"Let's Take This Outside" Boxing
6.111 Final Project Report

David A. Blau, Uzoma A. Orji, Reesa B. Phillips

May 18, 2006

TA: Javier Castro

Abstract

"Let's Take This Outside" Boxing is a one player or two player game in which fighter box to the death. The user interface is comprised of a camera and sensors equipped with accelerometers. The camera detects the position of the sensors and the accelerometers measure the force of a punch. The position of the boxer is inferred from the on-screen coordinates of the gloves. When a punch is detected, the control unit determines whether the punch hit the opponent and how forceful it was. The more forceful the punch, the more damage is done. The game continues until one player loses all her life. The output displays an image of the gloves and the opponent and health meter. The fighters, their health meters, and the boxing ring are drawn using sprites read from ROM.

Contents

List of Figures

1 Introduction

The motivation for the project came from wondering if Reesa can beat up Uzoma. Because of the obvious discrepancy in size between the two, actual boxing would not be plausible. There had to be a way to level the playing field. Hence, "Let's Take This Outside" Boxing is born.

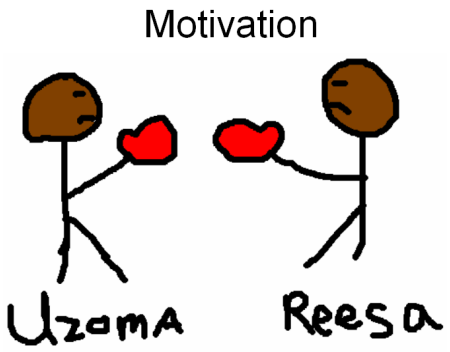


Figure 1: Motivation for the project. Can Reesa beat up Uzoma?

This project is a virtual boxing game. It will have two modes: a one player mode and a two player mode. In the one player mode, a single user will test his boxing abilities against the computer. In the two player mode, two friends (or enemies) get to slug it out against each other for bragging rights. Figure 2 shows a block diagram of the overall system. There are three main components for this game: the user interface, control unit and imaging block. Each component is discussed in the following sections.

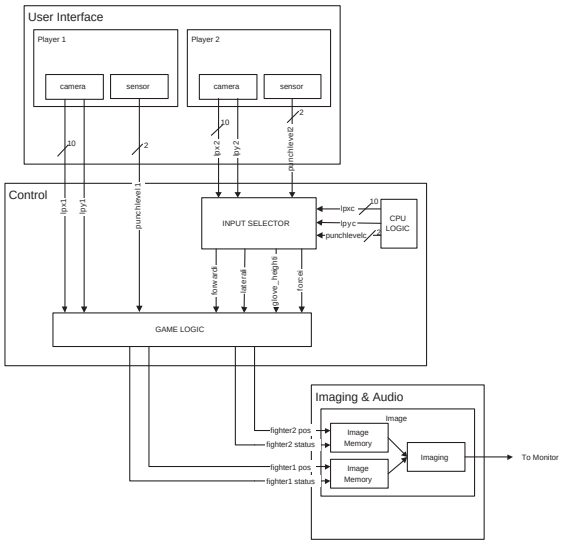


Figure 2: Block Diagram of the Overall Boxing Game.

2 User Interface

The user interface allows the player to control his or her boxer during the game. The first component is the camera which sits in front of the user. With the camera, the horizontal position and the height of the

gloves can be determined. The user wears a device equipped with bright LEDs. The camera detects the LEDs to determine the positions of the gloves. The sensors are the other component of the user interface. As mentioned, the sensors include bright LEDs used for position detection. They also sense punches. The block diagram for the user interface is shown in Figure 3. The sensors are described in detail next.

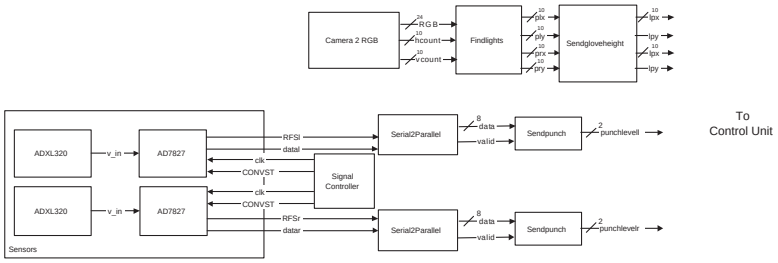
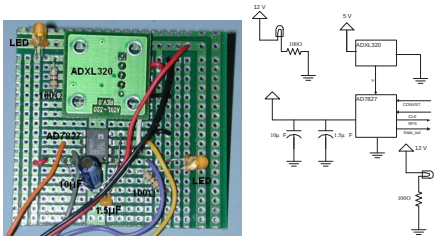


Figure 3: Block Diagram of Overall User Interface.

