Image Binary Classification

6.111 — Fall 2015 — Final Report

Juan Huertas and Andres Salgado-Bierman December 9, 2015

1 Abstract

The Project aims to implement a computer vision algorithm on the FPGA to extract features from an image. A new image would then be generated with these features and sent out via VGA. Current feature detection first acquires an image that has been preprocessed to extract key information, then creates features, and finally tests candidate objects against a model or classifier, typically to make a pass fail judgement. The process is often application specific and prohibitively expensive at high detection speeds. In this project we would start by generating texture values for each pixel using LBP (local binary patterns), then generate a feature space by binning the LBP values and creating a histogram of their frequency that can be compared against a precomputed linear classifier boundary. This allows the user to use the same hardware for different detections by just swapping out the classifier and letting the hardware handle feature generation. The overall effect should be real time candidate and feature generation, as well as, candidate classification where the generation of the classifier boundary is abstracted away. The texture values will be displayed as pixel luminance to display a filtered video stream.

2 Acknowledgments

We would like to thank Gim Hom the course instructor for having tailored this class over the years. The structure of the course prepared us to take on the final project. Understanding and implementing finite state machines in verilog was critical for our project and we learned this from earlier labs. The fifth course lab with the 31 tap low pass filter taught us the skills we needed to pipeline the calculations we needed to perform for classification. Finally lab four taught us how two write and debug verilog in a sensible way to ensure proper execution of our modules.

A big thanks to the teach assistants Miren and Ariana. Miren and Ariana both worked with the NTSC camera in their final projects for 6.111 and were able to help us out when we were having trouble with the camera. We would like to thank the CIM instructors and Jared in particular for his guidance in adding structure to our reports and making sure we were using clear language. Finally we would like to thank all the other students in 6.111. Lots of little problems were fixed by sharing experience with the teams working the next lab bench over.

\

3 Introduction

Image recognition is a broadly used practice in industries ranging from video games to manufac- turing. However most detection cannot be done in real time at high reliability on hardware. Current implementations use feature algorithms that are not as effective as traditional software approaches, or require large amounts of hardware to generate CNN models. Software on the other hand will never reach the specificity of creating efficient timing in a digital circuit. Our recognition system aimed to be able to generate a feature representation of the textures from a low resolution picture of a face to classify and display objects in real time. For our final project we wanted to design and implement our own facial recognition system in hardware using efficient feature representations as well as some hardware adjusted software approaches to object recognition via linear classifiers. The basic output was a yes/no response to whether there is a face in the video frame. In addition we displayed the texture features on screen as a filtered video stream.

This report will first cover our motivation for the project. There will be a brief discussion of previous work before we discuss our system architecture. Once the system architecture is clear we will discuss the design decisions that brought us to our implementation. The details of each module of the implementation will be covered next along with how we tested the system as it was built.Finally, we will discuss the successes and setbacks we encountered over the course of the project.

4 Motivation

On a higher level our interest in protecting personal privacy was a motivation for the project. In typical video security systems you are surveilled and your data is kept for months if not years. We would want there to be an end to the constant video surveillance that happens whenever you walk into a store or around the city. There is a big difference between video surveillance and human memory. When one looks or watches not every detail is recorded, but rather key abstractions of events are remembered. Imagine all of the circumstances where you would welcome the company of others yet feel uncomfortable if someone was filming you. The dream behind this project comes from imagining a world in which video security systems can look as a person looks, only storing abstract higher level information, as opposed to surveilling you and recording every little detail.

The aim of this project was to explore a hardware solution that could bring us closer to this dream. We aimed to implement a computer vision algorithm on the FPGA. The idea is that if machine learning algorithms can be successfully deployed in the digital systems that take video security feeds observations can be extracted without

having to record the video. For example, an extensive video security system was recently implemented across MIT dormitories, with multiple security cameras at each door. The video from these cameras is kept for an extend period of time by the MIT police. As a student with limited housing options it feels as if we are being surveilled in our home, with all of our comings and goings being recorded and time stamped. If there was an asic that identifies faces it could be attached to each of these video feeds. Then as a registered student or resident of the building is entering or exiting the camera would know it is them and not record any video. If the chip could recognize strangers then it could perform equally as well for security purposes, recording video of anyone who is not supposed to be in the building entering and exiting.

