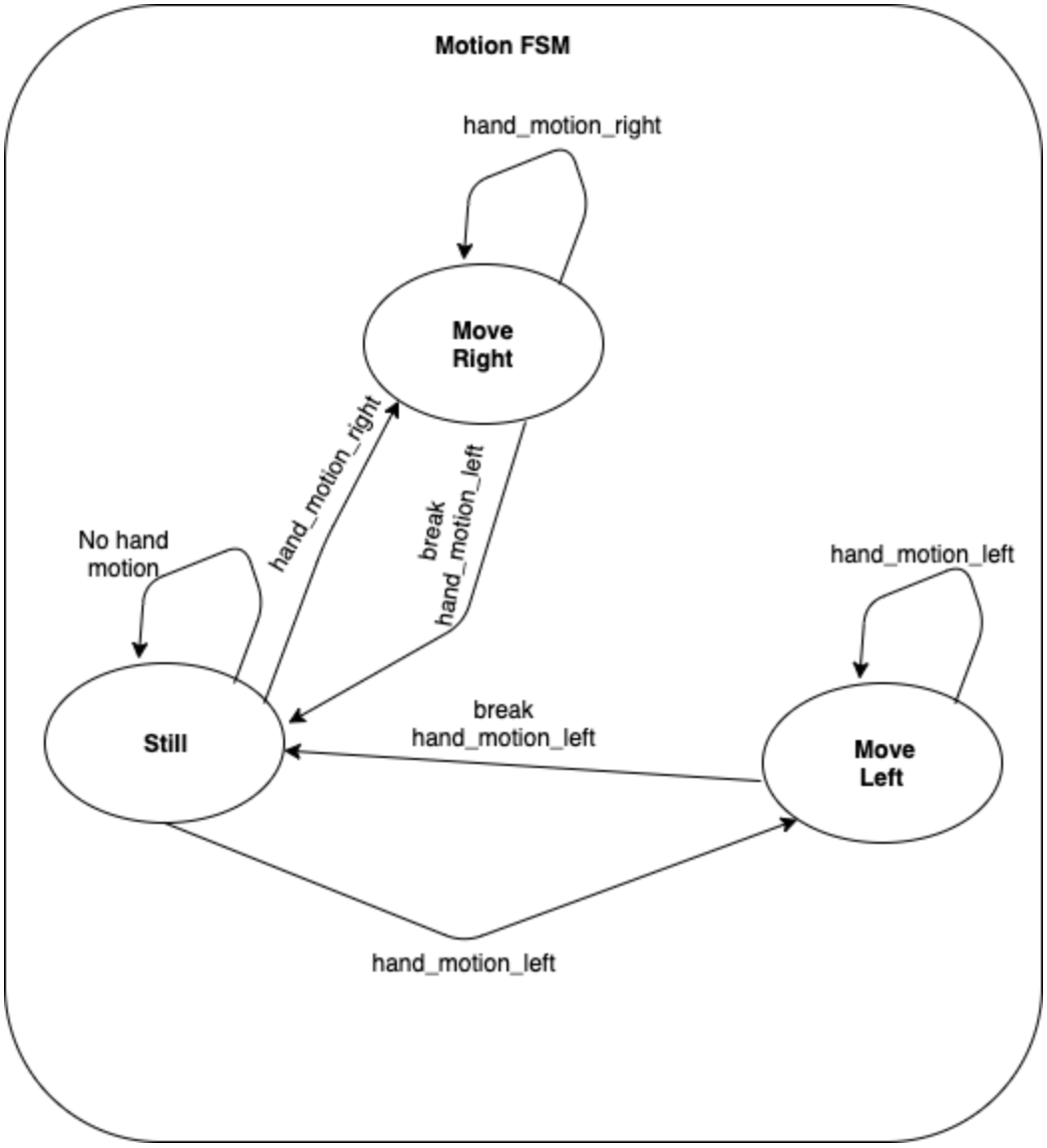
Game Logic (Diana):

