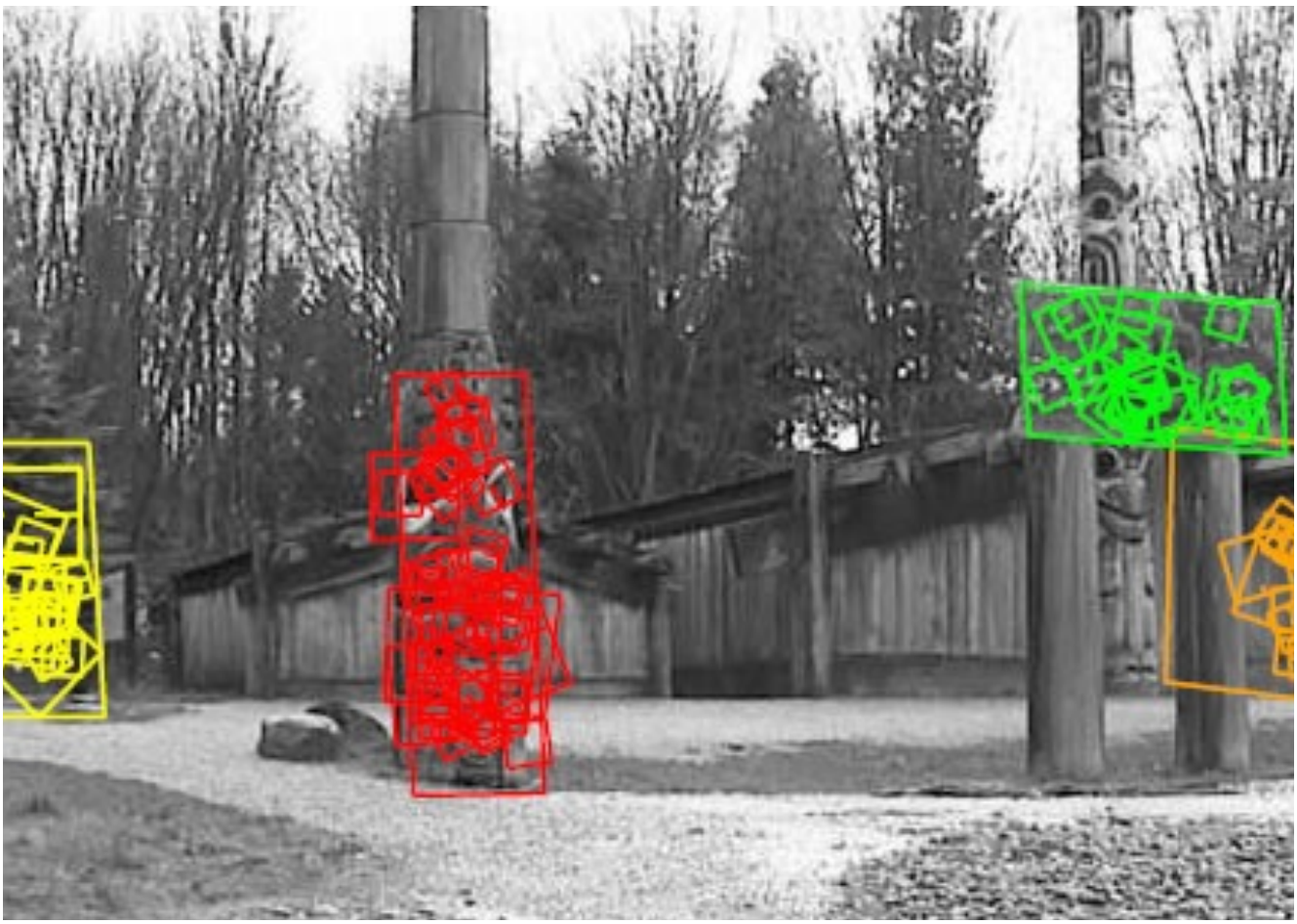
# To Detect or Not To Detect

## Implementation of scale-invariant feature transform (SIFT)

Nathan Brummel & Tyler McAtee - December 11, 2013



*(SIFT)*

# Abstract

This paper discusses our process for adding a hardware acceleration module to the Scale Invariant Feature Transform (SIFT) algorithm. When running an algorithm often about 90 percent of program runtime energy is consumed by 10 percent of the code. These parts of the code are frequently data processing intensive, and by adding custom hardware to speed up these sections, the overall algorithm speeds up drastically.

