ProtoFacer: FPGA-Driven Interactive Protogen Head

1st Benson Zhan Li Lin
Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology
Cambridge, MA, USA
bensolzl@mit.edu

***Abstract*—We present a design for a Protogen Head controlled by FPGA fabric. Using an inward-facing OV5640 camera inside the helmet, the Protogen Head reconstructs the cosplayer's face based on real-time video input. The FPGA processes 25 frames-per-second video input from the camera, (limited primarily by electromagnetic interference concerns).**

Experimental testing shows that the ProtoFacer is able to capture blinks and mimic mouth movement during speech, all while using under 400mW of power (excluding external power to the LED boards) and 1.5 MiB of block RAM. This makes it ideal for use in limited memory/power environments.

***Index Terms*—Digital systems, Field programmable gate arrays, Image processing,**

I. Introduction

Protogen heads are a type of mascot head which display a face using multiple LED boards behind a tinted visor. These allow for a greater range of expressions for the cosplayer to choose from since these displays can flexibly display many different facial expressions without mechanical components. Ideally, the displayed face should be representative of the cosplayer's face, allowing for common facial movements such as blinking and smiling to be expressed. However, most models construct faces from a fixed set of pre-determined expressions stored on the controller's memory.

The goal of the ProtoFacer project is to overcome this limitation by leveraging the parallelism of the FPGA fabric to implement a face reconstructor from real-time video input.

This presents a number of design challenges:

- The system needs to fit within the limited space of a protogen head (which is no larger than a 6-inch box) and leave enough space for the camera to see the cosplayer's face.
- The system should use a small amount of power; it would be difficult for cosplayers to carry around large power banks for their equipment.
- The system needs to process the video feed quickly enough to capture facial expressions, such as blinks and mouth movement.

This report will focus on the design and implementation of the ProtoFacer system. An overview is shown in Figure 4. The system is comprised of the following modules (elaborations in the respective sections):

- Head Frame (Section II-A)
- OV5640 Camera (Section III-A)
- Image Processing Pipeline (Section III-B)
- Expression Recognizer (Section IV-A)
- Face Image Buffers (Section V-A)
- Face Reconstructor (Section V-B)
- HUB75 Driver (Section VI-B)
- Power Modulator (Section VI-C)
- HUB75 LED Boards

We evaluate the performance of the ProtoFacer based on its resource usage in Section VII. We then end the report with a discussion of the system's limitations and future work in Section VIII.

II. Physical Construction

A. Head Design

Fig. 1. Exterior of the Protogen Head (with the tinted visor removed)

The Protogen head used is comprised of the following components:

- A pair of Waveshare Electronics RGB-Matrix-P2.5-64x32 LED displays [2], each containing 32 rows of 64 LEDs each. These are used to display the face of the protogen.
- An OV5640 camera, used to capture real-time video of the cosplayer's face for processing. This is mounted on the front tip of the protogen head.
- A 5V white LED strip mounted just below the camera for lighting the interior of the helmet.

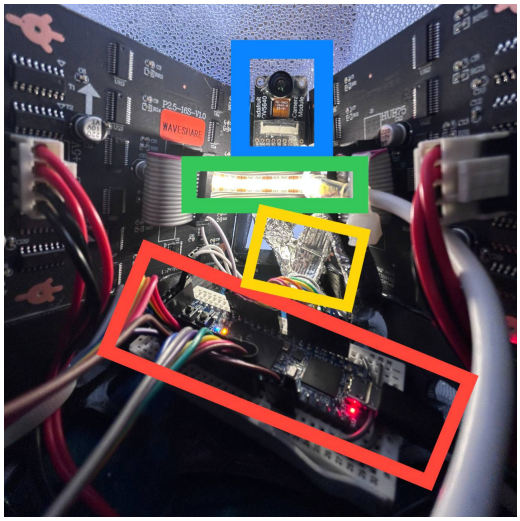

Fig. 2. Internals of the Protogen Head. Blue Box: OV5640 Camera. Green Box: LED Strip. Yellow Box: Shielded Camera Wire. Red Box: Cmod A7-35T board. Left and Right: LED Boards

