Identity Checker on FPGA

Author: Junye Chen, Sheng-Hao Huang, Andy Shen, Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—A facial detection and recognition system that runs on an FPGA. We want to explore the advantages of running the system on an FPGA rather than running the system purely in software. Our proposed hardware architecture is in SystemVerilog and implemented mainly on a Xilinx Kintex-7 FPGA.**

***Index Terms*—Facial Detection, Facial Recognition, FPGA**

I. Introduction

Facial recognition has become very popular in the field of computer vision and machine learning because of its wide range of applications. It can be used to check attendance in classrooms or in the workplace, and can improve security measures in critical areas. Facial recognition can be divided into two steps: detection and identification. Two of the most common methods for performing the detection step are the Viola-Jones algorithm and neural networks. One common method for performing the identification step is the Eigenface algorithm. Most systems, including the ones that use the aforementioned algorithms, implement facial recognition purely in software. However, since both steps involve many image processing techniques, there is significant potential for speedup if the techniques are performed in hardware. In this project, we want to explore whether implementing a facial recognition system on hardware will achieve a significant speedup. Our goals are to reach a detection accuracy of 90% for frontal faces, reach an identification accuracy of 80% for frontal faces and reach a 5x-10x speedup over a system completely implemented in software. We acknowledge that I/O may be a bottleneck in our system, so we will measure speedup both with and without I/O.

This report has the following structure: Section II covers what requirements our design has to meet. Section III describes our system architecture. Section IV elaborates on trade-offs that we considered before arriving at our final design. Section V elaborates on the subsystems that make up the system described in Section III. Section VI covers project management. Section VII lists related work.

II. Design Requirements

Our project needs to meet the following requirements in both accuracy and speedup.

- 90% Accuracy on Frontal Face Detection
  When taking frontal view pictures of people's faces, our system should be able to detect a bounding box that exactly encompasses the face 90% of the time. We will determine this visually.
- 80% Accuracy on Frontal Face Identification
  During testing, we will get people to take frontal view pictures in front of the camera and add their faces to a database of faces. When taking pictures of random people whose faces may or may not already be in the database, our system should be able to recognize the face and display the correct name or correctly determine the face isn't in the database at least 80% of the time.
- 5x Speedup over a Software Implementation
  We will measure the total time to recognize one face using our system in both software and hardware. In software, we will use the linux time command or the timing functions in standard C. In hardware, we will measure the number of clock cycles at our operating frequency. Our timing will start from when the camera takes a picture of the person whose face we are trying to detect. Our goal is to achieve at least 5x, ideally 10x, speedup in hardware. We acknowledge that the I/O transfer to our FPGA is a bottleneck of our system, so we will measure the speedup both with and without I/O.

III. Architecture

Our system is mainly divided into three components: a laptop (software), a FPGA (hardware), and a UART channel to transfer data between our laptop and our FPGA. We break our block diagram into two parts and explain each part in more detail here. The full block diagram can be found at the end of the report.

Laptop - The laptop is responsible for capturing camera input and preprocessing the corresponding images of people. The preprocessing includes downscaling the input image and converting the input image to grayscale.

- Downscale Module
  The size of the raw image from a laptop camera is roughly 4500 pixels by 2500 pixels. The facial detection algorithm is more optimal for smaller input image sizes, so the downscaling function converts the raw image to 160 pixel by 120 pixel.
- Grayscale Module
  We expect I/O to be one of the main bottlenecks in our system. Thus, the fewer bytes of data that we have to transfer over to our FPGA, the better. The laptop camera image is in RGBA scale, which takes 4 bytes per pixel. The grayscale function converts it to grayscale, which takes only 1 byte per pixel, reducing the amount of data to transfer by 75%.

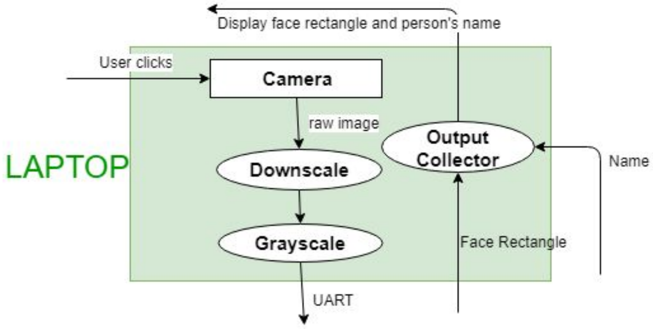


Figure 1. Laptop Block Diagram

FPGA - The FPGA we are using is the Kintex-7 XC7K325T FPGA, on the Xilinx KC705 Board. It is responsible for performing facial detection and identification on the input image. The image database will be stored in Block RAM and the input image will be stored in distributed RAM.