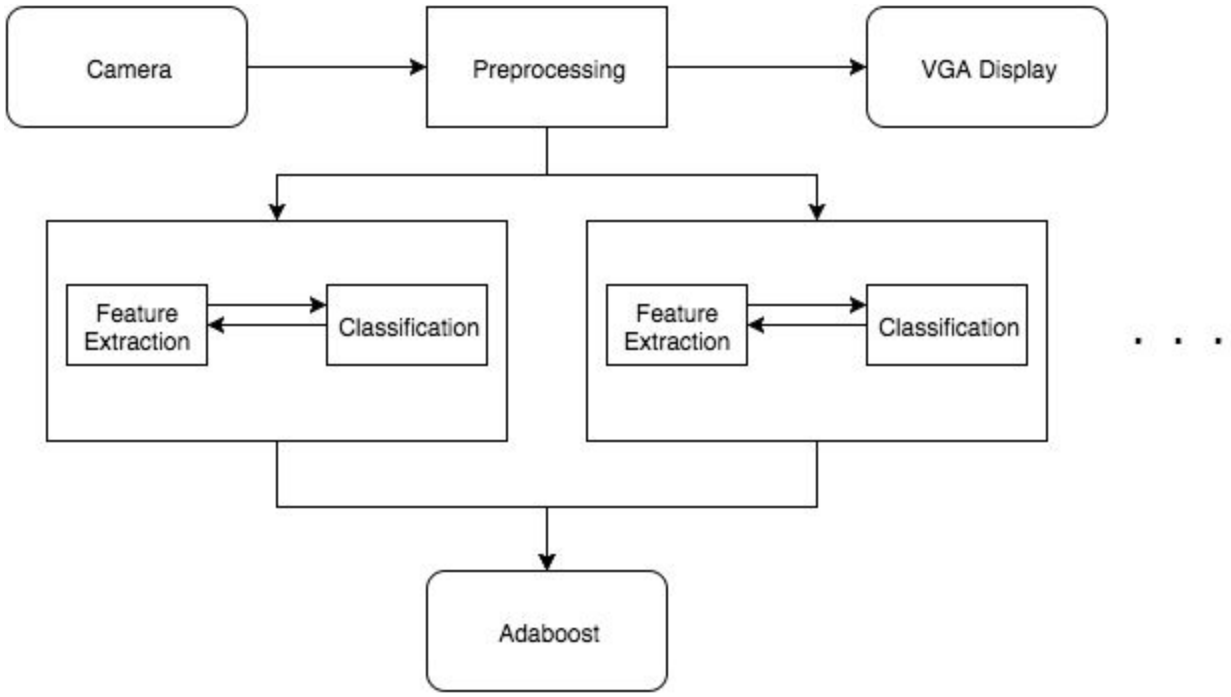
The computer vision approaches we were interested in are already implemented frequently in software. What is interesting about implementing in hardware is the ability to parallelize the calculations so that decisions can be made on each frame of the video stream.

5 Algorithm Selection

Several algorithms are required for object recognition, our proposed implementation opts for min- imizing complexity of the computer vision algorithms implemented while achieving high fidelity results in two key areas: feature kernel(Local Binary Patterns), and classifier (AdaBoost+SVM). Local Binary Patterns are broadly used in face detection algorithms, the kernel is used as a way to take color or chrominance data from images and translate it to texture data, the algorithm is not computationally intensive requiring no divisions or multiplications to be carried out. The computa- tion of the classifiers have been abstracted out into software such that only the final computation is carried out in hardware using precomputed, prestored coefficients to calculate the classifier boundary output. It is this pre calculation of coefficients that makes our system robust to classify a wide variety of objects. Example images are used to train the coefficients. To classify a different object the coefficients can be trained with different example images.