The SIFT algorithm is a computer vision algorithm used to detect and describe local features in images. We described several hardware modules using Verilog to aid in expediting the process of this algorithm. The first major module discussed in this paper is the SRAM Arbiter, which efficiently services two write ports and two read ports that are trying to connect to an external SRAM module. The second major module discussed is the Difference of Gaussians calculation module, which takes in a byte stream of pixel data, down samples it to a more manageable size (due to on-board FPGA memory constraints), runs a Difference of Gaussians calculation, and up samples it to the original size again.

## Overview

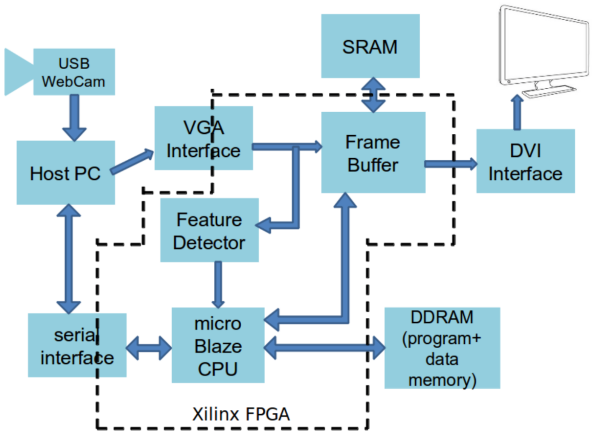



*Figure 1. Overview of Scale-invariant feature transform (SIFT) implemented in computer architecture (CS150 - Project).*

## Design

### Down Sampler

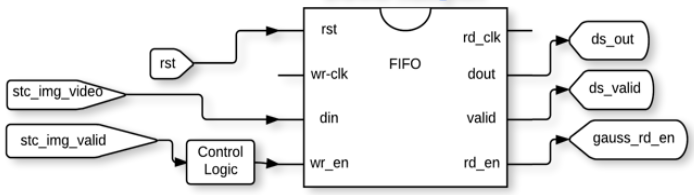



*Figure 2. The Down Sample module takes the video input and down samples it by half.*

When video data is transmitted through the down sampler (Figure 2) it becomes half the size and at a decreased image quality. This allows a system to need much less memory then if the image was filtered in its' original state. Following the operation the data is temporally stored in a transitional first in/ first out (FIFO) buffer.

### Gaussian Wrapper

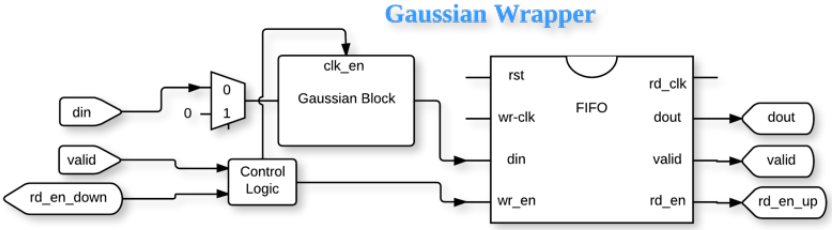



*Figure 3. The Gaussian wrapper takes a data input and prepares it to be filtered.*

Figure 3 is the gaussian wrapper module which prepares the input data to be filtered by the gaussian filter module. If needed the wrapper pads the incoming data to accurately allow the gaussian filter to perform the filtering operation. Additionally, the module stipulates which data is sent to the up sampler module in the pipeline.

### Difference of Gaussian

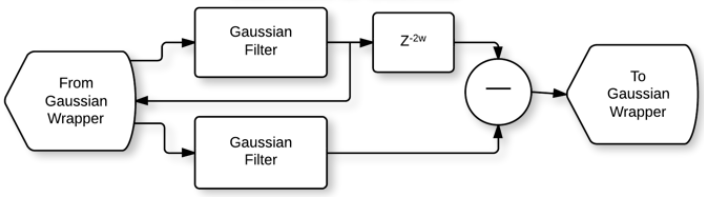



*Figure 4. The Difference of Gaussian takes the difference of two gaussian filtering operations.*

After data is prepared by the gaussian wrapper module it is fed into the difference of gaussian module (Figure 4), which subtracts the output of two gaussian filter module. Then the date is sent back to the gaussian wrapper.

