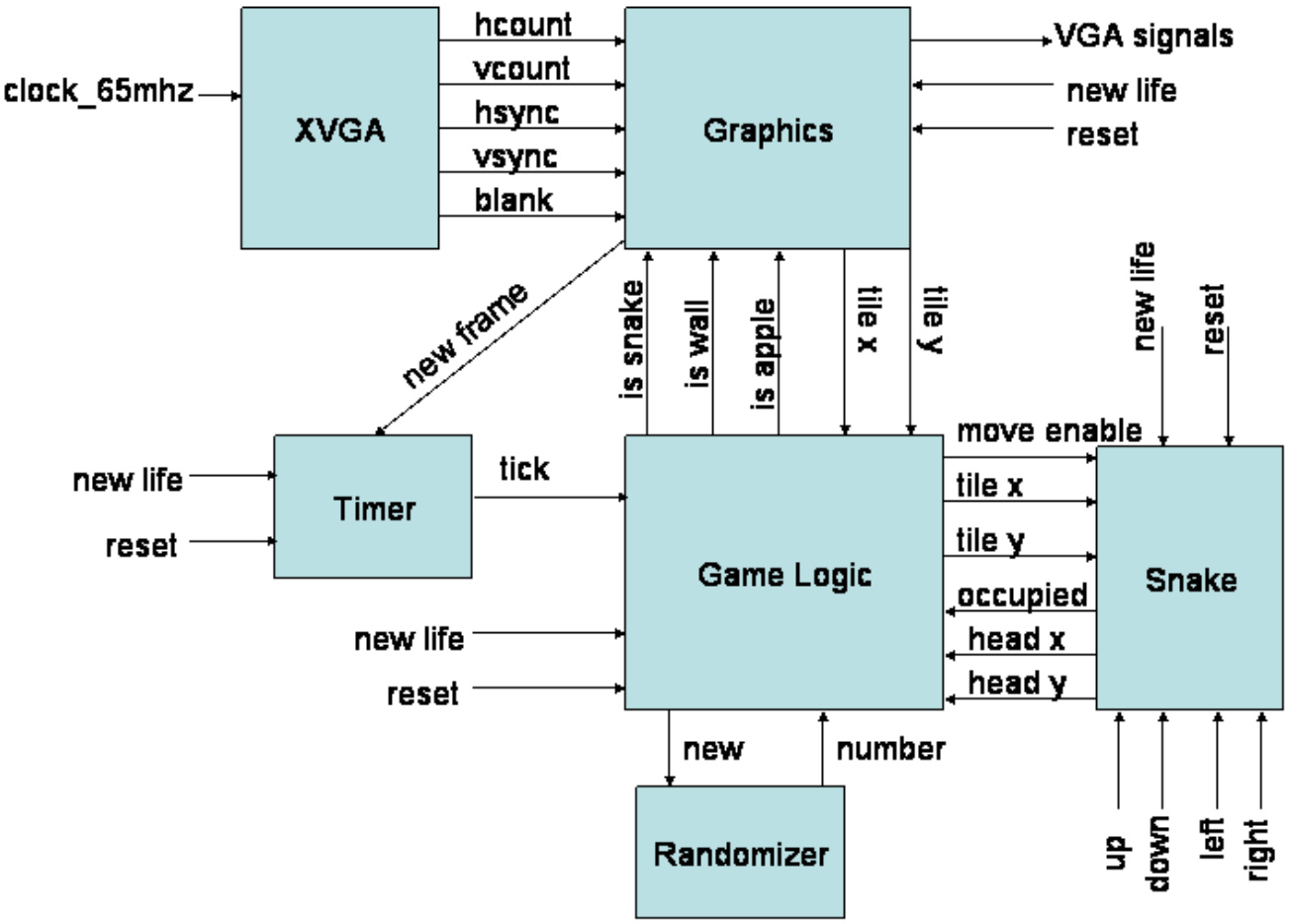
Project Proposal

I plan to create a system through which the user will play a version of the classic video game snake. In this game, the user controls a snake which navigates the screen, trying to eat apples and avoid obstacles. The snake moves at a constant pace, and the user presses directional buttons to cause it to turn in the direction given by the button. Turning in the direction opposite to the snake's current direction does nothing. The snake begins at a certain length, and its length increases whenever it eats an apple by running into it on screen. When the snake eats an apple, a new apple appears at a random location a moment later, and if the snake does not eat the apple within 30 seconds, it disappears and a new apple appears at another random location. While the snake tries to eat the apples, it must also avoid obstacles, which consist of the edges on the screen, walls, and its own tail. Hitting any of these causes the snake to lose a life. The snake can lose three lives before the game is over.