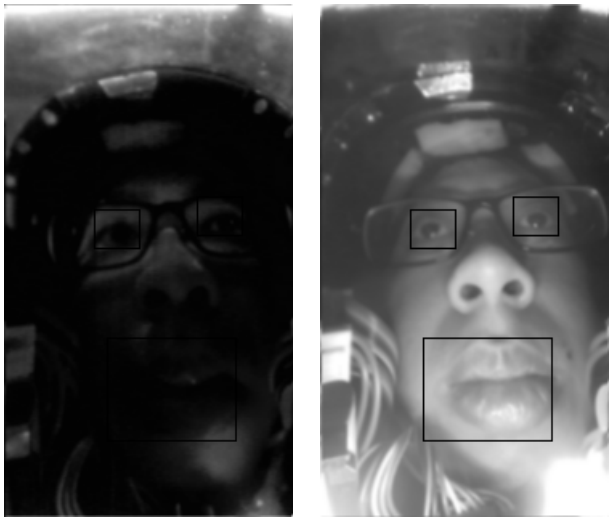

Fig. 3. Interior of the helmet with under various lighting conditions (Black bounding boxes are for calibration) Left: LED strip off. Right: LED strip on.

- A black-tinted visor made from clear Polyethylene Terephthalate Glycol (PETG) that was thermoformed into the right shape, then dyed with a synthetic black colorant.
- A Digilent Cmod A7-35T Breadboardable Artix-7 FPGA Module as the FPGA board [4]. This is mounted on the interior of the helmet just under the camera.
- A protogen head frame adapted from the Sigma Protogen model by Coelacant [1]. The frame is printed in black PETG using an Ultimaker S5 3D printer.
- Faux fur from BigZ Fabric

The LED strip inside the helmet is necessary for lighting the interior of the helmet. It ensures consistent lighting conditions for the camera. Testing showed that without the LED strip, the interior of the helmet often becomes too dark to make out any of the facial features at all (see Figure 3). This is usually exacerbated by the tinted visor blocking out most of the light from the exterior of the helmet.

B. FPGA Board Selection

This Cmod A7-35T was chosen as it has 52 digital input/output (I/O) pins for both the LED boards and the cameras [4]. The HUB75 LED boards and the OV5640 camera require 14 data wires each. To fit both the LED boards and the OV5640 camera as well as additional controller peripherals such as buttons and switches, we would need at least 30 I/O pins.

The small frame of the Cmod A7-35T also makes it viable for use inside the helmet itself, where space is severely limited. This allows most of the helmet to be self-contained, save for the power supply to the LED boards, LED strip and the FPGA itself.

Notably, the Cmod A7-35T only has a single micro-USB port. It has no external power supply port and no peripheral ports. Thus, the only way to power the FPGA is to use the same micro-USB used to load the FPGA bitstream. This also creates a significant debugging challenge: streaming video from the FPGA is impossible since all data needs to be sent over UART from the FPGA to the Micro-USB port.

To circumvent this, most of the code was tested on a Real Digital Urbana Board based on AMD's Spartan 7 [5]. The HDMI port of the Urbana was used to stream video from the camera to an external monitor for debugging purposes. That board was too big to fit inside the helmet and did not have enough I/O pins for both the camera and the LED boards.

The entire data flow was split into two stages. The first stage was the OV5640 camera to Face Constructor pipeline; the second was the Face Constructor to LED board pipeline. See Figure 4 for details. Each stage was tested separately on the Urbana board.

After both stages were completed and tested, the stages were integrated on the A7-35T. Notably, the A7-35T runs on a 12MHz internal clock [4] as compared to the 100MHz clock of the Urbana. A clocking wizard IP was used to generate a 100MHz clock on the A7-35T.

III. Camera Input

A. OV5640 Camera Settings

The OV5640 camera is initialized with the following settings:

- Resolution: 320 pixels high by 180 pixels wide
- Frame rate: 25 frames per second
- Format: 8-bit greyscale

The main reason for the low resolution, frame rate and format is to lower the clock speed of the camera by as much as possible. Since the camera sends 1 pixel (8 bits) per clock cycle, the clock speed of this camera is at least the product of the resolution and the frame rate.

The wire that transmits the camera feed to the FPGA is subject to significant electromagnetic interference from the surrounding electronics and the environment. This is due to the high frequency of the camera feed. To reduce electromagnetic