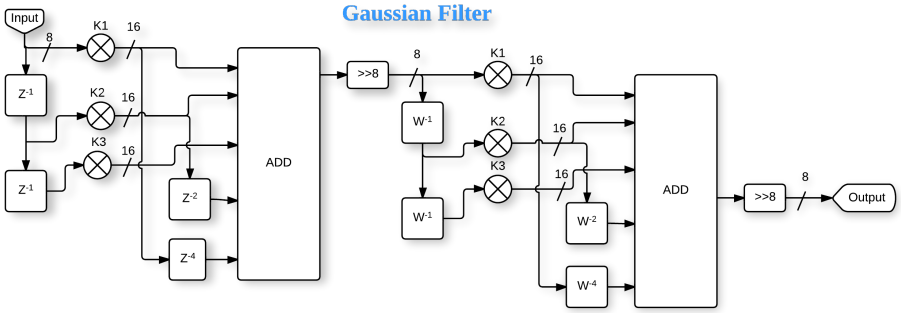



*Figure 5. The gaussian filter module preforms the mathematical gaussian filter operation on data. First the row operation is done then the subsequent column operation in calculated.*

The gaussian filter module (figure 5) performs multiplies a set of five adjacent row pixels in accordance with a gaussian operation. Then it accomplishes the same task with a set of five adjacent column pixels where the center pixel is the same center pixel in the row operation. The / indicates the bit width of the data line. K1 - K3 are constants of each gaussian filter stage. >>8 specifies a bit shift to the right of 8 bits. Z-1 and W-2 denotes a shift register where the Z indicates a bit and W indicates a row. The number show how many shift are needed.

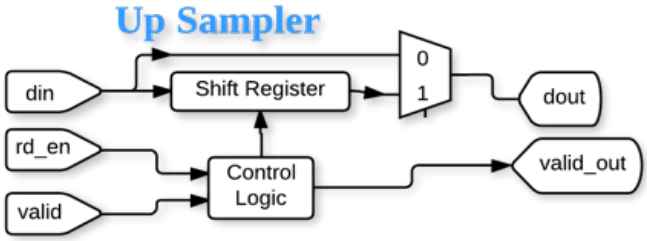



*Figure 6. The up sampler module takes a data input and increases the size by two. For example, if the data in is 400 x 300 it will be up sampled to 800 x 600.*

As a video frame transfers into the up sampler module (Figure 6) it will be converted to frame of double the size. For example, if the input frame size is 200 x 150 pixels then the up sampler module will convert the frame to a 400 x 300 pixel size.

# Detailed System Description

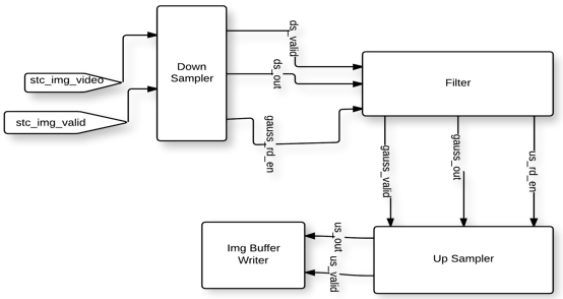



*Figure 7. A basic overview block diagram for an implemented difference of Gaussian filter to include the whole where a video signal is down sampled, filtered, and up sampled before it is presented to the image buffer writer module. The entire system runs on the video graphics array (VGA) clock.*

## Down Sampler

The video input, which can come from a static image writer or VGA, is feed into the DownSampler module (Figure 2). Figure 8 is a visual representation of the down sample operation. The x indicates the pixel that the DownSampler module (Figure 2) extracts from the input frame. Thus, the overall frame size is decreased by half.

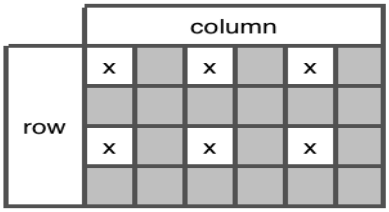



*Figure 8. Displays an example of which pixels the down sampler module extracts to decrease the frame size by half.*

This is accomplished with the use of two counters. The first counter permits the even column bits in a row (a row contains columns starting from 0, 1, 2, •••, N; where N is the last pixel in the row). While the other counter passes the even rows (the frame starts from 0, 1, 2, •••, N; where N is the last row). This data is fed into a FIFO in order to allow for an adequate buffer between the main gaussian modules and the input stream.

