_delimter_array(i)  <= (others=>'1');
  end loop;


  resetloop : for i in 0 to 7 loop

    TB_segmentarray(i).alpha <= (others=>'0');
    TB_segmentarray(i).beta <= (others=>'0');
  end loop;
```

```
READLINE(file_delim, line_delim);
while not endfile(file_delim) loop
  READLINE(file_delim, line_delim);
  READ(line_delim, delimiter);
  TB_delimter_array(to_integer(counter)) <= unsigned(delimiter);
  counter := counter + 1;
end loop;
counter := "000";


while not endfile(file_alpha) loop --Load segments:
  READLINE(file_alpha, line_alpha);
  READLINE(file_beta, line_beta);

  READ(line_alpha, alpha);
  READ(line_beta, beta);


  TB_segmentarray(to_integer(counter)).alpha <= unsigned(alpha);
  TB_segmentarray(to_integer(counter)).beta <= unsigned(beta);


  counter := counter + 1;
end loop;

wait until ( T_clk'event and T_clk = '1');
T_reset <= '1';
wait until ( T_clk'event and T_clk = '1');
T_reset <= '0';
---------- setup complete.
wait until ( T_clk'event and T_clk = '1');
--Random traffic on the buss conncetion:
T_m_write <= '1';
T_m_adr <= to_unsigned(79, 16);
T_m_data <= to_unsigned(79, 32);
wait until ( T_clk'event and T_clk = '1');
T_m_write <= '1';
T_m_adr <= to_unsigned(79, 16);
T_m_data <= to_unsigned(79, 32);
wait until ( T_clk'event and T_clk = '1');
T_m_write <= '1';
T_m_adr <= to_unsigned(79, 16);
T_m_data <= to_unsigned(79, 32);
wait until ( T_clk'event and T_clk = '1');
T_m_write <= '0';
T_m_adr <= to_unsigned(79, 16);
T_m_data <= to_unsigned(79, 32);

--Start loading data:
wait until ( T_clk'event and T_clk = '1');
loadstuff : for i in 0 to 7 loop
  T_m_write <= '1';
  v_adress := i + m00300_ofset;
  T_m_adr <= (to_unsigned(v_adress, 16));
  T_m_data(31 downto 13) <= (others=>'0');
```

```
    T_m_data(12 downto 0) <= (TB_segmentarray(i).alpha);
    wait until ( T_clk'event and T_clk = '1');
  end loop;

  loadbeta : for i in 0 to 7 loop
    T_m_write <= '1';
    T_m_adr <= (to_unsigned((i + m00300_ofset +8), 16));
    T_m_data(31 downto 25) <= (others=>'0');
    T_m_data(24 downto 0) <= (TB_segmentarray(i).beta);

    wait until ( T_clk'event and T_clk = '1');
  end loop;

  loaddel : for i in 0 to 6 loop
    T_m_write <= '1';
    T_m_adr <= (to_unsigned((i + m00300_ofset + 16), 16));
    T_m_data(31 downto 25) <= (others=>'0');
    T_m_data(24 downto 0) <= (TB_delimter_array(i));
    wait until ( T_clk'event and T_clk = '1');
  end loop;

  T_m_write <= '1';
  T_m_adr <= to_unsigned(79, 16);
  T_m_data <= to_unsigned(79, 32);
  wait until ( T_clk'event and T_clk = '1');
  T_m_write <= '1';
  T_m_adr <= (to_unsigned((m00300_ofset + 23), 16));
  T_m_data <= to_unsigned(0, 32); --Update curve
  wait until ( T_clk'event and T_clk = '1');
  T_m_write <= '0';
  T_m_adr <= (others=>'0');
  T_m_data <= (others=>'0');
  wait until ( T_clk'event and T_clk = '1');
  wait until ( T_clk'event and T_clk = '1');

  TB_setupcomplete <= '1';
  wait until ( T_clk'event and T_clk = '1'); --

  while not endfile(file_red) loop
    READLINE(file_red, line_red);
    READLINE(file_green, line_green);
    READLINE(file_blue, line_blue);
    READ(line_red, v_red);
    READ(line_green, v_green);
    READ(line_blue, v_blue);

    T_Rchan_in <= v_red;
    T_Gchan_in <= v_green;
    T_Bchan_in <= v_blue;
  wait until ( T_clk'event and T_clk = '1');
  end loop;

  --Image complete. Store the histogram:
  T_m_write <= '1';
  T_m_adr <= (to_unsigned((m00300_ofset + 23), 16));
  T_m_data <= to_unsigned(1, 32); --Store histogram
  wait until ( T_clk'event and T_clk = '1');
  T_m_write <= '0';
  T_m_adr <= (to_unsigned(0, 16));
  T_m_data <= to_unsigned(0, 32);
  wait until ( T_clk'event and T_clk = '1');
  --Wait four cycles for buffering of histogram.
  wait until ( T_clk'event and T_clk = '1');
  wait until ( T_clk'event and T_clk = '1');
  wait until ( T_clk'event and T_clk = '1');
  wait until ( T_clk'event and T_clk = '1');


  --Start to read the histogram:
  TB_starthistogramcheck <= '1';
  wait until ( T_clk'event and T_clk = '1');

  read_hist_loop : for i in 0 to 499 loop
    T_m_read <= '1';
    T_m_adr <= (to_unsigned((m00300_ofset + 24 + i), 16));
    wait until ( T_clk'event and T_clk = '1');
  end loop;
  T_m_read <= '0';
 wait;
end process;

verify_dut : process
file file_newred     : text open read_mode is "vectors\m00300_new_red.
    txt"; --8 bit
file file_newgreen   : text open read_mode is "vectors\m00300_new_green
    .txt"; --8 bit
file file_newblue    : text open read_mode is "vectors\m00300_new_blue.
    txt"; --8 bit
file file_histogram     : text open read_mode is "vectors\
    m00300_histogram_top.txt";  --21 bit
variable line_newred, line_newgreen, line_newblue : line;
variable line_bins : line;
variable bins      : std_logic_vector(20 downto 0);
variable counter   : unsigned(8 downto 0) := "000000000";
variable newred_ver, newgreen_ver, newblue_ver : std_logic_vector(7
    downto 0);


begin
  while not endfile(file_histogram) loop
    READLINE(file_histogram, line_bins); --Dump line into L
    READ(line_bins, bins); --Next argument into red
    TB_histogram(to_integer(counter)) <= bins; --Assign value to
        interface
    counter := counter + 1;
  end loop;

  wait until ( T_clk'event and T_clk = '1');
  setupdelay : while (TB_setupcomplete = '0') loop
    wait until ( T_clk'event and T_clk = '1');
  end loop setupdelay;

  delayloop : for i in 1 to num_delay loop
    wait until ( T_clk'event and T_clk = '1');
  end loop delayloop;

  while not endfile(file_newred) loop
    wait for clk_period/4;
    READLINE(file_newred, line_newred);
    READLINE(file_newgreen, line_newgreen);
    READLINE(file_newblue, line_newblue);