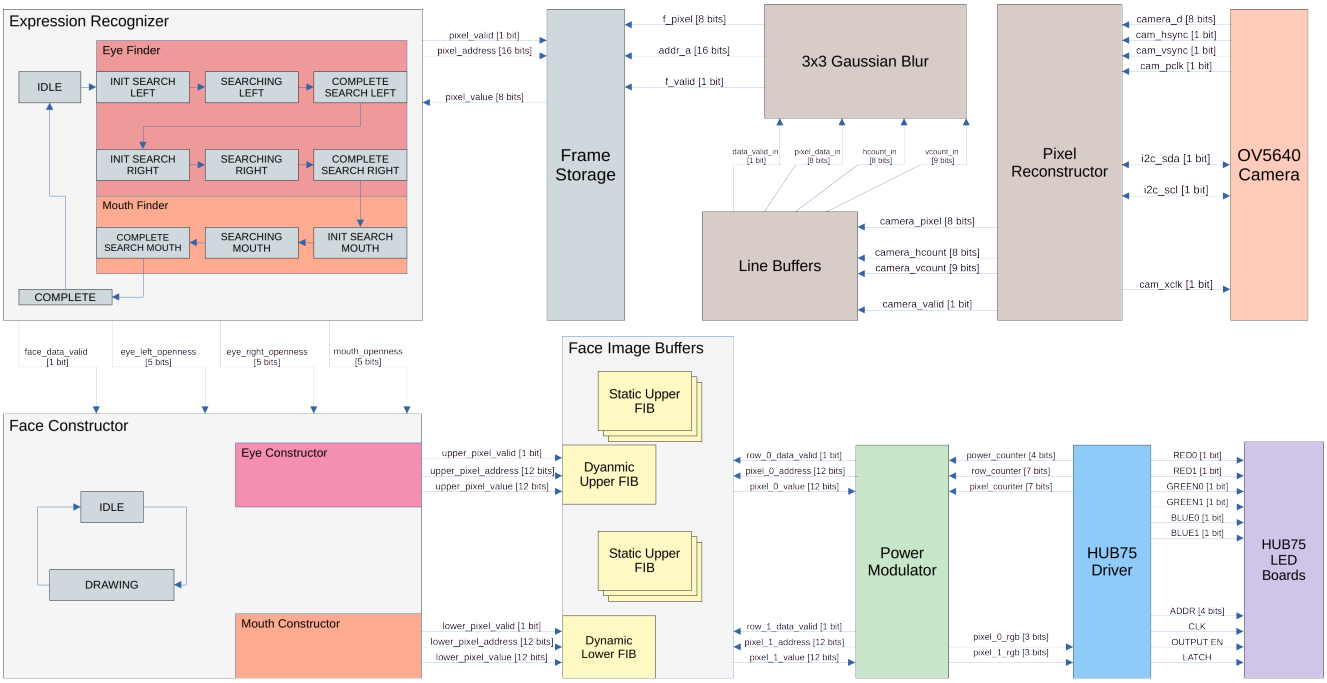

Fig. 4. Block Diagram of the ProtoFacer System

interference with the camera signals, the wire between the camera and the FPGA board is shielded using foil tape. Even with such precautions, it is difficult for the camera to send a stable signal over the wire. Thus, the clock speed is set as low as reasonably possible.

Initial testing with the camera feed showed that a greyscale image with 320x180 resolution was sufficient to capture important facial features. Reducing the frame rate would make it difficult to capture rapid facial movements such as blinks, while reducing the resolution would blur the image too much to distinguish facial features.

B. Image Storage

A pixel reconstructor samples the data from the camera on the rising edge of the camera's clock signal over the camera clock wire. This produces an 8-bit greyscale image.

To cross the clock domain boundary from the camera to the FPGA and to remove aliasing artifacts, the frame is first passed through a 3x3 Gaussian blur filter implemented with 4 horizontal line buffers. The raw frame is then stored in a $8 \times 180 \times 320 = 460800$-bit BRAM.

IV. Image Processing

Since the camera position is fixed relative to the protogen frame and the cosplayer's head is also generally fixed in orientation within the frame, the locations of the eyes and mouth are generally fixed in the visual field of the camera.

Thus, we only need to search a relatively small area of the frame to obtain the eye and mouth data. At the moment, these areas are hardcoded into the FPGA fabric at compilation time.

A possible area of exploration is to do another pass over the image to find the possible eye locations instead of using the hardcoded values. However, the fixed arrangement of the camera and cosplayer head within the helmet makes this unlikely to be a significant improvement in this context.

A. Expression Recognizer

The Expression Recognizer comprises a Finite State Machine (FSM) with the following states

- **Idle**: The Expression Recognizer is waiting for a new frame to be processed.
- **Eye Scanning**: We scan the left and right eye locations. We can then compute an estimate of how open the eyes are, then move to **Mouth Scanning**.
- **Mouth Scanning**: We scan the mouth location and compute an estimate of how open it is. We then move to **Complete**.
- **Complete**: The Expression Recognizer indicates that it has finished computing. This stays on for one cycle and the Face Reconstructor is expected to pull the data from the Expression Recognizer on this cycle. After one cycle, the Expression Recognizer moves back into the **Idle** state.