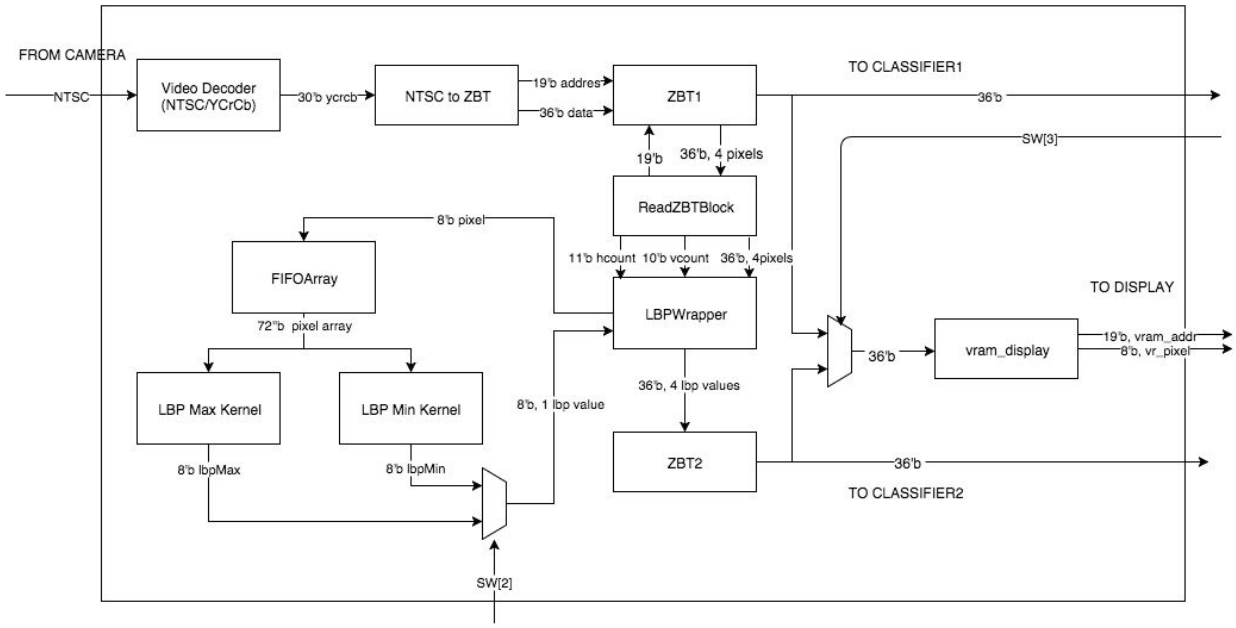
6 Module Descriptions

A big complication that arose came from originally wanting to do the image preprocessing directly from the live feed. Calculating the LBP kernel required us to change (the previously given) NTSC to ZBT and video decoder modules. After many hours of wasted effort, I decided to store the video-feed directly to ZBT and conduct the kernel computation to a secondary ZBT memory block, allowing us to be able to reuse the already existing and tested code with the tradeoff of effectively doubling our memory use. While this did not pose a problem on the Labkit boards, the change overhauls our previous architecture and is the reason for the changes in the overall block diagram (Figure 1).



6.1 Image Preprocessing (Juan)

6.1.1 ZBT Interface



Overall this portion of the project's implementation takes the input from the NTSC camera and preprocesses the image data to texture data, essentially bringing the luminance data into the feature space the classifiers will use to detect a face, and subsequently sending both the original black and white image as well as its feature space representation to separate classifiers, and the display in parallel. The output to vramdisplay module has been muxed to a switch so that images from ZBT1 and ZBT2 can be displayed within an XGA 1024x768 window. Several modules that were provided by the staff were reused. Rather than try to reinvent the wheel, we chose to focus our effort on the places where we would be adding new functionality to that end, the video decoder, ntsc to zbt, have been re-used directly, the vram display module has also been repurposed in the readZBTBlock module (Figure 2).