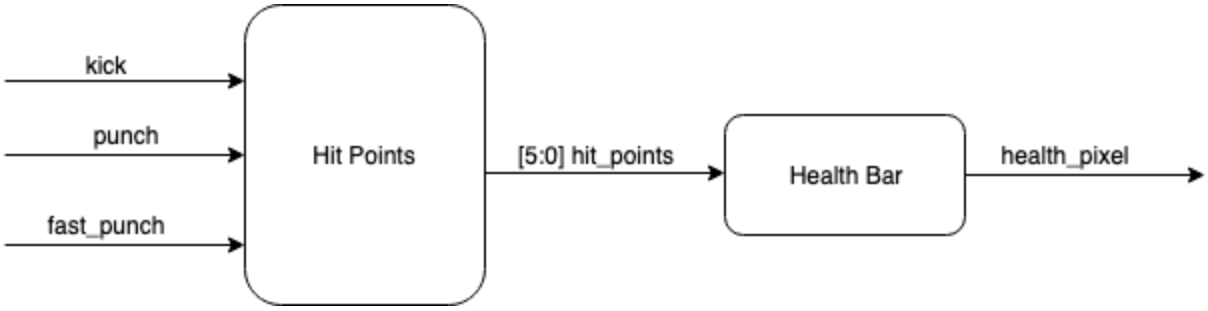
Overview

The game logic will describe everything involved in the various state machines and modules necessary to interact between all the Sprites (entities to be displayed). The main sprites are player_1, player_2, the AI, and the health bars for each player. Depending on time we may implement more sprites to enrich the visual experience. The intended functionality is to have two player modes, either player_1 versus player_2 or player_1 versus AI, controlled by switch 0 on the FPGA.

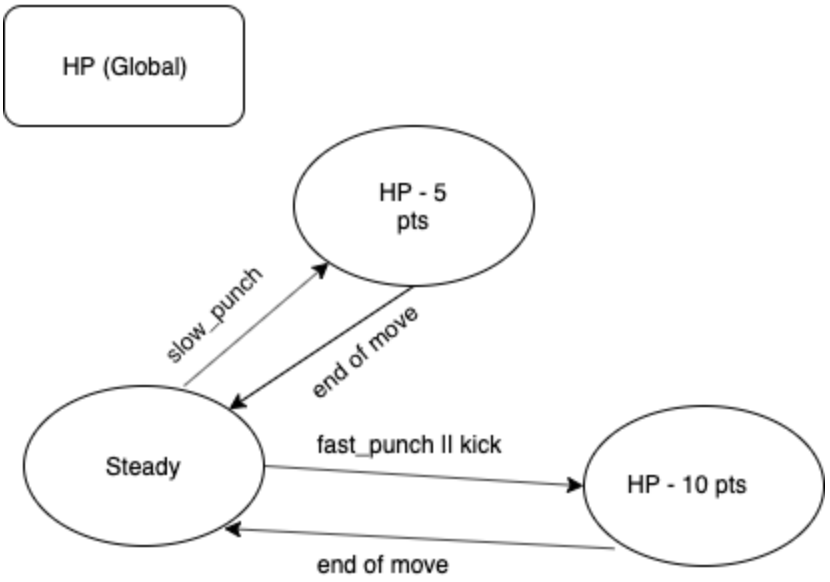
The goal of the game is to be the first to deplete the opponent's health from 100 to zero. There are two main states for the players- motion or attack. The players cannot attack while in motion, but they can be attacked if the other player is standing still. The rules for attacking are as follows: the player must be in range, the first to initiate an action gets the points, and a random number generator determines who hits in the event that both initiated actions at the same time.