- Facial Detection Module
  The facial detection module will use the Viola-Jones algorithm to detect the position of the face within the input image. Since we already have a pre-trained version of the algorithm from OpenCV, we will include all the pre-trained weights as part of the module when programming the FPGA. The detection module outputs a rectangle encompassing the detected face's location and sends its coordinates to the laptop and subsequent recognition modules.
- Face Cropper and Downscaling Modules
  Once the detection module outputs the face rectangle, the face cropper module crops the face out of the image, and the downscaling module downscales the cropped face into a 20 pixel by 20 pixel image, which is the input size to our identification module.
- Identification (Eigenface) Module
  This module uses the Eigenface algorithm to perform facial identification. It compares the data for the input image with all data for the faces in the database (stored in BRAM) and outputs a face index, which is sent back to the laptop through UART.

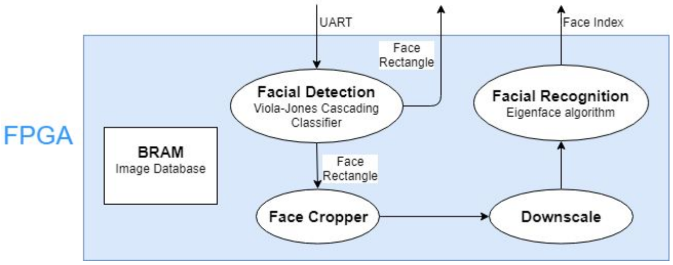


Figure 2. FPGA Block Diagram

UART - The UART channel is responsible for transferring the downscaled and grayscaled input from the laptop camera to the FPGA on system startup, as well as the face feature rectangle and the face index from the FPGA to the laptop. The transfer of the input from the laptop camera will take the longest. Assuming a baud rate of 460800 bits per second, the 160 pixel by 120 pixel converted input image, 153600 bits total, would take around 0.33 seconds to transfer.

IV. Design Trade Studies

A. Choosing an FPGA

We mainly considered the Nexys Video Artix-7 FPGA Board before we had access to the course inventory. This was the board with the most logic cells and Configurable Logic Blocks that we could find within our budget. Once we were given access to the course inventory, we realized that this board couldn't compare to the Kintex-7 KC705 board or the Virtex-7 VC707 board in the inventory, as these two boards had specs that reflected costs of 3x-6x our budget of $600. We put in a request for either board and ended up getting the KC705 board, so we ended up using that for this project.

B. Method of Data Transfer

We briefly considered PCIe for transferring data quickly between our laptop and our FPGA. However, getting PCIe to adapt to the ports on a laptop is really tough, and in the adaptation process the PCIe transfer speed get bottlenecked to the speed of the port it's trying to adapt to. Thus, we decided to go with UART for transferring data, as it is simple to understand and since we can dial up the baud rate easily.

C. Algorithm for Facial Detection

We have considered two algorithms for facial detection, the Viola-Jones algorithm we mentioned in the previous section, and a convolutional neural network. We summarize the tradeoffs between the two approaches in the following table.

Criteria	Viola-Jones	Neural Network
Familiarity	Good, a lot of documentation in OpenCV and related work on FPGA.	Not ideal because it is hard to find existing neural networks for facial detection on FPGA.
Computation Intensity	Not ideal, because of simple operations at each stage, but can pipeline stages.	Ideal on FPGA, because of heavy computations. More likely to see the speedup on FPGA.
Training	No training, because we can use pre-trained weights.	Long training, need to train it ourselves or use weights from a pre-trained network.
Difficulty	Conceptually easy to understand, but hard to optimize	Complicated network, but each layer does similar things.

Table 1. Tradeoff between Viola-Jones and Neural Network

After considering these criteria and the timeline of our project, we decided to use Viola-Jones because it has good documentation to refer to, and because we don't have to worry about training.