Creating a ZBT interface to use more than one ZBT memory block in tandem with the vram display took more time than anticipated but the result was crucial to being able to debug the rest of the project. In addition I have added an LBP wrapper module to serialize/deserialize the pixels written and read from each ZBT block, as well as implemented a secondary Kernel that is computed in parallel to the first and are then muxed to a switch to allow the manual change of input written to ZBT memory and subsequently used in the classification, and display.

6.1.2 FIFO Array

FIFOs were implemented by using a large shift register array to hold three complete rows of pixel values, while this allowed me to achieve my stretch goal for the Image Processing module, getting the timing right for this module was our main implementation bug. Each clock cycle the values in the array are shifted forward, using hcount[1:0] a different pixel value from the 36b ZBT data input is passed in while the next ZBT read gets the next four pixels from memory; the last pixel value in each FIFO is shifted out to the first register in the next and the module outputs the last three pixels in each row.

6.1.3 LBP (Max and Min) Kernels

The LBP Kernel block takes nine 8 bit luminance values for the brightness of each pixel and its neighbors to calculate the local binary pattern (LBP) for each pixel. The LBP is our measure of how the brightness of one pixel compares to the brightness of the pixels around it. In a 3x3 grid the middle pixel will be surrounded by 8 other pixels; The LBP uses the intensity of the centermost pixel to threshold the other pixels around it and create an 8 bit number where each number represents one of 256 different edge/texture values. While the Max kernel makes values above the threshold contribute a 1 to the LBP (Figure 3) the Min kernel raises the threshold value and makes values above the threshold contribute a 0 to the LBP.

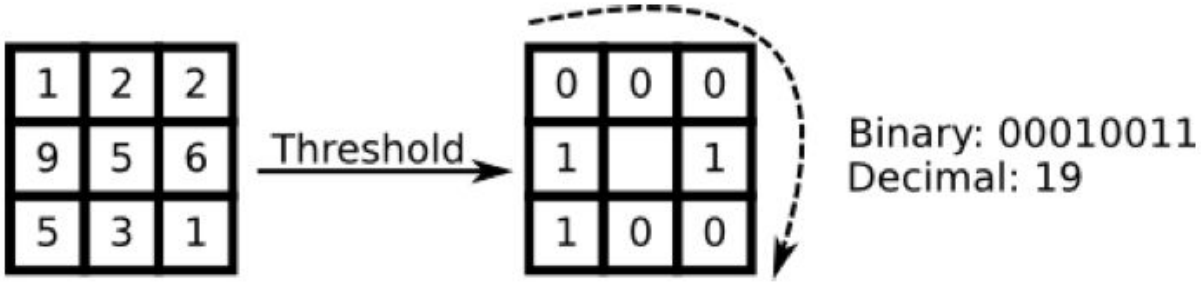


Figure 3: Example LBP Calculation

6.2 Feature Extraction (Andres)

A feature in our implementation is the distribution of LBP values over a region of the image. The image in this case is a frame of the NTSC camera. The feature extraction module listens to the vram display module. The vram display module takes care of indexing LBP values for each pixel from the zbt and outputting them along with the appropriate hcount and vcount for the VGA display. Using the hcount and vcount the first portion of the feature extraction module evaluates whether the incoming pixel is in the region for extraction. The module is parameterized to take as inputs the leftmost coordinate and uppermost coordinate of the feature on the image. The height and width of the feature are parameterized as well. Right before the first pixel of the feature the BRAM block storing the feature's distribution must have its values set to zero. This is important because values from the previous distribution's frame will still be store up until this point. When a pixel is recognized to be in range the BRAM block is told to increment the value at the index given by the LBP by one. This way at the end of the feature the address corresponding to each LBP value will have the tally of how frequently that value occurs in the feature window. There is also a logic value calculated for the occurrence of the last pixel in the window. This value serves as the done signal to let the classifier know that it can start to read values from the feature's BRAM. To obtain a better classification multiple parameterized feature extraction modules can be instantiated in the top level module of the system (Figure 4)

