Extended Sight with EOG

Crystal Wang and Elizabeth Mittmann

Abstract

For the 6.111 project we propose a digitally augmented EOG. The wearer of this device will have their eye muscle movement tracked by electrodes which will control the view in either a virtual reality or a camera in an alternate location. If there is time, more complex applications can be developed, for example a moving camera in a real world or a more complex virtual reality to explore (such as a maze). The user will be able to choose program options through blinking.

1 Overall Design

1.1 Project Description

An Electro-Oculogram (EOG) is a device which measures the position of the retina along with whether the eye is open or closed using electrodes. These electrodes can be placed in pairs above and below or to the sides of the eye. As the eye changes states, the potential difference between the pair of electrodes changes to indicate the new state (the eye acts a dipole). From an EOG the wearer's eye movement can be recorded.

For our 6.111 project we first aim to create a functioning EOG and output display. This display will simply show the state of the user's eyes as determined from the EOG data. From this base we will implement a pair of environments the user can interact with through the EOG.

- Real World Environment: In this environment an external camera will be displayed on the monitor. As the user moves their eyes the image displayed on the monitor will move to show what is in that direction (i.e. if the user looks up the monitor image will pan up slightly). We also aim to add a motor for panning the camera left/right.
- Virtual World Environment: In this environment a pre-generated image will be used instead of a live camera feed. This pre-generated image will allow the user a 360 degree ability to view the virtual world, as the image will wrap back on itself in a sphere-like format.

To switch between these monitor display options of eyes, real world, and virtual world, we will implement a menu which the user can enter with an extended blink. In the menu the user can scroll (again with eye movements) through the

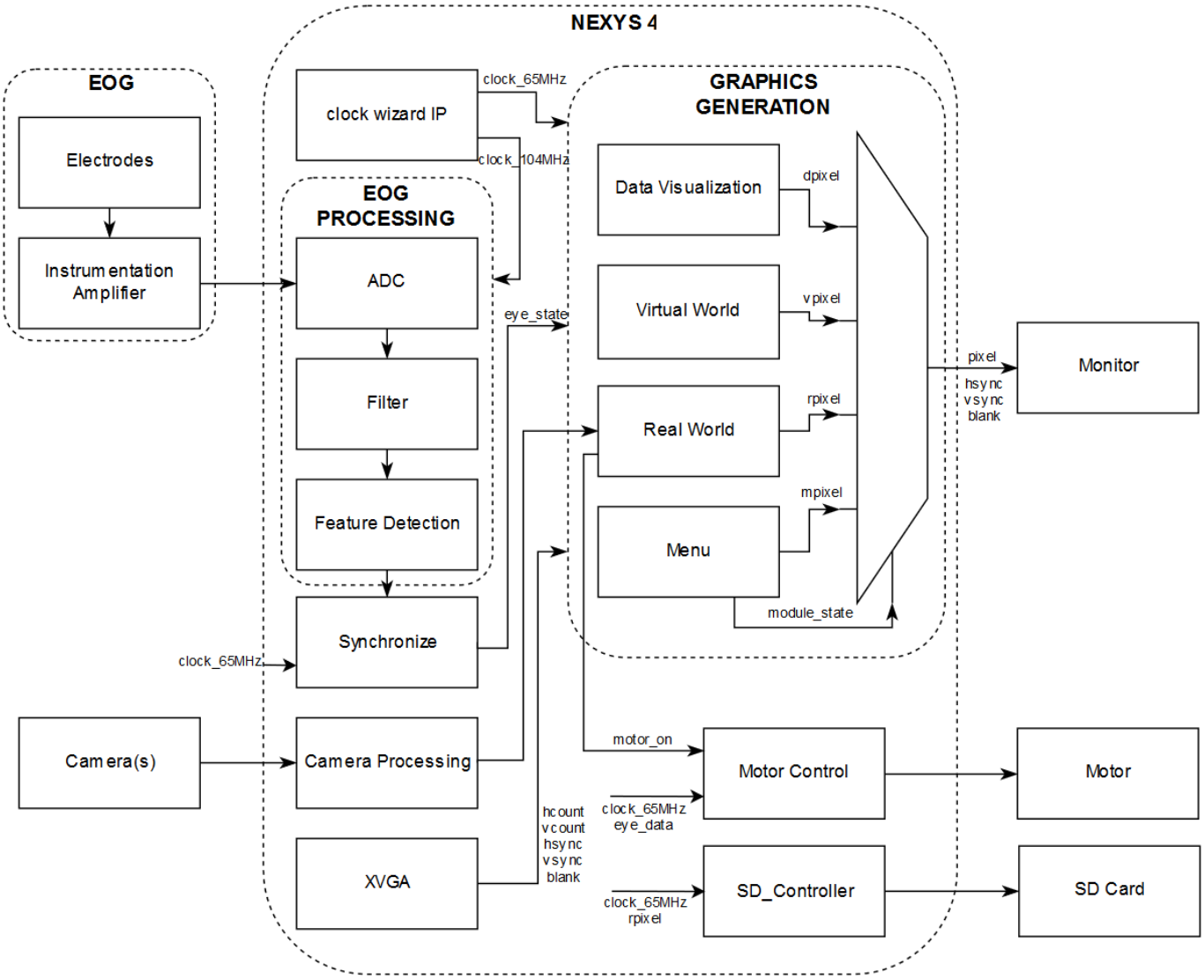
display options, and select one to view.