Internally, the Expression Recognizer is a wrapper around the Eye Size Detector and the Mouth Size Detector that routes the correct inputs and outputs to the BRAM storing the image

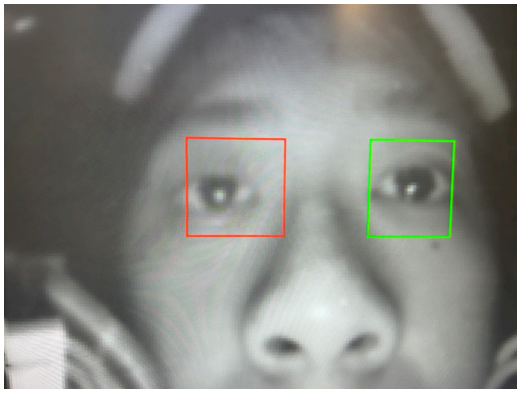

Fig. 5. Partial image of the camera output with eye bounding-boxes overlayed. Notice the bright spot in the middle of the eye as a result of the eye reflecting light from the LED strip within the helmet.

frame. Since it only routes data back and forth between the BRAM and the Size Detectors, each of the Size Detectors are responsible for handling the 2-cycle delay on the BRAM through pipelining.

This FSM produces the following output to the Face Reconstructor:

- `face_data_valid` [1 bit]: This is set to 1 if the Expression Recognizer is in the Complete state. This tells the Face Reconstructor that it should pull data from the Expression Recognizer.
- `eye_left_openness` [16 bits]: A measure of how open the left eye is.
- `eye_right_openness` [16 bits]: A measure of how open the right eye is.
- `mouth_openness` [16 bits]: A measure of how open the mouth is.

B. Eye Size Detection

To compute the size of the eye, we will be using the pupil as a proxy for the eye size. A colored pupil is generally darker than the rest of the face. Additionally, a closed eye forms a small line of dark pixels where the upper and lower eyelids meet.

The eye finder is given a subrectangle of the frame that contains the eye. The eye finder then performs two scans over the eye location. These scans are done column by column.

The first scan determines the minimum brightness values in the subrectangle. This gives us the darkest pixel in the search region.

The second scan determines how many pixels have a brightness value close to the darkest pixel. To reduce the effect of noise, we take a maximum of 3 consecutive pixels in the column scan. For pixels represented in (row,column) with $(0, 0)$ in the top-left corner, we take the maximum brightness of the pixels at $(r, c), (r + 1, c), (r + 2, c)$. If the maximum brightness of all 3 pixels is within 16 of the minimum brightness, we can conclude that the pixel should be in a dark region.

Fig. 6. Openness values when the eyes are open (top) and closed (bottom). The seven-segment display on the left of each image shows the computed openness values for the left and right eye.

This sliding window technique has the added benefit of emphasizing larger regions of dark pixels as compared to scattered regions of dark pixels. The pupil is generally a contiguous dark region of the eye, so this technique is effective in distinguishing the large dark pupil from the thin, dark, closed eyelid.

If we see a large number of pixels that are close in brightness to the darkest pixel, we can infer that the eye is probably open. Otherwise, we infer that the eye is probably closed. The exact value of the threshold depends on the size of the eye and the resolution of the camera and needs to be determined with further testing or with machine learning algorithms.

For eye size detection, only the pupil size is currently being used. There might be a way to determine gaze direction by using data about the sclera and the reflection of the LED strip within the helmet by the eye. The LED strip is always at the front of the eye and thus reflects from a consistent point on the frame. We may be able to determine gaze direction by comparing the centre of mass of the dark pixels with the bright reflection of the LED strip. However, this might be difficult at the current resolution as that bright spot is only 4-5 pixels wide, making it difficult to separate from noise.

C. Mouth Size Detection

We found that we could reuse the sliding window technique from the eye finder in the mouth finder. When the mouth is opened, the oral cavity is shielded from outside light and thus appears relatively dark compared to the rest of the face. This is true even in the presence of the LED strip within the helmet. Since the camera is offset from the LED strip, the upper mouth casts a shadow on the interior of the mouth which can be detected by the camera.