2.1 Sensors

Each glove contains a circuit board with the ADXL320EB two axis analog accelerometer. The analog voltage is converted to a digital signal using the AD7827 8 bit serial analog to digital converter (ADC).

A picture and a circuit schematic of the sensor used by the user is shown in Figure 4.



(a) (b)

Figure 4: Picture and schematic of the sensor equipped with an accelerometer, ADC and LEDs.

The ADC converts the input voltage from the accelerometer and outputs the the 8 bit number serially. A Serial2Parallel FSM is created to convert the serial data into an 8 bit number. A state transition diagram for the Serial2Parallel FSM is shown in Figure 5.

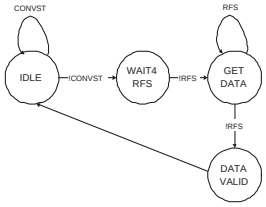


Figure 5: State Transition Diagram for Serial2Parallel FSM.

Once a CONVST signal has been sent to the ADC, a conversion begins. The FSM waits until the ADC

sets RFS low. This transition indicates the start of the serial transmission. The FSM registers the data until RFS goes high meaning the 8 bit number has been sent and the data is now valid.

Long wires are needed to interface the board with the labkit. These wires have a delay associated with them which means that the synchronization between the ADC and the labkit must be handled carefully. At 25.175 MHz, the clock is too fast that the labkit samples incorrect data because of the delay. To circumvent this problem, the data sent between the labkit and the ADC is synchronized to a slower 3.375MHz clock. Moreover, the data from the ADC is sampled by the labkit on the falling edge of the clock. This gives the data a half period to travel the length of the long wires.

Once the 8 bit serial data is received by the labkit, a punch detection modules determines if the user has thrown a punch and the strength of the punch if one was thrown. The game only has 3 levels of punches: soft, medium and hard. The SendPunch FSM handles the detection of punches. Its state transition diagram is shown in Figure 6.

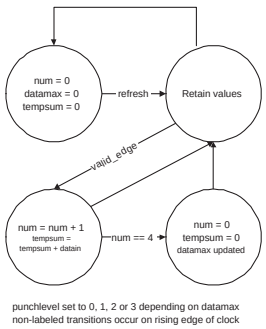


Figure 6: State Transition Diagram for SendPunch FSM.

The SendPunch receives nearly 100 samples from the ADC during a 3.375 MHz clock cycle. It averages 4 samples to remove noise and retains the maximum value detected every 10 frames of the boxing game. The maximum value during the time interval is the punch data sent to the control unit. The last remaining piece of data from the User Interface is the position of the gloves. The camera is the main hardware used to detect position.

2.2 Camera

The first step in using the camera is decoding its output signal. The camera uses an NTSC decoding scheme in which the luminance (Y) and chrominance values (Cr and Cb) are given for each pixel. Moreover, the pixels of the odd lines of a frame are provided by the camera before the pixels of the even lines. For simplicity, only the oddlines of a given frame are written to memory. For processing purposes, the oddlines are essentially stretched by a factor of 2 to interpolate a full frame and this new frame is the frame written to memory. The end goal is to write each pixel's YCrCb to RAM so that the data can be converted to RGB color space and used to detect the sensors worn by the users. The overall block diagram for decoding the camera input to an RGB value is shown in Figure 7.

Conversion from YCrCb to RGB is simply a matrix multiplication defined as:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.164 & 1.596 & 0 \\ 1.164 & -0.813 & -0.392 \\ 1.164 & 0 & 2.017 \end{bmatrix} \begin{bmatrix} Y - 64 \\ C_r - 512 \\ C_b - 512 \end{bmatrix}$$

The hardware implementation of the matrix multiplication is shown in Figure 8.

It should be noted that a filter is placed in front of the camera to block out the fluorescent lighting. These filters are essentially sunglasses that make the detection of the LEDs much easier. The original LEDs used were the typical directional types. Therefore, they needed to be pointed directly at the camera in order

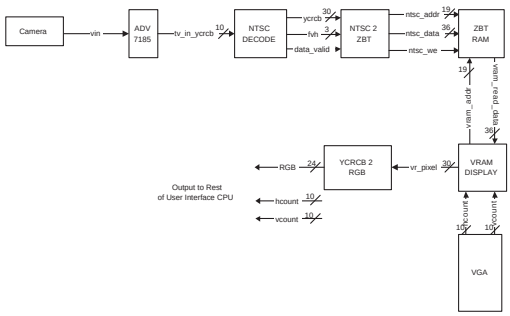


Figure 7: Block Diagram for Conversion of Camera Input to RGB Color Space.