    READ(line_newred, newred_ver);
    READ(line_newgreen, newgreen_ver);
    READ(line_newblue, newblue_ver);

    assert T_Rchan_out = newred_ver
      report "Calculation done wrong. T_Rchan_out: " & Integer'image(
          to_integer(unsigned(T_Rchan_out))) & " while expected value
          is: "
      & Integer'image(to_integer(unsigned(newred_ver))) & " according
          to testfile"
      severity warning;

    assert T_Gchan_out = newgreen_ver
      report "Calculation done wrong. T_Gchan_out: " & Integer'image(
          to_integer(unsigned(T_Gchan_out))) & " while expected value
          is: "
      & Integer'image(to_integer(unsigned(newgreen_ver))) & "
          according to testfile"
      severity warning;

    assert T_Bchan_out = newblue_ver
      report "Calculation done wrong. T_Bchan_out: " & Integer'image(
          to_integer(unsigned(T_Bchan_out))) & " while expected value
          is: "
      & Integer'image(to_integer(unsigned(newblue_ver))) & " according
          to testfile"
      severity warning;
    wait until (T_clk'event and T_clk = '1');
  end loop;
  wait;
end process;

verifyhist : process
  variable histcounter : unsigned(8 downto 0) := "000000000";
begin
  while(TB_starthistogramcheck = '0') loop
      wait until (T_clk'event and T_clk = '1');
  end loop;
  wait until (T_clk'event and T_clk = '1');
  wait until (T_clk'event and T_clk = '1');
  histcounter := (others=>'0');
  check : for i in 0 to 499 loop
      wait for clk_period/4;

      assert TB_histogram(i) = T_m_rddata(20 downto 0)
       report "Something wrong with the histogram, position " &
           Integer'image(i) & "## Actual: " & Integer'image(to_integer
           (unsigned(T_m_rddata))) & " ## Expected: "
        & Integer'image(to_integer(unsigned(TB_histogram(i))))
        severity  warning;
      histcounter := histcounter + 1;
      wait until ( T_clk'event and T_clk = '1');
   end loop;
   wait;
end process;

