Knight’s Resolve

Syed Ali Raza Rizvi - //

Project Description:

Originally designed as a story game, this project was reworked into a puzzle game with two possible endings. The game is implemented as a Moore machine, with button inputs driving state transitions. The buttons A (KEY 0) and B (KEY1) are used as a one bit input (eg; 1 = A, 0 = B). By using both of them in combination a one-bit-like input can be replicated.

Features:

vga_controller

The vga_controller module is used to output a 64x48 PNG image. It is modified from a github source[5]. A smaller size was chosen to maximize the amount of available space on the on-chip memory. The on-chip memory contains 240KB. To read a PNG, it needed to be converted to a hex file first, this was one of described memory types compatible with the board as said in the user manual. Each pixel is converted to a hex character, since all the images were drawn from pixilart.com by myself, we could choose the colours as well. Each image was created using a 4-bit color depth, which provides a total of 16 possible colors. This is why each pixel is a hex character, each letter from 0-F represents a colour. The method to convert the PNG was by using a python script. With the pillow library, we can map each pixel to a hex character. Each hex file has 3072 lines (64x48 = 3072) and each line has a hex character. With this information, we can use readmemh to process memory at compilation. Each hex file is 8.99KB, we are limited to how many images we want to read.

The horizontal and vertical counter for the vga module is used to determine where and when we can display a pixel on the 640x480 resolution. We read the 1 pixel’s information from dividing the horizontal and vertical counter by 10 and multiplying the y counter by 64(width). This ensures that each memory index of our image is read out in a 10x10 block to fit the 640x480 resolution. Each pixel read is also sent to the get_rgb function to decode the hex character to its red green and blue colour.
Additionally, the hex files are required to get the output. If they are saved in the project directory we can access them easier.

state_machine_controller

The design was based off of a Moore machine design, combining both parts in figure (a) and (b). In figure (a) below, contains the more simple linear logic. Figure (b) was more complicated since I wanted to make an immersive walking simulator. The labyrinth logic . We have input buttons a and b to progress through each state. The entire state machine is controlled by the two button inputs.
The state changes from either button_a or button_b depending on my state diagram

The game has two endings, each of the endings loop back to the title page. The state machine works on a much slower clock than the vga_controller. This is because the buttons pressed, changes between the states if held for a little bit. Therefore, I found a speed that is enough to be responsive for when we just press the button. This is shown in the video.

The output for each state is different for the story parts. However in the labyrinth part, I have some states that reuse images, this makes it seem like we are actually walking around the space adding some immersion as some sort of a walking simulator.

Conclusion:

This project demonstrates how to design a time-controlled state machine system using Verilog. By combining a state machine controller and a vga output, the system is able to simulate a game-like experience with timed state transitions. The design is scalable, allowing for easy expansion of the number of states or changes in the clock division ratio to adjust the timing of button inputs. The labyrinth part of the game can also be its whole own game with a more complicated maze and intersection. I had the drawings finished for 3 way intersections, however due to limited on-chip memory, I was only able to make a small proof of concept version while also sticking to making a story as described in the project proposal.