D. Software Storage Method for Face Database

We considered two methods of storing the face database on our laptop: storing it locally and storing on a cloud storage, such as Amazon S3. For storing in the cloud, we would need to use additional protocols to access the images and vectors in our database. Additional latency would be introduced if we were to try and store our data in the cloud. The benefit would be that we would essentially have no limit in terms of how much storage we have to work with. However, once we computed how much storage we actually needed, we found that we only need a weight vector of 59 elements and a 20 pixel by 20 pixel picture for each face that we want to store, so there would be no storage issues with storing the face database locally. Thus, we decided to store face database locally on our laptop.

E. Hardware Storage Method for Face Database

We considered two methods of storing the face database on our FPGA: block RAM and using an sd card. We have limited block RAM on our FPGA, so if our face database gets too large, it would cause space issues. An sd card would resolve this problem, since it has so much storage. On the other hand, an sd card would be off chip, requiring us to use communication protocols to access the data on it. This plus lengthy access times would increase the amount of time face identification takes, changing it into a bottleneck. Block RAM is on chip, so if we use it we wouldn't have to worry about complicated communication protocols or lengthy access times. In the end, we decided to go with block RAM and just limited the size of the face database to the data for 100 faces, 100 being a large enough number to encompass all the faces that we might possibly add.

F. Using Vivado HLS or Pure HDL

We considered using Vivado HLS to program our FPGA, since it easily converts C code to Verilog HDL and VHDL. However, we felt that using HLS would make the project too software oriented, as it would obfuscate any hardware optimizations that were performed under the hood. We felt that our HDL skills were good enough to convert C code to HDL ourselves and then optimize it, so we decided to not use HLS.

G. Storing Viola-Jones Classifier Weights

We considered storing the Viola-Jones classifier weights in distributed RAM vs storing the weights in block RAM. Storing them in distributed RAM would mean that we lose a lot of distributed RAM just to store a bunch of floats, but it would save us memory latency, which is important since each stage in the Viola-Jones classifier is computationally heavy. Storing the weights in block RAM would give us more distributed RAM to implement all our modules with, but we would have to deal with memory latency and most likely have to deal with waiting multiple clock cycles to read all the weights out of each block RAM (we can't store each weight in its own block RAM since we have finite block RAMs available). We chose to use distributed RAM to store the classifier weights for convenience.

V. System Description

A. Facial Detection Module

Once the downscaled and grayscaled 160 pixel by 120 pixel image arrives at the FPGA via UART, it will be stored in distributed RAM. A 24 pixel by 24 pixel section (called a scanning window) of the stored image will be inputted into a 4608 bit (576 pixels times 8 bits per pixel) input of the cascading classifier portion of this module once every clock cycle. This scanning window input is denoted as a "face candidate" since it will go through each stage of the cascading classifier to check if it has the necessary features for a face. The specific bits being inputted will be determined by indexes being incremented by counters every clock cycle. The cascading classifier portion of this module will be pipelined such that the inputted face candidate goes through one stage in the classifier every clock cycle. Each stage only needs to transfer on the face candidate and whether or not it found the feature(s) it was searching for onto the next stage, allowing for a very pipeline-able design.

To ensure high accuracy, we decided to use a multiscale cascading classifier. Since a face cannot always be captured by our scanning window, we need to scale the input image down continuously until we detect a face. We chose to use a scaling factor of 1.2 based on past projects that we looked at. Once all the scanning windows have been inputted into the cascading classifier, we will have an finite state machine that waits for the classifier to finish and checks if any faces were detected. If no faces were detected, it will reset the index counters, as well as replace the image in distributed RAM with a downscaled version of itself calculated previously in parallel when the scanning windows were being inputted. This will repeat until a face is detected or until the dimensions of the image in distributed RAM are smaller than the dimensions of the scanning window.

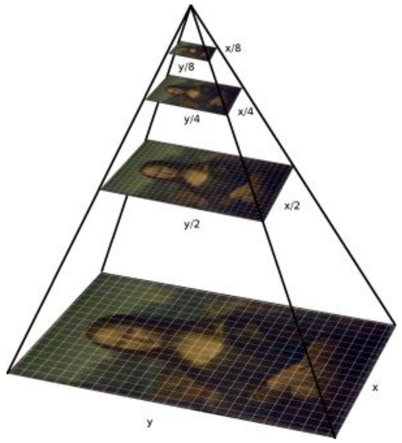


Figure 3. Pyramid Scaling for Cascading Classifier

B. Face Database

The face database needs to store an average face from our training images of size 20 pixel by 20 pixel, as well as 59 eigenvectors of size 20 pixel by 20 pixel. For each face, it also needs to store a weight vector of 59 16-bit elements. These are all for performing the Eigenface algorithm[3] in our facial identification module. These will all go into block RAM. If we store the average image in its own 18 Kb block RAM, store each of the 59 eigenvectors in their own 18 Kb block RAMs (why we do this is covered in Section *V.C.*) and store the weight vectors for 100 faces in six 36 Kb block RAMs, this comes out to sixty 18 Kb block RAMs and six 36 Kb block RAMs. The XC7K325T FPGA has 890 18 Kb block RAMs (each pair can be converted to a 36 Kb block RAM) so we have more than enough block RAM.