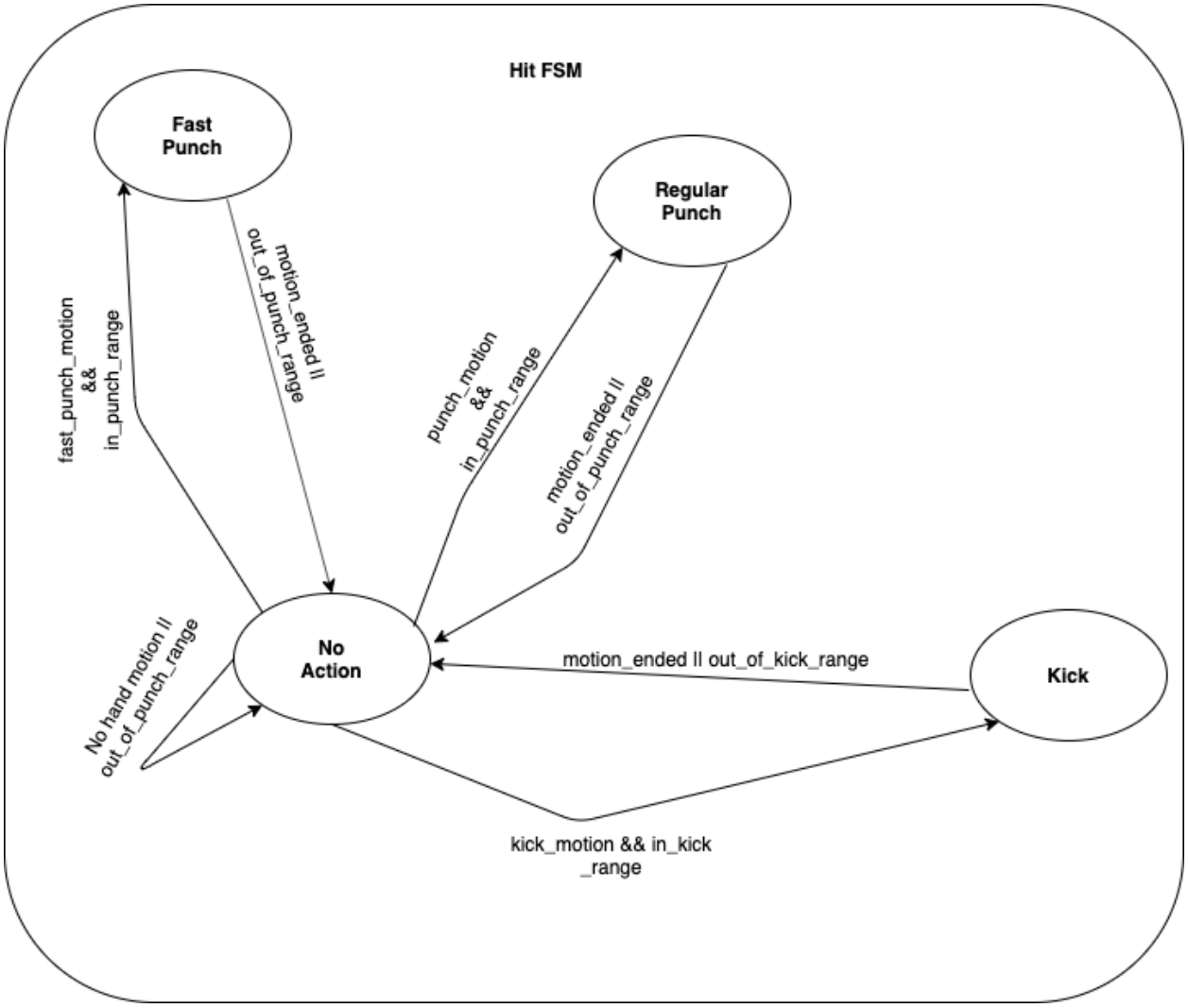


There are three forms of attack: punching, which requires the distance between the fighting sprites to be within the range of the sprites' width and the appropriate hand motion; fast punching, which requires both punch requirements be fulfilled but also meets speed criteria (the AI simply has a 5% chance of fast punch); and kicking, which requires the sprite to do the kick hand motion and be within a distance of half the sprites' width. A punch results in a damage of 5 health points, while a fast punch and a kick both result in 10 points of damage.


kick
punch
fast_punch
Hit Points
[5:0] hit_points
Health Bar
health_pixel

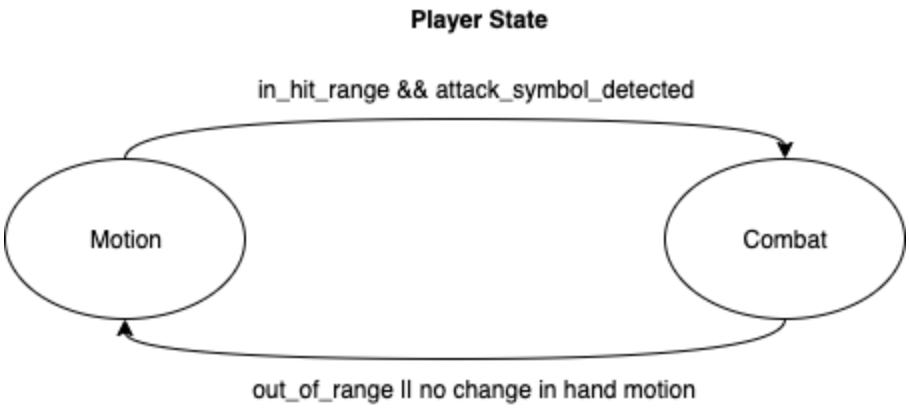
HP FSM


HP (Global)
HP - 5 pts
slow_punch
end of move
Steady
fast_punch II kick
HP - 10 pts
end of move



