ECE532 Digital Systems Design - Final Report

Enhanced Object Detection Using HDR Methods

Group 6: Alex Papanicolaou, Rose Li, Yuanfang Li, Kenan Hu

1 Overview

1.1 Motivation

High Dynamic Range (HDR) imaging is an old technique that has recently been applied to modern cameras such as in the iPhone. HDR imaging combines sets of Low Dynamic Range (LDR) photographs that have been taken with different exposure times, enhancing images taken by cameras with a fixed range, to the full range visible to the human eye. This is particularly useful for bringing out details in images taken at night or in bright sunlight. In a typical smartphone, this process requires significant time to complete. FPGAs are capable of accelerating the computation in hardware, enabling the application of HDR image processing to real-time applications. Figure 1 shows the general concept of HDR through a diagram. A Panasonic DMC-ZS25 camera model was used to capture the images used for this project.

1.2 Goals

The goal of this project is to apply HDR imaging technology to enhanced object detection problems. The working system will be demonstrated by first creating a static arrangement of objects, some hidden by bright lights, others blending in with darker backgrounds. A picture will be taken and the scene observed using a non-HDR capable device. The same arrangement will then be photographed at different exposure times and processed using HDR. After processing, the system will identify the hidden objects.

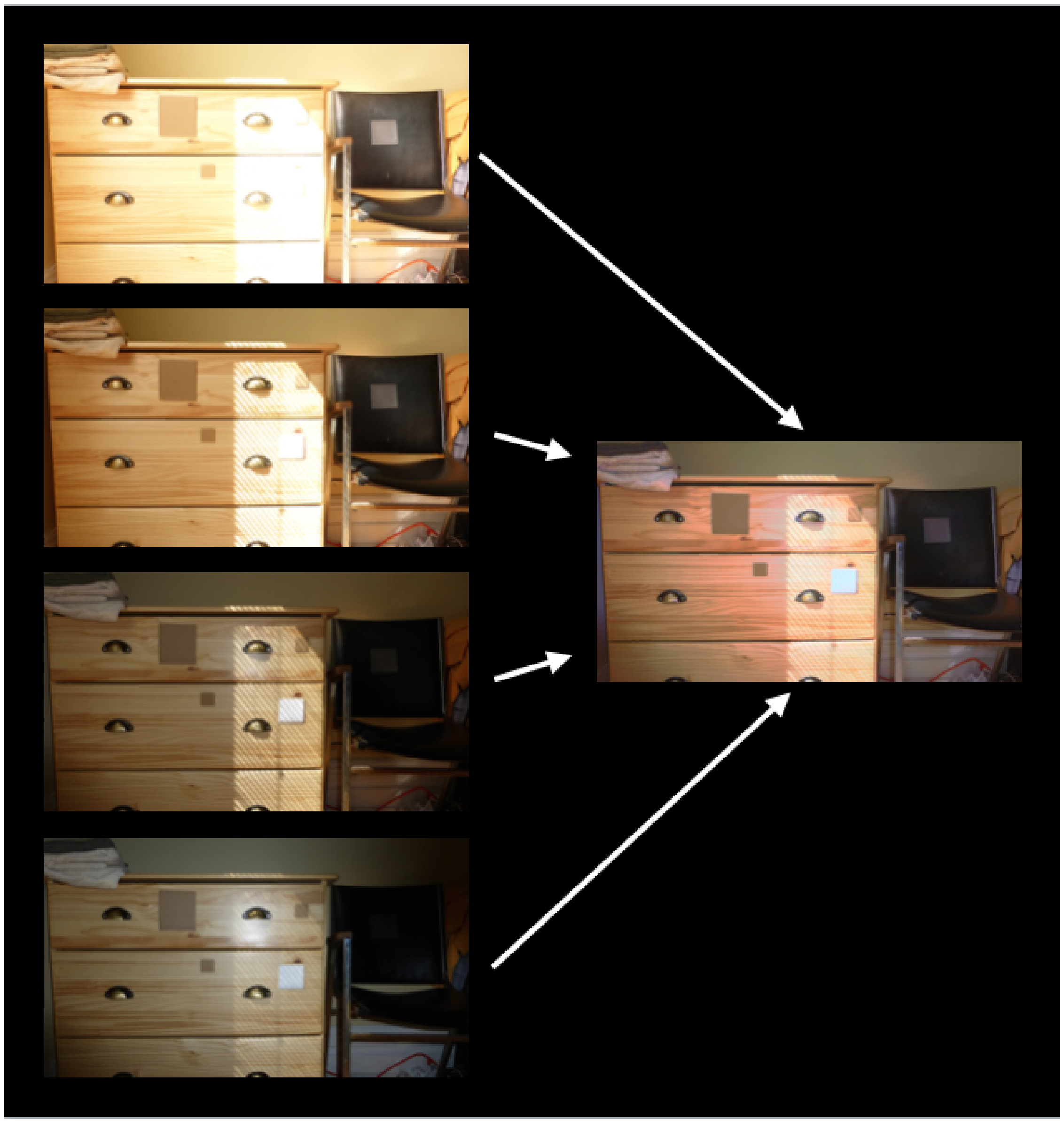

Figure 1: 4 Exposure composition of an HDR image. Note the various items brought out more in some exposures than others.

1.3 Block Diagram and IPs Used

Processor:

1. MicroBlaze Soft Processor Core:

   The Microblaze is used to run the program that controls each component of the system.

2. Software IPs

(a) DMA Controller:

    Macros provided by Xilinx and used to control operation of the DMA block

