Interactive Adventure Game

Greg Luthman

Akash Shah

Description

Our generation was the first to grow up with video games as a large part of our entertainment. Perhaps the most popular of these games was Super Mario Brothers, a side scroll adventure game for the Nintendo Entertainment System. (NES) As children we could become enveloped in the game world and imagine that we were the main character on screen fighting to save the princess from terrible monsters. With our 6.111 final project, we hope to do just that, put the player into the game world. We will create a live action, side scroll adventure game that is similar in concept to Super Mario Brothers, except that instead of playing with a controller and seeing a character move on screen, everything is controlled by the player's actions in front of a camera. We will not only use the video of the player to determine the proper commands to send to the game, but also place that video of the player into the game world. The player will be able to duck objects, jump over objects, move forward or backward in the game world. Other commands are a possibility as we continue designing this project. Akash Shah will develop the input and output modules dealing with video and Greg Luthman will develop the game logic. If there is enough time, we will try to implement sound in the game.

Module Breakdown

Game Module

The game module takes the various control signals from the video portion of the project and uses them to figure out how the player is interacting with the game world. The game module also takes a single bit signal from the video modules that is high if the player occupies the pixel indicated by the current vcount and hcount. Using this, the game module can tell if the player has contacted any of the other elements in the game, and can respond accordingly. There are several submodules that are used to support a main game finite state machine (FSM). These smaller modules includes character sprites, physics engines, timers, and level memories. The game module outputs several signals back to the video modules. It outputs a data bus corresponding to what the game world is to look like without the player video. This is accomplished by changing the color of the pixel to be displayed as the hcount and vcount signals cycle through the entire screen. The game module outputs two scale factor signals, one for the x direction and one for the y direction. This is to tell the video modules how to scale the video of the player to fit appropriately in the game world. The last signals that the game module outputs to the video modules are two signals telling the video modules where to place the video of the player within the game world. This is done by outputting the desired x and y coordinates of the top left corner of the video. These last signals will enable the player to see “himself” jump through the air, or do other special

things based on the game world. The game world will probably be implemented as a one-way world. This means that once an object is off the left side of the screen, the player cannot go back to that part of the level again, but only up to the edge of the screen. This should make the implementation of the game world less memory intensive, because the module doesn't have to store the state of all objects that have already been interacted with.

Game FSM

The game FSM will be the brains of the game module. It's main inputs are the commands from the video modules and the player_pixel_on input, also from the video modules. Other inputs will be the character generation pixels from the character sprite modules, timers to keep track of in game events, and an interface with a memory that contains the layout of the level. The levels will probably be created by storing only the coordinates of the desired characters, and then generating those characters on demand. Registers are used to keep track of the position of the player within the game world, as well as the position of other objects in the game world. Each object will be assigned an object number, so storing and detecting collisions and determining whether or not special actions can happen, should be much easier.

Game World

The game world should be familiar to anybody that has played a side scroll adventure game similar to Super Mario Brothers. There will be enemies that walk slowly across the screen, and to defeat these enemies the player must jump on them. There will also be inanimate objects, both in the background that cannot be interacted with, as well as objects in the foreground that the player must jump on top of and over. When the player jumps, the video representation of the player will move through the “air” in a path determined by some simple physics engine. There will be objects that will affect the player's state, such as coins that can be collected for an extra life, mushrooms (or another object) that will cause the player to “grow” bigger, and possibly objects such stars or capes that give the player temporary abilities such as invincibility and flight. A player defeats the level by making it to the end of the level's map within a certain amount of time. A player loses if he/she loses all of his/her lives. When a player contacts an enemy without jumping on it, one of two things happens. If the player is “big” then the player reverts to a “small” state. If the player is “small” then the player loses a life and the level restarts.

If there is enough time, sound and multiple levels will be implemented, possibly levels with different themes and different physics. (ie. A dungeon level, or an underwater level.) There will be two types of sounds, sound effects and a sound track. The track will be playing in the background and will correspond to the mood of the current level. The sound effects will play at the appropriate actions of the player, such as jumping, squashing an enemy, or busting a block.

Video in

The video input from the camera will be mostly handled by the hardware that comes bundled with the labkit. The labkit's ADV7185 should be able to handle the video input that would be necessary for this project. Furthermore, the sample code from the website should help in this task as well.

Green Screen Detector

This module is primarily responsible for handling detection of the green screen background that will be used in this project. This detection will allow for an overlay that replaces the background color with pixel coloring of choice during the output stage. The calibration signal will be used to allow the module to determine initially the color of the background that is being input from the camera. This will allow for handling of variations in intensity and color for the background itself and allow for variations in these parameters caused by the camera itself. (In principle, this should work with any color background). Furthermore, the module will allow for variations in color that occur by allowing for a slight tolerance.

Character

This module will utilize the green screen detection module in combination with the incoming video feed to communicate to the game sub-circuit the location of the character on the captured image. This module will also allow for scaling and translation of the character image from that which is input from the camera.

Gesture Recognition

This module will allow for recognition of simple gestures that the player may choose to execute. The module will use a block memory to store old stances and gestures of the player in an effort to deduce gestures based on comparison to a set of known or previous states. The module will then set a group of control signals indicating specific movements have been enacted by the user.

Overlay Module

Responsible for overlaying the video feed with the game world and generating a proper output based on hcount and vcount signals.

VGA output

Will output video signal to the VGA hardware for a crowd-pleasing display.