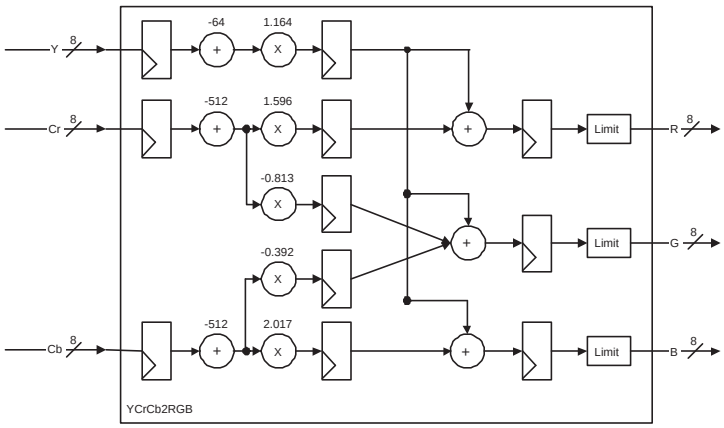


Figure 8: Hardware Implementation for the Conversion of the YCrCb to RGB.

to obtain good results. There are two LEDs used in the design to try to solve this problem. If the user had her hands up, one LED would be detected by the camera. When the user extended her arm to punch, the other LED would be used for detection. These LEDs were later replaced by mini lamps which send light in all directions and were much easier to find.

The purpose of the camera is not to display to the VGA but to detect the bright LEDs on the gloves of the user. As the RGB values of the pixels are read from RAM, the findlights module looks for colors that correspond to a bright light. It looks for two clumps of bright light which should correspond to the left and right hands of the user. The center of these clumps are the assumed hand positions and are sent to the control unit.

3 Control Unit

The Control Unit (CU) is responsible for manipulating the inputs to simulate a boxing match. The CU's inputs are the horizontal, vertical, and strength values for each glove, and a pulse from the VGA controller signaling the end of a frame. Using the glove information, the control unit determines the position of the player's head, whether a punch was thrown and if that punch connects, and decrements the life and energy levels appropriately. The CU outputs the position of the player's left and right gloves, whether or not each hand is punching, life and energy levels, and the time left in the round to the display module.

There are three game types in which the CU can be. First is single player mode, in which the player boxes against the AI. The second and third modes are two player master and two player slave. In the two player modes, all computation is done by the CU of the master labkit; the slave labkit sends glove data and receives display information.

Along with the game types, there are four game modes which determine the different stages of interaction. The first mode is the TITLE mode, in which the player selects the game type. Second is the ROUND mode where the bulk of the game play takes place. The last two modes are the REST mode for between rounds and the GAMEOVER mode for after three rounds or a player gets three knockouts.

The Control Unit is a major finite state machine (FSM) with seven states: IDLE, STARTGETDATA, WAITGETDATA, STARTCOMPUTE, WAITCOMPUTE, STARTSENDDATA, and WAITSENDDATA. Initially, the CU begins by entering the TITLE mode for selecting game type and awaits a start pulse from the VGA controller in the IDLE state.

In ROUND mode of both single player and two player master game types, the CU enters the STARTGETDATA state on the frame pulse. Once the appropriate minor FSMs have reported that they are busy, the CU enters the WAITGETDATA state. For single player mode, the AI serves as the minor FSM charged with providing opponent glove data. In two player master mode, the opponent's glove data is received from the inter-labkit bus through the Get User Data module. When the appropriate minor FSM has produced valid data, they report that they are no longer busy, and the CU enters the STARTCOMPUTE state. In this state, the CU starts the minor FSMs governing each fighter determine the position, punches, life, and energy of each player and the minor FSM that handles the time. Once all three computational minor FSMs have started, the CU enters the WAITCOMPUTE state until the FSMs have all finished. When these minor FSMs finish, the CU returns to the IDLE state, in single player mode, or continues to STARTSENDDATA and starts the Send Display Data minor FSM, in two player master mode. When the module has started, the Control Unit goes into the WAITSENDDATA state. The CU returns to the IDLE state when the Send Display Data module reports that it is no longer busy. A finite state machine diagram of ROUND mode in single player and two player master game types is shown in Figure 9.

In ROUND mode of the two player slave game type, the CU has significantly less dependence on the frame pulse. The Control Unit only assumes the IDLE state when ROUND mode is first entered or if some illegal state is registered. From the IDLE state, the CU enters the STARTSENDDATA state to send the glove data to the master CU through the Send User Data minor FSM. When the minor FSM is busy, the CU enters the WAITSENDDATA until the transfer completes. Because the slave CU performs no computation, the CU immediately goes to the STARTGETDATA state, starting the Get Display Data minor FSM. When the Get Display Data FSM is busy, the Control Unit enters the WAITGETDATA state. When display data has