end architecture;
```

code/m00300_TB_toplevel.vhd

F Simulations of VHDL modules

F.1 rgb2ychan

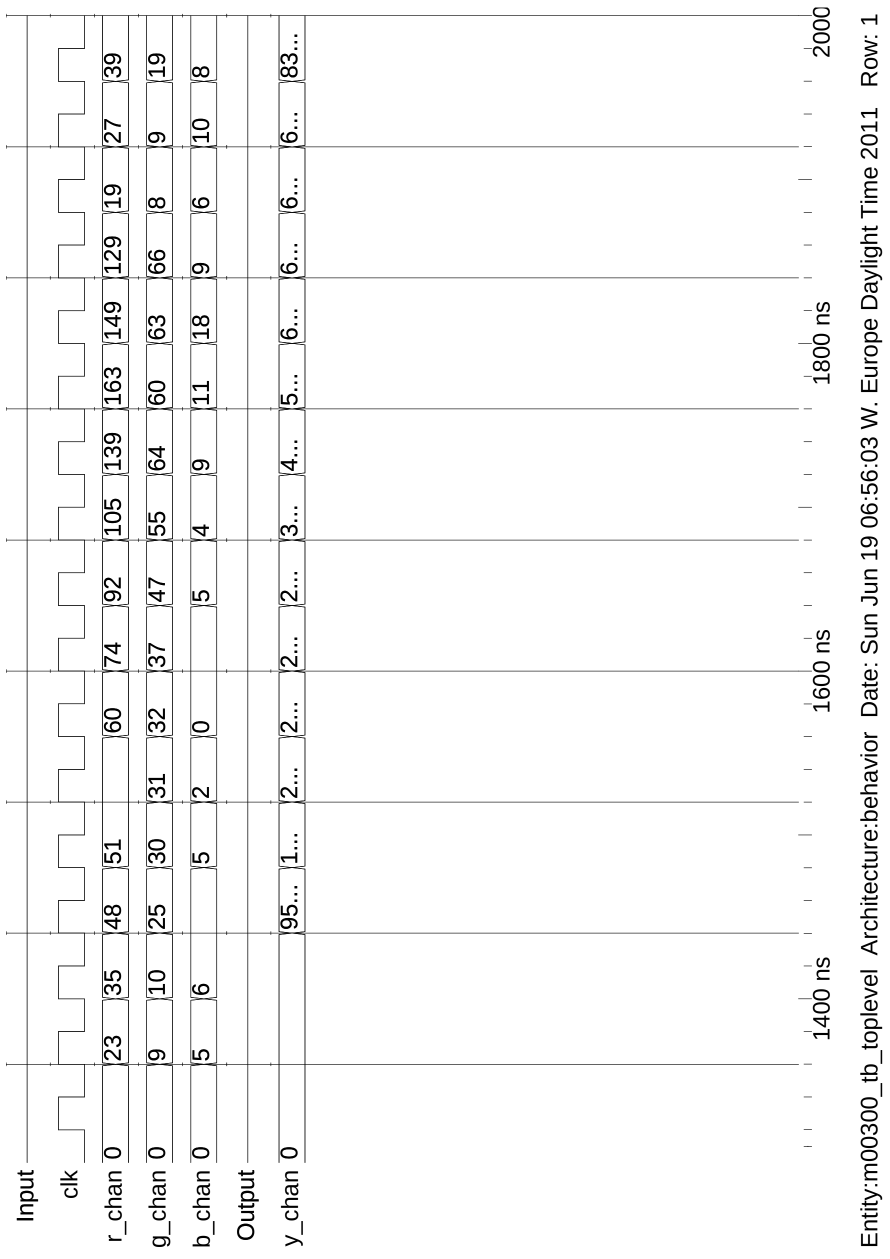

Figure 75: *Simulation of rgb2ychan*

F.2 TMA

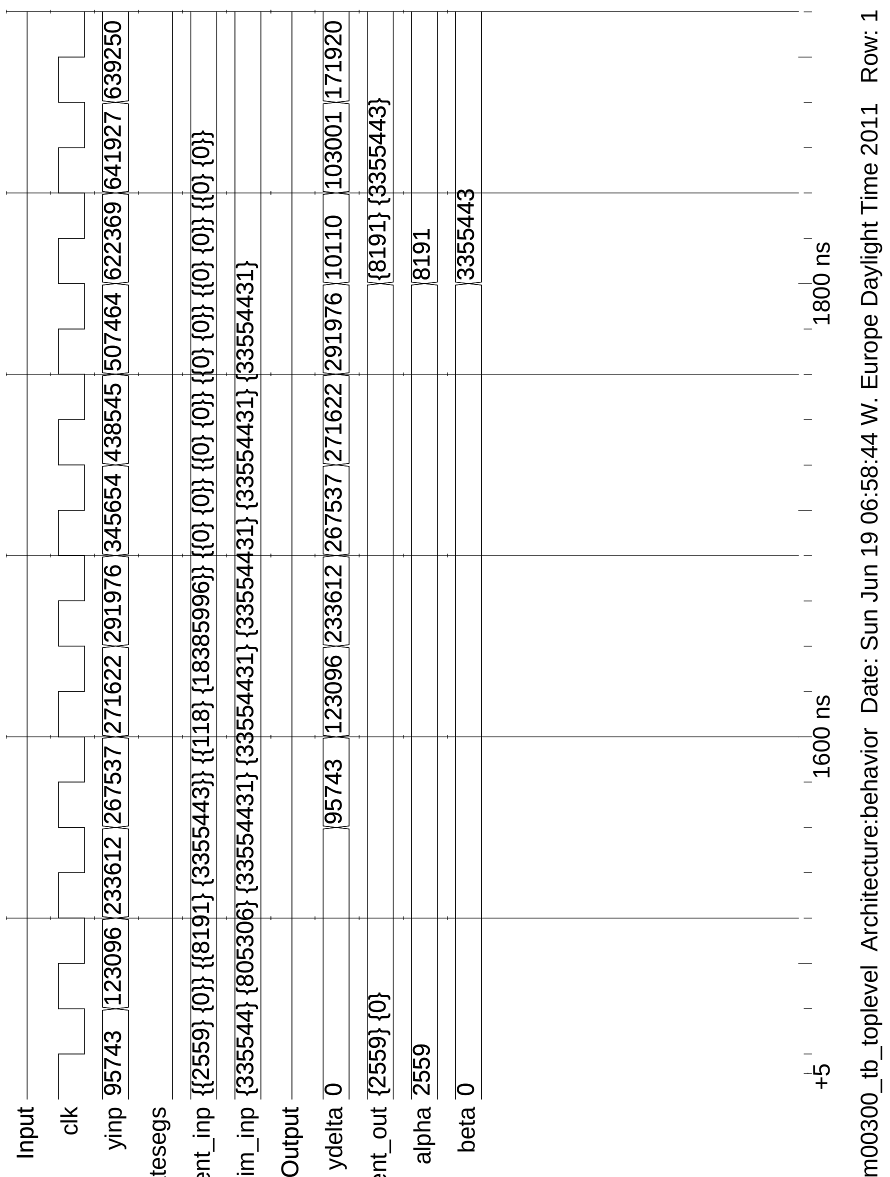

Figure 76: *Simulation of TMA showing the loading of the segment coefficients.*

F.3 TMB

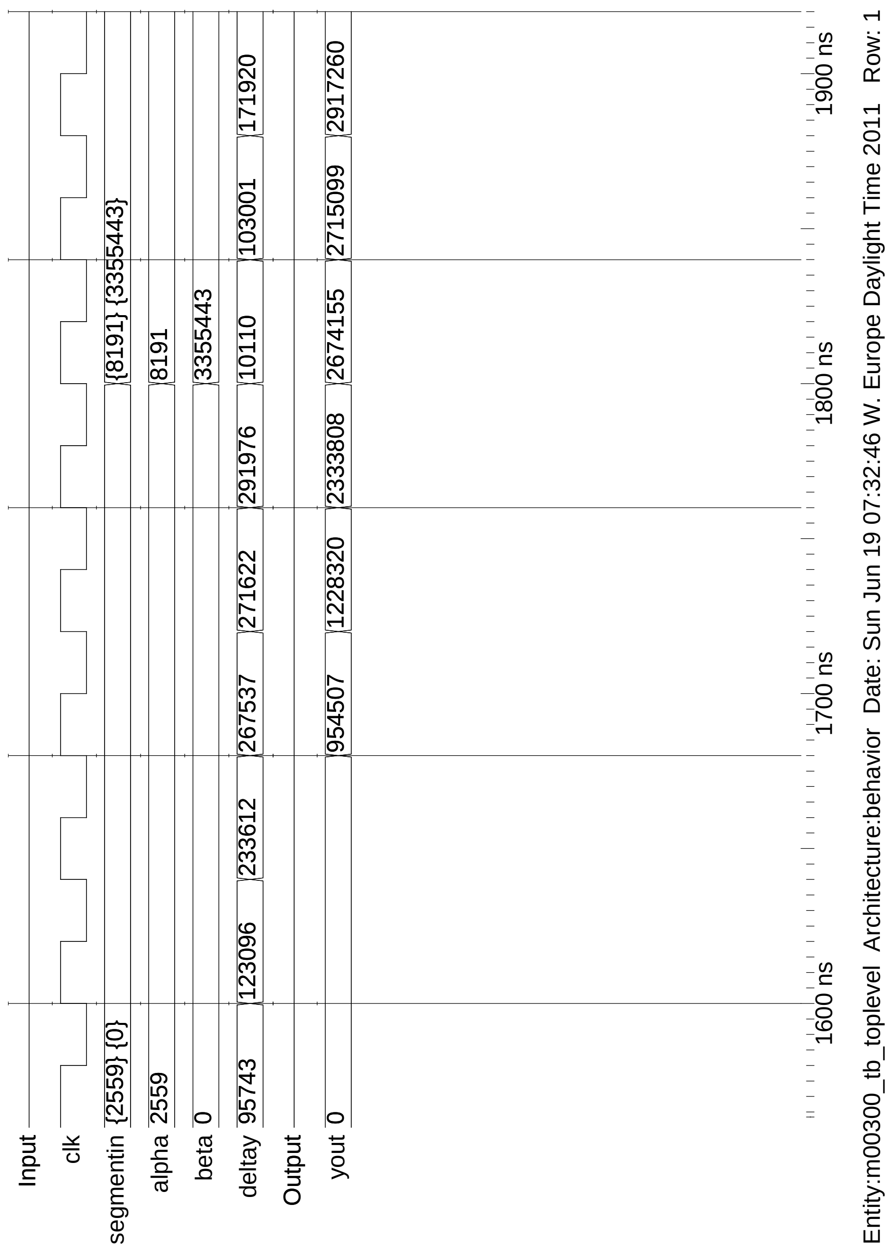

Figure 77: *Simulation of TMB showing the loading of the segment coefficients.*

F.4 Array divider

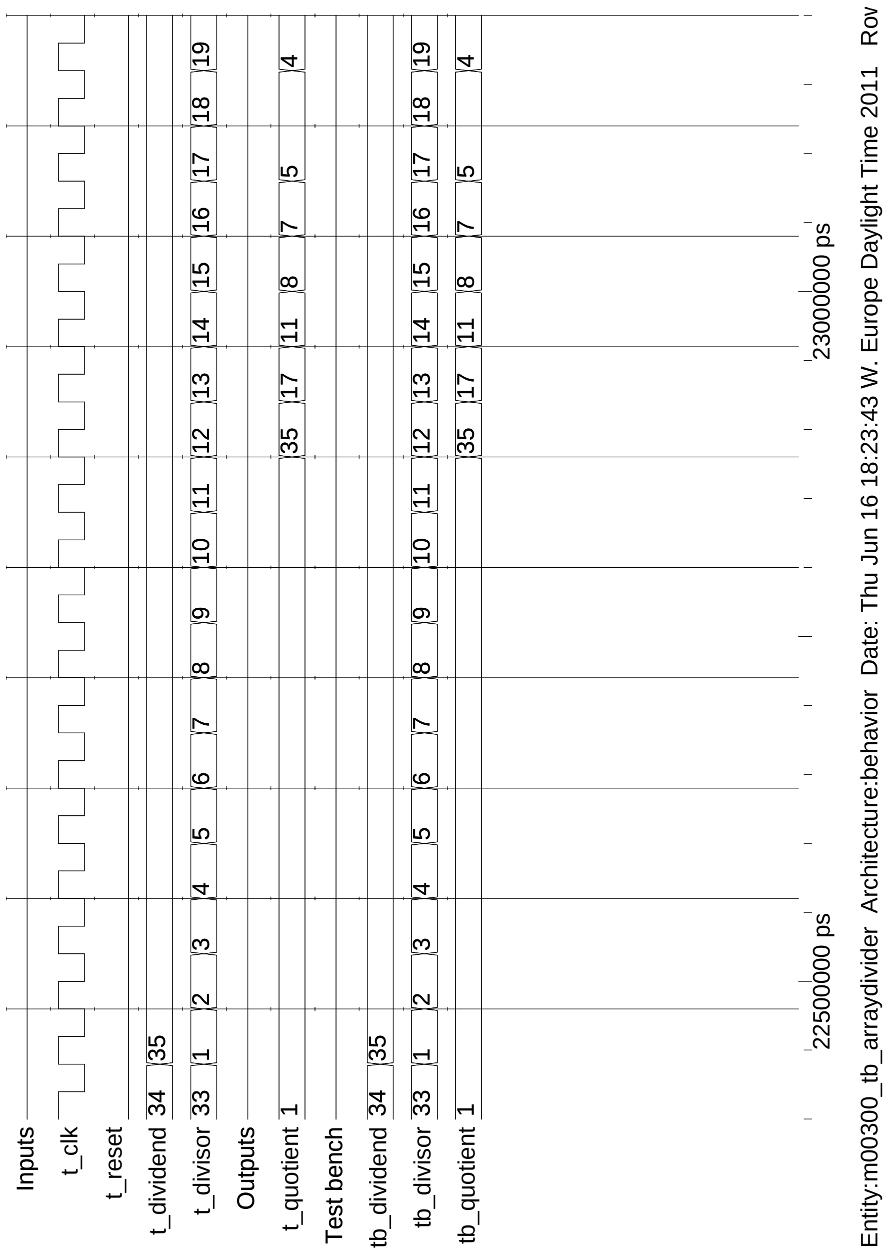

Figure 78: *Simulation of array divider, using eight bit vectors. This is just a small part of the test bench.*

F.5 Buffer

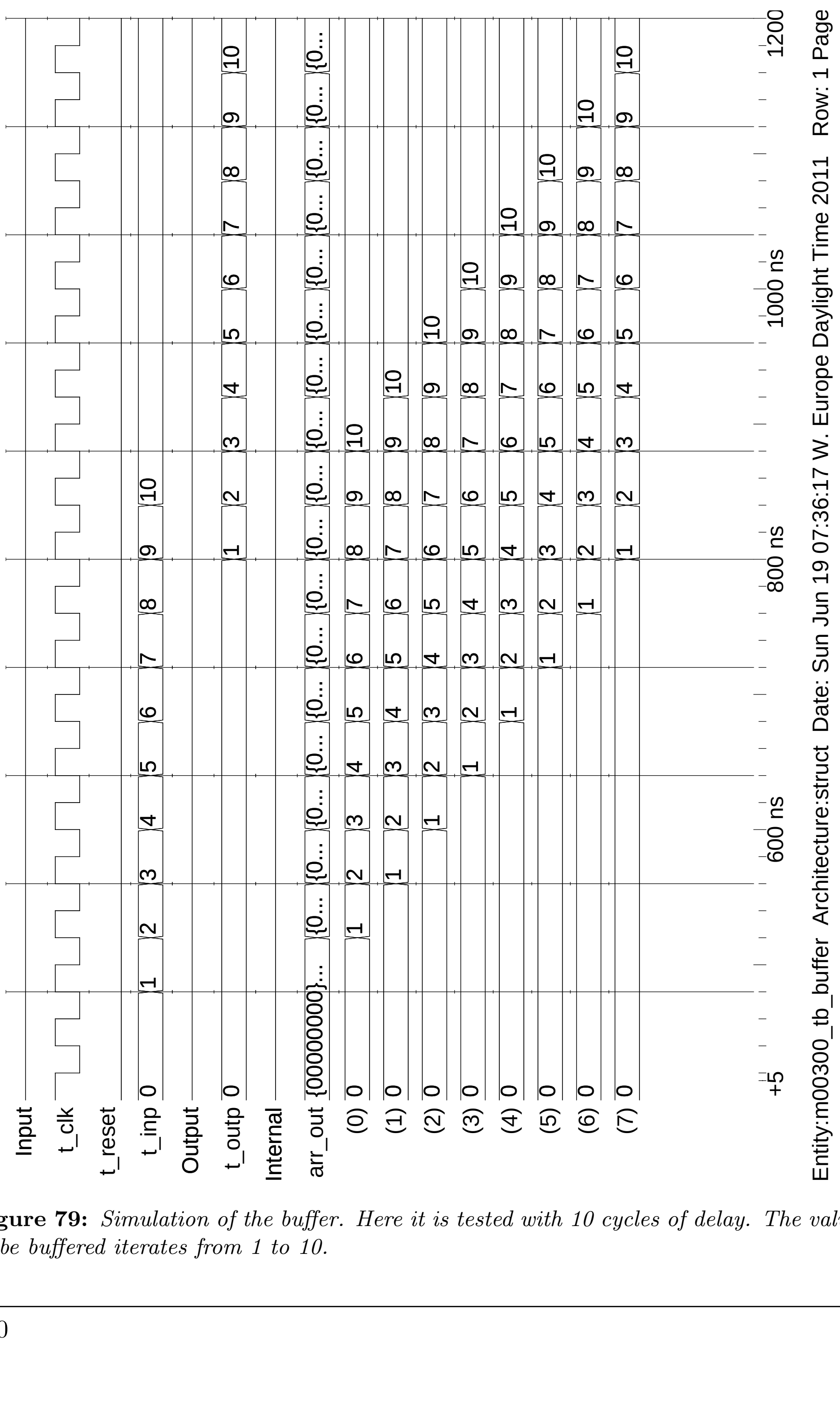

Figure 79: *Simulation of the buffer. Here it is tested with 10 cycles of delay. The values to be buffered iterates from 1 to 10.*

F.6 Memory interface

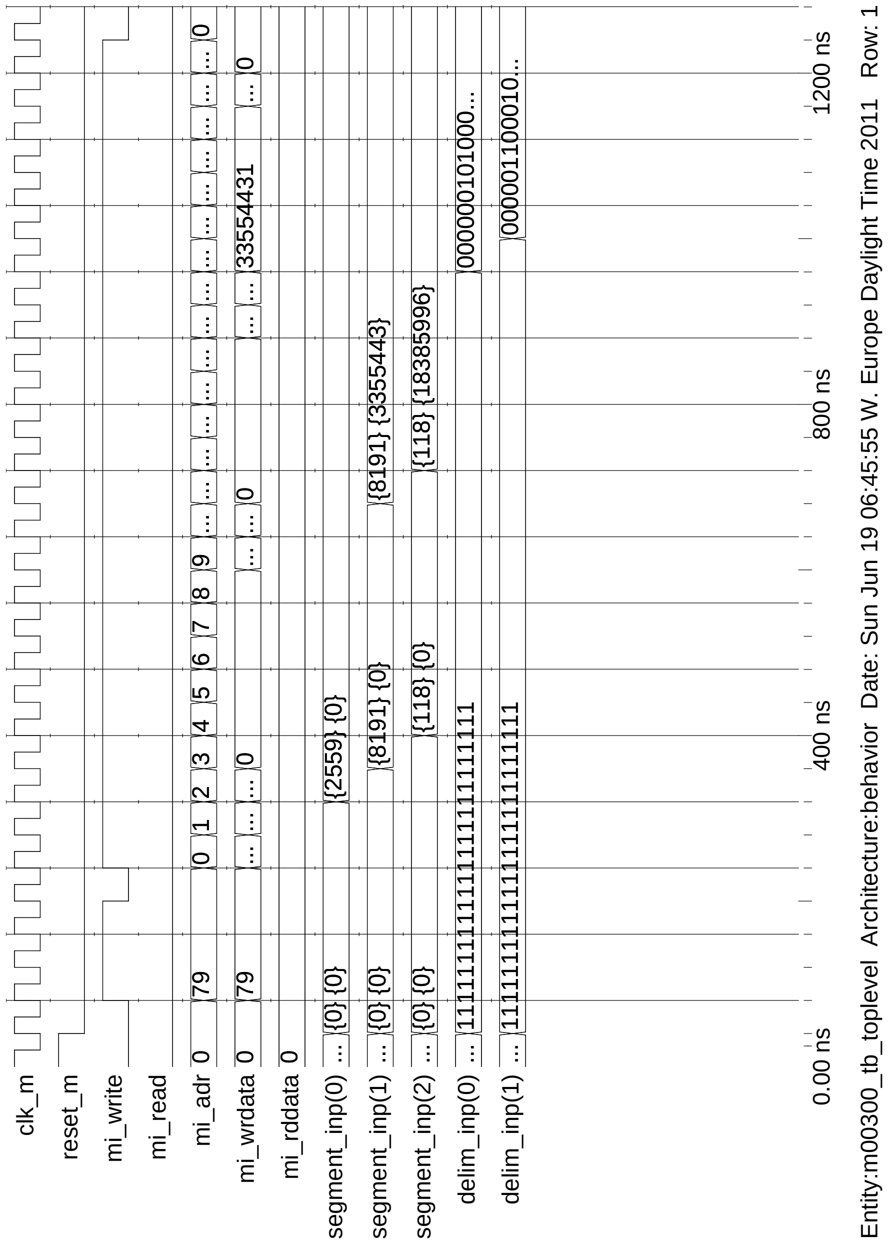

Figure 80: *Simulation of the buffer: This show how a segmented curve is loaded into the arrays. Only the loading of three segments are shown here. At the end, when update goes high, the arrays are loaded into TMA.*

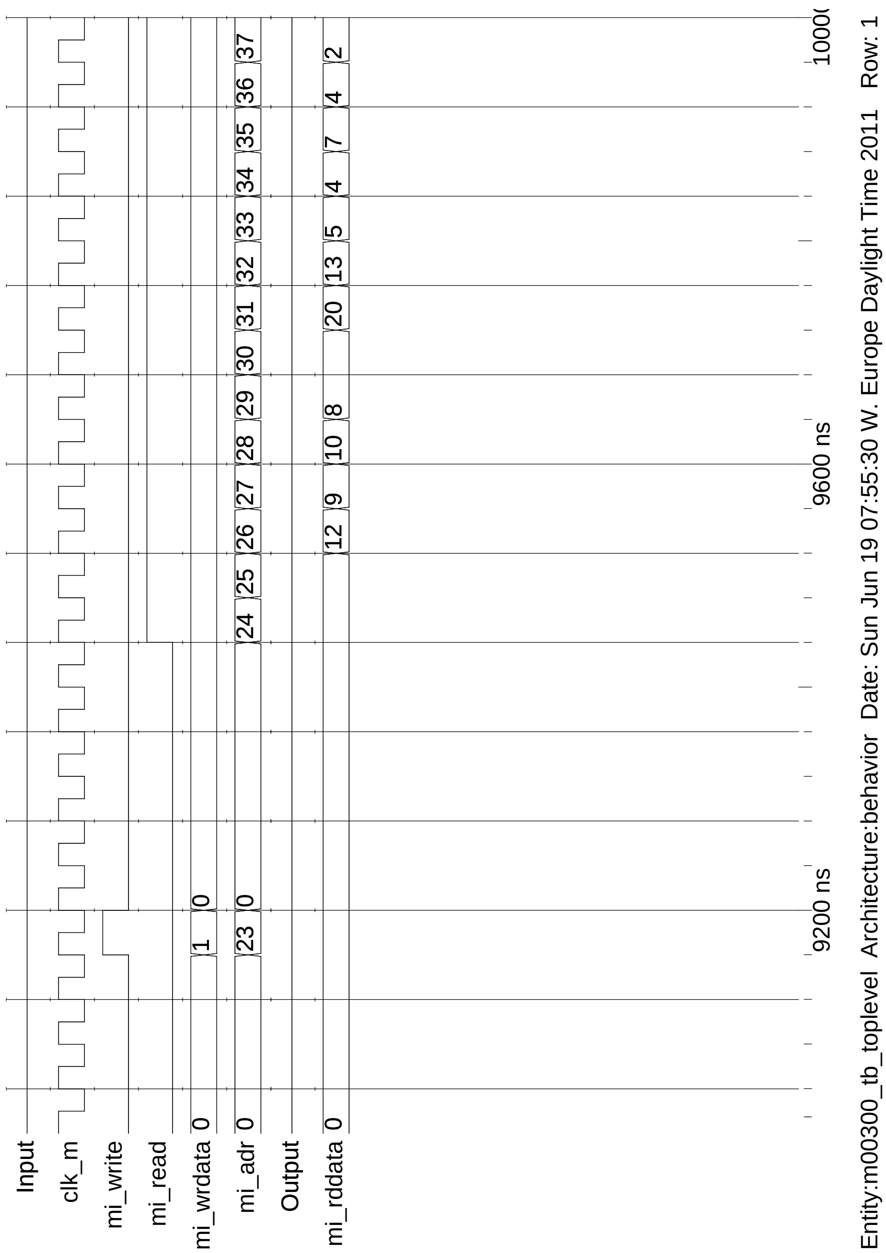

Figure 81: *Simulation of the buffer: This show how the histogram is read out of the tone mapper. Showing the first few reads out of 500.*

F.7 Reducer

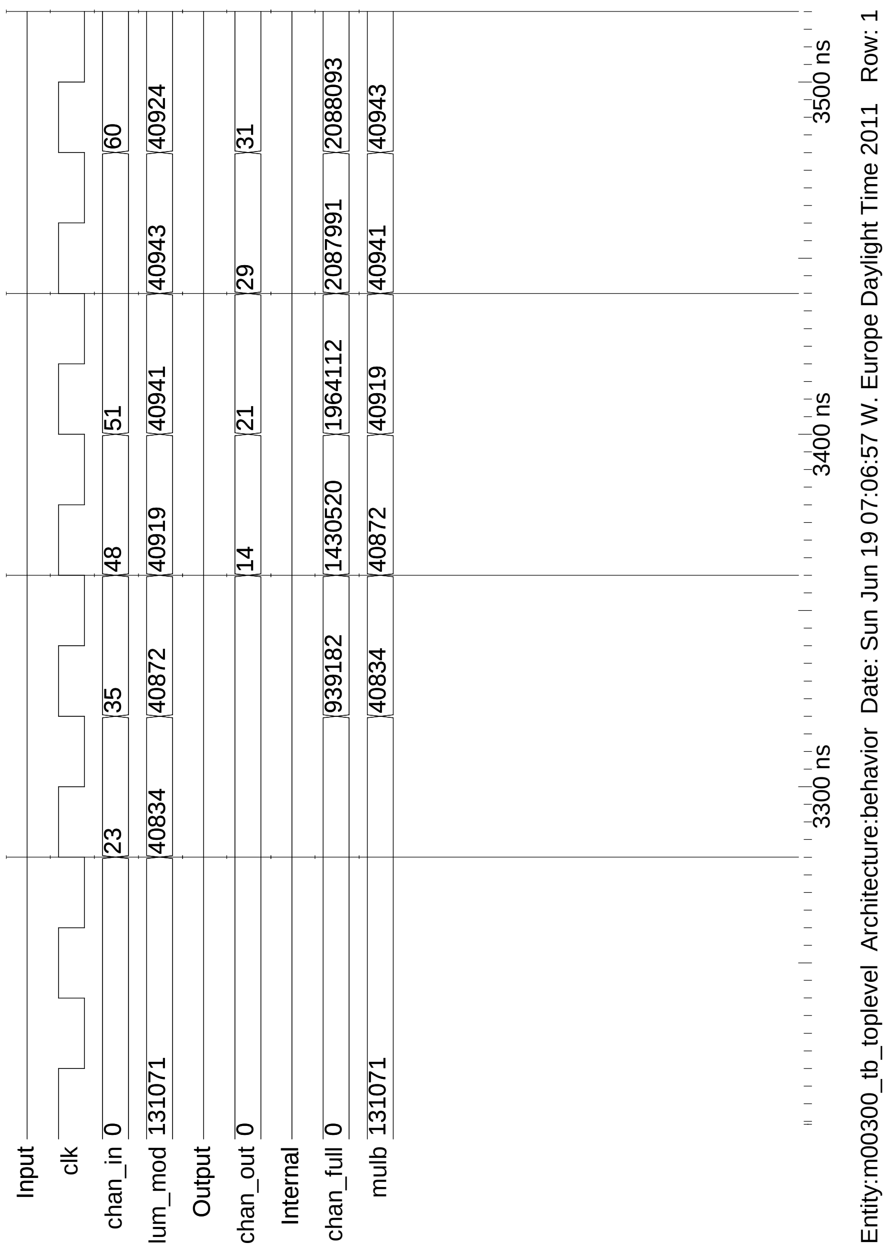

Figure 82: *Simulation of the reducer. In this example, the reduction of the red pixel is shown. Lum_mod is 25 bit large, but reduced 8 bit before the multiplication.*

G Matlab code for generating test vectors

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Thesis 2011: Tone mapping in video conferences