(b) SPI Controller:

    Low level drivers provided by Patrick Payne used to control operation of the SPI block.

    FAT32 file system module provided by elm-chan.org

Data Transfer and Display Blocks

Xilinx IP Cores:

1. 7 Series Memory Interface Generator:

   Used to interface with off-chip DDR memory

2. AXI Direct Memory Access Controller v7.1:

   Used to move from memory mapped to streaming and vice versa

3. AXI Thin Film Transistor Controller v2.0:

   Used to control the VGA display port

4. AXI Quad SPI v3.2:

   Used to interface with the SD card

5. AXI UART Lite:

   Used to interface with computer terminal during debugging

Custom IPs:

1. Video to AXI:

   Used to convert from video out to AXI streaming interface

HDR Processing Blocks:

Xilinx IP Cores:

1. Gamma Correction v7.0:

   Used to gamma encode the HDR image

2. RGB to YCrCb Color-Space Converter v3.0:

   Used to convert the HDR image into YUV space

3. YCrCb to RGB Color-Space Converter v3.0:

   Used to convert the edge-detected image into RGB space

Custom IPs:

1. HDR:

   Used to generate the HDR image from four images of different exposures

2. Object Detection:

   Used to detect edges in the HDR image

Figure 2 shows the block diagram of the entire system. We created two custom IPs, the HDR block and the edge detection block, both of which use the AXI streaming interface to communicate with the Xilinx IPs.

2 Outcome

2.1 Results

As a team, we were relatively successful at implementing the initial design features. The main missing feature is the application of the tone-mapping algorithm prior to object detection. This feature was omitted from the final design largely for 2 reasons:

1. Complexity: Standard tone-mapping algorithms like Reinhard involve complex calculations on each pixel and are difficult to implement in hardware, especially given our limited time and expertise.