To allow the user to see the screen while controlling the camera's view we will implement an incremental movement system. If the user wants to pan the camera to the right they can simply look right and then look back at the screen–the degree of turning will depend on the length of time they're looking right.

As stretch goals we could expand each of the environments by:

- Real World Environment: We would like to add functionality such that when the user can take a photo of the current displayed view and it will be stored on an SD card. Super-stretch goals (to show the potential of the project but probably will not have time to implement) include adding multiple cameras (a security system for example), displaying the saved pictures, and adding a second motor (for up/down movement)
- Virtual World Environment: We would like to make the virtual world more interactive by allowing the user to walk through and look around a virtual world maze. Super-stretch goals include cursor control and a more complex virtual environment

1.2 Block Diagram



1.3 Design Description

The design of our EOG relies on electrodes measuring the differential voltage up-down and left-right across the eyes. Both of these differences are then amplified so that they are readable once put into the Nexys4.

The next large module is EOG processing which consists of reading the analog signals into a digital format in the ADC and then filters these signals so that

feature detection can be run on them. EOG processing will output variables indicating the direction the user is looking in or if they are blinking for an extended period of time.

From here the next large block is Graphics Generation where a MUX switches between displaying the menu, data output, a "real world" and a "virtual world". This large block is also connected to the camera (through a camera processing module), a motor (through a motor control module), and the SD card (through the SD controller module).

2 Implementation

2.1 Internal Components

2.1.1 Clock Wizard IP (in Vivado)

Inputs: internal 100MHz clock

Outputs: `clock_65MHz`, `clock_104MHz`

Complexity/Level of performance: This module will create a 65MHz clock (for graphics) and a 104MHz clock (for data processing) to use and sync with

- Number and type of arithmetic operations: created by Vivado
- Size of internal memories: none
- Required throughput: will output every 104 MHz and 65 MHz

How it will be tested: Display clock signals on logic analyzer

2.1.2 ADC

Inputs: Differential amplified analog signals from electrodes coming out of instrumentation amplifier, `clock_104MHz`

Outputs: Three 8 bit values at a rate of 62.5kHz (ground, left/right, up/down), `ready`

Complexity/Level of performance: The Nexys 4's XADC has dual ADCs that each return 1M sample per second but needs to run on a 104 MHz clock to process the sample at that rate. The ADC will also digitize the sampled voltages, and output sequences of 12-bit two's complement numbers (referred to as the pulse-code modulated or PCM data). We will down sample and take one of every 16 samples for a overall rate of 62.5kHz and take the 8 most significant bits