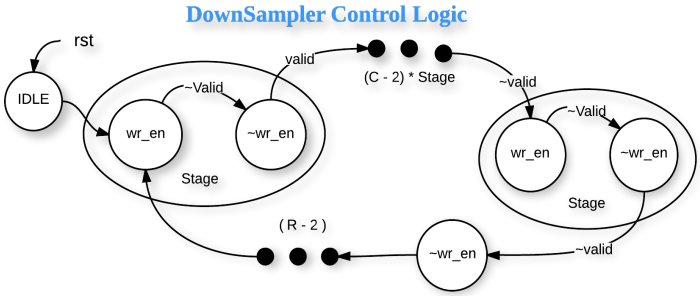



*Figure 9. The finite state machine (FSM) for the DownSampler module. Where C denotes number of columns and R denotes number of rows in the frame.*

The FSM of the control logic for the DownSampler module is displayed in Figure 9. After each frame the module is reset. Data is fed into the FIFO when wr_en is high. The frame data is then sent to the GaussianWrapper module (Figure 3) to be filtered.

## Gaussian Wrapper

Upon a valid high from the DownSampler FIFO the GaussianWrapper (GW) module (Figure 3) begins to process the frame data. Due to the fact that the GW contains the difference of

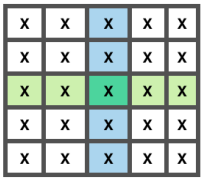


*Figure 10. Displays the five x five grid which is used to perform the gaussian operation on. The horizontal calculation is in light green and the vertical is in light blue for the dark green block (pixel).*

gaussian (DOG) module (Figure 4) and the Gaussian (GFM) module (Figure 5) it is one of the most integral components in the pipeline. We chose to separate the frame into five x five grids which is showcased in Figure 10. The figure shows that to calculate the filter operation

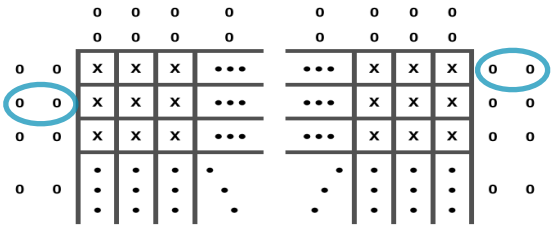


*Figure 11. Illustrates the padding applied to the frame by the GW. The zeros that are circled are the same zeros since the date is in reality a stream of data not a two x two array. The zeros above the frame are in the pipeline due to reseting the GW.*

on a particular pixel one must use five pixels in the horizontal and vertical directions as displayed in Figure 10. Thus, to perform the gaussian operation affectively the GFM must be provided with a set of five valid data bits in the row and column directions. This is vital to obtain an accurate calculation. To illustrate this Figure 11 displays where the padding is applied.

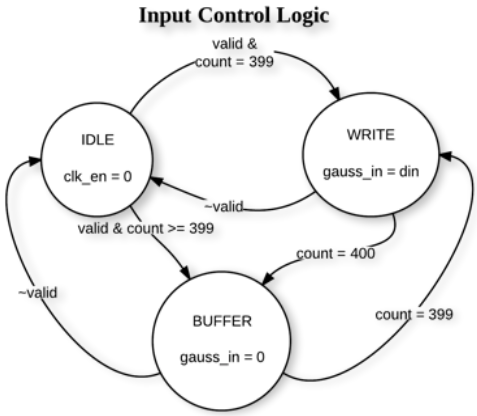



*Figure 12. The finite state machine (FSM) for the input of the GW module.*

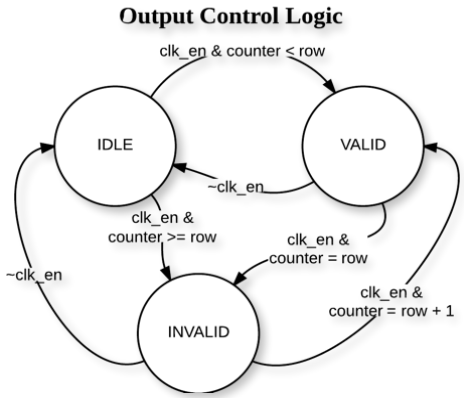



*Figure 13. The finite state machine (FSM) for the output of the GW module.*

The zeros above the frame are present due to the fact that the GW is reset after every frame. Additionally, zeros must be present after each row in order to correctly calculate the last bit of the row and those same zeros are also needed to calculate the first bit of the next row. An example of this is illustrated by the zeros that are circled in blue in Figure 11. At the conclusion of the frame, the last two rows, there must be two rows of zeros present in order to finish a correct filtering operation of the frame. As Figures 12 and 13 exemplify the GW

keeps track of the valid signals throughout the procedure and informs the up sampler when to accept valid data.