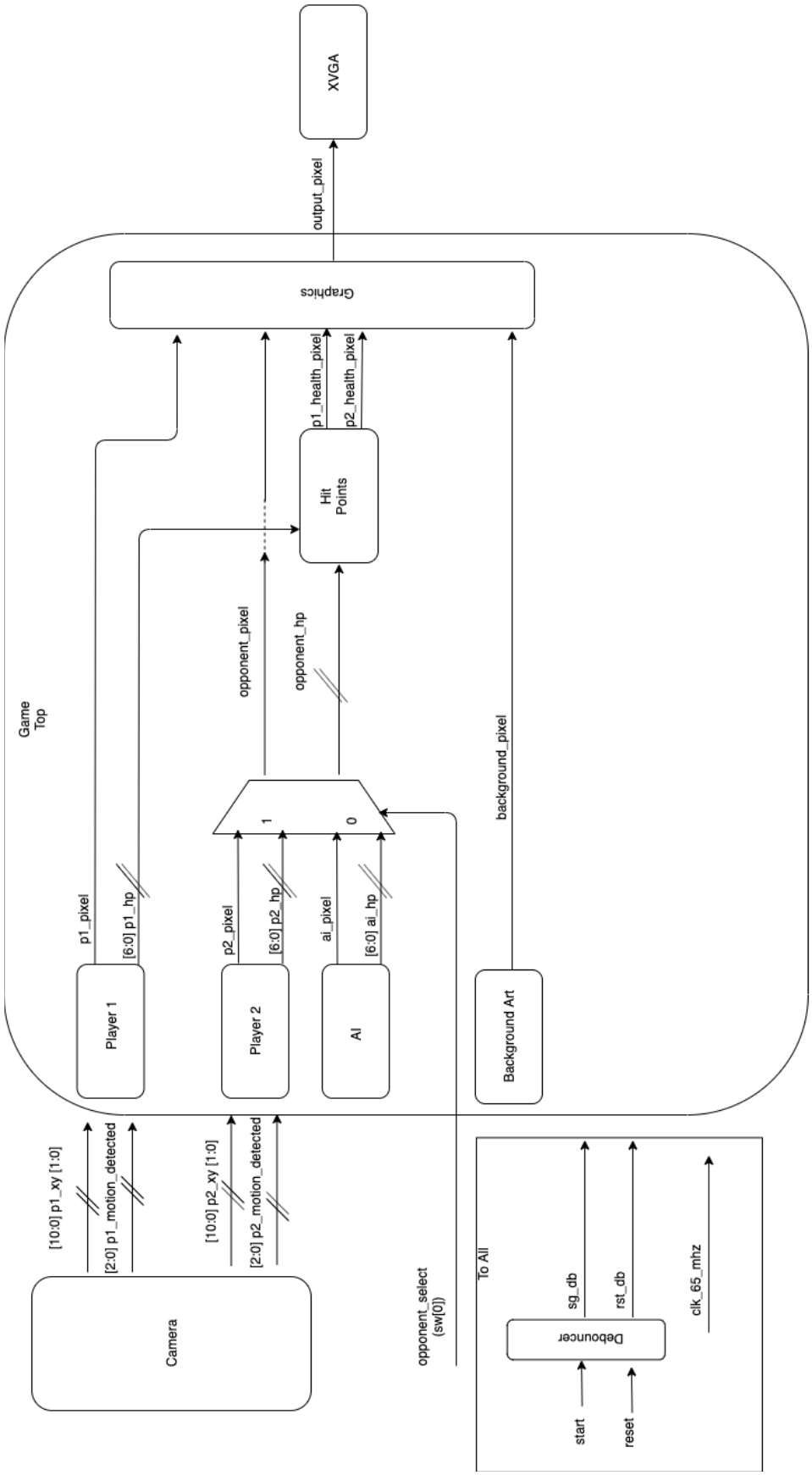
Motion is platform based, that is, sprites can only move forwards and backwards, based on hand motions. Speed is determined by switches 1 to 4, in a multiple of pixels per second.

Implementation



There are two main states in which a player can exist: motion/inaction, or hitting, which includes punch, fast punch, and kicking. These states depend on five variables: Whether or not the player is in motion, if the player is in range of punching, if the player is in the range of kicking, which hand gestures are used, and who has right of way. In punching range is defined as whether or not the opponent is the width of the player sprites in pixels apart. Kicking range is the same except the distance must be less than half the width of the player's sprite apart. The hand gestures are determined by computer vision and enumerated later. Right of way is the way hits are determined- whoever hits first gets the point, but in the event that both get the hit within a certain time range, a random number generator will determine which player gets the damage.