%
% Bit level implementation of proposed tone mapper
% Requires imageprocessing package for matlab
%
%@Author Erik Strmme
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear;
clc;

% Generating test vectors for VHDL testbenches
% Coordinates for parts of image used in testing:
y_start = 2467;
y_end = 2479;
x_start = 3583;
x_end = 3597;
% Enabletxt = true to generate vectors
enabletxt = true;
%%Remove gamma, normalize and limit to 12 bit.

im1 = double(imread('studentroom.tiff'))./65536;
%im1 = srgb2rgb(double(imread('homeoffice.tif'))./65536);

RGB = floor(im1.*4095); %% Limit to maximum 12 bits.
RGB(RGB > 4095) = 4095;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if(enabletxt)
    figure;imshow(im1(y_start:y_end, x_start:x_end, :), []);

    redchan = dec2bin(RGB(y_start:y_end, x_start:x_end,1), 12);
    greenchan = dec2bin(RGB(y_start:y_end, x_start:x_end,2), 12);
    bluechan = dec2bin(RGB(y_start:y_end, x_start:x_end,3), 12);
    sizeofmatrix = size(redchan);

    endofline = repmat('X', sizeofmatrix(1), 1);
    dlmwrite('m00300_red.txt', [redchan, endofline], 'delimiter', '',
        'newline', 'pc');
    dlmwrite('m00300_green.txt', [greenchan, endofline], 'delimiter',
        '', 'newline', 'pc');
    dlmwrite('m00300_blue.txt', [bluechan, endofline], 'delimiter', ''
        , 'newline', 'pc');
    clearvars redchan greenchan bluechan
end;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% RGB too Y:
Ytransform =  floor([0.2126729; 0.7151522; 0.0721750] .* 8192);

%Ychan now require 25 bit.
Ychan = RGB(:,:,1).*Ytransform(1) + RGB(:,:,2).*Ytransform(2) + RGB