C. Facial Identification Module

Once a 20 pixel by 20 pixel face arrives from the downscaling module, the facial identification module will retrieve the average face from block RAM and perform normalization in parallel on each of the 400 pixels. It will then compute each of the 59 weight vector elements for the image in parallel, by retrieving each of the 59 eigenvectors from block RAM in parallel and multiplying those by the normalized image. The final weight vector of 59 elements will then be compared to the existing weight vector for each image stored in block RAM, using one clock cycle for each comparison, meaning the number of clock cycles to find the closest weight vector is equal to the number of existing faces in our database. The difference between the closest weight vector and the current weight vector will then be compared to a threshold, to determine if the face is actually within the database or not.

VI. Project Management

A. Schedule

Below is a rough deadline for all our tasks. We have attached the full gantt chart at the end of this document.

Task	Deadline
Viola-Jones cascading classifier in C UART setup	3/3
Viola-Jones testing	3/7
Viola-Jones on FPGA	3/24
Test face detection only on FPGA	3/31
Eigenface recognition in C	4/7
Eigenface testing	4/14
Eigenface recognition on FPGA	4/21
Integration test	4/28

Table 2. Rough Deadlines

B. Team Member Responsibilities

Junye - Implement Viola-Jones cascading classifier, implement eigenface algorithm in C, assist in converting C code to RTL code.

Sheng-Hao - Re-acquaint with Vivado and show Hans how to use it, work with Hans to convert algorithms to RTL and optimize them.

Andy - Design app interface, preprocess image (grayscale, downscale), create a local database to keep track of face data.

C. Budget

We have included a spreadsheet of our budget at the end of this report.

D. Risk Management

I/O - We identify that I/O is a major bottleneck for the speedup of our system. Assume our input is a 160 by 120 image. If we set baud rate to 460800 bits/second, it takes about 160x120x8 / 460800 = 0.33s to transfer the data from software and FPGA. The FPGA we have right now can support as much as 921600 bits/second, but we use 460800 bits/second here just to be safe. We will also make sure that we account for the speedup with and without I/O.