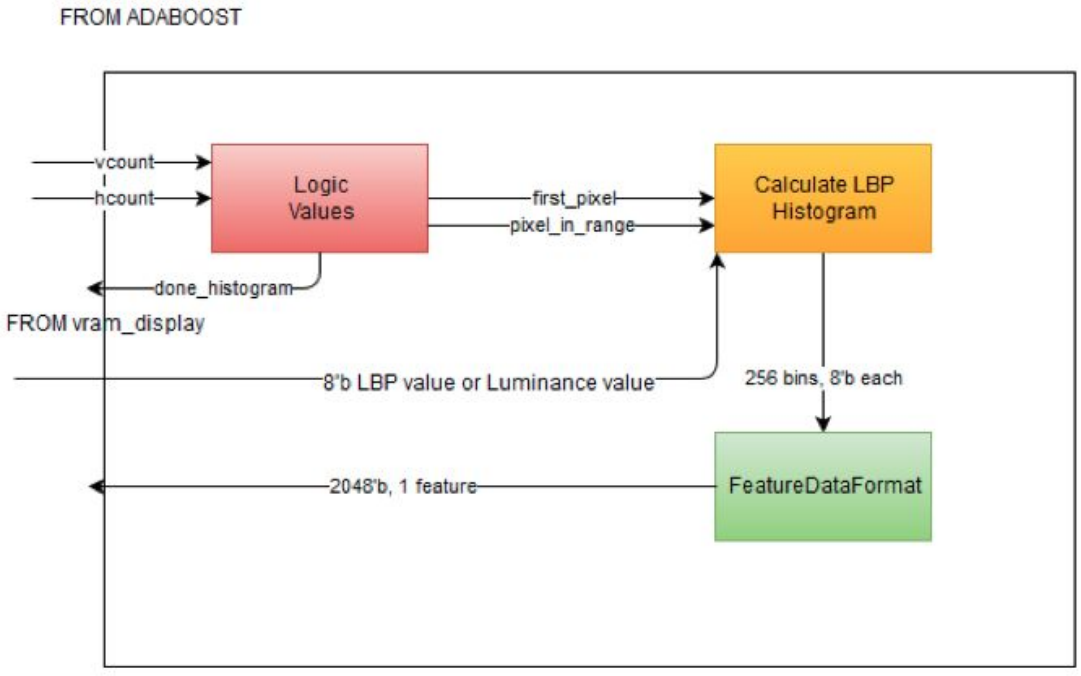


Figure 4: Feature Extraction

6.3 Classification (Andres)

A classification module is instantiated for each feature. The classification module waits on the done signal of the feature extraction module. The classification module is used to take the dot product of the feature vector and the coefficient vector. The way this calculation is performed is pipelined. Over 256 clock cycles takes the product of one component of the vectors and adds this to the running sum. The coefficients are stored in a ROM as 8 bit numerators. For the calculation the coefficient is multiplied by the feature component and bit shifted by 8 for division and averaging. Once the classification module as performed the multiplication and division for each component of the vectors it sends the final classification sum along with a done signal to the AdaBoost module (Figure 5).