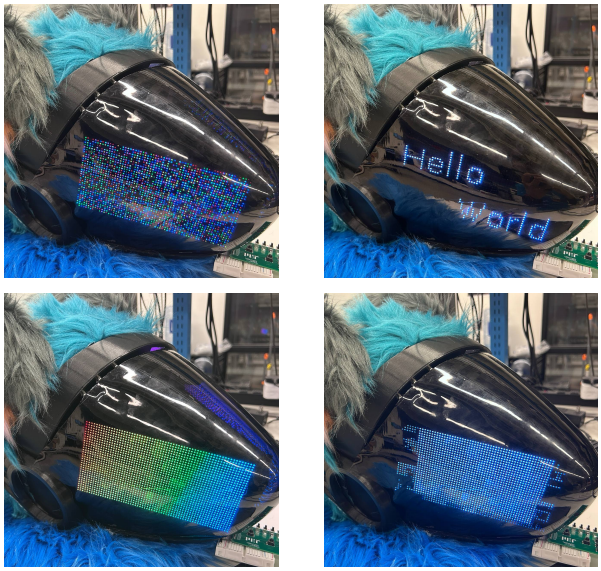

Fig. 7. Examples of static images displayed on the LED Boards. Clockwise from top left: static noise, Hello World, a low-resolution screenshot of a 6.2050 lecture, rainbow gradient

It is possible to use a similar technique to Nguyen et al. in [3] where a path is traced between the left and right endpoints of the lips to determine the size of the mouth. We decided not to use that method as it was more complicated and unnecessary when the position of the mouth is relatively fixed.

V. Face Construction

Once we have information about the mouth and eyes, we can construct the face needed. The LED boards take RGB inputs on two rows at a time; providing an address of row i to the LED boards will let us write into both row i and row $i+16$. The downstream LED Board Controller needs to read pixel data from a pair of rows at a time Thus, we store the data in two BRAM blocks to separate the upper 16 rows from the lower 16 rows. More details about the protocol can be found in section VI-A.

A. Face Image Buffer (FIB)

The Face Image Buffer (FIB) is a pair of 24576-bit blocks of memory that stores the brightness data (4 bits red, 4 bits green, 4 bits blue) of each pixel (16 rows x 128 pixels per row). The FIB is written by the Face Constructor and read by the LED Board Controller.

We can load static FIBs for faces that are generated externally. A Python script is used to convert 32×128 bitmap (.bmp) images into memory files (.mem) that the FIBs initialize at compile time.

One FIB is reserved for dynamic face generation by the Face Reconstructor. The others can be used for hand-drawn / externally generated faces, at the cost of additional memory use (Figure 7).

B. Face Constructor

The Face Constructor is an FSM with two states:

- **Idle**: We are not currently drawing a face and are waiting for input from the Expression Recognizer. Once the Expression Recognizer provides us with the face_data_valid signal, we can move to the **Drawing** state.
- **Drawing**: We iterate over all 32 rows of 128 pixels and use the openness values obtained from the Expression Recognizer to determine the color of each pixel. These values are then written into the FIB.

We note that the Face Constructor *should* complete more quickly than the Expression Recognizer. Still, we should store data generated by the Face Constructor even if we are in the drawing state.

Having the Face Constructor as a separate module completely parametrizes the face. This means that we can easily test the Face Constructor by fixing the inputs and observing the generated faces.

Within the Face Constructor are the Eye Constructor and Mouth Constructor. These run in parallel: the Eye Constructor draws in the pixels within the upper 16 rows of the LED boards, while the Mouth Constructor draws in the pixels within the lower 16 rows of the LED boards.

C. Eye Construction

The eye constructor takes in the eye openness values from the Expression Recognizer and uses this information to construct the eye. Let o_e be the eye openness value, while l and r are the left and right endpoints of the eye to be displayed.

This construction is done by bounding all pixels (x, y) of the eyes between two parabolas, where x is the column and y is the row. Note that all coordinates are bounded by $0 \leq x < 64$ and $0 \leq y < 16$ on a single board, and that the top left corner is $(0, 0)$. The upper parabola has the equation

$$y \leq 15 - \frac{o_e(x-l)(r-x) + 8(r-l)}{4(r-l)}, x \in [l, r]$$

The lower parabola has the equation

$$y \geq 15 - \frac{o_e(x-l-1)(r-x-1)}{18(r-l)}, x \in [l+1, r-1]$$

Computing a division is impractical on FPGA hardware, so we instead use the following equations for the upper and lower parabolas respectively:

$$4(r-l)(15-y) \leq o_e(x-l)(r-x) + 8(r-l)$$

$$18(r-l)(15-y) \geq o_e(x-l-1)(r-x-1)$$

Due to the large number of arithmetics operations needed, this computation is pipelined over 4 clock cycles in order to meet timing constraints.

As o_e increases, both the upper and lower parabolas rise. Since the upper parabola rises faster than the lower parabola, the eye appears to grow as o_e increases.