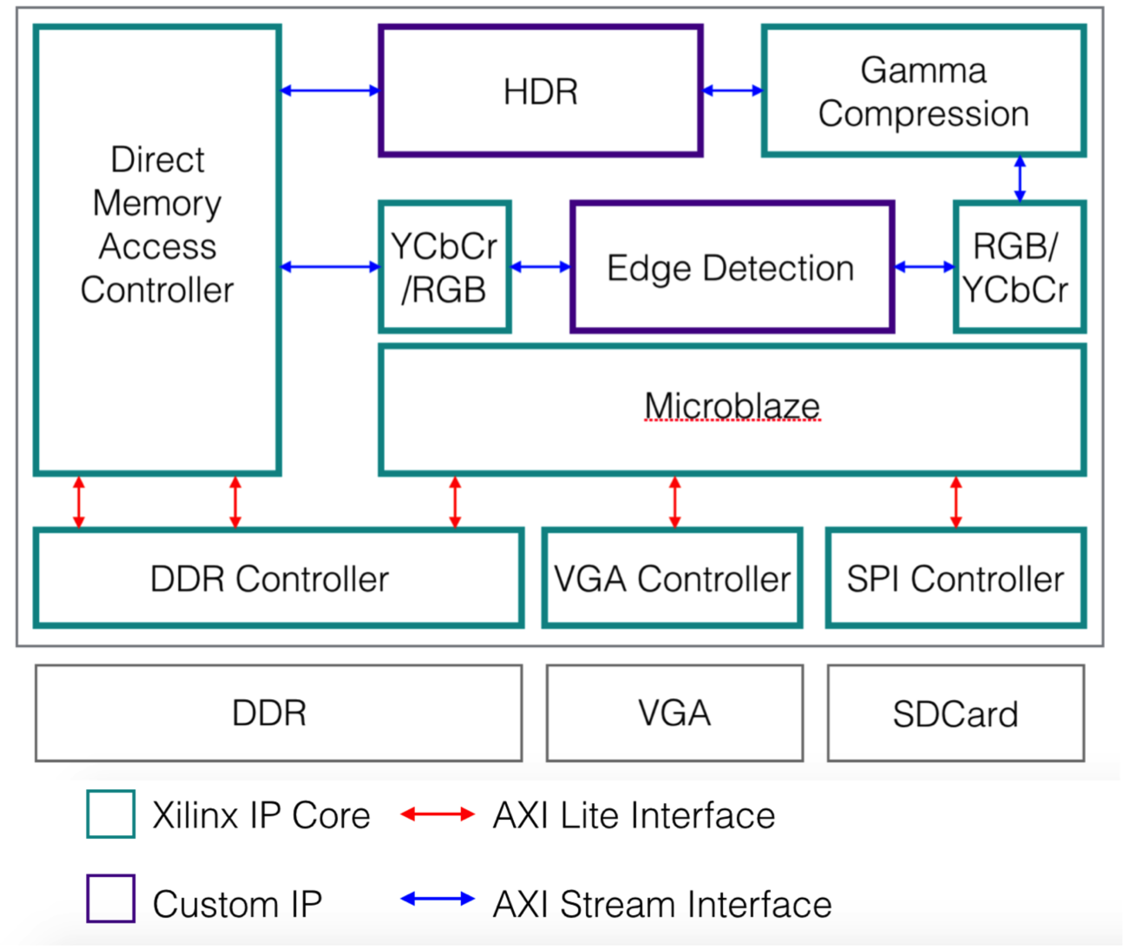

Figure 2: System Block Diagram

2. Necessity: Simpler algorithms can also produce better results in terms of colour hue and vibrance but have little effect on clarity of edges. The chief goal of object detection using HDR can be met without performing tonemapping.

We did implement software tone-mapping as well as a preliminary (untested) Verilog module of a simple algorithm but ultimately decided that time spent simulating and testing this block could be better spent on system integration and improving edge detection.

Another feature we were not able to implement is using externally generated image filters to allow detection of any kind of object. In our proposal, we believed that this feature could be easily added since our first design implemented the object detection algorithm in software. However, as

Initial Design Features	**Status**
Access raw images of different exposures from SD card	Complete
Transfer images to DDR memory	Complete
Run HDR algorithms to combine multiple-exposure frames into one composite image	Complete
Tone-map images for better display results	Removed
Access externally generated image filters from the SD card defining the set of objects to be detected	Incomplete
Scan composite with the image filters using object detection algorithm (software)	Complete (hardware - requires additional finetuning)
Mark composite image around centroid of detected objects and combine into a new image (software)	Incomplete
Display image on VGA monitor	Complete

Table 1: Initial design features.

we progressed with the project, we found that the HDR algorithm did not need to be as complex as we initially believed, so we decided to also implement object detection in hardware. In order to eliminate any divisions or multiplications by fractions, the object detection algorithm was designed to convert these operations into shifts (i.e. multiplications and divisions by powers of 2) and so is optimized for square detection only. The resulting edge detected image being generated by the hardware had a high noise content with lots of white pixels filling in the image. Future modifications include adjusting the pixel discriminating threshold to remove pixels below a certain value of light quantity.