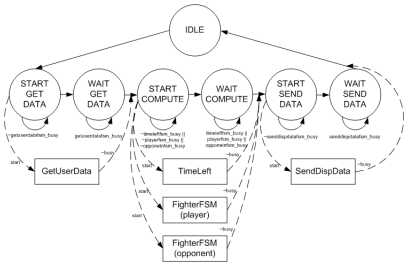


Figure 9: The finite state machine transition diagram for single player and two player master game types

been fully received, local registers are updated, and the Control Unit returns to the STARTSENDDATA state, bypassing the IDLE state. By remaining in the send glove data state, the slave is guaranteed to be prepared to send glove data at the master's request. Because the slave has no computation state, it enters the get display data states before the master, allowing the master to continue without delay. To prevent deadlocks, the slave has an override condition. If the slave is currently waiting to send data and receives a request to receive data, the slave CU immediately transitions to the get display data states. While this case mediates the effects of clock skew, it desynchronizes the CU's FSM from the slave's display module and causes periodic glitching. Fortunately, the glitching is negligible compared to the stable movement during game play. A finite state machine diagram of ROUND mode in two player slave game types is shown in Figure 10.

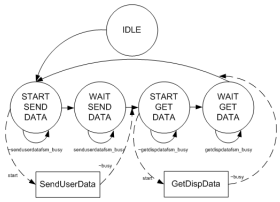


Figure 10: The finite state machine transition diagram for the two player slave game type

When the time left in the round has decreased to zero, the CU enters REST mode for thirty seconds, and increments the round number. During the REST mode, both players' life and energy bars are replenished. After thirty seconds have expired, CU returns to the ROUND state.

If a player's life ever decreases below zero, characterized by register underflow, the player's number of knockouts is incremented. If both players' knockout counts are below 3, the CU enters the REST mode and the round ends. Otherwise, CU enters the GAMEOVER mode.

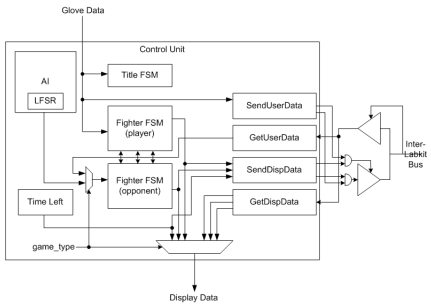


Figure 11: The block diagram for the Control Unit

The modules of the Control Unit are connected according to the block diagram in Figure 11. The glove data is fed directly into the Title FSM for selection at the title screen, to one instance of the Fighter FSM

to handle the player's punches, and into the Send User Data module to send glove data across the bus. The second Fighter FSM instance receives inputs from the AI or the Get User Data module depending on game type. A Time Left FSM is responsible for handling time during and between rounds. The outputs from the Fighter FSMs and the Time Left FSM are output to the Send Display Data module. If the CU is in the single player or two player master game type, the Fighter FSM outputs are also output to the display module. If in two player slave game type, the output to the display module is read from the Get Display Data module.

3.1 Title FSM

The Title FSM is only active during the TITLE mode of play. The FSM awaits a punch before sending the Control Unit into ROUND mode. If the punch is in the left-most third of the screen, the game type is set to single player. If in the middle third, the game type is two player master. The right-most third is for two player slave. When the next mode is set, the Title FSM initializes the life bars, energy bars, and time left to 31 units, 31 units, and 180 seconds, respectively. The bars are represented as five bit numbers; 31 is their maximum value. 180 seconds is the standard time for a boxing round.

3.2 Time Left FSM

The Time Left FSM is active during the ROUND and REST modes of play. When started, the FSM receives the current time left as input. Every frame, the FSM outputs the next number of seconds left. During most frames, the output value is equal to the input, but every half second, the output value is the input decremented by one. Half seconds are used because a full three minute round is a lot longer than most people realize. When the FSM is no longer busy, the Control Unit updates its time left to this FSMs output value.

3.3 AI

The AI is driven by four 16-bit pseudorandom number generators (PRNG), which are implemented as linear shift feedback registers. The module also takes the player's horizontal position (the midpoint between the gloves) and the horizontal position of both gloves. The first PRNG generates a decision value that determines whether or not the AI will move or punch. If moving, the decision value also determines whether or not the AI moves to the left, to the right, closer to the player, or away from the player. The second PRNG determines how far the AI will travel if moving. The third PRNG determines the hand to throw the punch, whether the punch will be high or low and whether to throw a light, medium, or hard punch. The last PRNG randomly positions the gloves relative to the AI's position to mimic spontaneous random hand movement.

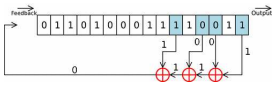


Figure 12: The diagram for a 16-bit linear feedback shift register

The linear feedback shift register is implemented as a simple 16-bit register that shifts to the right when enabled. The most significant bit of the next value is a function of a subset of the bits of the current value. The register is initialized with a seed value. Although the bits are deterministic in binary radix, as unsigned decimal numbers they appear to be random. Figure 12 shows a diagram of the pseudorandom number generator. Choosing an appropriate seed value requires understanding of the function chosen to generate the most significant bit. Unfortunately, many seeds lead to an AI that prefers to do nothing rather than move around or punch.