### Gaussian Filter Module

Within the GFM revealed in Figure 5 the down sampled frame is then filtered using gaussian function. Essentially, the data is processed per Figure 10 where first the horizontal operation is completed. Then the vertical operation is calculated. The calculation is as follows:

$$g(x,y) = \frac{1}{2\pi\sigma^2} \cdot e^{-\frac{x^2+y^2}{2\sigma^2}}$$ (R.A. Haddad)

The calculation was separated into two distinct steps since the gaussian function is separable in x and y coordinate. Additionally, the function is symmetric on the mean. The the center pixel from Figure 10 was taken as the mean. This allowed for the ability to decrease the amount of multiplication operations. Upon the multiplication operation the data was converted to a 16 bit size to keep sensitivity; however, after the data has been added it needs to be subsequently converted back to an eight bit number. Thus minimizing the hardware utilization of the entire GFM. The shift registers were need to align the data in the horizontal and vertical direction.

Furthermore, the '>>8' blocks are used to normalize the data by 256. The large shift registers were created using the Coregen tools, which created a block ram implementation of series of shift registers. This was accomplished by essentially using two pointer, one N number of spaces away from the other. Then as both pointers increment the output pointer will reach the first data after N times. This will continue to occur until the generated register is reset or clock enable is not asserted.

### Difference of Gaussian

The DOG (Figure 4) takes the output of one GFM and inputs it into another GFM. Then the two outputs of the GFMs is subtracted to obtain an difference. For the same pixel to go through both GFM and for the difference to be taken the output of the first GFM must be delayed by two widths. This is accomplished by using a large shift register as explained in the section for the Gaussian Filter.

### Up Sampler

The UpSampler (UP) module (Figure 6) takes the filtered image and up samples the image to double the size. For every input, (A,B,C•••), then the output will be (A,A,B,B,C,C,•••), which is illustrated in Figure 14. To do this the clock signal of the FIFO was half the speed of the VGA clock and is showcased in the FSM for the input control logic of the UP in Figure 15.

ABCD••• => AABBCCDD•••
AABBCCDD•••

*Figure 14. Show the input (left) and the resulting output (right) of the UP.*

This was accomplished by using a register and an inverted to cause the VGA clock to double in length per cycle. Additionally, the row is fed into a shift register which stores the entire row.

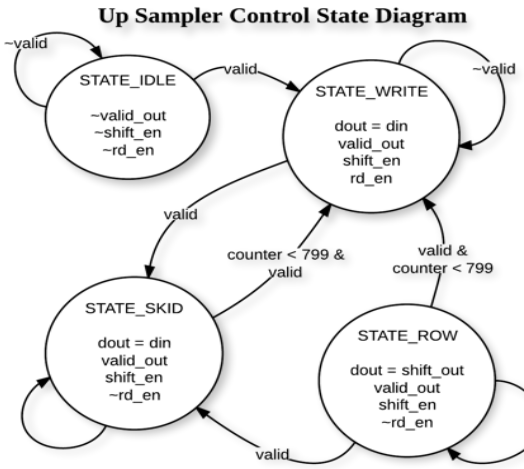



*Figure 15. The FSM for the input data control of the UP module.*

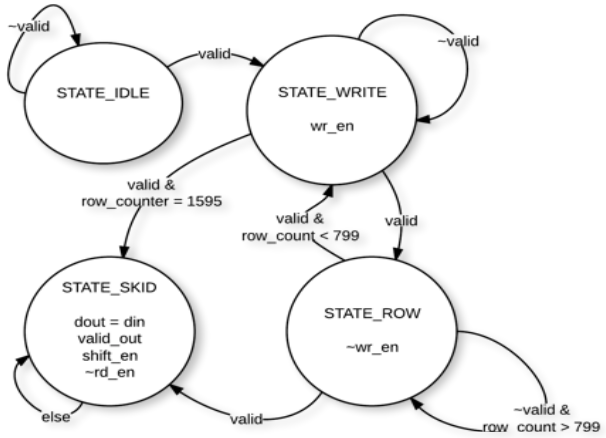



*Figure 16. The FSM for the FIFO control of the UP module.*

Upon completion of a row the multiplexer changes to the shift register's output which is displayed in Figure 16. Thus, the row is output a second time. When this is occurring the rd_row is low. This prevents the FIFO from outputting and preserves the next row to repeat the up sample process.

