Identity Checker with FPGA

Junye (Hans) Chen (junyec)
Sheng-Hao Huang (shenghah)
Andy Shen (asshen)
Team C7

Use Case

- Facial detection + facial recognition on FPGA
- Steps:
  - Detect 1 face on camera
  - Add face to database OR
  - Recognize face
- Areas:
  - Software systems
  - Hardware Systems
- Image processing algorithms have significant parallelism, and thus large potential for speedup

Requirements

- Enough FPGA block RAM to store grayscale face database
  - 50 people * 5 faces/person * 20x20 pixels/face * 1 byte/pixel = ~ 0.1 MB of RAM
- Enough LUTs, flip-flops, and DSP slices on FPGA
  - 44800 LUTs, 89600 FFs, and 128 DSPs used by reference paper
- Correctly detect a still face on screen within 5 seconds
- Recognize a face correctly at least 80% of the time
- Speedup of 5x ~ 10x
  - Measure with I/O and without I/O

Key Technical Challenges

- Having the right FPGA
  - Reference paper uses Virtex-5 XC5VFX70T
  - Artix-7 100T or 200T have similar/better specs
  - $300 and $500 respectively
- Minimizing UART time
  - Need high baud rate
  - Also need high operating frequency for FPGA
- Minimizing time to scan laptop camera image during facial detection
  - May limit the laptop camera size
  - May also scale down the image before using it
  - Convert image to grayscale (1 byte/pixel vs 4 bytes/pixel for color)

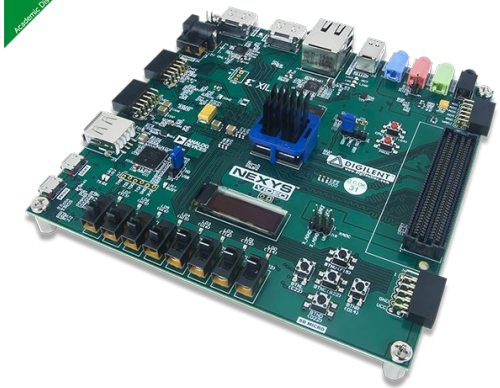
Key Technical Challenges

- Training multi-stage classifier for Viola-Jones algorithm (facial detection algorithm)
  - Find optimal number of stages and types of facial feature classifiers
  - Need high training and validation accuracy for detecting faces
- Increasing parallelism for eigenface algorithm (facial recognition algorithm)
  - Need to compare face data against all existing face data
  - The more simultaneous comparisons the better
  - The more parallel matrix multiplication the better
- Learning Vivado
  - Be able to use module wizards for UART and block RAM
  - Also may need to learn how to use Vivado HLS to synthesize C code onto FPGA

Key Technical Challenges

- Keeping track of existing face data in cloud/local storage
  - Need to send this over on startup of device
- Finding a suitable method for data transfer from the app to the FPGA
  - Current idea: Transfer data through UART
- Trial and error with image preprocessing
  - What type of preprocessing (B&W, scaledown, etc)
  - Keep image file size low

Components



FPGA + Board

Laptop with Camera

UART Cable

Design

- Device boots up
- App sends existing face data bank to FPGA via UART transaction
- FPGA stores data in block RAMs
- Scenario A: person tries to add face to data bank
  - Person stands in front of laptop camera and takes a picture
  - App preprocesses the image (scale down, apply grayscale) and sends it to FPGA
  - FPGA sends face picture through facial detection logic
  - FPGA sends back signal that indicates whether face was detected

Design

- Scenario B: person tries to check their identity
  - Person stands in front of laptop camera
  - App samples image from laptop camera every 5 seconds
  - App preprocesses the image (scale down, apply grayscale) and sends it to FPGA
  - FPGA sends face picture through facial detection logic
  - If a face is detected, FPGA sends face data through recognition logic
  - Recognition finds face that matches the most and gives similarity measurement
  - FPGA sends back signal that indicates whether similarity measurement was high enough