We were successful at implementing edge detection algorithms that demonstrate the difference between an image with no HDR composition and one with HDR. Figure 3 illustrates our implementation using software to prove the premise of the project. Normal exposed image on the left, while HDR image is on the right. Note how the white square in sunlight and the square on the chair is fully visible using HDR techniques, while the midtone brown squares remain relatively the same demonstrating the non-linear operation HDR algorithms apply to create the composite image.

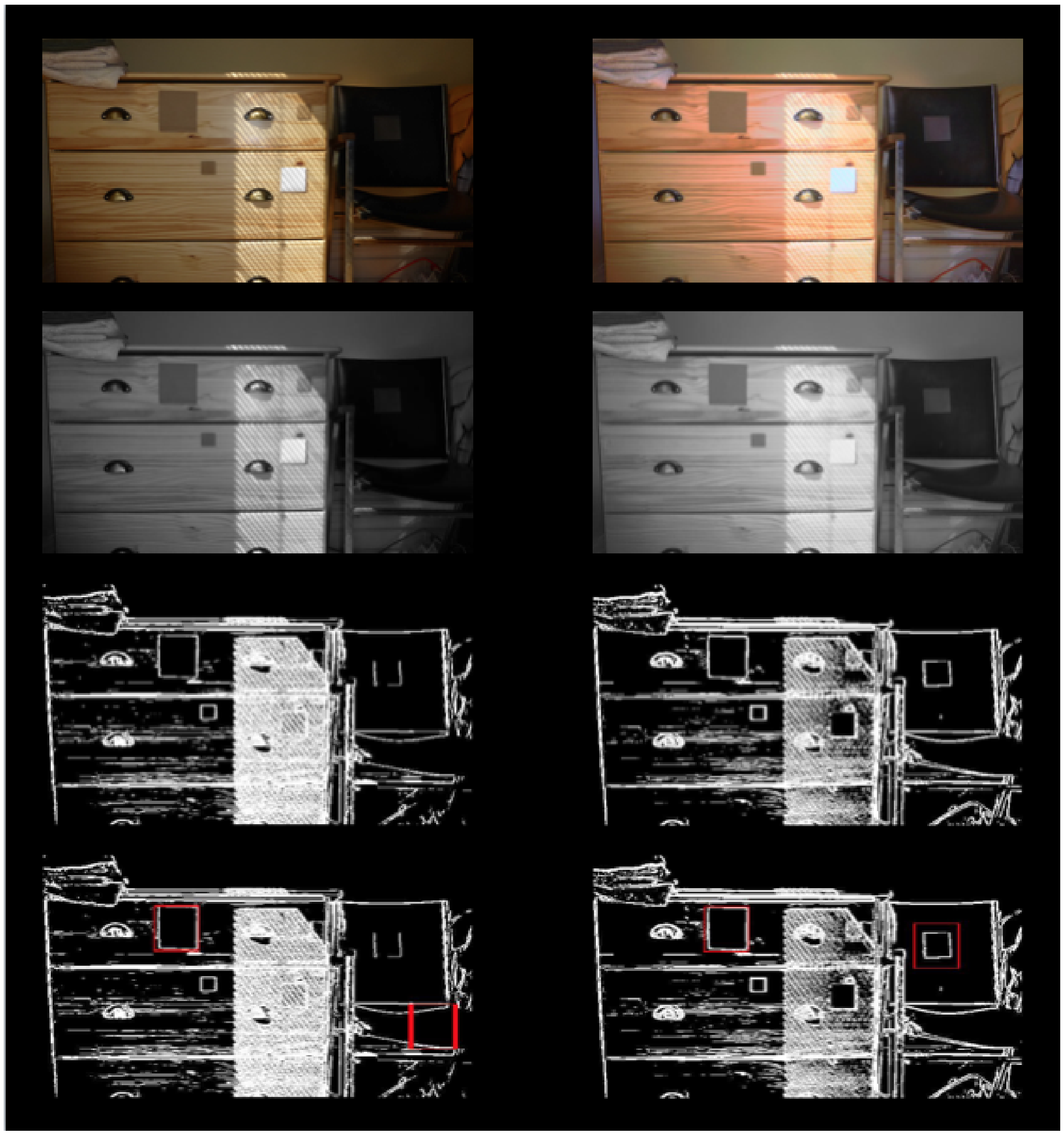

Figure 3: Normal exposed image (left) vs HDR image (right) edge detection comparison