## SRAM Arbiter

The SRAM Arbiter module is a controller that shares the interface with the SRAM and has the ability to control two write and two read ports even thought the SRAM is a single read write memory. It also takes an input from the ImageBufferWriter. Each port can submit a request regardless of the behavior of the other ports. Thus, this maximizes the throughput and provides fairness to each port using. The fairness is implemented using a round-robin arbitration scheme. For instance, the order is R0, R1, W0, W1 and the overall operation can be followed by looking at Figure 18, which describes the FSM for the SRAM Arbiter. The main block diagram for the arbiter (Figure 19) establishes the correct data paths for data to transmit

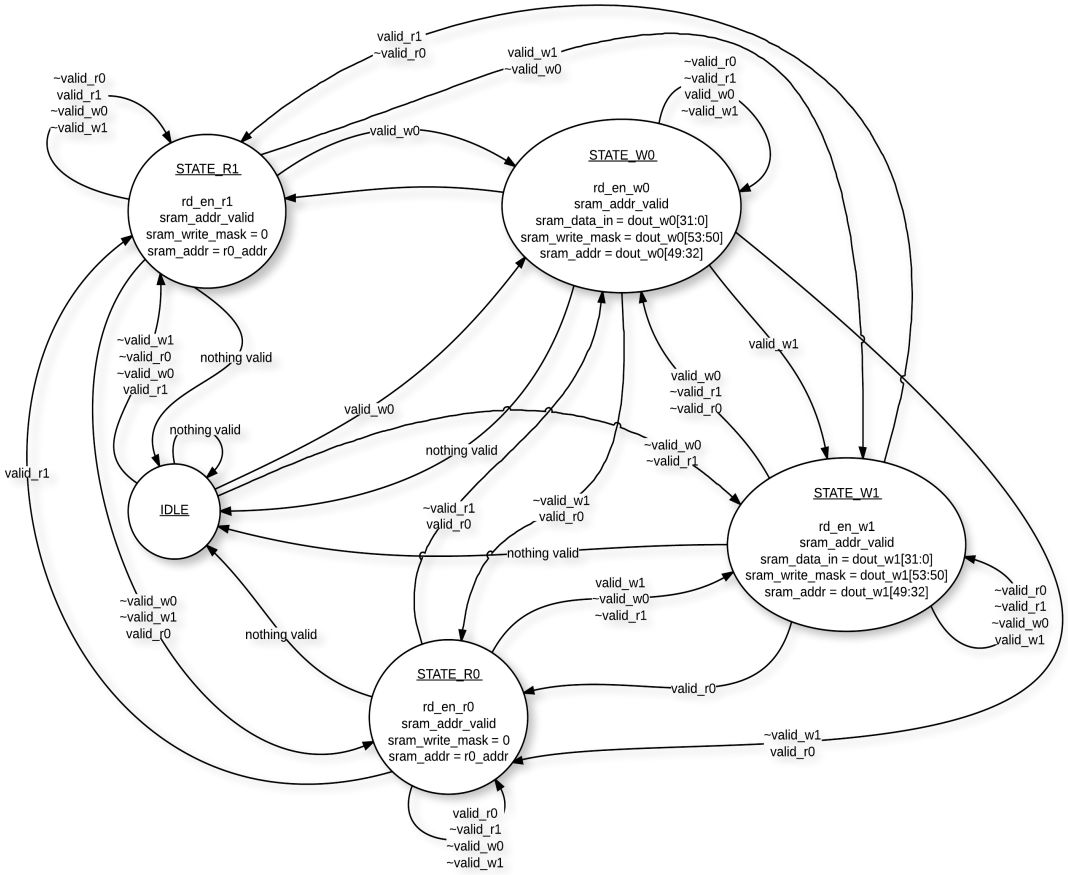



*Figure 18. The FSM for the control logic of the SRAM Arbiter module.*

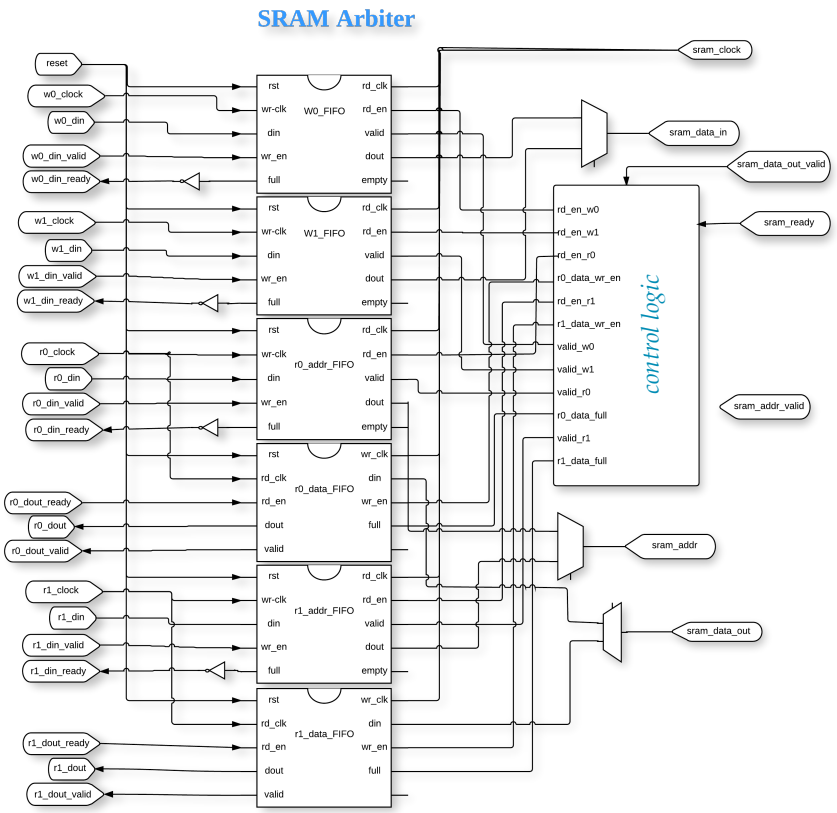



*Figure 19. The block diagram of the SRAM Arbiter.*

to the SRAM or to be read from the SRAM.

## Design Decisions and Tradeoffs

As stated in Figure 7 the decision was made to use the VGA clock. The main reason was that implementation of the pipeline had began with the VGA clock and it was determined that due to the use of FIFOs as buffers that another clock was not needed. However, the clock signature for the UpSampler module for reasons discussed in the Up Sampler section was altered.

The decision to pad the incoming data was made to establish a consistent basis to calculate the extrema. This allowed for the gaussian function to accurately calculate the filtered pixel output. Consequently, this added a new layer of difficulty to the control logic of the GW. By carefully developing an efficient FSM for the module the correct output was realized.

