6.111 Final Project – Project Proposal

Eric Grebing and Erik Stafl

Our implementation of SkiFree aims to integrate a wireless controller with the traditional Windows PC game. There are a number of blocks that will provide functionality, but the project can be broken into two main parts – the controller and the game itself. The controller is simply a T-shaped bright orange block that the user moves to control the motion of the skier. A camera tracks the movement of the block and the image is processed to show changes in angular and lateral position. The data from this controller is then fed into the game and changes the skier sprite on-screen. The game is implemented using logic on the FPGA and has the same general gameplay of the original Windows application.

The skier is controlled using an image-based system. The input to this part of the system is a single camera that takes data about the block in the user's hand. The images from the camera are sampled at a 60 Hz rate and passed into a frame buffer for analysis. Angular movement is determined by the user's movement of the controller in a circle. The angle is determined by the angle of the base of the "T" in relation to a line perpendicular to the ground. The lateral movement is determined by the deviation of the controller from a pre-determined center location.

The user is able to track their movements through on screen feedback. A small in-game window provides a feed of the user's hand and controller from the camera. This screen shows the user how far off-center his or her movement has become and allows for recalibration. This image can also be viewed on full-screen by flipping a switch on the labkit.

The image processing block takes in images from the frame buffer and isolates the pixels corresponding to the orange controller using filtering. Image processing also involves filtering of the image to reduce noise. The image processing takes the angular and lateral positions of the controller into account and calculates values for these positions. This data then passes into the game to actually control the motion of the skier.

The control inputs are used to control the skier in a 2D environment inspired by the early Windows SkiFree game. The background is white to resemble a snow covered mountain. All other images are created using sprites. The skier in motion is implemented using eight different sprites – one corresponding to each orientation of the skier. If the head-on view is thought of as the skier's position at 0 degrees, there will be sprites representing the skier at -90, -67.5, -45, -22.5, 0, 22.5, 45, 67.5, and 90 degrees. Fortunately, one sprite can be saved by duplicating the -90 degree orientation and flipping the image 180 degrees. The skier is also characterized by other sprites designated for special events. These events include jumps and crashes that come in response to other objects in the skier's environment.

In addition to the skier, other sprites populate the skiing track. Three different types of trees are implemented using three separate sprites. Different types of hills are also implemented using sprites. Each of these other sprites that occupy the map affect the skier's movement. The trees, for example, serve as obstacles for the skier on his path. If the skier runs into a tree for example, the system detects a crash and changes into a crash sprite. The hills, on the other hand, will either impair the skier's path or allow him to jump by asserting the jump control.

The flow of the game takes place from top to bottom on the screen. The skier sprite is relatively static in the middle of the screen. The other sprites move with an upward velocity to provide the illusion that the skier is moving downward. Throughout the skier's path, other sprites, such as hills and trees, serve as obstacles that make the skier deviate from a straight path down the hill.

Simple physics must be implemented in this game. Since the skier moves downhill, we assume acceleration due to gravity. Thus, if the longer the skier moves down the hill, the more his velocity increases.

The gameplay requires a number of logic statements to deal with different situations for the sprites on screen. At any point when the skier is in motion, the "landscape" sprites, including the trees and hills, will be moving upward on screen in accordance with the velocity variable.

The game also implements a timer that provides the skier's time from top to bottom down the hill. This timer is formed using a simple division of the system clock. This time is displayed in the top corner of the screen and enhances game play. In addition, the skier's velocity is displayed using the same module.

Beyond simple functionality, the game may be expanded to include audio components. One possible addition involves the use of voice commands to make the skier make additional movements. Another possible extension is the use of sound effects for crashes and other events in the game.

This project will require us to successfully obtain data from a camera, isolate the pixels of the T controller and provide the system with information about the controller's position. This input will then be fed into the game interface to control the motion of the skier. One challenge ahead will involve us dealing with a color image from the camera, isolating specific pixels, and eliminating noise. We will be doing a fair amount of image processing research to learn how to deal with these issues.

The clean input from the controller will then be fed into the game to create a playable version of SkiFree. While basic implementation does not appear to be extremely difficult, replicating parts of the game such as jumping and other features will prove to be a challenge. In all, this project will allow us to develop our image processing and video logic skills.