The AI will move forwards and backwards with the goal of maintaining the same distance between itself and the player, just out of range of punching (as is a common strategy in fencing). If it is in range it will always hit. With a 10% probability it will attempt to close the distance into fighting range. Hit detection will be based on pixel overlap. If both player sprites are not zero, we know the pixel in question is an overlap. The overlap must persist for four iterations of the 240mhz clock in order to count.


XVGA
output_pixel
Graphics
Game
Top
p1_health_pixel
p2_health_pixel
Hit
Points
opponent_pixel
opponent_hp
background_pixel
p1_pixel
[6:0] p1_hp
p2_pixel
[6:0] p2_hp
ai_pixel
[6:0] ai_hp
1
0
Player 1
Player 2
AI
Background Art
[10:0] p1_xy [1:0]
[2:0] p1_motion_detected
[10:0] p2_xy [1:0]
[2:0] p2_motion_detected
Camera
opponent_select
(sw[0])
To All
sg_db
rst_db
clk_65_mhz
Debouncer
start
reset

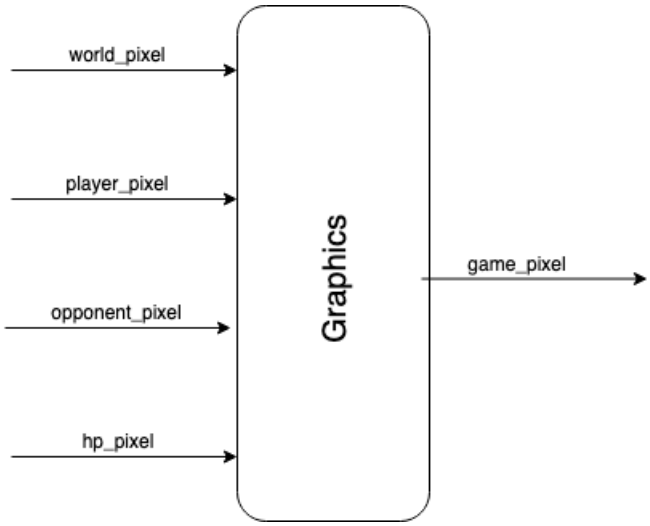
Graphics (Diana/Ray)

Display (Diana)

The display will be a 1024 x 768 pixel screen that will refresh at a rate of 60 MHz. Player sprites will move right/left at a rate of N pixels per frame, where N is controlled by the difficulty level. We will use ROMs for the sprites, as we did in lab 3. The data will be transferred over VGA.

The output of the camera will be a motion detected state- allowing a mapping between the outside world and the monitor. X and Y coordinates on the screen will be calculated based on the actions.

At the start of the game, the sprites will be at opposite ends of the screen (left edge and right edge).

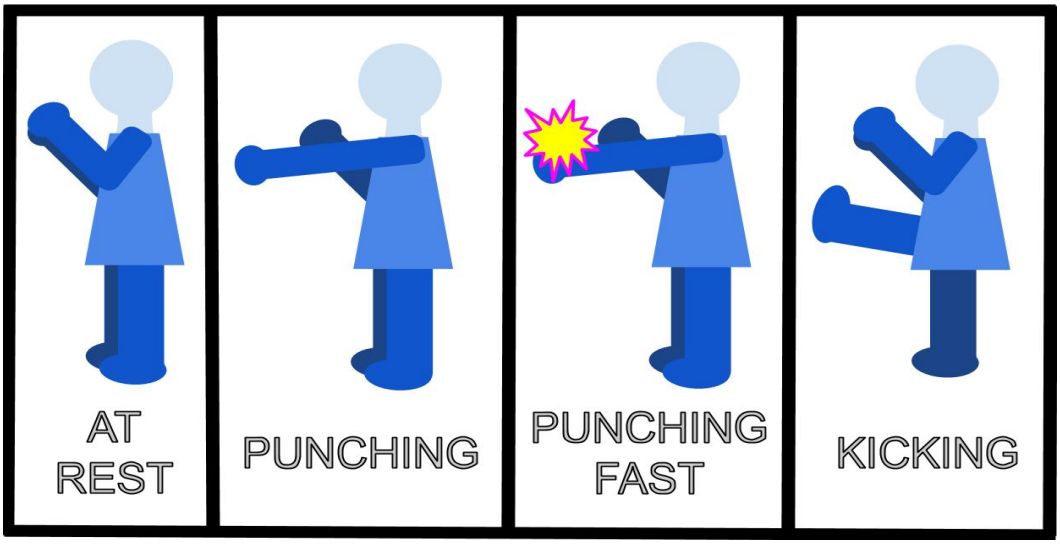


Sprites (Ray)