VII. Related Work

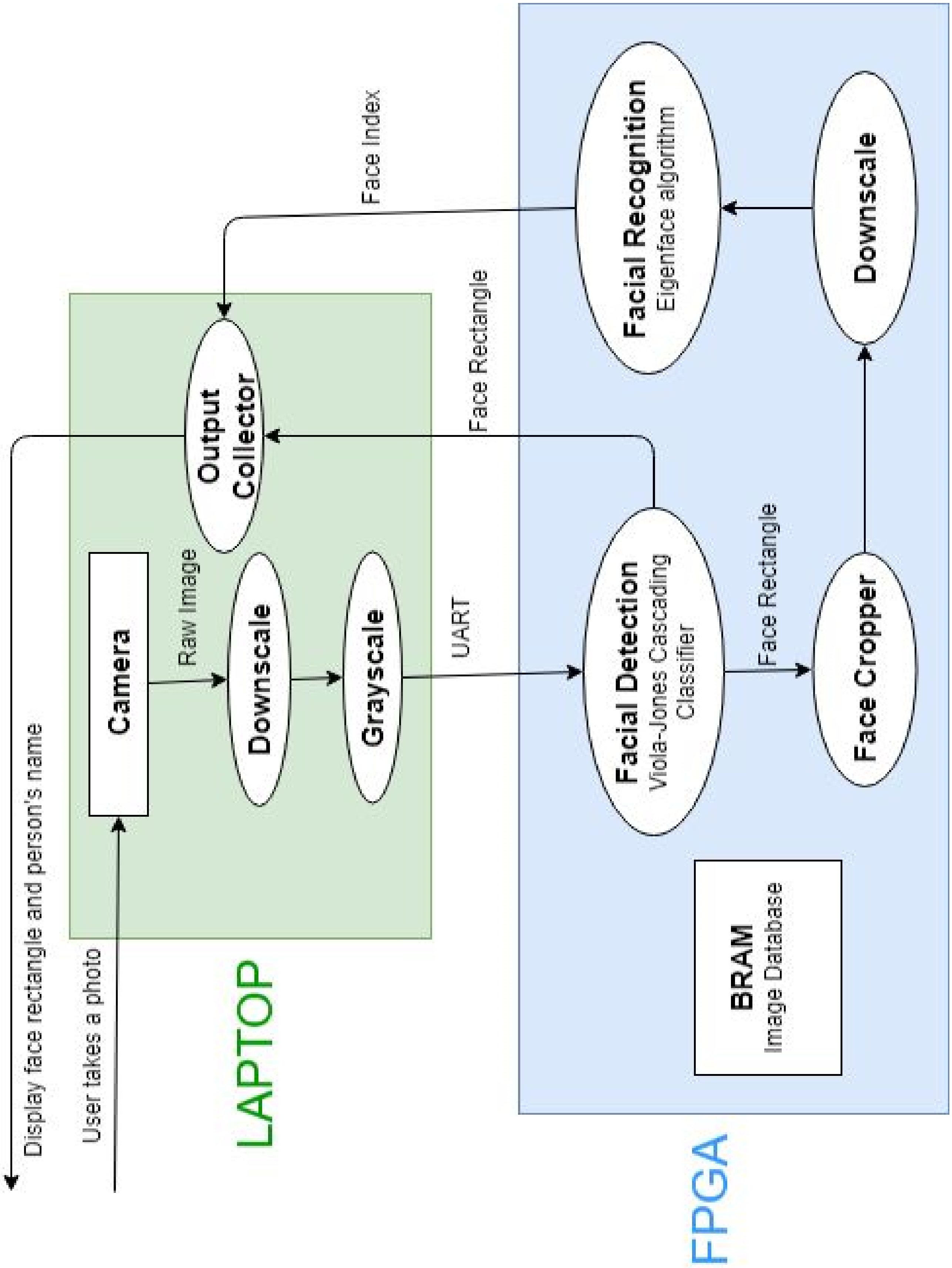
[1] J. Cho, S. Mirzaei, J. Oberg, and R. Kastner, “Fpga-based face detection system using haar classifiers,” in International Symposium on Field Programmable Gate Arrays, February 2009.

