Introduction

This project is a recreation of the classic arcade game Snake on the NEXYS A7. The primary purpose of this is to make something but to also showcase the capabilities of the NEXYS A7. From being able to generate and handle multiple clocks to mimicking analog signals using PWM the NEXYS A7 is capable of much more. The following is a brief rundown of how the project works and some of the many components used, which include but are not limited to the aforementioned examples given earlier. It'll go over what each module is doing and explain the general idea and concept that the normal user usually never sees. Finally, this paper will explain how all of the following modules will work together to create the end product

Sub-modules

se_clk_div.v

The clock divider's purpose is to take in a clock signal of 100MHz and output a new clock based on the original clock. This new clock will help us run the other components in the circuit that may run on different frequencies. It can do this by counting the number of times the clock pulse goes high and incrementing a counter till it equals the div value. Once the value equals the div value it will reset the counter and flip the output for the divided_clk. The div_value can be calculated by using this equation div_value=100MHz/(2*desired frequency)-1.

randomGrid.v

Reading the randomGrid may not seem like it's random but the function will continue to be implemented with each clock pulse. This means that if we sample the output for the x and y for this function at a random moment it produces an almost random number for human perception especially since the frequency of the clock that will be used will be too fast for a human to perceive a trend.

pwmgen.v

Pwm is a way for a digital circuit to mimic an analog signal by affecting the duration of a duty cycle. In this case, it can accomplish this with the use of a counter. The counter in this case will count to its parameterized size from there it will remain at its current state or flip based on the duty load which is controlled by the code's load and duty inputs

pwm_top.v

The Pwm_top will use the previously mentioned pwmgen module and take its output to drive the RGB LEDs on the board. this will allow the board to display different colored LEDS based on the condition of the game.

decoder.v

This function is rather simple as its only function is to take in one of the cardinal keyboard inputs and make that the semi-permanent direction the snake follows. Without it, the snake would stop without any keyboard inputs this way the user does not need to hold down the key on the keyboard.

Debouncer.v

The concept of a debouncer is to act like a filter and remove any unintended user inputs such as noise. This circuit can be created by assigning this value to a temporary value if the the value remains the same the counter will increase and when the counter reaches 19 it means the input has been long enough for it to be considered a user input. Otherwise, the temp value will be overwritten with another input and the counter will reset.

VGA_gen.v

The VGA_gen will help drive an image on a VGA monitor. Put simply the VGA will go across the pixels in the monitor almost like a book; it'll start at the top left corner of the monitor and move horizontally to the right side of the monitor. Once it reaches the end it will go back to the beginning and move one row down on the vertical axis and repeat moving from the very left corner to the right. Till it reaches the end of the vertical axis and re-run it all over again. In the case of this program, it'll have a 640x480 display. This means for the horizontal axis it will count from 0-799 and 0-524 for the y-axis. The extra pixel counts in each axis can be attributed to their respective front porches, back porches, and sync pulses minus 1 due to the count starting on 0.

SSG_control.v

The seven-segment controller will help manage data given to it and display it on its respective segment on the board. Depending on what input is given the board has pre-programmed outputs to make the segments on the boardlight up in the shape of a hex value. Since this is a common anode this means for seven segments its anode must be set to 0. Thus when the cathode is 0 it will light up that specific segment on its respective seven segments.

PS2Receiver.v

This file works by receiving the keyboard inputs and outputting them as a 32-bit binary. To do this it needs the debouncer to filter out any unwanted signal. Once the debouncer has filtered out all the nonuser inputs it will check the negative edge of the clock assign the data given to a temp value and increment the counter to assign the next bit value. Once the bits are assigned the parity bit should flag the next part of the code to assign the data output in groups of 8.

Top file

top_snake.v

The top snake is the longest code however once it is broken down it is evident that it is simply either functions that have already been explained or it is drawing out the parts of the game on the screen. For starters, the function will declare regs and wires to store and route value for later and use the clk_divider module 4 times to produce 3 separate clocks the CLK50MHZ, the VGA_clk, update, and slow_clk.

Additionally towards the top 2 switches are assigned to LEDs as a way to indicate that the switches work and display if they are active or not. Afterward, the pwm_top module will take the original clk input and use that to display green if the user wins or red if the user loses.

Moving on the function calls for modules previously explained it calls for the VGA_gen to retrieve the x/ypixel coordinates and h/vsync values, RandomGrid to generate a random coordinate for the apple after it has been eaten, PS2Reciever to receive the keyboard inputs, decoder to take that input and translate it to a constate direction for our snake to follow, and ssg_controller to display the number of apples eaten.

Afterward, the program enters separate always loops that are driven by the VGA_clk to draw the borders of the game, the snake, and the apple by assigning RGB values to the current pixel

value of x and y., the function will randomly place them based on the randomGrid module and account for if the op is outside the display bounds.

Since the function now has the necessary starting components for the game it needs to account for when it runs into the bounds, itself, or an apple. When the snake collides with the apple it increments a counter and runs randomGrid to create new coordinates for the next apple making sure it is within the bounds of the game. Should the snake collide with its own body it will trigger the end game. A similar occurrence will happen if the snake runs into the border.

Once the apple count increases a new body segment will appear behind the snake by cascading the head to a new value the preceding bodies being an afterimage of their former selves. Additionally, the apple count will change the number of pixels the snake will traverse causing it to speed up as the snake collects more apples.

Conclusion

In conclusion, this project shows the versatility of the NEXYS A7 by running a not-so-simple game of snake. As can be seen in the code a lot goes into making even a simple game as snake. The multiple different modules from managing and driving different modules, to receiving inputs and displaying outputs. However, the NEXYs 7A is capable of running all of this, and by combining all these modules it can be pieced together to create this beloved arcade game and many more. The project above is only a small explanation and example of what the NEXYs 7A is capable of. Hopefully, this demonstration has presented some of the many underlying applications that run beneath the many things around you and presents an opportunity to attempt something similar with the tools explained in the project.