Daniel Southern
Rachel Bainbridge
October 30th, 2007
6.111 Project Proposal

Optical Input Targeting Game

Our Duck Hunt game will function similarly to the Duck Hunt of the original NES. A laser aimed at a computer screen will function as the "gun" and a camera will be used to locate this laser and a red dot will be displayed by the game at its location. The player will also be able to fire the gun by pressing the enter button on the lab kit. The computer screen will display a gaming environment similar to that of the original Duck Hunt. Animated ducks will fly across the screen one at a time and the score will also be displayed in the upper left hand corner of the screen.

Our implementation of the Duck Hunt game consists of two major parts: the video output to the screen which will be handled by Rachel Bainbridge and the input from the camera that locates the laser pointer which will be handled by Dan Southern.

Video Output Module – Rachel Bainbridge

There are three modules planned for the video output, thought more may be added to deal with frame buffering. Sprites from the original Duck Hunt game will be used for the background and for the animated duck. They will be stored on a ROM and read off through the indexer module, which will allow the Drawer to reference a sprite simply as "tree" or "duck2." the indexer will interpret such statements into physical memory addresses and read out the correct pixel values. For instance in Duck0 were to be stored in the first ten lines of memory, the indexer would know that Duck1 began on the 11th line of memory and be able to find it there.

Drawer Module – *Draw background duck and cursor*

The two main modules are the Drawer and the Duck module. The drawer will take an x and y input, two 10-bit values, from the camera and draw a small red blob at those coordinates. It also takes a control signal from the camera logic to help with callibration. Depending on the signal, the screen may flash horizontal or vertical lines, or a border around the screen. Lastly, it takes an input from the enter button that tells the module whether a shot has been fired. If that signal goes high, the drawer will compare the duck coordinates to the laser coordinates to determine whether the duck has been hit and send a signal to the duck module. The drawer module is also in charge of creating the graphics of screen. It will ask the indexer for the sprites it needs and draw them in the correct locations. There will be a bunch of static sprites used for the background and an animated sprite for the duck. The pixels of the background will be or'ed together, but muxes

will be used to make sure the duck is displayed on top of background elements. The duck module will send the players score to the drawer and that will also be displayed on the screen in the form no. duck hit / no. ducks released. The output to the screen will be in HSV.

Duck Module – *manage score and duck behavior*

The duck module will handle the movement of the duck and the calculation of the score. The duck will like the puck in the pong game, except that it will be allowed to bounce out of the top of the screen. The player can change the speed of the duck to increase or decrease difficulty. If a duck is hit, it will fall vertically to the ground, the player's score will be increased, and a new duck will be started. A new duck will also be started if the duck flies beyond the top of the screen. The duck module will also tell the drawer what type of duck needs to be drawn. The duck sprite will change every few frames to give the illusion that it is flapping its wings.