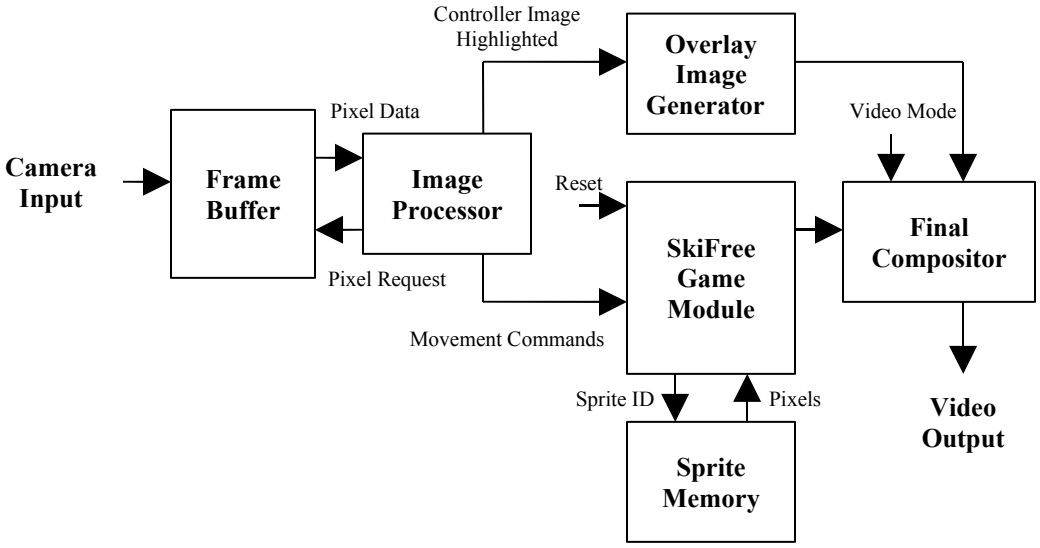
Module Overview

We plan to implement the project using a set of modules similar to what is described here. Any additional functions, such as audio input or output would require more modules, but those are not listed as they are not part of the core project.

Most modules listed can be individually tested by simply wiring its inputs and outputs to controlled sources or to the screen directly. For modules that do not output video data, their performance can be tested in Modelsim or on the FPGA using the logic analyzer.

Finally, many of these modules are actually comprised of a small number of sub-modules, but those details have not been determined yet. Once more implementation level design work has been finished, the details will be incorporated into the overall module listing and specification.

High Level Block Diagram



1. **Frame Buffer**
   Inputs: S-Video Data decoded from Camera.
   Outputs: Pixels requested by the Image Processor
   The frame buffer is simply a memory large enough to store a single frame from the video source, so it can be processed.

2. **Image Processor**
   Inputs: Pixels from the Frame Buffer
   Outputs: Movement commands to the SkiFree game module, pixel request to the frame buffer, and highlighted controller pixel data.
   The image processor uses color to determine the pixels that represent the position of the colored "T" block. It then determines the position and orientation of block itself from these pixels. Finally, the module then uses this position and orientation to output movement commands to the SkiFree game module.

3. **Overlay Image Generator**
   Inputs: Pixels data from the image processor
   Outputs: Pixel data from the image processor with overlays
   This module simply takes the highlighted pixel data indicating the controller's position and draws a circle on top of it to indicate where the center of the screen is.

4. **SkiFree Game Module**
   Inputs: Movement Commands, Reset
   Outputs: Pixel data for the game itself.
   The game module takes care of all of the major gameplay aspects of the project, including physics, object placement and collision detection. It simply takes movement commands from the image processor to update the state of the skier, and a reset signal to restart the game.

5. **Sprite Memory**
   Inputs: Sprite ID
   Outputs: Pixel data for the sprite
   The sprite memory is a simple ROM that stores the image data for each sprite. It is used by the game module to reference the pixel data needed to display each image on the screen.

6. **Final Compositor**
   Inputs: Pixel Data from Overlay Generator and Game Module, and Options flags.
   Outputs: Pixel data for the entire project to be displayed on screen.
   The final compositor simply uses the options flags to determine which video mode the user is playing in, and it will output the correct pixel data according to that. The options may include full screen game, full screen view of camera image, full screen view of enhanced camera data, or full screen game with small overlay of enhanced camera image.