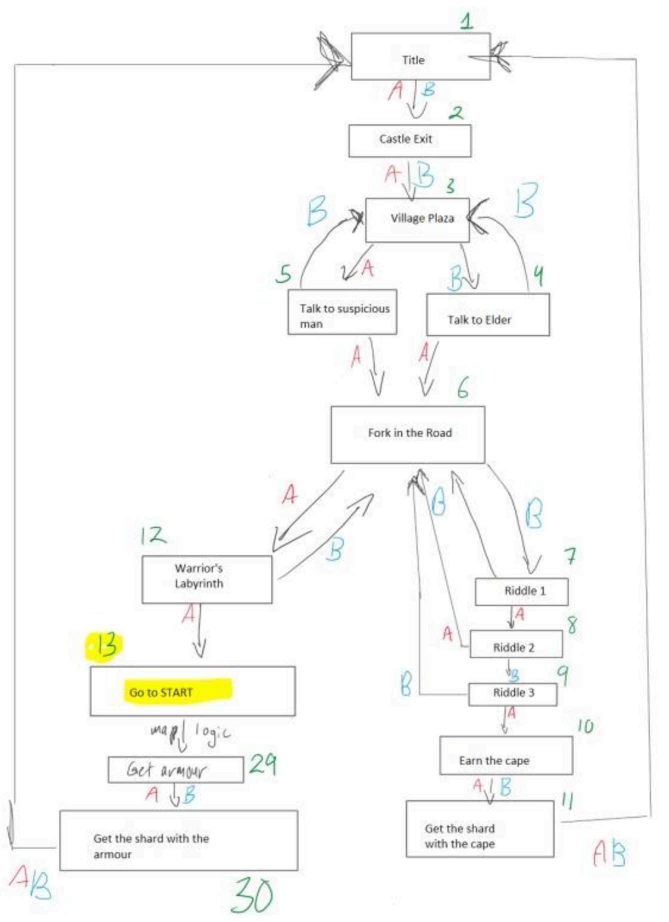

Figure (a), story and riddle

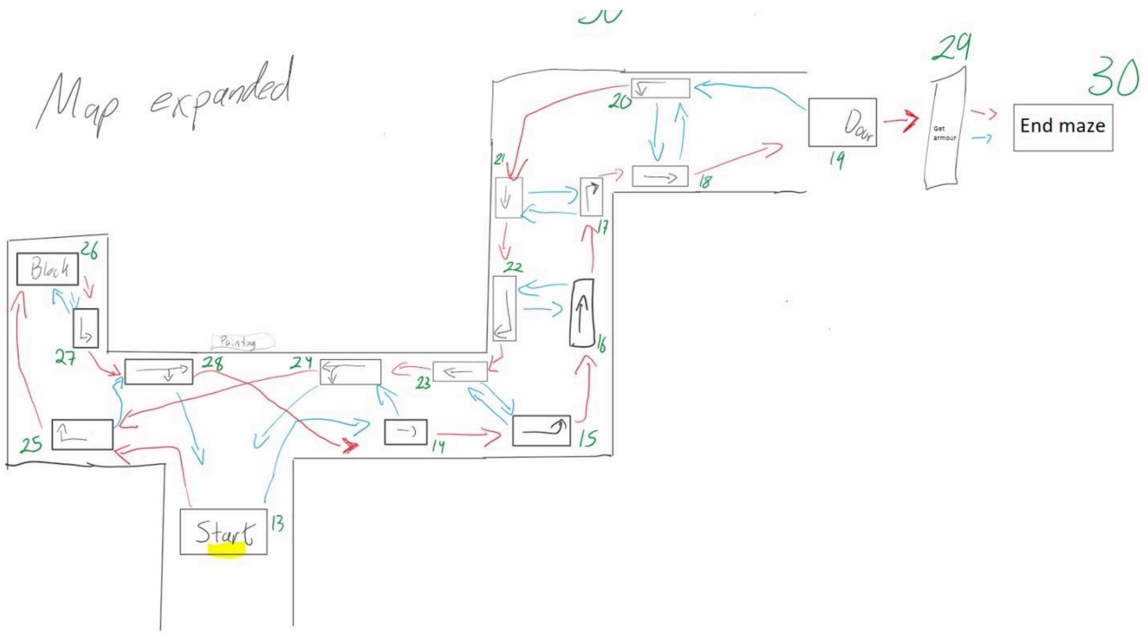

Figure (b), labyrinth map

References and Sources

[1] DE10-Lite User Manual
https://ftp.intel.com/Public/Pub/fpgaup/pub/Intel_Material/Boards/DE10-Lite/DE10_Lite_User_Manual.pdf

[2]
https://www.youtube.com/watch?v=4enWoVHCykI&t=437s
How to Create VGA Controller in Verilog on FPGA? | Xilinx FPGA Programming Tutorials

[3]
https://www.youtube.com/watch?v=mR-eo7a4n5Q
VGA image driver (make a face) on an Intel FPGA

[4] RGB Color Codes Chart
https://www.rapidtables.com/web/color/RGB_Color.html

[5]
https://github.com/dominic-meads/Quartus-Projects/blob/main/VGA_face/smiley_test.v