3.4 Fighter FSM

The Fighter FSM is responsible for translating punches into a boxing match. The Fighter FSM takes in the glove data for a given player, the horizontal position of the opponent's head, the state of the opponent, and the vertical position of one of the other opponent's hands. The FSM outputs the player's energy, the player's state, the picture parameter for the player's state, the opponent's life. The two instances of the Fighter FSM are tightly coupled because there is heavy interaction whenever a punch is thrown. The picture parameter is updated every change in state or significant hand movement.

The Fighter FSM has four states: STAND, BLOCK, PUNCH, and RECOIL. The FSM also has a stall counter, used to simulate the pausing effects of punching, blocking, or getting hit. In all states, the stall counter decrements unless it is zero. While the stall counter is nonzero, activity is suspended. In the STAND state, the output reflects the input directly. When the player's gloves are within 100 pixels of each other, horizontally, and on the same vertical level, the FSM transitions into the BLOCK state. If the player has sufficient energy and throws a punch, the stall counter is incremented and the energy bar is decremented proportional to the strength of the punch. If the punch is close enough to the opponent's head position, the opponent's stall counter is incremented. The opponent's life bar is only decremented if their state was not BLOCK. After the PUNCH state, the fighter FSM enters the RECOIL state with a nonzero stall value and a smaller combo-stall value. In the RECOIL state, if the stall value is less than the combo-stall value and the player issues the next punch of a combo, the rest of the stall delay can be bypassed and a punch thrown relatively quickly. While combos allow for faster sequences of punches, they drain the energy bar much more quickly than they automatically replenish. When the stall counter reaches zero, the FSM transitions back to the STAND state.

3.5 Inter-Labkit Communication

The communication between labkits uses eighteen of the user4 pins. Because one labkit is designated master and the other designated slave, eight of the pins are used exclusively by either labkit; only the 10 data pins are bidirectional. More than ten bits of data are transferred during every exchange. As a result, data is transmitted with a 3-bit address. The receiving labkit acknowledges the transmission by matching the 3-bit address, signaling the transmitting labkit to proceed to the next 10 bits of data. The directionality of the 10 data pins is controlled by either labkit's enable pin, which is also transmitted on the bus. To prevent both labkits from driving the data pins, the slave labkit will only drive the pins if its enable is high and the master labkit's enable is low. This slave backout during collisions works in tandem with the slave control unit to prevent deadlock. Figure 13 shows the allocation of the data pins on the bus.

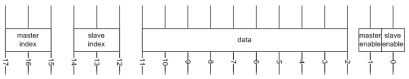


Figure 13: The inter-labkit bus. Index and Enable pins are designated by labkit. Only the data pins are shared between labkits.

The original communication design was based on an acknowledgement pulse instead of matching addresses. In test benches on a single labkit, the pulse was always received and sufficient to continue data transfer. When communicating between two labkits, factors such as clock skew, attenuation from the ribbon, and external noise all reduce the probability of a single cycle pulse being received correctly. Using level sensitive acknowledgements may take a few more cycles per data chunk, but is much more reliable and actually works outside of test benches. For all communication modules, the address zero indicates that no more information is available to send.

3.5.1 Send User Data

The slave labkit must transmit the glove data to the master labkit in order for the frame to be computed. The Send User Data module sends the player's glove data across the bus to be interpreted as the opponent

by the other labkit. The first ten bits sent (with address 1) are the horizontal position of the left glove. Then next three chunks of data are the, the horizontal position of the right glove (10 bits), the vertical positions of both gloves (1 bit and 1 bit), and the punch strengths of both gloves (2 bits and 2 bits). The fact that vertical positions and punch strengths are not grouped together is a relic of there previously being two address pins and using all four addresses. When the master labkit's synchronized address pins match the outgoing address, the next address and chunk is sent. After address 4, the module sends address 0 and all zero data and lowers the enable pin, relinquishing the bus, and indicating to the Control Unit that it is no longer busy.

3.5.2 Get User Data

The Get User Data module is complimentary to the Send User Data module. The module is initialized to address zero, waiting for an incoming address that does not match the current address. When the slave's enable pin is high, the incoming address and data pins change. The Get User Data module decodes the new data into appropriate registers and matches the address. This process continues until the Get User Data receives an address of zero, in which case the module reports that to the Control Unit that it is no longer busy, and the output values are available to update the CU registers.