    (:,:,3).*Ytransform(3);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if(enabletxt)
```

```matlab
    Ychanbin = dec2bin(Ychan(y_start:y_end, x_start:x_end), 25);
    sizeofmatrix = size(Ychanbin);
    endofline = repmat('X', sizeofmatrix(1), 1);
    dlmwrite('m00300_ychan.txt', [Ychanbin, endofline], 'delimiter', ' ', 'newline', 'pc');


    %%%% Specific for the histogram:
    %Smallest value: 11 0000 0000 0000 0000
    smallest = bin2dec('11');
    biggest = bin2dec('111110110');
    zebins = linspace(smallest, biggest, 500);
    values = floor(Ychan(y_start:y_end, x_start:x_end)/65536);
    zehist = histc(values(:), zebins);

    histogrambin = dec2bin(zehist, 21);
    sizeofmatrix = size(histogrambin);
    endofline = repmat('X', sizeofmatrix(1), 1);
    dlmwrite('m00300_histogram_top.txt', [histogrambin, endofline], 'delimiter', '', 'newline', 'pc');
    clearvars values
    clearvars zebins
    clearvars zehist
    clearvars histogrambin
    clearvars Ychanbin
end;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Make histogram:
%zebins = (0.01:1/65535:0.99) .*33554432; %2^25 %% 65537 bins
zebins = (0.01:1/500:0.99) .*33554432; %2^25 %% 502 bins
zehistogram = histc(Ychan(:), zebins);


%% Define number of segments and range of each segment:
zpop = [0, 0.10, 0.90, 1];
zeqpointspop = [0, 0.10, 0.90, 1];
zvalue = zpop.*sum(zehistogram);

zeqpoints = double(floor(2^25 .* zpop))+1;


%% Producing a reduced histogram equalization curve:
% Find segment delimiters:

zpoints = zeros(size(zvalue));
zpoints(1) = 0;
zpoints(end) = 2^25-1;


akkumz = 0;