Testing and Debugging

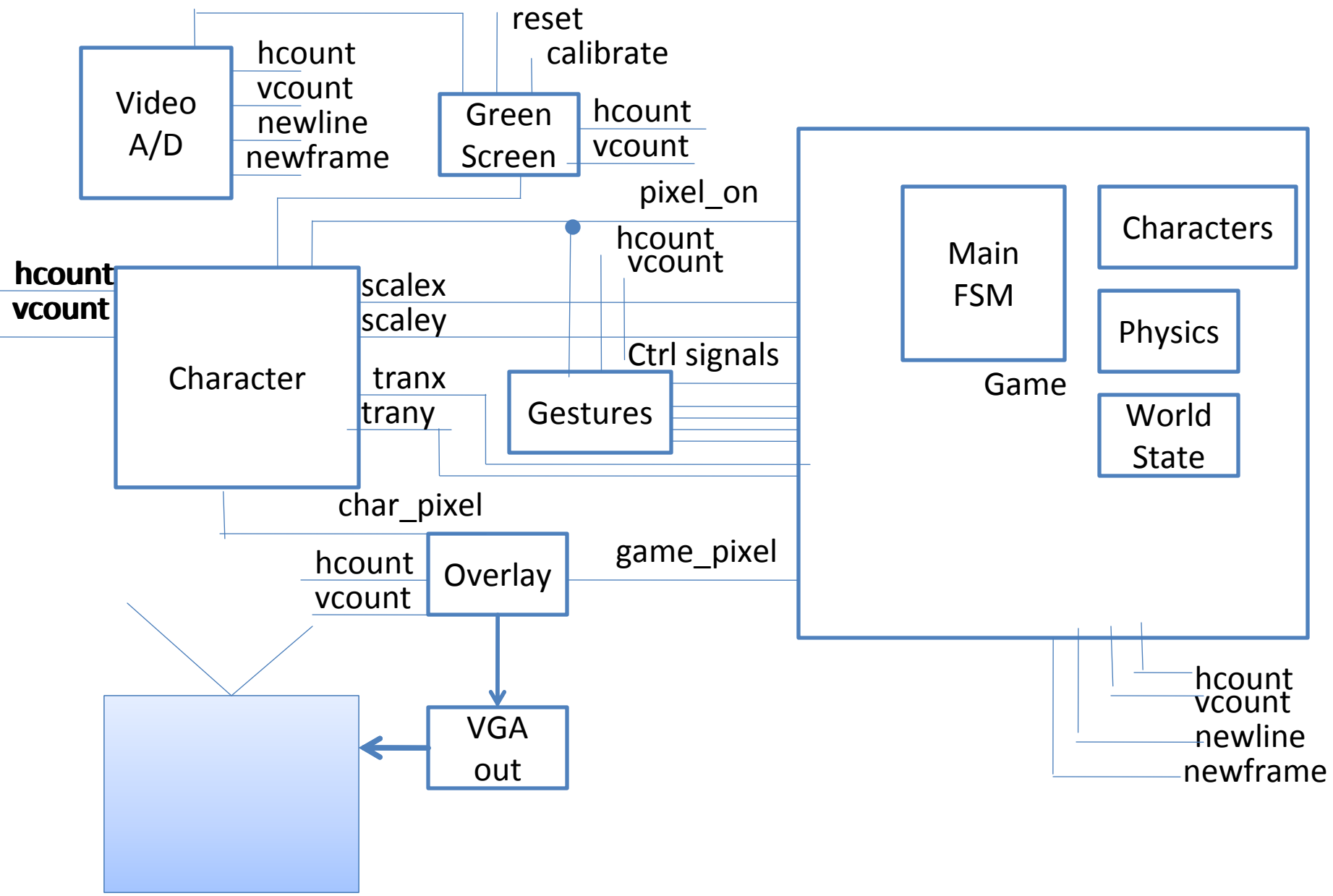
Testing of Video Sub-circuit

Testing and debugging of the video subcircuit will occur in stages. Functionality will be tested in a piecewise/unit test fashion initially, making sure each module works individually. Testing using simulation or a logic analyzer will likely be insufficient as much of the functionality relies on video/camera input to the FPGA.

Thus, as each module is completed, more complicated tests will be performed by programming the Labkit and evaluating the performance of the modules. Once this works to our satisfaction, the video circuitry will be combined together, independent of the game circuitry, and further testing will then be done. During this phase, extreme scenarios will be tested as well to ensure the proper functionality of the video processing circuitry. Once such testing is completed to satisfaction, the video and game circuitry will be connected in an effort to test the system as a whole.

Testing of the Game Sub-circuit

The game will be implemented in a graduated fashion. The first version will use the output of a blob sprite as the image of the player, and will use inputs from the pushbuttons of the labkit to move the player around a blank screen. Slowly different classes of objects will be added at increasing levels of graphical sophistication. The game will start out with blobs of different colors representing all the different objects, but by the end of the project, it should be possible to use the memories to store relatively complicated objects or even animations. The output of the game will be added with the blob representing the player to show a simple game world to start with, but as the video modules are created and the game module progresses, the game should grow to a fairly sophisticated game.


Video A/D
hcount
vcount
newline
newframe
reset
calibrate
Green Screen
hcount
vcount
pixel_on
hcount
vcount
hcount
vcount
Character
scalex
scaley
Ctrl signals
tranx
trany
Gestures
Main FSM
Characters
Physics
Game
World State
char_pixel
hcount
vcount
Overlay
game_pixel
VGA out
hcount
vcount
newline
newframe