For all module control logic Moore machines where used instead of Mealy to prevent placing a Mealy after or before another Mealy machine. This allowed the design to become more simplistic leading to fewer design errors and overall less bugs.

Problems occurred in filtering & lining up the pixels when preforming the Difference of Gaussian blocks operation. To alleviate this a control block which coordinate the correct lining up of the pixels. This was due to the implementation with no regard to where a particular pixel is on the screen. From the UP the pixels are fed to the ImageBufferWriter module which then assigns the pixel to the mask and the corresponding location on the screen.

Additional problems developed when testing. When designs where initially instituted it was extremely hard to unit test individual components. Therefore, the designs were redesigned

| | 6-LUT | FF | BRAM | DSP |
|---|---|---|---|---|
| **GAUSSIAN** | 498 | 168 | 0 | 6 |
| **GAUSSIANWRAPPER** | 2128 | 931 | 0 | 9 |
| **DOG** | 3207 | 1331 | 0 | 9 |
| **UPSAMPLER** | 243 | 78 | 0 | 0 |
| **DOWNSAMPLER** | 363 | 179 | 0 | 0 |

*Figure 20. The utilization of the various blocks to include number of 6-input Look Up Table (6-LUT), Flip Flop's (FF), Block Ram (BRAM), and DSP48E (DSP).*

to permit easy testing. This had an added benefit of minimizing bugs and making the modules easier to understand and implement with other modules.

# DESIGN METRICS

Figure 20 displays the utilization for the GFM, GW, DOG, UP, and DownSampler modules for 6-LUT's, FF's, BRAM, and DSP. The critical timing path was defined from the GW to GFM to a width shift register then to the same GW. After the GW the signal is fed to another GFM and then through the the adder for the GFM. The maximum clock frequency was determined to be 48.888 MHz with a clock period of 20.455 ns. The utilization leads to that fact that ten GFM could be created due to the use of nine DSP's out of 64.