If I have additional time, I plan to create multiple levels, which the user would reach by having the snake reach some specified length, to add sounds, and to interface with an NES controller instead of just the buttons on the labkit.



My design consists of six main modules: the XVGA, the graphics, the game logic, a timer, a randomizer, and a snake module.

My XVGA module will be very much like the one from lab 5. It will take in a clock which will control the amount of time sent signaling each pixel to the VGA monitor, and from that will return signals for the current hcount, vcount, hsync, vsync, and blank. These signals will go to my graphics

module, which will use them to determine what values to pass to the VGA monitor.

The graphics module will be in charge of interpreting the game state graphically and sending that information to the monitor. It will send the game logic an x and a y tile location, and in return it will receive signals representing whether the snake, an apple, or an obstacle is in that tile location. The module will then cache the pixel information based on these tiles and its knowledge of what these things look like, outputting each pixel to the VGA display as appropriate. In addition, the graphics module will send a signal to the timer whenever it reaches the end of a frame.

The game logic module will be the major portion of my project. Whenever this module hears a tick from the timer, it tells the snake to move, then it analyses the new position of the snake's head to see if it collides with a boundary, a wall, the rest of the snake, or an apple. When the snake collides with a boundary, a wall, or itself, the game logic module checks to see if there are any lives left. If the snake has lives remaining, the game logic decreases the number of remaining lives and then asserts the new life signal, telling all other modules to reset, but with one less life. If there are no lives left, the game logic module asserts a global pause signal, telling all action to stop until reset is asserted. If the snake collides with an apple, the game logic waits two timer cycles while getting a new set of tile coordinates from the randomizer, which it then checks to be sure that this tile does not collide with anything currently on screen. If the tile is collision free, then on the next tick of the timer, the game logic creates an apple at that tile, otherwise it continues to inspect random tiles until it finds one suitable for the next apple. The game logic module also sends the graphics module information about what is present in a particular tile and tells the snake to move a tile once every timer tick.

My timer is a simple counter which asserts a high enable signal once every n new frames, where n is the number of pixels in the hight of a tile. In this way, the game logic is enabled when the snake will have moved a full tile, thus keeping the snake aligned with the tiles and also eliminating the need to make collision calculations when the snake is not turning and therefore no information has changed.

My randomizer generates new locations for the apple. I plan to do this using a combination of a linear feedback shift register and the system time. Every time the game logic receives an enable signal from the timer, it will forward it to the randomizer, which will then carry out one iteration of the linear feedback shift register output stream. In this way, once an apple is eaten, the randomizer will return a new location based on how long it took to eat that first apple. This should hopefully prevent the game from being identical each time it is played.

Finally, the snake module will keep track of the current length of the snake and where each of its 1 tile long tail segments are. Given a tile location, it will return whether the snake is on the tile. The snake module takes in the user input corresponding to the four directional buttons, and it keeps track of the last direction button that was pressed. When the module receives the move enable signal, it turns the snake in the direction of the last button which was pressed, unless that direction is opposite of the snake's current direction, at which point nothing happens. The snake module outputs the position of the head of the snake, as this is the part with the potential to collide with things on screen, to the game logic so that it can check for collisions.

I plan to start by creating a simplified version of the graphic system, where each tile is a solid color. I will feed a fake game state into it for the sake of testing. Once that works, I will try to create a moving game which does not care about collisions and has no apples. After that, I plan to add collision detection and the ability to lose lives. Once all of that is working, I will try to implement the randomizer, testing mostly using Modelsim, and then use it to create new apples along with having apple collision detection. If I manage to get all of that working, I will begin to work on creating multiple levels

As I am the only member of my team, I will be creating and testing these modules on my own.