2.2 Future Work

The system could be significantly improved by allowing easier customization and fine-tuning. In the current design, the inverse camera response function values are initialized in the HDR block so any changes would require modifying the IP block and re-synthesizing the entire system. Consequently, the system is only guaranteed to work well with images taken with a single camera. The object detection algorithm is also hard-coded in the IP block as opposed to being able to use different image filters given by the user; again this limits the applications of the system. Ideally, the system should be able to use any set of image filters as described in our initial design.

The final image displayed could also be improved by applying tone-mapping prior to object detection. Although aesthetics is not the main goal of the project, it would be still be helpful for the HDR composite to look nicer than the original images.

3 Project Schedule

Week	Original Milestone	Achieved Milestone
1	**Alex:** SD card SPI block	**Alex:** Imported spimaster core from OpenCores into Vivado
	Kenan: DDR memory controller block	
	Rose: UART control interface	**Rose:** UART control interface
	Yuanfang: Microblaze, DDR and UART integration	**Yuanfang:** Microblaze, DDR and UART integration
2	**Alex:** VGA control interface	**Alex:** AXI to WB, VGA control interface from OpenCores to modify.
	Kenan: SD command protocol interface	
	Rose: Image decompression in software	**Rose:** Wrote Matlab program to extract pixel data from JPEG - due to complexity, project was modified to use bmp format instead
	Yuanfang: Image decompression in software	**Yuanfang:** Program to extract pixel data from JPEG - due to complexity, project was modified to use bmp format instead

3	**Alex:** Object detection algorithm implemented in software **Kenan:** Object detection algorithm implemented in software **Rose:** Image decompression in hardware **Yuanfang:** Image decompression in hardware	**Alex:** Discrete grey-scale conversion, edge detection methods using the Sobel Edge Detection kernels **Rose:** Modified Matlab file to extract pixel data from JPEG, output bin file and text file **Yuanfang:** Interfaced DMA, MIG and Microblaze, able to stream data from memory through FIFO and back into memory
4	**Group:** Take picture from SD card and display on monitor	**Alex:** Simple greyscale and sobel edge detection algorithms written for verilog. Simple square detection script written in python **Rose:** Obtained SD cards and successfully powered on and initialized SD card using SPI protocol **Yuanfang:** Added TFT controller to system, able to stream out image from DDR into frame buffer and display on monitor

5	**Group:** HDR LUT and tonemapping in software	**Alex:** Experimented with various algorithms to get the norm of x and y edges without implementing square root function. Better filtering techniques developed. **Rose:** Integrated SD to VGA and SD to DDR to VGA projects with help from Patrick Payne's low level drivers **Yuanfang:** Started experimenting with HDR algorithms, camera response function, etc
6	**Group:** HDR LUT and tonemapping in hardware	**Alex:** Continuation of edge detection implementation in verilog. Validation of project premise that HDR is indeed better for detecting objects than normal images. **Rose:** Successfully displayed a bin file image from SD card onto VGA monitor **Yuanfang:** Used Matlab to generate camera response functions, create HDR image and tonemap for use in edge detection testing, experimented with different calibrations
7	**Group:** Spare week, polishing features	**Alex:** Edge detection integration with rest of subsystems.

		Rose: Worked with Yuanfang to integrate HDR block with SD to DDR to VGA system **Yuanfang:** Created HDR block in hardware, integrated with system to read from DDR, perform HDR and display on monitor

There is a significant discrepancy between our original milestones and what we actually accomplished. Our achieved milestones usually occurred a week later than originally planned due to some challenges and modifications that occurred during weeks 2 and 4. We had originally planned to read in JPEG images from the SD card but found decompression to be much harder than anticipated, even in software. As a result, we chose to store our images as bmp files instead. We also spent much longer than anticipated interfacing with the SD card due to having to use a SPI controller instead of a SD card controller and some mysterious issues with file names on the SD card.