3.5.3 Send Display Data

After computing the current frame, the master labkit must send the display data back across the bus to the slave. Similar to sending glove data, the Send Display Data module loads the data and address pins with new data whenever the slave labkit matches address pins. First the opponent's left hand horizontal position is sent (10 bits), then right hand horizontal position (10 bits), the vertical positions of both gloves and the time left (1, 1, and 8 bits), the player's horizontal head position (10 bits), then the player's picture parameter, end-of-round bit, and both opponent's gloves' punch strengths (5 bits, 1 bit, 2 bits, 2 bits), then both life bars (5 bits and 5 bits), and last both energy bars (5 bits and 5 bits). After the slave labkit matches the last index, the module sends the zero address and zero data, signifying the end of transmission. Also, the module reports that it is no longer busy and the Control Unit may proceed.

3.5.4 Get Display Data

The Get Display Data module, analogous to the Get User Data module, decodes the incoming data and address into appropriate registers to output to the slave labkit's display module. Initialized to the zero address, the module begins reading data pins when the master labkit places a nonzero address on the bus. When the Get Display Data receives a subsequent zero address, it indicates the end of transmission, the module reports that it is no longer busy, and the Control Unit can output the display data to the display module.

4 Imaging

4.1 VGA Module

The game timing is determined by the timing behind the VGA module. The parameters for this module correspond to a 640 x 480 resolution display with a 60Hz frame rate. The module outputs the active low, horizontal and vertical, sync and blank signals to the ADV7125 DAC. It also outputs the pixel count and line count values that map to a single on-screen pixel and a frame pulse that is set high for one clock cycle at the end of a frame (when pixel count equals 639 and line count equals 479). On every rising edge of the clock, the module increments its pixel count. When the pixel count reaches 639, the module moves from horizontal active video to front porch and sets the horizontal blank signal to zero. When equal to 655, the module moves from horizontal front porch to sync and sets the horizontal sync signal to zero. When it reaches 751, the module moves from horizontal sync to back porch and sets the horizontal sync signal to one. When the

pixel count reaches 799, the module moves from horizontal back porch back to active video, the counter is reset, the horizontal blank is set to one, and the line count is incremented. For the line count, the vertical blank is set to zero at line 479 which is the end of vertical active video, the vertical sync is set to zero at 490 which is the end of vertical front porch, the vertical sync is set to one at 492 which is the end of vertical sync, and the vertical blank is set to one at 523 which is the end of vertical back porch (and the line counter is reset). Each sync signal is passed through a two cycle delay register chain to match the delay of the other VGA signals from the DAC. The frame pulse is used by the Control Unit and the pixel and line counts are used by the by the Display Field to output the appropriate color for each pixel.

4.2 Display Field Module

The Display Field module outputs eight-bit red, green, and blue signals to the DAC which illuminate the screen. The inputs include the pixel clock, reset signal, and the frame pulse, pixel count, and line count from the VGA module. They also include the game information from the Control Unit such as the mode, round, and time left; the player information such as the current vertical signal, x coordinate, and status of each hand and the life and energy left; and the opponent information such as the current x and y coordinates, status, life, and energy. Instead of controlling the pixel colors directly, each object is drawn using submodules. The pixel and line counts are input to each submodule, and, with the exception of the submodule that creates the round number, the submodules output a color or black (all 0s).

The Display Field module determines which of the submodule outputs it needs to output based on the mode. In the start mode, the module outputs the submodule outputs that create the player's gloves and the one player and two player options as seen in Figure 14.

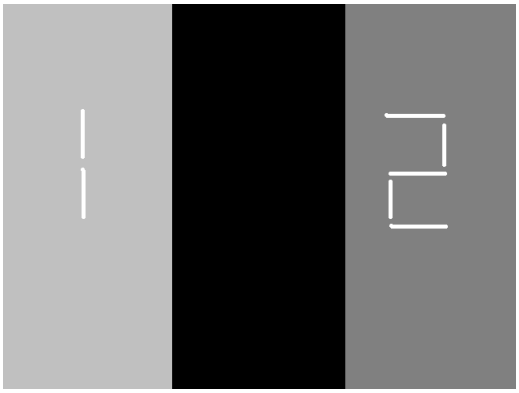


Figure 14: A screenshot of the start mode menu.

In the round mode, the module outputs the submodule outputs that create the player's gloves, the opponent, the time left, the boxing ring, and the life and energy bars as seen in Figure 15.

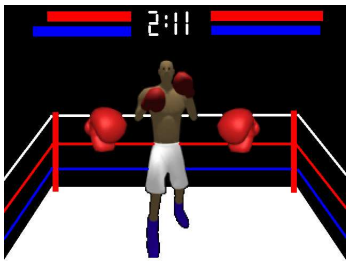


Figure 15: A screenshot of the game during a round.

In the in-between-round mode, the module outputs the submodule outputs that create the round number card as seen in Figure 16.