- Number and type of arithmetic operations: Conversion will require arithmetic operations, however this has already been written and optimized on the Nexys 4's ADC
- Size of internal memories: none
- Required throughput: For every incoming 16 samples on a 1MHz

clock, the ADC will output one sample. This will result in a throughput of 62.5k per second

How it will be tested: We will hook up the electrodes through the instrumentation amplifier and view the output after it has passed through the ADC on the logic analyzer

2.1.3 Filter

Inputs: Signals from electrodes coming out of ADC, `clock_104MHz`

Outputs: Two 8 bit values at a rate of 62.5kHz (left/right, up/down)

Complexity/Level of performance: Sample EOG filters needed a low pass filter, high pass filter, notch filter, and a non-inverting amplifier. Thus this will be roughly 3 times as complex as the filter we created for lab 5, and will likely be pipelined

- Number and type of arithmetic operations: multiplication, addition
- Size of internal memories: We will store the incoming samples in a 32 register array, each 8 bits wide
- Required throughput: This module will be pipelined with respect to a 104 MHz clock, but will be able to process one frame of the signal through three filters before the new data arrives from the ADC

How it will be tested: We will test each filter first individually with sample inputs like we did in fir31_test.v for lab 5, then some overall simulations. When both this module and the ADC have been tested individually, we will calibrate and test using the logic analyzer

2.1.4 Feature Detection

Inputs: Two 8 bit values at a rate of 62.5kHz (left/right, up/down) from filter, `clock_104MHz`

Outputs: A 6 bit number (`eye_state_104Hz [5:0]`) describing the instantaneous state of the eyes (Left, Right, Up, Down, Closed, Forward)

Complexity/Level of performance: Output will be delayed roughly one second to filter out blinking (which are roughly 1/3 second long) and distinguish between them and intentionally closed eyes

- Number and type of arithmetic operations: To determine from the data which category the eye direction falls into we will need a simple set of cutoffs in a case statement. From there we determine blinks vs. closed by looking at these states over time
- Size of internal memories: We plan to start with a 2 BRAMs of 8x64k which will store the last second or so of data
- Required throughput: new data at a rate of 62.5kHz

How it will be tested: This can be tested in simulation with overly simplified inputs to ensure integration and replacement for blinking works. Then it can be tested with the process signal (filter) module, as electrodes

can be hooked up and the binary output numbers can be displayed on the FPGA

2.1.5 Synchronizer

Inputs: `clock_104MHz`, `eye_state_104Hz [5:0]`

Outputs: `eye_state [5:0]` on 65 MHz clock

Complexity/Level of performance: Synchronizes EOG data (104MHz) with graphics clock (65MHz)

- Number and type of arithmetic operations: none
- Size of internal memories: none
- Required throughput: 65 MHz

How it will be tested: Input sample data at one clock rate and see it synched on logic analyzer

2.1.6 XVGA (1024x768 @60Hz)

Inputs: `clock_65MHz`

Outputs: `hcount`, `vcount`, `hsync`, `vsync`, `blank`

Complexity/Level of performance:

- Number and type of arithmetic operations: none (same as lab 3's xvga module)
- Size of internal memories: none
- Required throughput: 65MHz

How it will be tested: Shouldn't need (same as lab), can have color bars to check

2.1.7 Virtual World

Inputs: `eye_state [5:0]`, `clock_65MHz`

Outputs: `vpixel [11:0]` (Nexys 4 has 12 bit color)

Complexity/Level of performance:

- Number and type of arithmetic operations: depends on complexity of world, if attempt to make 3D will take more cycles–definitely will be pipelined
- Size of internal memories: The module will need to store a virtual world, which in it's simplest state will be an image that is roughly 4 times as tall and 4 times as wide as the actual screen. Thus it will store 4*1024x768 pixels in an internal BRAM
- Required throughput: We plan to display 1024x768 on the monitor with a 65MHz clock to achieve a 60Hz refresh rate

