Pong in VHDL—Hardware Only

Brian Lien, Lucas Mueller, Nikodem Gazda

Block Diagram:

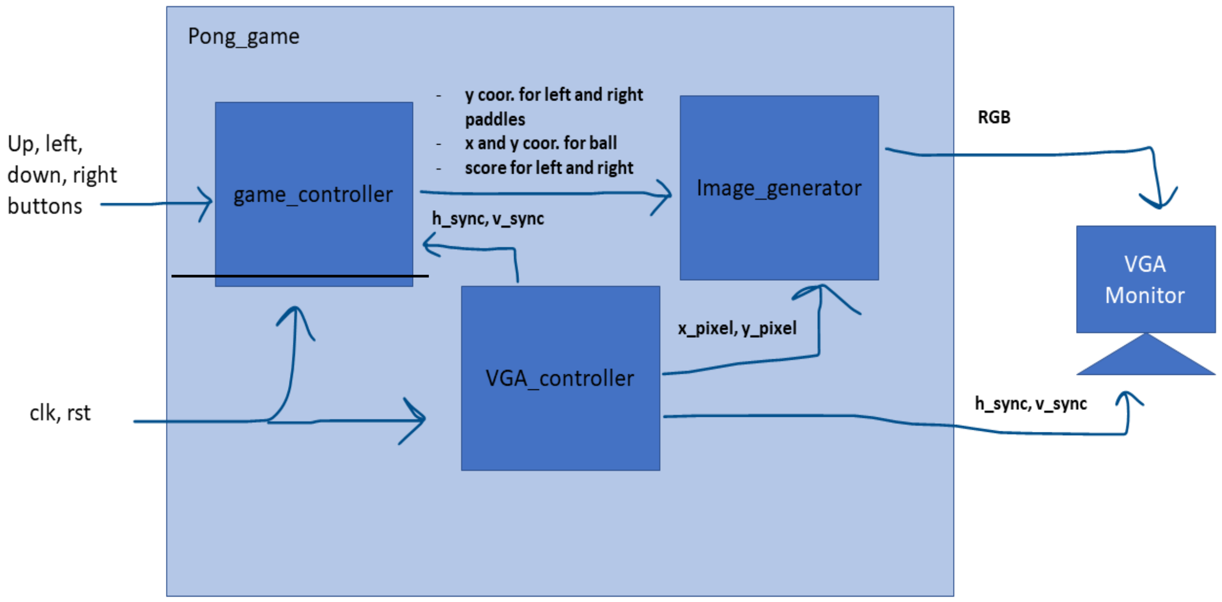

Figure 1: Block diagram of our pong implementation.

Implementation Description:

Pong Game (Pong_Game.vhd):

This VHDL file is the top-level design entity for this project. It is a structural architecture that exists solely to connect each of the other components and interface with the board I/O. There is no additional functionality in this module apart from what is described in the block diagram above.

Apart from this top-level entity, the project contains 3 sub-modules: the VGA controller, the game controller, and the image generator. The VGA controller manages the synchronization of the VGA display output. The game controller manages the logic regarding the behavior of the game, such as entity movement, collision, and scorekeeping. The image generator takes the positions of all the elements that need to be displayed and draws them to the screen.

Game Controller (gamecontroller.vhd):

The game controller uses a state machine and button inputs to output the current position of the ball and paddles as well as the score to the image generator. This module consists of three process statements: UpdateFSM, “MovePaddle”, and “MoveBall”.

Upon reset, the UpdateFSM process returns the paddles and ball to their center positions and the score to 0. On each clock cycle, this process updates the registers for the scores, the ball position, velocity flags for the ball, paddle positions, and the game state.

The “MovePaddle” process is responsible for reflecting the button inputs in the position of the paddles as well as updating the state of the game. Depending on how the buttons are pressed, the paddle position registers are updated to move them up or down. When the ball is in a position where it collides with the wall without hitting a paddle, the game state is set to 0 and the paddles are reset to the middle of the screen to signify the game is currently paused until the users start gameplay again. If the buttons cause the paddles to move while the game state is cleared, the game state is then set, and the ball is allowed to update its position in the “MoveBall” process. The movement of the paddles updates on the rising edge of v_sync, meaning every time the entire screen refreshes.

The “MoveBall” process is responsible for moving the ball and keeping track of the score. On each rising edge where the game state is set, the ball’s position is checked to see if it has collided with the left and right walls. The ball is then checked to see if it’s made contact with the paddles and reflects the ball off the paddles if so. If not, the score is updated and the ball position is reset to the center. This process also checks if the ball has collided with the ceiling and floor and reflects the ball if this is the case.

Image Generator (image_generator.vhd):

The image generator works by taking all the positions of the elements that need to be displayed and drawing them on the screen by using a series of if statements and RGB values. By defining dimensions for the ball and paddles we were able to use their positions to draw them to the VGA monitor by defining their boundaries with greater than and less than conditional statements. Then, when the active pixel, the pixel that is currently being drawn by the VGA Controller, is within these boundaries the color is set to the proper value. The positions of all the game elements are fed to the image generator as inputs from the game controller. And the position of the active pixel is sent to the image generator via the VGA controller. For displaying the score, we used a collection of bitmaps to represent the numerals so the controller would write a color to the RGB when the position overlapped with a 1 on the bitmap and black otherwise.

VGA Controller (vga_controller.v):

As the provided VGA controller was nonfunctional, an alternative component was sourced. The VGA controller used in this design was provided by David J. Marion (https://github.com/FPGADude). This component counts through the x and y coordinates of the display and outputs the VGA synchronization signals(video_on, hsync, vsync) and the (x,y) coordinates.

- The 25MHz clock signal is produced by a simple clock divider.
- The video_on signal is high by default and is brought low when the current (x,y) is not within the 640x480 display range.
- The hsync and vsync signals are pulled high at the end of each horizontal and vertical scan, respectively.

Problems Faced:

We faced several issues completing this project. Using and understanding Vivado for the first time took the largest amount of time. The VGA controller also took a considerable amount of time to complete, though the issues with the VGA controller related directly to using Vivado for the first time.

The remaining issues were lapses in thought when designing the logic of the game, all of which were solved after minimal debugging.

Lessons Learned:

We learned how to interface with Vivado. This assignment also was a good refresher for VHDL, and we learned how to integrate Verilog code with VHDL.