Player sprites are 8-bit color side-view images of size less than or equal to 64 x 64 pixels. Each sprite will have four states: (1) at rest/moving, (2) punching, (3) punching fast and (4) kicking. The sprites will punch and kick with the arm and leg that is “closest” to the screen. To differentiate between them, the two players will be different colors.

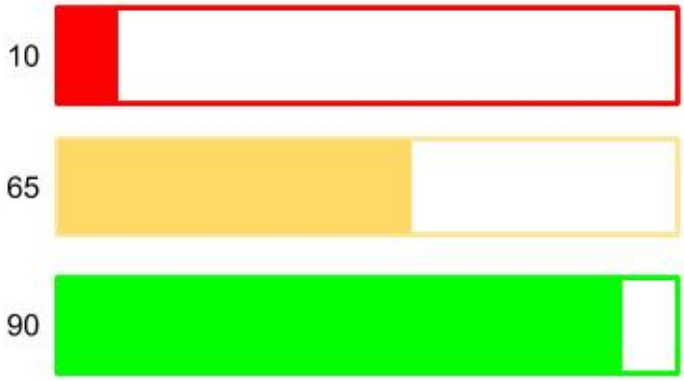
To show a punching (fast) motion, the sprite will transition from at rest/moving to punching, stay at punching (fast) for some time T, then transition back to at rest/moving. To show a kicking motion, the sprite will transition from at rest/moving to kicking, stay at kicking some time T, then transition back to at rest/moving. To show a moving motion, the sprite will stay

in the at rest/moving state and move right or left at a rate of PLAYER_SPEED pixels per second.

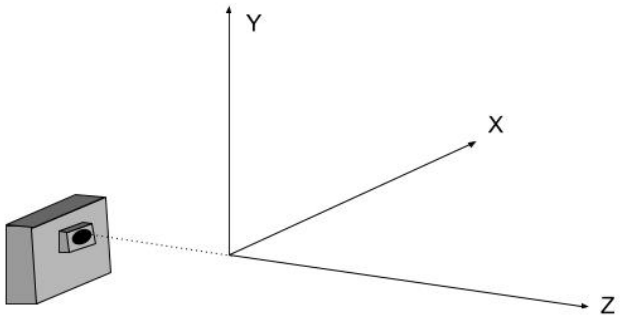


Health (Ray)

Each player will have a health bar that ranges from 0 (dead) to 100 (full life). At 100, the bar will be filled and green. At 0, the bar will be empty and outlined red.

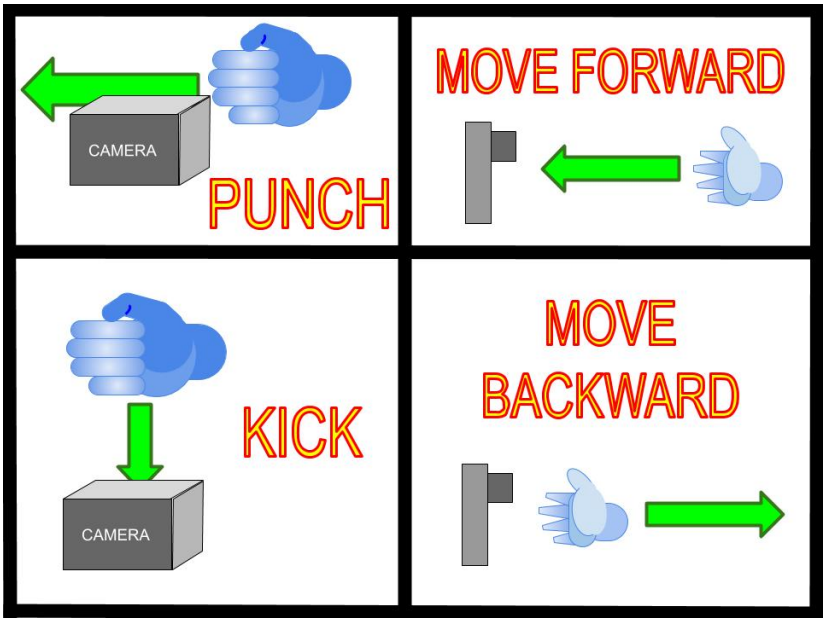


Computer Vision (Ray)



Gestures

To punch, the player will make a fist and punch in front of the camera in the x direction (right or left). To kick, the player will make a fist and move their hand in the y direction (up or down). To move forwards or backwards, the player will make an open hand and move it in the z direction (towards the camera or away from the camera). (Note: Forwards will correspond to the direction the sprite is facing (right or left), and backwards will correspond to the opposite direction.)