How it will be tested: This can be tested by connecting the inputs to FPGA buttons and viewing the output on the monitor

2.1.8 Real Logic

Inputs: `eye_state [5:0]`, camera data, `clock_65MHz`

Outputs: `pixel [11:0]`, `motor_on`, `wr` (write enable, to SD_Controller)

Complexity/Level of performance: Based on the input of eye state the section of the camera view displayed will change. This will be done through changing vertical and horizontal offset variables which will then be referenced when placing a given camera pixel on the monitor screen. Since our camera only has a resolution of 640x480, we do some math to scale the input by 2 in both dimensions

- Number and type of arithmetic operations: Addition, subtraction, and boundary comparisons will be needed
- Size of internal memories: Ideally images video would be stored in BRAM+DRAM, will store 6 frames to minimize frame tearing. Thus, 6*12*640x480bits. However given that the posted code can only store in BRAM, we will start out referencing and scaling the 1 12*640x480bit image in BRAM
- Required throughput: 65MHz for a 60 Hz refresh rate

How it will be tested: How it will be tested: This can be tested by connecting the inputs to FPGA buttons and viewing the output on the monitor

2.1.9 Data Visualization (eyes)

Inputs: `eye_state [5:0]`, `clock_65MHz`

Outputs: `dpixel [11:0]`

Complexity/Level of performance:

- Number and type of arithmetic operations: The state of the eyes will be input into a case statement to return the center dimensions of the pupils to be displayed. For each pixel a computation on whether it is inside the circle (of the eye and/or the pupil) will be run, and then the pixel will be assigned a color. To run the calculations for the circles, this process will need to be pipelined
- Size of internal memories: none
- Required throughput: 65MHz for a 60 Hz refresh rate

How it will be tested: This can be tested by connecting the inputs to FPGA buttons and viewing the output on the monitor

2.1.10 Menu

Inputs: `eye_state [5:0]`, `clock_65MHz`

Outputs: `mpixel [11:0]`, which module has been selected (`module_state [1:0]`–real world, virtual world, or data visualization; may be bigger if we decide to add more modules)

Complexity/Level of performance: The state of the eyes will be input into a case statement to return the button which the eyes are selecting. From

this selection, the module (virtual, real, or data out) will be selected and the monitor output will change to that

- Number and type of arithmetic operations: There will be a waiting period as a given option must be "clicked on" for a second or so before the user is transitioned to that output. There will also be a selection to show the user what they are clicking on which will need to be refreshed with the screen refresh but will not involve complex arithmetic
- Size of internal memories: menu display images (12 bit pixel x 1024 x 768) (3, one for each possible state)
- Required throughput: 65MHz for a 60 Hz refresh rate

How it will be tested: This can be tested by connecting the inputs to FPGA buttons and viewing the output on the monitor

2.1.11 Camera Processing

Inputs: Raw camera data, `clock_65MHz`

Outputs: `pixel [23:0]`, `href`, `vsync`

Complexity/Level of performance:

- Number and type of arithmetic operations: The camera will have storage operations to the BRAM.
- Size of internal memories: In the BRAM one image of 640 by 480 by 12 bit depth. This totals 3686400 bits or .46 Mb of BRAM
- Required throughput: 65MHz for a 60 Hz refresh rate

How it will be tested: This can be tested by creating a simulation with a preset camera input and an output variable to display the color at a specified location

2.1.12 Motor Control

Inputs: `eye_state[5:4]` (left/right), `motor_on` (from real world), `clock_65MHz`

Outputs: PWM signal to the servo

Complexity/Level of performance: This can be accomplished with a case statement that sets the servo output to one of two pre programmed settings– turning left or turning right

- Number and type of arithmetic operations: none
- Size of internal memories: none
- Required throughput: to sync with the system logic, this can trigger on the 65MHz clock, although for the camera's use throughput speed is not critical

How it will be tested: Simulation, then connected to servo with FPGA button inputs, and then paired with real world module