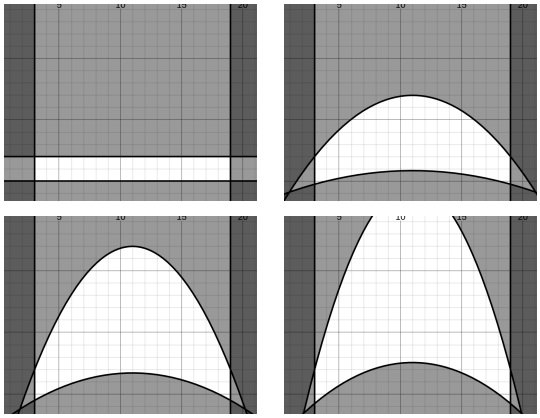

Fig. 8. (Top left, Top right, Bottom left, Bottom right) Openness of the eye at openness values of 0, 5, 10, and 15. White areas represent the part of the eye that is displayed.

Fig. 9. Display of the eyes at openness values of 0, 8, and 16.

D. Mouth Construction

Instead of using parabolas, the mouth constructor computes the upper and lower boundaries of the mouth using four pairs of line segments. The distance between these line segments can be increased based on the openness of the mouth. Figure 10 shows how the area between the line segments creates the face.

Let o_m be the mouth openness value. The upper boundary equations are:

$$x \leq 4 + 2y \ \{10 \leq x < 20\}$$

$$4y + x \geq 52 \ \{20 \leq x < 32\}$$

$$x \leq 3y + 17 \ \{32 \leq x < 44\}$$

$$4y + x \geq 84 \ \{44 \leq x < 64\}$$

The lower boundary equations are:

$$20y + 10o_m + 40 \leq 10x + o_m x \ \{10 \leq x < 20\}$$

$$8y + 2x \leq 104 + 4o_m \ \{20 \leq x < 32\}$$

$$200x + 4o_m x + 172o_m \leq 600y + 3400 \ \{32 \leq x < 44\}$$

$$50(4y + x) \leq 4000 + o_m(72 + x) \ \{44 \leq x < 64\}$$

The lower boundary equations are significantly more complicated than the upper boundary equations. This is because the upper boundary remains constant regardless of the openness value, while the lower boundary moves down with increasing openness. The constants are carefully computed such that the computed line segments intersect precisely at $x = 20, 32, 44, 64$.

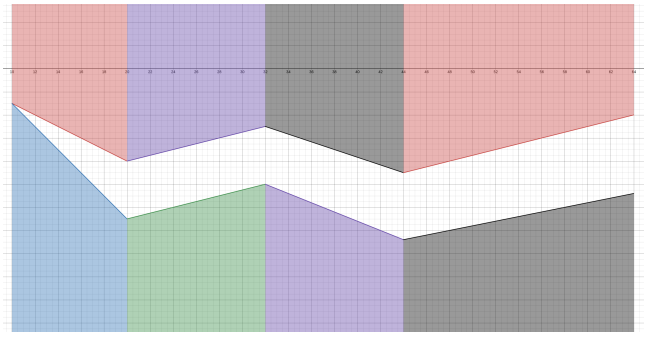

Fig. 10. Plotted graph showing the mouth area at an openness value of 10.

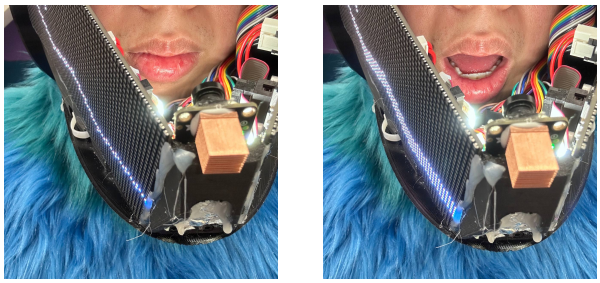

Fig. 11. Change in the displayed mouth based on the mouth of the cosplayer. Left: Closed Mouth. Right: Open Mouth.

These calculations are also pipelined over 4 clock cycles to meet timing constraints. This also synchronizes the mouth constructor with the eye constructor.

VI. Display Controller

To control the LED boards from FPGA fabric, we wrote a module to communicate the correct pixel values to the LED boards.

A. HUB75 Protocol

HUB75 LED Boards are split in half horizontally in terms of addressing. For example, if the board has 32 rows, then providing an address of $1001_2 = 9$ will address both row 9 and row $9 + 16 = 25$.