Our original milestones also specified group milestones for weeks 4 to 7 as opposed to individual milestones. In practice, we found this difficult to achieve due to timetable conflicts between group members, so the milestones were adjusted and divided amongst group members accordingly.

4 Description of Blocks

4.1 MicroBlaze Soft Processor Core

The MicroBlaze processor is used to control the system and several of the blocks. The software (found in main.c) works as follows:

1. Enable TFT controller

2. Write images from SD card to DDR memory. The TFT controller will simultaneously display these images on the screen as they are being written.

3. Loop for various image sets: Initialize the DMA controller and begin streaming images from DDR. At the same time, stream the HDR image into the frame buffer. The TFT controller will simultaneously display the final image as it is being written.

4.2 Data Transfer and Display Blocks

Xilinx IP Cores:

1. 7 Series Memory Interface Generator

   This block is used as a memory controller for the off-chip DDR memory.

2. AXI Direct Memory Access Controller v7.1

   The DMA controller is used to perform memory-mapped to streaming operations and vice versa. Since the input images are stored in contiguous blocks in DDR, direct register (simple) mode is used with a maximum burst length of 128 words.

3. AXI Thin Film Transistor Controller v2.0

   The TFT controller is used to display images on the VGA monitor. When configuring the core, 32-bit VGA mode should be selected. Note that the TFT controller uses 6 output pins for each colour channel whereas the Nexys4 board supplies only 4 pins per colour channel. To work around this, the toplevel Verilog wrapper generated by Vivado from the project block design should be modified to include only the 4 most significant bits from each channel as output.

4. AXI Quad SPI v3.2

   The SPI block is used interface with the SDCard, where the input images are stored.

5. AXI UART Lite

   This block is used to communicate between the computer and the Nexys4 board.

Custom IPs:

1. Video to AXI

   The output of the edge detection block is in 24 bit video out format while the input to the streaming to memory mapped port of the DMA block uses standard 32 bit AXI streaming. The video out data is the final HDR image that needs to be displayed and is streamed directly into the frame buffer to be read by the TFT controller. The Video to AXI block facilitates communication between video out and AXI streaming ports and also converts the pixels to the format required by the VGA display.

4.3 HDR Processing Blocks

Xilinx IP Cores:

1. Gamma Correction v7.0

   The Gamma Correction block is used to encode the HDR composite image so that it can be properly displayed on the monitor. For correct functionality, the input data width should be set to 10 bits per pixel and the output data width to 8 bits per pixel (in order to divide by 4). The pixels per scanline should be set to 640 and the scanlines per frame should be set to 360. Finally, to produce the results, the gamma value should be set to 0.45 for all three colour channels (assuming that the monitor uses a standard gamma of 2.2 so that the overall system gamma is approximately 1 for a linear result).
   NOTE: A Core Licence Agreement is needed to implement this IP in hardware.

2. RGB to YCrCb Color-Space Converter v3.0

   This block is used to convert the HDR image from RGB to YCrCb in preparation for the edge detection block, which operates using Y values only. The pixels per scanline should be set to 640 and the scanlines per frame should be set to 360. The standard selection used is YUV, with values ranging from 0 to 255 for computer graphics and 0 offset for all channels.

3. YCrCb to RGB Color-Space Converter v3.0

   This block is used to convert the generated edge detected image back to RGB. The pixels per scanline should be set to 640 and the scanlines per frame should be set to 360. The standard selection used is YUV, with values ranging from 0 to 255 for computer graphics and 0 offset for all channels.

Custom IPs:

1. HDR

   Functionality:

   The HDR block is an AXI streaming IP that receives a stream of pixels on the slave port from four images of different exposures and produces a pixel stream of the HDR composite image on the master port. The input stream should consist of each row of the four images: i.e. row 1 of the first image, followed by row 1 of the second image, followed by row 1 of the third image, followed by row 1 of the fourth image, etc. The block produces one output row for every four input rows received. For correct functionality, each image must be 640x360 and each pixel must be stored in RGB888 format. The AXI streaming interface requires a 32 bit data width, so the 8 least significant bits should be padded with zeros.