2.1.13 SD_Controller

Inputs: `miso`, `rd`, `wr`, `din [7:0]` (from `pixel [11:0]` of real world), `reset`, `address [31:0]`, `clock_65MHz`

Outputs: `cs`, `mosi`, `sclk`, `dout [7:0]`, `byte_available`, `ready`, `ready_for_next_byte`, `status [4:0]`

Complexity/Level of performance: We will reference the SD card information on the "Tools" tab

- Number and type of arithmetic operations: none
- Size of internal memories: none needed
- Required throughput: probably no more than 1 photo per second (blinking speed limitations)

How it will be tested: We will find test photos and store them onto the SD card and then put the SD card into a computer to verify images have been stored correctly

2.2 External Components

2.2.1 EOG Electrodes

Cost: We can use electrodes from 6.169's EKG class project for free as they bought hundreds. If those don't work or we decide other electrodes would be better we can ask the EKG group where they are getting their electrodes as high quality ones run about $100 for a set (https://mfimedical.com/products/reusable-gold-cup-eeg-electrodes)

Complexity: Not particularly complex, we just need to find ones that work

How to interface: The wires from the EOG sensors will connect to the hardware amplifier

2.2.2 Implementation Amplifier

Cost: We could use two INA 128 as our amplifier as it has a high CMRR (common more rejection ratio) for it's cost ($7.92 each), or we could use an AD620 which is used in 6.169 as the amplifier for their EKG circuit, and thus we might be able to get some to use for free

Complexity: Not particularly complex - we can follow the wiring diagram

How to interface: Supply power and ground, input signals to amplify

2.2.3 Monitor

Cost: None, already exists

Complexity: Same as past labs

How to interface: Over VGA cable

2.2.4 Camera

Cost: None, will borrow a Nexys 4 camera

Complexity: We plan to use the code Weston posted, and we will only have to crop the frame rather than any image or color recognition, so shouldn't be too different from sample code

How to interface: Over the provided cable

2.2.5 Motor (servo)

Cost: None, we own springRC SM-S4303R servos we can use

Complexity: Just need to get it to turn slowly in the correct direction - precise positioning or speed is not critical

How to interface: Need to be sent a PWM signal, along with power (4.8 - 6V) and ground

2.2.6 SD Card (memory for pictures)

Cost: None, lab already has 2GB SD cards we can use

Complexity: We need to configure the SD card

How to interface: Stick it in the Nexys 4

3 Timeline

- Week of October 31
  - EOG hardware and amplifier connected to ADC and outputting
  - Data visualization tested and working (display) + menu selection interface
- Week of November 7
  - Filter and feature detection tested and working
  - Camera data displayed on monitor & camera data processed given eye state
- Week of November 14
  - Virtual world
  - Integration, end-to-end testing
- Week of November 21
  - Virtual world
  - Motor integration + mount
- Week of November 28
  - Debug and stretch goals
- Week of December 5
  - Debug and stretch goals
- December 12 - Final Project Check Off

	10/31-11/6	11/7-11/13	11/14-11/20	11/21-11/27	11/28-12/4	12/5-12/11
EOG hardware and ADC	E					
Data visualization + menu	C					
Filter + feature detection		E	EC			
Real world + camera		C				
Virtual world			C	C		
Integration			E			
Motor integration + mount				E		
Taking pictures					EC	EC
Maze game					EC	EC

Key: E = Elizabeth, C= Crystal

4 Conclusion

In conclusion, our project will be an EOG with data, camera, and virtual world display modes. Selection between these modes will be controlled by a menu which can be selected and sorted through with eye movements. Possible extensions include taking photos with the camera and/ or navigating a maze in the virtual reality. We foresee the filtering of input data to reliably trigger on user's eye movements, and creating a seamlessly patched 360 degree 3D world to be challenges. Our goal is to integrate and expand on what we have learned about FPGAs to effectively capture and intriguingly display eye movement data. Similar systems to this have played a key role in communicating with people who have limited movement capabilities. This system can help expose others to how this technology and interface works.