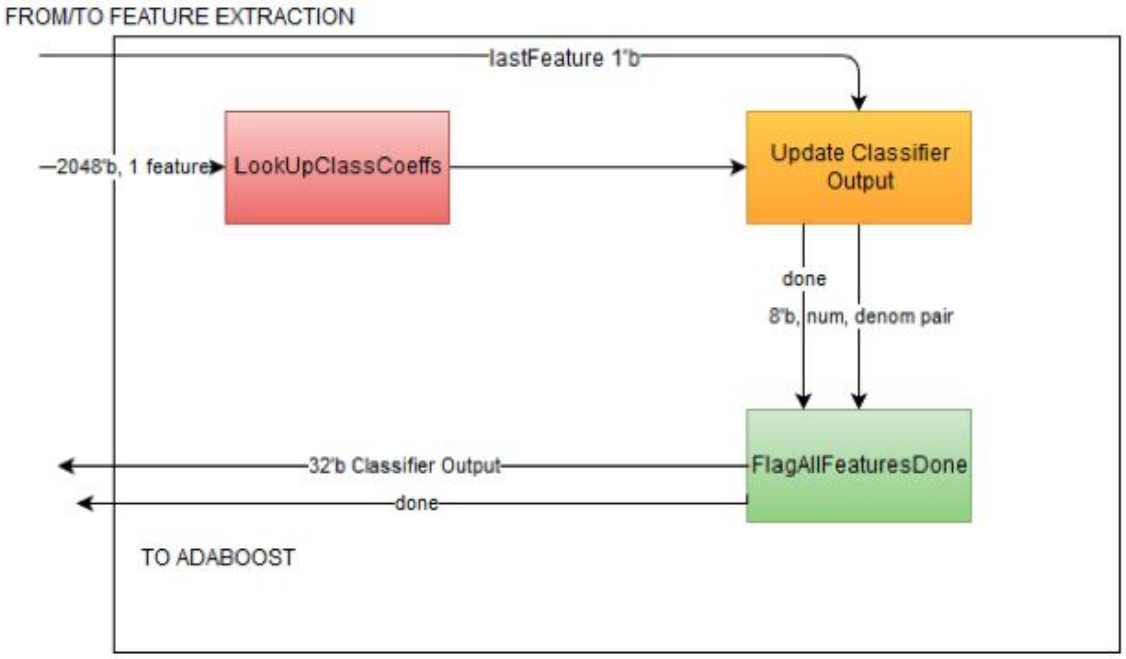


Figure 5: Classification

6.4 AdaBoost (Juan)

AdaBoost functions in a similar way to the candidate classification (Figure 5). Each 32 bit value from candidate classification is multiplied by a 32 bit coefficient. The coefficients are representative of the importance of each feature to the overall classification, and are used to improve accuracy. The sum of each of these multiplications passed through an activation function that gives a binary answer to the question of whether or not there is a face in the photo. Coefficients are obtained in the same manner as in the candidate coefficient block. While we are currently only boosting the decisions of two sub-problems, the architecture I designed is capable of expanding to calculate many more subproblems across separate classifiers (Figure 6). Finally the binary decision and not yet thresholded classification value is output to a Hex Display and an LED on the board (respectively) to visualize the output.