Internally, the boards store 3 bits of data per pixel for a single row, one for each of the RGB channels to determine whether that colored LED should be on. For the 32x64 LED boards used here, they store $3 \times 32 = 96$ bits. The address wires act as a mux to determine which row should be lit based on the stored LED pattern.

HUB75 communicates over 16 wires in a 2x8 header. The HUB75 pinning can be found in the Waveshare Wiki [2]. The 16 wires consist of

- 6 RGB data wires. 3 wires control the RGB values of the upper 16 rows, while the other 3 wires control the RGB values of the lower 16 rows.
- 5 address wires. These exist for compatibility with HUB75 LED boards with 64 rows. The highest bit of the address is unused here.

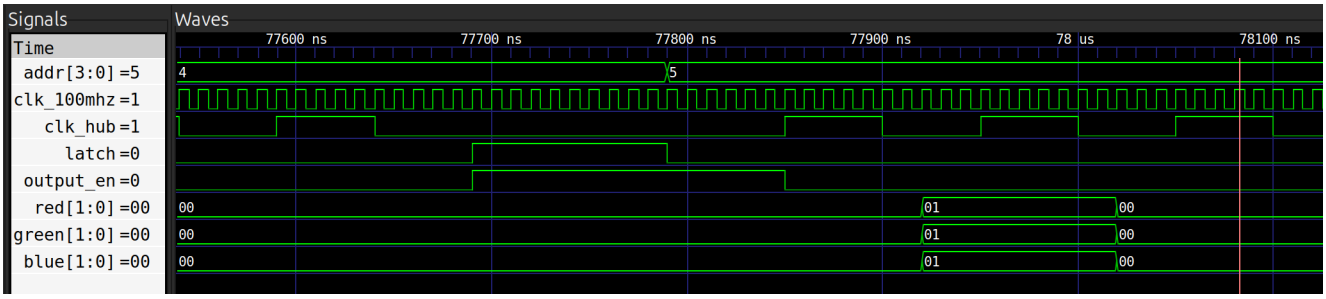

Fig. 12. Example waveform of the HUB75 output run with a 100MHz internal clock and sending data over a 10MHz clock.

- 1 clock wire.
- 1 output enable wire. This is an active low wire, so the LED boards are only lit when this wire is low.
- 1 latch wire.
- 2 ground wires.

The LED boards have a separate 5V power supply attachment point to power the LEDs.

To send data, a clock signal is sent along the clock wire. On the rising edge of the clock, the data wires are sampled and stored in the LED boards, with the pixel being sampled incrementing with each rising edge. When the latch wire is pulled high, the data stored since the previous latch are now set as the new pixel settings.

The address wires act as a mux for the current pixels and not for the data currently being sent over the wire, if any. This leads to the unintuitive behaviour that the address wires control the current row being lit (and thus the previous row that was sent) and not the current row being sent over the wire. For example, if the pixel data for row 9 and 25 are sent over the wire and the latch is pulled high and then low, the address wires need to be set to 9 even if row 10 and 26 are being sent immediately afterwards.

B. HUB75 Driver Design

The HUB75 driver controls the transmission of the pixel data to the HUB75 LED boards. Internally, the driver is a Finite State Machine that maintains 4 counters

- `power_counter` [4 bits]: Counter for the power modulator to determine when a pixel should be on to modulate brightness
- `row_counter` [4 bits]: The current row number
- `pixel_counter` [7 bits]: The current pixel number
- `cycle_counter` [4 bits]: Counter to determine the signal sent over the clock wire that increments on every FPGA clock cycle. For example, if we a send clock signal over the wire at one-tenth the rate of the FPGA clock, this counter would count from 0 to 9.

For each pixel, the driver will request the pixel on/off value on each of the 3 colors from the Power Modulator (explained in the next subsection). In particular, the HUB75 Driver will send the `power_counter`, `row_counter` and `pixel_counter` to the Power Modulator and receive the 6 pixel values that it needs to send over the wires after 2 FPGA clock cycles. We pull the data wires to the new pixel values 2 cycles after the falling edge of the data clock (see 12).

C. Power Modulator (PM)

The Power Modulator (PM) reads from the FIB and determines the duty cycle of the colors on each pixel. To maintain a high enough refresh rate on the LED boards, the modulation width is limited to 16 different values. Given the current power_counter, the PM determines whether the pixel should remain on or off.

Querying the FIB takes 2 clock cycles, so the PM is pipelined to ensure that the correct pixel data is captured and sent back to the driver.