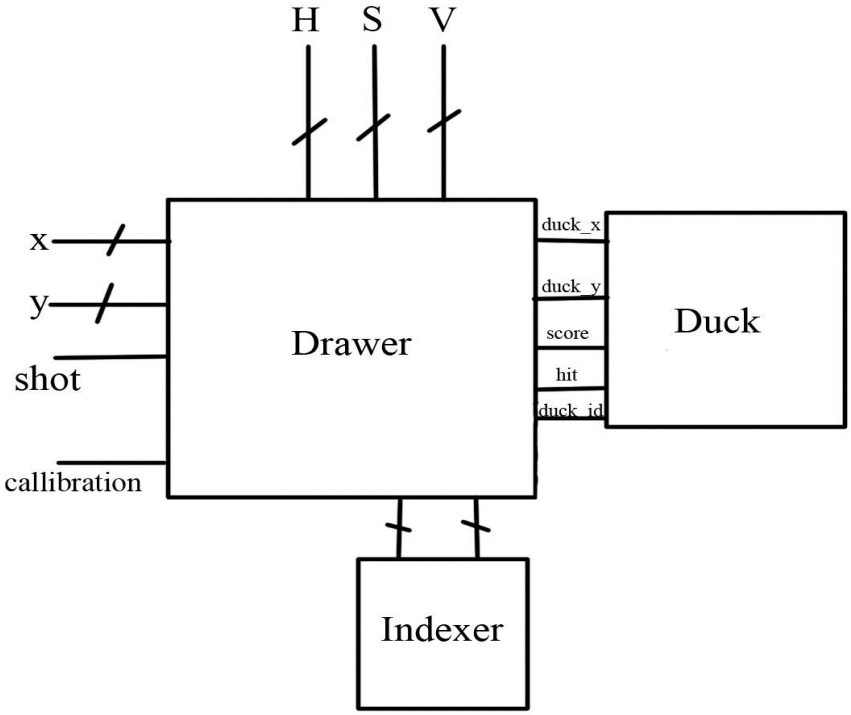

Figure 1. Video Output Module Block Diagram

Testing

Each module will be tested individually and then together. Since the duck is similar to the pong game, code can be reused and is almost guaranteed to function correctly. Testing will most likely begin with getting the background to display correctly and have the duck moving around. Switches can be used to turn on and off different parts to see if something is functioning correctly independent of other modules. In the end, the input and the output will need to be tested together.

Video Input Processing Module – Daniel Southern

The project's other primary component will be a module to process incoming video information through the labkit's ADV7185 NTSC decoder. The module will scan through the incoming video in real time searching for the “cursor” in the form of a dot of red light (supplied by aiming a red laser pointer at an object in the camera's field of view). The module will identify the location of the dot by examining the relative luminosity in a range of chrominance of pixels to find a spot of reddish light that is brighter than the pixels surrounding it.

List of Modules Required:

ADV7185 To Frame Buffer Interface – *Module to write output of NTSC decoder into frame buffer*

- Input – pixels out of the NTSC decoder (8-bit Y, 8-bit Cb, 8-bit Cr)
- Output – frame buffer write port memory control signals (write address, write enable)
- Output – pixel information (8-bit Y, 8-bit Cb, 8-bit Cr)

This module will capture the pixel output information from the ADV7185 and write it into a frame buffer memory. This module will need to operate at the speed of the incoming NTSC signal, which is 27 Mhz. The module will determine which position of the frame buffer to write the data into based on the position data which is also generated by the ADV7185.

When the ADV7185 signals that there is data ready to be read, this module will generate the necessary control signals for writing into the frame buffer, perform the write, then signal to the ADV7185 that the pixel information has been read.

Pixel Analyzer – *Module to scan through the frame buffer, find position of the cursor*

This module will read through the frame buffer pixel by pixel performing a search for the “cursor” described above. Operations on each pixel will include a conversion in the HSV color space to simplify color identification and relative luminosity comparisons. The amount of computation necessary for each pixel force this process to run at a speed slower than 27Mhz. However, there will be not be significant performance degradation even at much slower speeds since there are segments of the NTSC signal which do not contain active pixel information, etc.

Even if the processing turns out to be computationally intensive such that we cannot completely analyze each frame before it is being replaced by the next image, it shouldn't affect our cursor location algorithm since the location of the cursor will be fairly continuous over $1/60^{th}$ of a second intervals.

Decoupling the processes of writing incoming camera data into a frame buffer and analyzing the data largely eliminates timing concerns with our pixel

processing circuitry since data can be read from and written into the frame buffer at different rates.

Submodules

YCbYcr to HSV Converter – *Reads a pixel out of the frame buffer on each cycle and converts it to an HSV pixel*

- Input – frame buffer read port data (1 Pixel – 8-bit Y, 8-bit Cb, 8-bit Cr)
- Output – frame buffer read port control signals (read address)
- Output – (x, y) coordinate of the cursor (two 10-bit numbers)

This Module is responsible for generating control signals for the frame buffer to read the pixels out in the correct order (i.e. iterating across each row and column). The value will be converted to the HSV color space producing 8-bit values for H, S, and V normalized to the range 0 - 255.

Center of Mass Calculator – *Filters out pixels according to their HSV values and a preprogrammed set of rules, finds center of mass of matching pixels*

- Input – pixel in HSV format (8-bits for each of H, S, V)
- Output – Center of mass coordinate (two 10-bit numbers)

This module will look at HSV values, filtering for sets of coordinates corresponding to pixels matching these criteria. We will compute the center of mass of this set of values, and produce a new (x, y) coordinate after each pass through the frame buffer.

Input Coordinate to Output Coordinate Converter – *Takes the coordinate produced by the weighted average of matching pixels and and produces the corresponding coordinate in the output image.*

- Input – (x, y) coordinate of the cursor (two 10-bit numbers)
- Input – Calibration Parameters
- Output – calibrated (x', y') position of the cursor (two 10-bit numbers)

This module will perform some transformation on the (x, y) coordinates in order to map the input image space onto the output image space. Displaying a dot on the screen in the new coordinates (x', y') should theoretically display the dot immediately underneath the red dot of light. If this is not the case, then the calibration parameters will need to be adjusted – either manually or through some automatic routine programmed into the system, depending on the level of complexity we are able to include.