Motion Detection

Our setup will consist of one camera that will be positioned facing the player(s). In a player vs. player setup, both players will sit in front of the camera with the hand that they are using to gesture motions (punching, kicking, moving forward, moving backward) in front of the cameras. Each player will wear a different colored glove to differentiate between the two players.

The camera will take a video of the players' fighting hands. From each frame, the top left and bottom right points on the two hands will be extracted. The rate of change of these coordinates over a set of F consecutive frames will be used to calculate the average velocity of the players' hands and the axis and direction in which they are moving. This information will be used to classify the hand gesture as one of the following states: none, punching, fast punching, kicking, moving forwards, and moving backwards.

For instance, if the distance between the top left and bottom right points on a hand stays the same, but both are moving down, that means the hand gesture is the kicking gesture. If the distance between the top left and bottom right points on a hand is increasing, that means that hand is moving closer to the camera, which indicates that the hand gesture is the moving forward gesture.

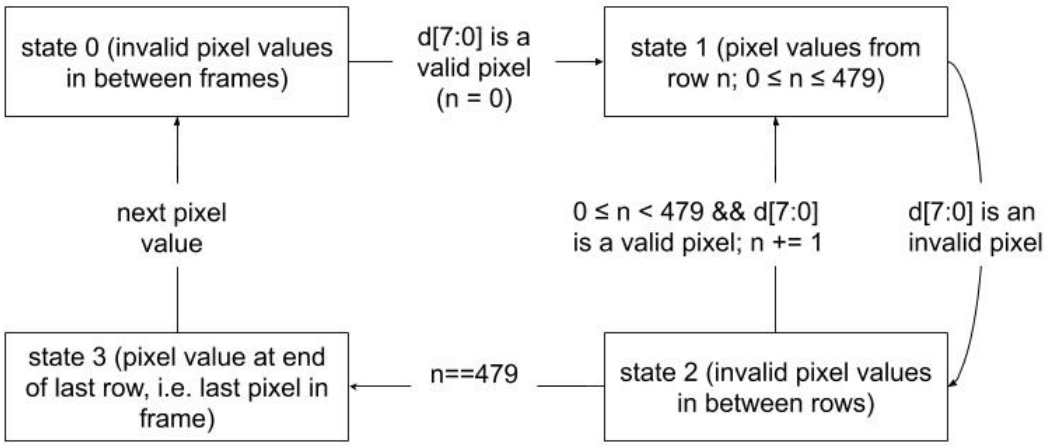
Parsing Video Data

For our project, we will be using a OV7670/OV7171 CAMERACHIP™. The OV7670/OV7171 CAMERACHIP™ is a low voltage image sensor with the functionality of a single-chip VGA camera and image processor, and can operate at up to 30 frames per second

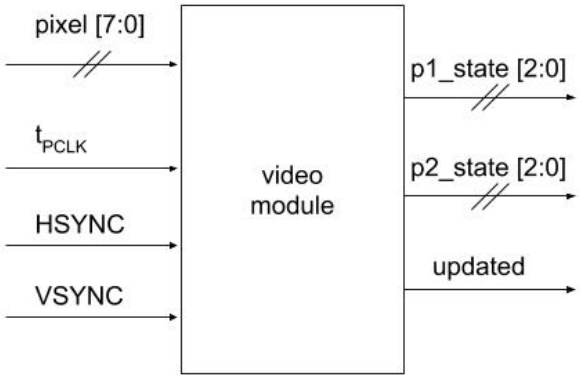
(FPS). We will be using the 8-bit color option (GRB 4:2:2). The video module will receive 8 bits at a time from the image sensor at some rate t_{PCLK}. Once the data for an entire frame has been passed through, the video module will extract the position data needed from the frame and store it.