VII. Evaluation

The resource allocation for the current build is as follows:

- Each upper and lower FIB requires one 36-kilobit BRAM each, while each frame buffer requires 18 36-kilobit BRAMs. A minimal build would use only 20 36-kilobit BRAMs in total (under 1 megabit!). A debug build with an extra frame buffer for UART transmission would use 38 36-kilobit BRAMs.
- We use 34 out of the 90 available DSPs on the Cmod A7-35T.
- Lookup Table (LUT) utilization is at approximately 4.6$ of available LUTs.

We have a significant amount of room available for additional logic, but not for additional memory as the Cmod A7-35T board only has 50 36-kilobit BRAMs slots.

In terms of timing, the main bottleneck is the incoming camera data. The Expression Recognizer only performs two passes over the data, while the Face Constructor builds a very small image on the fly. Both are also pipelined, so they complete their computations long before a new frame is able to be captured and sent by the camera.

Overall, the system achieved all of its major goals.

- It fits entirely within the helmet, save for the power supply to the LED boards and the FPGA.
- Vivado's build log puts the power utilization of the FPGA under 400mW.
- It can detect blinks and mouth movement in real-time, and reproduce them in the face reconstructor.

VIII. Limitations and Future Work

The current ProtoFacer system functions with low power requirements and little memory usage. It also fits entirely within the helmet, save for three power cables for the FPGA, LED boards and LED strip.

However, the current system has significant room for improvement.

- The eye and mouth size estimation could be made more consistent. Rapid head movement can cause the bounding boxes to become inaccurate, and this system does not easily generalize to different head shapes and sizes. Any damage that permanently moves the camera will also cause inaccurate estimation. While it might not be feasible to fit a full machine learning algorithm on the FPGA fabric at this scale, better algorithms could be used that perform more processing over the search regions.
- The internal wiring of the system can be made more robust. The camera wires are only shielded using foil tape and testing showed that this was insufficient to prevent catastrophic electromagnetic interference at the full camera resolution of 1280x720 pixels. Using a twisted pairs cable and better shielding could help improve the reliability of the system and allow for us to use a higher resolution image and RGB instead of greyscale.
- Three wires are necessary to power the system; one for the FPGA, one for the LED boards and one for the camera. It would be much more convenient to power the system from a single power source. This would require changing the FPGA to support an external power supply, and rewiring the power cables that support the LEDs.

IX. Acknowledgment

We would like to thank the MIT 6.2050/6.111 class staff for their guidance and help in the development of this project.

- Joe Steinmeyer for his invaluable advice and feedback throughout the semester as the team's project mentor.
- Kiran Vuksanaj for her work on the OV5640 camera driver and providing the necessary ROM files to modify the camera settings.

Outside of the class, we would like to thank

- Speck Snazzagen* for his suggestion to build off Coelacant's protogen designs.
- Coelacant* for the free-to-use design of the Sigma Protogen head frame.
- Unsigned Long* for their assistance with sewing the fur and general design of the head.
- Nocoffei* for his guidance with sewing the fur to the head and the design of the ears.
- Bleppy* for his advice on the design of the visor.
- Tempestas Rex* for always being there for me even when I stayed up past 4 am hunting hardware bugs.
- Kenny Zhang and Fayleon Lin for their feedback and suggestions on this report.
- MIT Project Manus for the open maker space resources such as the Ultimaker 3D printer used to print the head frame, sewing machines as well as miscellaneous electronics used in the project. In particular, we want to thank staff member Lee Zamir for his insight into making projects and assistance in procuring the parts needed for the protogen head.
- MIT Architecture Shop for allowing us to use their thermoformer to mould the visor.
- MIT Technicolor Furs for providing the fur used in the project.

*online pseudonyms, not their real names.

References

[1] Coelacant, "SigmaProtogen" (2024), GitHub repository, https://github.com/Coelacant/SigmaProtogen

[2] Waveshare. (n.d.). RGB-matrix-P2.5-64X32. RGB-Matrix-P2.5-64x32 - Waveshare Wiki. https://www.waveshare.com/wiki/RGB-Matrix-P2.5-64x32

[3] D. Nguyen, D. Halupka, P. Aarabi and A. Sheikholeslami, "Real-time face detection and lip feature extraction using field-programmable gate arrays," in IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), vol. 36, no. 4, pp. 902-912, Aug. 2006, doi: 10.1109/TSMCB.2005.862728.

[4] Digilent. (n.d.). CMOD A7 Reference Manual. Cmod A7 Reference Manual - Digilent Reference. https://digilent.com/reference/programmable-logic/cmod-a7/reference-manual

[5] RealDigital. (n.d.). Urbana Board Reference Manual. https://www.realdigital.org/doc/496fed57c6b275735fe24c85de5718c2