zvaluepointer = 2;
for i=1:length(zehistogram)
    akkumz = zehistogram(i) + akkumz;
    if(akkumz >= zvalue(zvaluepointer))
        zpoints(zvaluepointer) = zebins(i);
        zvaluepointer = zvaluepointer + 1;
    end;
    if(zvaluepointer == length(zpop))
        break;
    end;
end;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Calculate coefficents for hardware:
alpha = zeros([1 length(zpop)-1]);
beta = zeros([1 length(zpop)-1]);
segx = zeros([1 length(zpop)]);

for i=1:length(segx)
    segx(i) = floor(zpoints(i)); %25 bit -Related to the size of Ychan
end;

for i = 1:length(zpop)-1
    deltax = (zpoints(i+1) - zpoints(i));
    alpha(i) = (zeqpoints(i+1) - zeqpoints(i) - 1) ./ deltax;

    maxincrease = 31.9961; %Max for (5+8) bits.
    if(alpha(i) > maxincrease)
      alpha(i) = maxincrease;
      nextbeta = deltax*maxincrease + zeqpoints(i);
      zeqpoints(i+1) = nextbeta;
    end;
end;

for i = 1:length(zpop)-1
    beta(i) = floor(zeqpoints(i) - 1); %25 bit -Related to the size of
         Ychan
end;

alpha = floor(alpha.*256); %Extend to 13 (5+8) bits

if(enabletxt)
    Alphabin = dec2bin(alpha, 13);
    Betabin = dec2bin(beta, 25);
    sizeofmatrix = size(Alphabin);
    endofline = repmat('X', sizeofmatrix(1), 1);
    dlmwrite('m00300_alpha.txt', [Alphabin, endofline], 'delimiter', '
        ', 'newline', 'pc');
    dlmwrite('m00300_beta.txt', [Betabin, endofline], 'delimiter', '',
         'newline', 'pc');

    Segxbin = dec2bin(segx, 25);
    sizeofmatrix = size(Segxbin);
    endofline = repmat('X', sizeofmatrix(1), 1);
    dlmwrite('m00300_segdelimter.txt', [Segxbin, endofline], '
