Lab 4: Completing My Project

Charlie Coleman
2018-12-18

Objective: The objective of this lab is to complete the design laid out in labs one through three using hardware and software components.

Equipment:
- Computer
- Digilent Zybo 7000
- Xilinx Vivado 2018

Procedure:
- Hardware
  1. Connect the Zybo 7000 you computer and ensure you have the board files for Vivado.
  2. Create a new project with the Zybo as the target
  3. Create an empty block design.
  4. From the IP catalog, place the:
     - ZYNQ7 Processing System
     - XADC Wizard
  5. Create a custom IP block called VGA_controller
     (a) Give the IP block 16 registers
     (b) It will need 4 outputs,
     - `hs - std_logic`
     - `vs - std_logic`
     - `row - std_logic_vector(9 downto 0)`
     - `col - std_logic_vector(9 downto 0)`
     (c) Create a 100MHz → 25MHz clock divider and a new clock signal
     (d) Create your horizontal sync (hs) and vertical sync (vs) signals
        i. HS should toggle with the 25MHz clock
        ii. VS should go high for 1 clock period when HS reaches the width of the frame
     (e) row & col should increment with VS & HS accordingly
  6. Create a custom IP block for the ship
     (a) Give the IP block 16 registers
     (b) It will have 2 inputs and 3 outputs
     - `in:  row - std_logic_vector(9 downto 0)`
     - `in:  col - std_logic_vector(9 downto 0)`
     - `out:  red - std_logic_vector(4 downto 0)`
     - `out:  green - std_logic_vector(5 downto 0)`
     - `out:  blue - std_logic_vector(4 downto 0)`
     (c) Create a 32x32 array of 16 bit vectors for the pixel values
     (d) Use a register to hold the current player position
     (e) If the row & column correspond to a pixel in the player's sprite, output that

value over the red, green, & blue lines

(f) Use a register to input new colors into the array of pixels for the player

(g) Use a register as a data valid signal for the hardware

(h) Use another register as the index of the array to write a value to.

(i) When the data is valid, over the position given with the new pixel value.

(j) Also read in a new player position when data is valid.

7. Create a custom IP block for your maze
   (a) Give the IP block 16 registers
   (b) The block will need 5 inputs and 3 outputs
   ```
   - in:  r_in - std_logic_vector(4 downto 0)
   - in:  g_in - std_logic_vector(5 downto 0)
   - in:  b_in - std_logic_vector(4 downto 0)
   - in:  row - std_logic_vector(9 downto 0)
   - in:  col - std_logic_vector(9 downto 0)
   - out:  r_out - std_logic_vector(4 downto 0)
   - out:  g_out - std_logic_vector(5 downto 0)
   - out:  b_out - std_logic_vector(4 downto 0)
   ```
   (c) The RGB inputs to this block come from the ship to aid in hit detection.

   (d) Draw a maze by checking if the row & col values are in certain ranges, creating rectangles.

   (e) Check if the input RGB is not equal to zero while you are drawing a wall. If so, write 0xFFFFFFFF into a register to signal to the software side that a collision has been detected.

   (f) Output the maze RGB or the ship RGB out the RGB out signals, depending on which is activated.

8. Run block automation, connection automation
9. Create 5 output ports in the overall block design
   - vga_hs
   - vga_vs
   - vga_r[4:0]
   - vga_g[5:0]
   - vga_b[4:0]
10. Create 4 input ports for the ADC
   - vauxn7
   - vauxp7
   - vauxn14
   - vauxp14
11. Open the XADC Wizard & enable channel sequencer
12. Activate channels 7 & 14 (these are used by the Zybo)
13. Connect the inputs to the XADC Wizard
14. Connect vga_hs & vga_vs to the VGA controller
15. Connect the VGA RGB signals to the maze controller
16. Generate bitstream & export to SDK

- Software

1. Create a new hello world project
2. Clear the helloworld.c file
3. Create a helloworld.h file
4. Make an array of pixel positions for the ship, treating 16,16 as the origin.
5. Create a struct for the player with its position, speed, and angle.
6. Create an initialize function in the C file
   (a) Initialize the player variable, set it's position to the start of your maze, speed to zero and it's angle to point it whichever direction it will start moving.
   (b) Write the player sprite to the hardware using the data valid/pixel value/pixel position/player position registers.
7. Create the main game logic
   (a) Write a function to rotate points around the origin given a point and an angle. (you will need to link the math library in order to use sin & cos)
   (b) Write an updatePlayerPosition() function, which will take the speed, a single double, and turn use it to displace the player correctly based on it's angle. You need to create a new point (0, speed) and rotate it to the player's angle, then add this point to the player's position. You also need to decrease the player's speed gradually, so decrement by 0.1
   (c) Write a updatePlayerSprite() function, this should take every point in the player sprite array and rotate it to the new angle, then write these values to the hardware.
   (d) Write a getInput() function, which will poll the ADC registers and check their values. They run much faster than 60Hz, so there shouldn't be any waiting required. Use the X/Y values of the joystick to increase player speed/change angle accordingly.
   (e) Write a checkCollisions() function, which checks the maze register that will hold ones if a collision has occurred. If it has, reset the player to it's original position, speed, and angle
8. Combine all of these functions into a game loop function. This function will need to wait for the frame to be done drawing to restart.

⋆ Program the Zybo & launch the C code on the hardware. You should have a maze!

Recalculations and Predictions: N/A

Data and Observations: There is no real data to be looked at in this lab. It was observed that the design ran successfully on multiply monitors, which points to the fact that it was running at 60Hz. The player was able to interact with the game through a joystick, and the game was able to successfully detect a collision with a wall and punish the player for it. The collision detection was very accurate, and the game was playable.

Analysis and Discussion: No data was collected in this lab, the behavior and observations from the design matched expectations very closely.

Conclusions: In the end, this lab was a success. A maze was successfully implemented using a combination of software and hardware designs. This knowledge base will be very helpful in future situations as it has helped me become more familiar and comfortable designing things

in hardware, while normally I would simply default to software. The connection between hardware and software using the AXI interface seems to be a somewhat common method and will be very useful if I am to use a device with this functionality in the future.

I gained experience in C and VHDL, and while I had used both before, I gained a lot of useful information from this project. I had never utilized C (or any software) without an operating system before. I had also never created multiple designs in VHDL to connect together with other, commercial IP blocks. Overall, this lab was a great success.