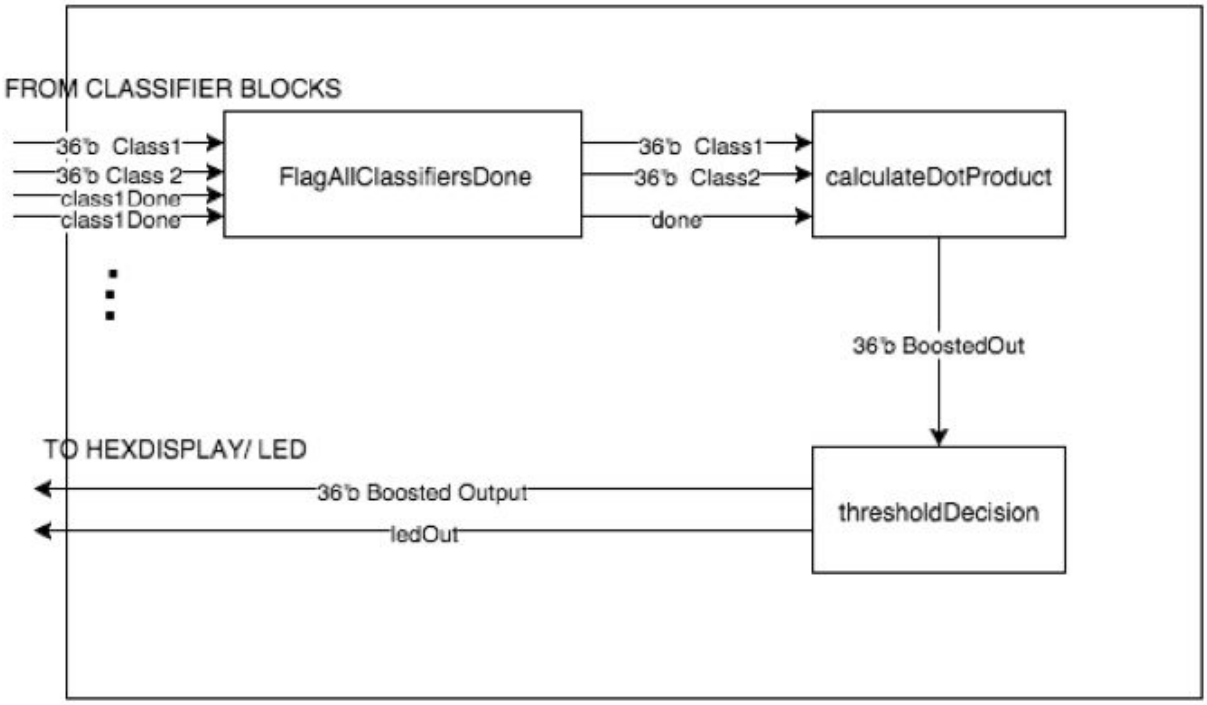


Figure 6: AdaBoost

7 Testing

Learning from the course labs modules were written along side with a test bench. The biggest challenge we faced in debugging was indexing; the ZBT memory had to be index, pixels had to be indexed, histograms had to be indexed, and coefficients had to be indexed. Write test benches along with the modules helped to find and fix indexing issues early. For the feature extraction module first the logic values of indexing the pixels were tested. A test bench was run for the classifier to ensure in the waveform that the correct coefficients were being multiplied by the correct feature component and that the overall classifier was being calculated correctly. The visual display for the LBPs help with debugging the preprocessing module.

8 Results

We were able to complete the preprocessing and zbt interface modules to display the feature rep- resentation of the image (LBP value of each pixel) on the screen (Figure 7). The image from the camera was also stored in ZBT such that we could switch between displaying the grayscale image and the filtered image. We were also able to instantiate the feature extraction module, for a couple of different windows on the black and white image as well as its processes counterpart to extract the features and run pipelined calculations with different coefficients that we had precomputed in matlab and stored in ROM. Unfortunately, we were not able to generate a robust enough classifier from images taken with a different camera, to be able to detect faces reliably. However we were able to implement the code in matlab to appropriately. In matlab we calculated th LBPs and performed feature extraction. As well we were able to generate the verilog code to make the coefficients file. We started with a script that calculated floating point coefficients. Then these values were normalized and converted into 8 bit integers representing the numerators of the numerator denominator pairs we used for the coefficients on the FPGA.

Figure 7: AdaBoost

9 Conclusion

While the degree of accuracy of our described method remains to be proven, we completed our projects hardware focused goals and were able to recover from a large change in architecture before it was too late. We were able to create a feature representation of the image though local binary patterns, as well as compute a boosted classifier decision (from a precomputed perceptron model) with only hardware at 30 frames per second. Real time image processing, feature extraction and classification can be accomplished in real time by exploiting parallel architectures.

Our advice to others would be to reuse as much of the previously tested core modules that are provided by the staff. Rather than try to reinvent the wheel, its important to focus your effort on the places where you will be adding new functionality . Furthermore, when accounting for time spent on the project you should budget ample time for testing and debugging, even if your project is technically feasible, integrating modules togethers could bring about unforeseen complications.