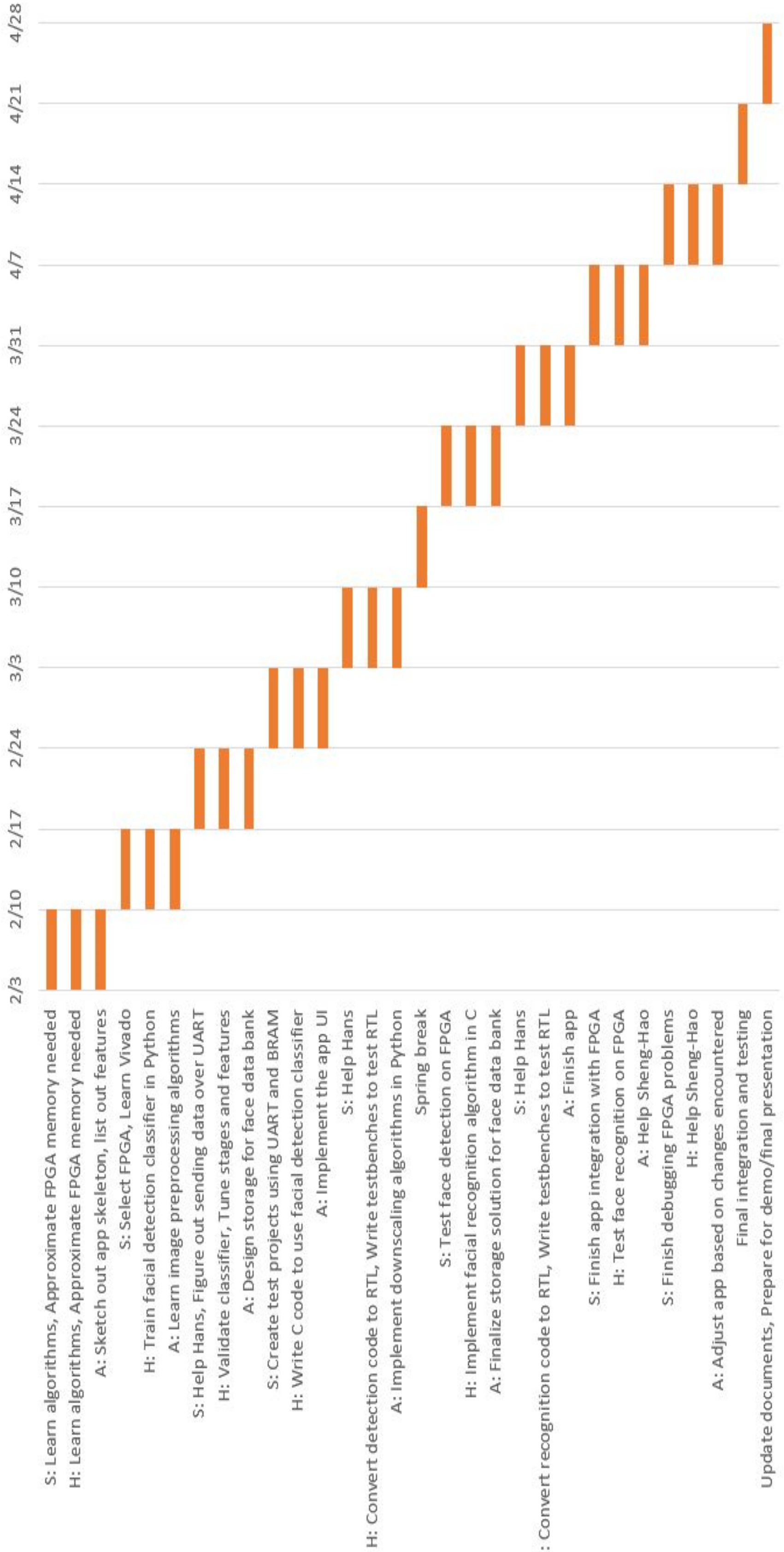
[2] J. Cho, B. Benson, and R. Kastner, “Hardware acceleration of multi-view face detection,” in IEEE Symposium on Application Specific Processors, July 2009.

[3] J. Matai, A. Irturk, R. Kastner, “Design and Implementation of an FPGA-based Real-Time Face Recognition System,” in IEEE

VIII. Summary

We hope our system will be able to meet the design specifications. For the final report, we will talk about future work and lessons learned here.


LAPTOP
FPGA
Camera
Raw Image
Downscale
Grayscale
UART
Output Collector
Display face rectangle and person's name
User takes a photo
Facial Detection
Viola-Jones Cascading Classifier
Face Rectangle
Face Cropper
Downscale
Facial Recognition
Eigenface algorithm
Face Index
Face Rectangle
BRAM
Image Database


2/3
2/10
2/17
2/24
3/3
3/10
3/17
3/24
3/31
4/7
4/14
4/21
4/28
S: Learn algorithms, Approximate FPGA memory needed
H: Learn algorithms, Approximate FPGA memory needed
A: Sketch out app skeleton, list out features
S: Select FPGA, Learn Vivado
H: Train facial detection classifier in Python
A: Learn image preprocessing algorithms
S: Help Hans, Figure out sending data over UART
H: Validate classifier, Tune stages and features
A: Design storage for face data bank
S: Create test projects using UART and BRAM
H: Write C code to use facial detection classifier
A: Implement the app UI
S: Help Hans
H: Convert detection code to RTL, Write testbenches to test RTL
A: Implement downscaling algorithms in Python
Spring break
S: Test face detection on FPGA
H: Implement facial recognition algorithm in C
A: Finalize storage solution for face data bank
S: Help Hans
H: Convert recognition code to RTL, Write testbenches to test RTL
A: Finish app
S: Finish app integration with FPGA
H: Test face recognition on FPGA
A: Help Sheng-Hao
S: Finish debugging FPGA problems
H: Help Sheng-Hao
A: Adjust app based on changes encountered
Final integration and testing
Update documents, Prepare for demo/final presentation

Part	Cost
Laptop (from us)	$1000 ($0)
KC705 Board Kit (from course inventory)	$1685 ($0)
Vivado License (from CMU) - UART/BRAM modules	$3595 ($0)
Pre-trained Viola-Jones weights	$0
Total	**$0**