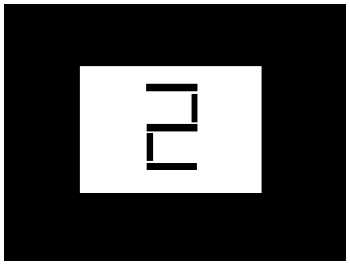


Figure 16: A screenshot of the in between rounds mode.

In the game over mode, the module outputs the same submodule outputs as in the round mode as seen in Figure 17 (the "GAME OVER" image displayed across the screen is created in the Fighter 1 Control module which the Display Field module outputs in both modes).

Figure 17: A screenshot of the game over mode.

The layout of which image to overlay on another is created by simple logic within the module. With the exception of the in-between-round mode, the module implements the following order with that logic. First, the module outputs the submodule output that creates the player's gloves if that output is not black. Then, the module outputs the submodule output that creates the opponent if that output is not black (except during the start mode). Finally, the module outputs the bit-wise OR of all the submodule outputs that create the background (such as the one and two player options and the boxing ring) since the background images do not overlap. The in-between-round mode operates differently because the submodule that creates the round number outputs black for the number and white (all 1's) otherwise. Therefore, in this mode, the module outputs the submodule output that creates the round number and the submodule output that creates the white card only if these outputs are black. This creates the black number and black background on the screen. Then the module creates the white card by outputting the rest of the outputs of the submodule that creates the white card.

4.3 Glove and Fighter ROMs

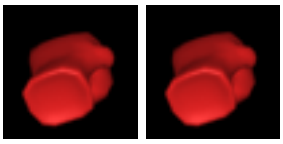
In order to display more complex images, bitmap files of the player's gloves and the opponent were created and each image was stored in its own synchronous ROM. A java program was created to convert the bitmap files to coe files to upload to the Glove and Fighter ROMs. In order to conserve space, the data in the ROMs are 4-bits wide. The ROMs take the pixel clock and address as inputs and output the 4-bit data. Each ROM has one clock cycle of latency.

4.4 Fighter 1 Control Module

The Fighter 1 Control module creates the image of the player's gloves. It takes as inputs the pixel clock and reset signal, the pixel count and line count, the vertical signals, current x coordinates, and status of the player's left and right hands, and the mode and outputs the 24-bit RGB signal. The module sets the vertical position of each glove to one of two positions based on its corresponding vertical signal. There are four ROMs within the module that correspond to two different images of the left glove and two of the right and one ROM that corresponds to the game over image. It takes one clock cycle to compute and register the address, another to get the data from the ROM, and another to convert the 4-bit data into a 24-bit signal corresponding to the right color. Due to this delay, the position three pixels/clock cycles ahead is calculated within the module. If that position is within the boundaries of the gloves or the game over image (in the game over mode), the next ROM address is determined and assigned and then the ROM data is registered and converted during the three cycles in order to ensure that the converted data is available when it is needed.

The images within the ROMs need to be scaled by a factor of two. Therefore, the ROM address is incremented by one every other pixel to scale it horizontally. Every other line, the ROM address is decremented by the image width to repeat the previous line; and on the following line, the ROM address is incremented by one in order to continue to the next line of the image, thus scaling the image vertically.

The module chooses between the different pictures for the left and right gloves based on a 1-bit status input for each glove. If the status is low for the left glove, the module chooses the ROM containing the image in Figure 18(a); otherwise, it chooses the ROM containing the image in Figure 18(b). If the status is low for the right glove, the module chooses the ROM containing the reverse image of that in Figure 18(a); otherwise, it chooses the ROM containing the reverse image of that in Figure 18(c).



(a) Low left glove

(b) Hight right glove

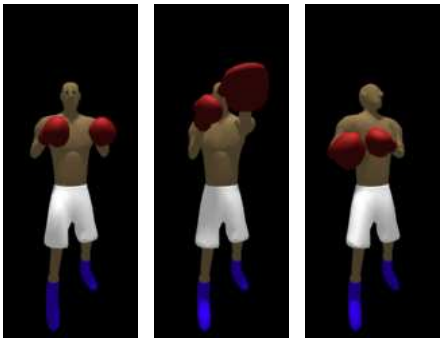
If the game is in the game over mode and the current x and y coordinates are within the boundaries of the game over image, the module outputs the converted data from that ROM. If the current x and y coordinates are within the boundaries of the gloves, the module outputs the converted data from the corresponding glove ROM. Otherwise, the module outputs the 24-bit RGB signal that forms black.

4.5 Fighter 2 Control Module

The Fighter 2 Control module creates the image of the opponent. The inputs are the pixel clock and reset signal, the pixel count and line count, the current x and y coordinates and the status of the opponent, and the mode and outputs the 24-bit RGB signal. There are twenty ROMs within the module that correspond to twenty different images of the opponent. The timing and the procedures to scale the images and convert the 4-bit ROM data to a 24-bit RGB signal in the Fighter 1 Control Module are all exactly the same for this module.