Other Considerations

One fundamental element of this design is the mapping of positions in the camera input space to position in the NTSC output space. There are several degrees of freedom which could be taken into account, such as

- Scaling – even in if the camera was perfectly aligned with our video output source, we would still have the issue of differences in size between a single pixel in the input image and a pixel in the output image. We plan to operate

with the output image being located entirely inside the input image to ensure that we can capture the entire screen. We also, for example, plan to use calibration patterns such as a rectangle around the edge of the screen which will require that the entire output image be visible in the input image.

- Skew, Both Horizontal and Vertical – If the plane of the image received by the camera is not parallel to the plane of the screen, then we will observe some combination of two effects: the top of the screen will appear larger/smaller than the bottom of the screen in the input image, and the left side of the screen will appear larger/smaller than the right side.
- Rotation – If the camera is not rotationally aligned with the screen, (i.e. the edges of the image in the input signal are not perfectly horizontal or vertical), then we will have to compensate for some rotation. This operation can be characterized by a rotation around the center of the image

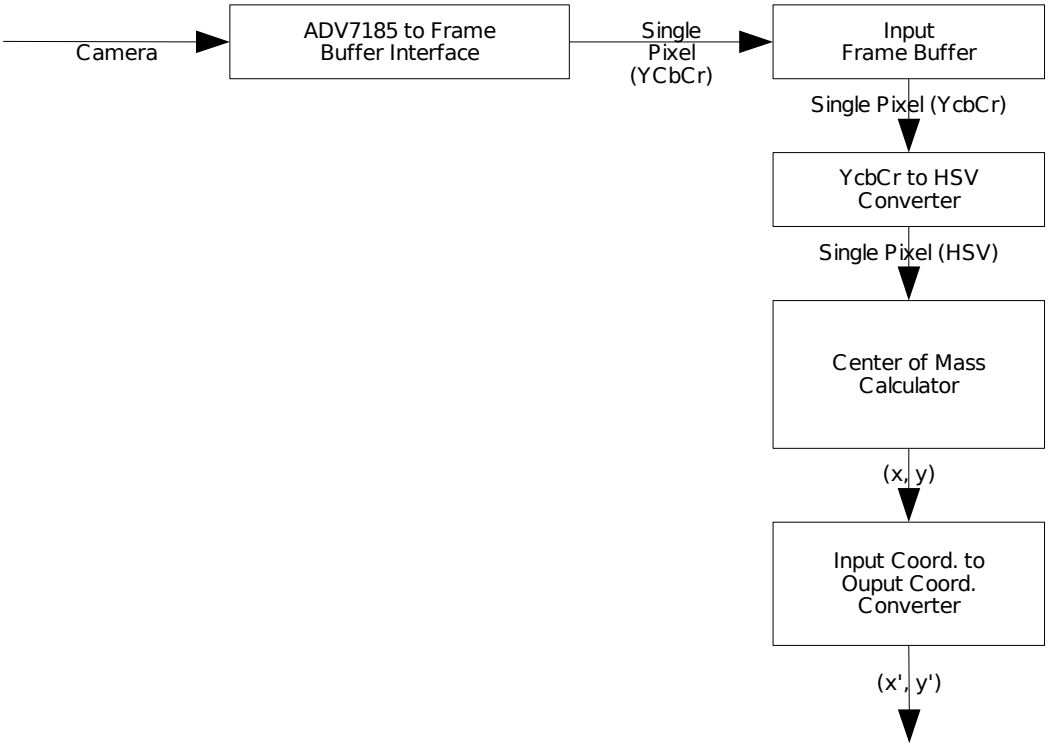

Figure 2. Video Input Processing Module Block Diagram

Testing

I plan to develop each module separately according the design specification, testing along the way to ensure that each component works as required. I plan to test the HSV module in ModelSim to ensure that all mathematical formulas produce the correct results in all situations. I also plan to test the Center of Mass Calculator Module in ModelSim, as its numerical results will be easy to verify in this way. I will test the frame buffer and the two frame buffer interfaces by capturing video from the NTSC, storing it in the frame buffer, then reading from the frame buffer to display it out on the VGA. I will then add each module in succession to the complete system and ensure that at each step we have the desired functionality and there are no new incompatibilities between the modules interfaces.