(Note: The following description will explain what to do in the case that the game is in Player v. Player mode. In the case of Player v. AI mode, the module will do the same operations, but with only one hand.) After position data has been extracted from some number N of frames, the module will classify the hand gestures from the video data. The module will then output the updated states of the two sprites, where 0=None, 1=Punching, 2=Punching Quickly, 3=Kicking, 4=Moving Forwards, and 5=Moving Backwards. On the first clock cycle after the sprites' states have been updated, an output variable called "updated" will be set to true to indicate that the sprites' states have been updated.

Video Module FSM



Video Module Diagram



Clock Rates

We will use three different clocks in this project: the video clock rate, the display clock rate, and the action clock rate. The video clock rate will be the rate at which the FPGA receives frames of the video from the camera. The display clock rate (65 MHz) will be the rate at which the game module receives updates regarding the state of the sprites, as well as the rate at which the display screen is updated. The action clock rate (1 Hz) will be the rate at which the sprites' actions last.

Player/Game Communication (PJ)

In our design, we will be using computer vision to track the movements of our player(s) and incorporating haptic feedback and sound outputs into our game logic. We intend to use colorful gloves that will allow us to map the motions and appearance of our players hands, as well as a belt that uses vibration to tell the player(s) when they take damage. We will use the output from our motion detection module to generate sounds which the player will hear during gameplay.

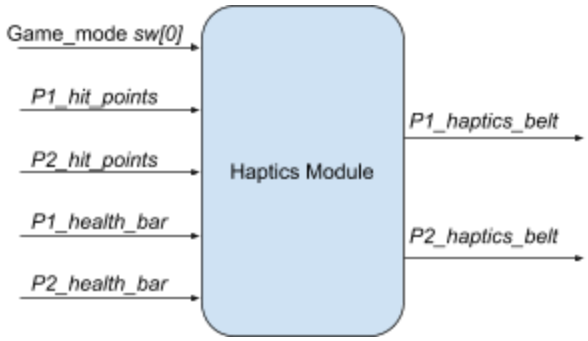
Player Input

The only input to our system should be video output from one camera. This will track the motion of our player(s) using colorful gloves which we will use to differentiate between players. Inside of our system, we can run the output from the camera into a module that will track the two dimensional location of the gloves. Since objects will appear larger when they are closer, we can track forward and backward movement by accounting for the object getting larger. Through this we can then track the gestures described in the computer vision section in 3 dimensions.

Haptics

Our players will wear vibration response belts during gameplay. This belt will incorporate linear resonant actuators (LRA) type vibrating motors, specifically either G1040003D or G1040001D. Using this type of vibrator, we will also be able to vary the intensity of the vibration, depending on the extent of damage dealt to the player (low vibration for low damage, highest vibration for highest damage) by varying the frequency input to the vibrating motor. The highest intensity of vibration will result from the resonant frequency input to the motor, and the other intensities will be chosen from the frequency characteristics graph of the motor. When the player dies/loses, a customized death pulse will execute only to the dying player as the game ends.

The inputs will be the game mode (single player/pvp), hit points applied to each player, and the health bar status of each player.

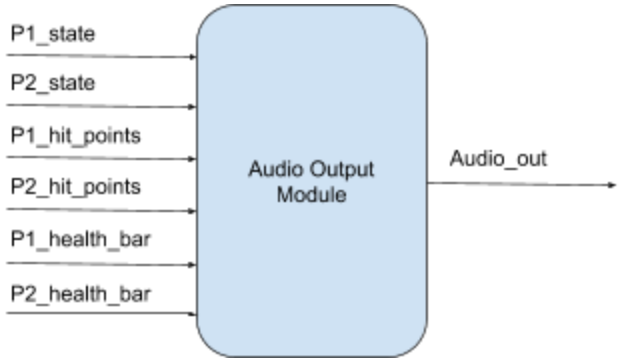


Sounds

We also want to have some sounds playing during our game. We intend to have sounds for the beginning of the game, the different types of attack (punch, fast punch, and kick), for movement, and for the different intensities of damage, of which we will have 3: small damage, medium damage, high damage. Like our haptic feedback, we will also have special sounds for when the game ends, such as a knock out noise and then a game over noise.

For this module, the inputs will be from the motion detection, which will indicate what type of attack or motion happened, and a feedback input, which will come from the damage module and make noise if a player is hit.

To generate these noises, we use a simple speaker, like the ones provided to us in lab 4.



Testing (All)

Logic

To test our code, we will have waveforms for punching slowly, punching quickly, and kicking. We will use the waveforms determine the motion, and set the state based on this. For debugging, we will output our motion parameters to the seven segment display. Each module will have its own testbench and expected behavior on the fpga and on the monitor display.