        delimiter', '', 'newline', 'pc');
    clearvars Alphabin Betabin Segxbin
end;

%% Visual test of curve:
testlum = 0:2:2^25;
testsegmentos = zeros(size(testlum));
for i=1:length(segx)-1;
    testsegmentos(testlum>=segx(i)) = i;
```

```matlab
end;

testcurve = ((alpha(testsegmentos)).* (testlum - segx(testsegmentos)) ./ 256) + beta(testsegmentos);
%% Calculate new lum:
segmentos = zeros(size(Ychan));
%Find which segment each pixel belong in
for i=1:length(segx)-1;
    segmentos(Ychan>=segx(i)) = i;
end;

deltay = Ychan - segx(segmentos);
newlum = ((alpha(segmentos)).* floor(deltay./256)) + beta(segmentos);

%Newlum can be max 2^26-1
% Bigger alpha => Smaller beta, Bigger beta => Smaller alpha
newlum(newlum > 2^26-1) = 2^26-1;

if(enabletxt)
    Segmentosbin = dec2bin(segmentos(y_start:y_end, x_start:x_end)-1, 4);
    Newlumbin = dec2bin(newlum(y_start:y_end, x_start:x_end), 26);
    sizeofmatrix = size(Segmentosbin);
    endofline = repmat('X', sizeofmatrix(1), 1);
    dlmwrite('m00300_segment.txt', [Segmentosbin, endofline], 'delimiter', '', 'newline', 'pc');
    dlmwrite('m00300_newlum.txt', [Newlumbin, endofline], 'delimiter', '', 'newline', 'pc');
    clearvars Segmentosbin Newlumbin

    deltaybin = dec2bin(deltay(y_start:y_end, x_start:x_end), 25);
    sizeofmatrix = size(deltaybin);
    endofline = repmat('X', sizeofmatrix(1), 1);
    dlmwrite('m00300_deltay.txt', [deltaybin, endofline], 'delimiter', '', 'newline', 'pc');
    clearvars deltaybin
end;
%%%%%
%% Calculate lum modifier
lum_modifier = floor((newlum.*4096)./Ychan);
lum_modifier(isnan(lum_modifier)) = 1;