   The input port is a standard AXI streaming slave port while the output port is a modified video out port to match the requirements of the Gamma Correction block.

   Hardware Operation:

   The HDR block contains three lookup tables that are used to map the red, blue and green pixels from images of different exposures into the linear domain for processing. Conversion is done using the inverse camera response function; since this function differs for each camera, the lookup table values must be re-initialized for different cameras. The values in the current block have been calculated using Matlab.

The converted pixels from the first three images are stored in line buffers. Upon transfer of the fourth image, the corresponding pixels from all four images are summed (resulting in a 10 bit value) and stored in a separate line buffer. At this point, the block is ready to begin streaming out pixels for the composite HDR image.

Algorithm:

Several methods of HDR conversion were tested in Matlab prior to hardware implementation, including multiple camera response functions, exposure weighting, and logarithmic scaling. The purpose of these tests was to maximize hardware performance by selecting the simplest algorithm in terms of mathematical calculations while still showing a definite improvement in edge detection over the non-HDR images. Since the ultimate goal of this project was to use HDR as a method of enhancing object detection, higher quality of edges in the final image was chosen over more aesthetic features such as vibrance and correctness of the colours. Given this requirement, it was found that a significant increase in edge detection could be observed by simply calculating an average of the four exposures.

Some HDR algorithms also recalculate the inverse camera response function for each set of images to produce more tailored results based on the environment of the image (i.e. daytime vs nighttime): however these calculations are slow and difficult to implement in hardware. To avoid overfitting the function to a specific environment, several sets of images in different environments were used. Finally, to avoid floating point calculations, the function values were rounded to the nearest integer. This introduced some banding in the output image but had no significant impact on the edges.

2. Object Detection

   Functionality:

   The Object detection block is also an AXI streaming IP like the HDR IP module. The input stream initially waits for three rows to initiate the sobel edge detection transformation. After this, it waits for one row of pixels before initiating another round of operations to start sending data out.

The input to this IP is an 8 bit greyscale value taken from the Y component of the RGB to YCrCb conversion block before it. The output port is also an 8 bit greyscale value of either 0 or 255 connected to a YCrCB to RGB block.

Hardware Operation:

The Sobel transformation is a 3x3 matrix centered around the pixel being sent to an output buffer. In order to obtain the values for this matrix, the block must always have 3 full rows of pixels before initiating a calculation. After the completion of the calculation, the output row buffer transfers the pixels to the next block in the streaming line, the last two rows shift upward and a new row from the pixel stream is added to the third row. This process continues until a counter hits 360 rows.

Algorithm:

The Sobel transformation module computes both the x edge and y edge values at the required pixel location and determines what value to assign to it, either 0 (black) or 255 (white). It does this using three conditional statements. First, if the combination of x and y edges is greater than 255, the value at that pixel is strongly white and the output value for that pixel is 255, the maximum light quantity. Next, the sum of the squares of the x and y edges are taken and compared to a threshold value (originally 1225) that measures how strongly there is light in that pixel without taking a square root (the square rooted value for threshold is squared). If it is below this light threshold, the output pixel is 0. Lastly, any other pixel falling outside of those conditions gets assigned a white value (255).

5 Design Tree

- docs: Contains a copy of the presentation slides and final and report.
- src/software/edge_generation: Contains python scripts used to compute the HDR images.

- src/fpga/hdr_final/toplevel/ddr_dma_v2.xpr: The Vivado project for the hardware
- src/fpga/hdr_final/toplevel/ddr_dma_v2/ddr_dma_v2.sdk: Contains the SDK project files for the Microblaze program
- src/fpga/hdr_final/edge_new: Contains the Vivado project files for the object detection IP block
- src/fpga/hdr_final/hdr_ip_sof_fix: Contains the Vivado project files for the HDR IP block
- src/fpga/hde_final/video_to_axis: Contains the Vivado project files for the video to AXI IP block