The module chooses between the different pictures of the opponent based on a 5-bit status input. Figures (18(c) - 18(e)) show three examples of images stored in ROMs that the module can choose from. If the current

x and y coordinates are within the boundaries of the opponent, the module outputs the converted data from the corresponding glove ROM. Otherwise, the module outputs the 24-bit RGB signal that forms black.



(c) (d) (e)

Figure 18: Sample images of the fighter

4.6 Boxing Ring Module

The Boxing Ring module is responsible for creating the image of the boxing ring. It takes the pixel clock and the current x and y coordinates (which are the pixel count and line count from the VGA module) as inputs and outputs the eight-bit red, green, and blue signals. If the current x and y coordinates are within the boundaries of the poles, ropes, or floor of the boxing ring, the module outputs the red, green, and blue signals that form the color corresponding to that object. Otherwise, the module outputs the red, green, and blue signals that form black.

4.7 Number Module

The Number module is a simple logic circuit that takes as inputs the pixel clock, the current x and y coordinates, the x and y coordinates of the top left corner of the number, and a 3-bit number between 0 and 9 to display and outputs the eight-bit red, green, and blue signals. The module defines seven segments and determines which segments correspond to the input number. If the current x and y coordinates are within the boundaries of the segments corresponding to the input number, the module outputs the red, green, and blue signals that form the color corresponding to that object. Otherwise, the module outputs the red, green, and blue signals that form the background color.

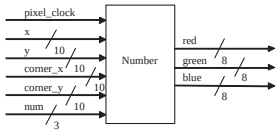


Figure 19: Block diagram of the Number Module.

4.8 Rectangle Module

The Rectangle module is a simple logic circuit that takes as inputs the pixel clock and reset signal, a blink signal, the current x and y coordinates (which are the pixel count and line count from the VGA module),

the x and y coordinates of the top left corner of the rectangle, and the width and height of the rectangle and outputs the eight-bit red, green, and blue signals. The blink signal is used to determine whether or not the rectangle should blink once a second like a cursor does. Within this module, there is an instance of the Counter module which creates a 1Hz enable signal in order to implement the blinking effect. Therefore, if the blink signal is low and the current x and y coordinates are within the boundaries of the rectangle, the module outputs the red, green, and blue signals that form the color of that rectangle. Otherwise, it outputs the red, green, and blue signals that form black. However, if the blink signal is high and the current x and y coordinates are within the boundaries of the rectangle, the module only outputs the red, green, and blue signals that form the color of that rectangle every other second. Otherwise, it outputs the red, green, and blue signals that form black.

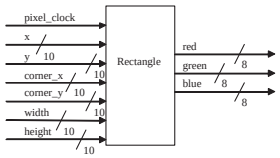


Figure 20: Block diagram of the Rectange Module.

4.9 Timer Module

The Timer module is responsible for converting the time left from seconds into minutes and seconds. The module takes as inputs the pixel clock, reset signal, and the time in seconds and outputs the minutes and seconds corresponding to the time. The time left only ranges from 0 to 180 seconds. Therefore, the module implements the simple logic to convert 180 seconds to 3 minutes and 0 seconds, 128 seconds to 2 minutes and 8 seconds, and so on.

The block diagram of the entire Imaging component is shown in Figure 21.

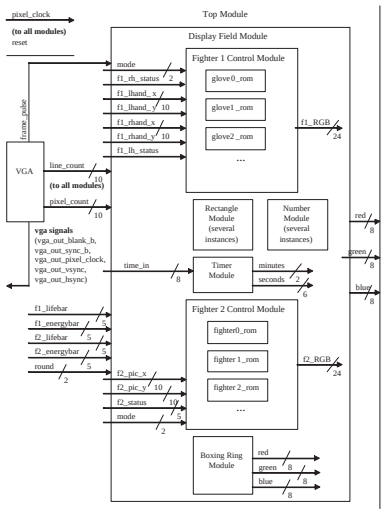


Figure 21: Block diagram of the entire Imaging component.

5 Acknowledgments

We would like to thank the 6.111 staff for supervision of the project: Professor Anantha Chandrakasan, Javier Castro, Theodoros Konstantakopoulos, and Kyeong-Jae Lee. We extend deep gratitude towards Gim Hom and Christopher Falling for their extensive help during debugging. We thank David Dunmeyer for providing the filter used with the camera. Last but not least, we thank Group 2 for providing comic relief during late night hours in the lab. Those boys are silly. That's It.

6 Conclusion

Our objective was to create a boxing game and we accomplished the daunting feat. We were able to integrate the individuals blocks successfully (although at the last possible minute). Our video turned out better than expected and despite the long hours in the lab, the experience produced fruitful results. And that concludes this report. We're now officially done with 6.111.