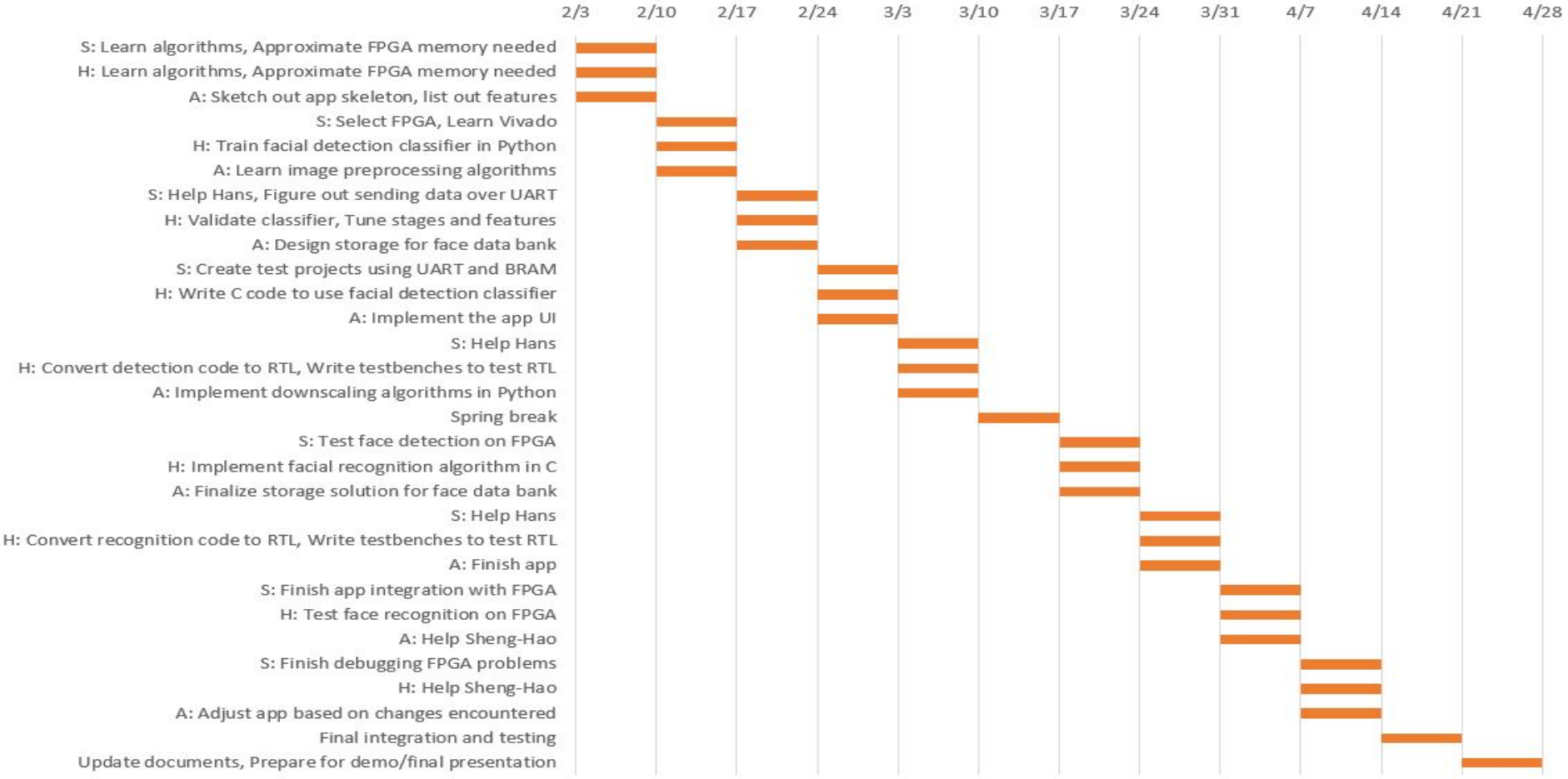
Testing

- Vivado’s reported FPGA resource usage must be less than 100%
- We want the accuracy of the facial recognition system to be at least 80%
- Be able to add a face during demo, then recognize the person
- Record the time to run the complete pipeline of the system, and compare the running time on FPGA vs on software
    - Measure the system runtime on FPGA, and specifically measure I/O transfer time
    - Measure the system runtime on selected CPU model
    - Measure the speedup between the two models, and check if it scales as our face bank gets bigger

Tasks/Division of Labor

- Sheng-Hao - Hardware / FPGA
  - Re-acquaint with Vivado and shows Hans how to use it
  - Learn algorithms with/from Hans
  - Work with Hans to code up algorithms in RTL and optimize them
- Hans - ML / Algorithms
  - Learn Viola-Jones algorithm
  - Learn Eigenface algorithm
  - Work with Sheng-Hao to code up algorithms in RTL and optimize them
- Andy - Software / App
  - Design the app interface
  - Figure out when to preprocess image (B&W, scale down, etc.)
  - Create a local or cloud database to keep track of face data

Schedule


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