### Division of labor

The beginning of each stage there was a discussion at the beginning of who was going to tackle what parts, where one person was to design and implement the module while the other was to develop the test. This insured that both people understood the module in question and that it actually preformed as required.

Tyler designed the SRAM Arbiter, Gaussian Wrapper, Difference of Gaussian, SRAM Arbiter test, and Up Sampler. Nathan designed the Gaussian Wrapper Test, Difference of Gaussian Test, Gaussian Filter and Test, Down Sampler, and Overlay. The ratio of design to debug was approximately one hour of design to 2 — 2.5 hours of debugging/testing was done. The estimation of about 240 man hours were spent during the entire project. This would equate to nearly 80 hours of design and 160 hours of debugging and testing.

# Conclusion

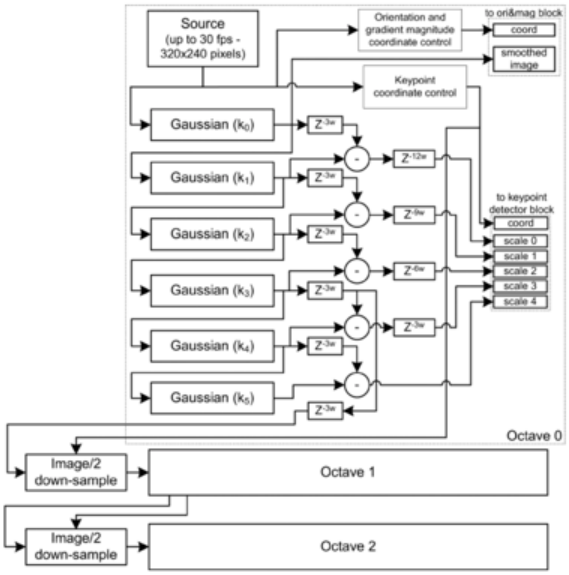



*Figure 13. Pipeline architecture implementing the Gaussian filter cascade and the difference of Gaussian function (Bonato).*

## Summary of main features

The final iteration of our project was able to quickly and reliably display the static image of Campanile Tower, a single filtered Gaussian image, a double filtered Gaussian image, and the difference of Gaussian displayed in two different formats: a scaled format done by shifting the result left 3 bits, and a result where every pixel has a constant added to its value. The output can also be configured to display the test grayscale pattern, as well as the shifting grayscale pattern. These seven options can be successively displayed and swapped between using the GPIO DIP switches.

The final design leaves room to grow -- the resource usage was low enough to add more gaussians and finish multiple octaves. After this, the next step would be to implement the feature detection module and pass on the results of that to software to finish the Scale Invariant Feature Tracking process.

## What would you do differently next time?

A lot of problems occurred in filtering & lining up of the pixels when doing the Difference of Gaussian blocks. A coordinate control block is included. The implementation accepted pixels with no idea of where on the screen they belong, and then feed them into the ImageBufferWriter module that assigns a pixel and mask. A better implementation would take the output of the ImageBufferWriter and keep track of the pixel location belonging to each pixel. This would make lining up the pixels for the difference of gaussian calculation easier alleviating the problem of shifting that occurred on many implementation.

Beyond this, more extensive testing and clearer goals would help. The arbiter threw some very subtle errors when working on the Difference of Gaussians that were not detected at all before.

# References

Bonato, V.; Marques, E.; Constantinides, G.A., "A Parallel Hardware Architecture for Scale and Rotation Invariant Feature Detection," Circuits and Systems for Video Technology, IEEE Transactions on , vol.18, no.12, pp.1703,1712, Dec. 2008 doi: 10.1109/TCSVT.2008.2004936

CS150 - Project. (n.d.). *CS150 - Project*. Retrieved December 11, 2013, from http://www-inst.eecs.berkeley.edu/~cs150/fa13/project

R.A. Haddad and A.N. Akansu, "A Class of Fast Gaussian Binomial Filters for Speech and Image Processing," IEEE Transactions on Acoustics, Speech and Signal Processing, vol. 39, pp 723-727, March 1991.

SIFT: Scale Invariant Feature Transform. (n.d.). *AI Shack*. Retrieved December 12, 2013, from http://www.aishack.in/2010/05/sift-scale-invariant-feature-transform/