%Need to limit how many bits lum_modifier can be. Lets say
% 17 bits (5 + 12), essentially saying max mod is
lum_modifier(lum_modifier>131071) = 131071;

if(enabletxt)
    Ychanbin = dec2bin(lum_modifier(y_start:y_end, x_start:x_end), 17);
    sizeofmatrix = size(Ychanbin);
    endofline = repmat('X', sizeofmatrix(1), 1);
    dlmwrite('m00300_lum_modifier.txt', [Ychanbin, endofline], 'delimiter', '', 'newline', 'pc');
    clearvars Ychanbin
end;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Multiply with image to find the new image:
newimageHW = zeros(size(RGB));
```

```
newimageHW(:,:,1) = floor((RGB(:,:,1).*lum_modifier)./(16.*4096));
newimageHW(:,:,2) = floor((RGB(:,:,2).*lum_modifier)./(16.*4096));
newimageHW(:,:,3) = floor((RGB(:,:,3).*lum_modifier)./(16.*4096));

newimageHW(newimageHW > 255) = 255;
figure;imshow(uint8(rgb2sRGB(newimageHW./255).*255), []);

if(enabletxt)
    redchan = dec2bin(newimageHW(y_start:y_end, x_start:x_end,1), 8);
    greenchan = dec2bin(newimageHW(y_start:y_end, x_start:x_end,2), 8)
        ;
    bluechan = dec2bin(newimageHW(y_start:y_end, x_start:x_end,3), 8);
    sizeofmatrix = size(redchan);
    endofline = repmat('X', sizeofmatrix(1), 1);
    dlmwrite('m00300_new_red.txt', [redchan, endofline], 'delimiter',
        '', 'newline', 'pc');
    dlmwrite('m00300_new_green.txt', [greenchan, endofline], '
        delimiter', '', 'newline', 'pc');
    dlmwrite('m00300_new_blue.txt', [bluechan, endofline], 'delimiter'
        , '', 'newline', 'pc');
    clearvars redchan greenchan bluechan
end;
```

code/verification/implementation.m