VGA Pong in Hardware

Chris Fallin <cfallin@nd.edu>

Summary

In order to exercise my newly-acquired knowledge of digital logic structure and design in Verilog, it was decided that I should complete some practical, nontrivial project on the Digilent Basys demo-board platform built around a Xilinx Spartan3E FPGA. I chose to create a hardware-based Pong implementation, generating valid VGA signals on the built-in display port and accepting input from a PS/2 keyboard. What follows is an account of the design as implemented.

VGA Port

The first milestone in the design process required that I produce a valid image on a standard VGA monitor attached to the demo board. A VGA connector has five signal lines of concern to this application: horizontal synchronization (H-sync), V-sync, red, green, and blue. Digilent has wired these directly to digital GPIO pins on the FPGA, with the implication that only eight colors are possible.

VGA timing information is available freely throughout the Internet; the general idea is that one scans the display surface (CRT or LCD) left-to-right, top-to-bottom, and pulses the horizontal or vertical sync line low to reset to the left edge or the top of the screen, respectively. For some subset of each line, and for some subset of the lines within one frame, the electron beam is "active" – ie, not blanked (off-screen).

With this information in hand, I first created a Verilog module to accept the onboard 50 MHz clock and produce five outputs: H-sync, V-sync, active (not blanked), and horizontal/vertical coordinates.

With the synchronization in place, the remainder of the display hardware simply has to determine the appropriate pixel value given the coordinate registers and send to the R/G/B lines, gated by the 'active' line.

The display requirements for this particular application – a game of Pong that displays a square ball and two rectangular paddles – are very simple. These three objects are displayed by three "box detectors" – Verilog modules that take the coordinate registers and top, bottom, left, right coordinates and determine whether the current scan position lies within the given box. Each box detector drives one of the three colors, with the result that the game logic merely has to provide coordinates for the paddles and ball.

Keyboard

The keyboard support was trickier than expected, and in the end does not work completely robustly; however, this is likely due to race condition(s) for which a thorough

solution would not be a practical use of time given the success of the remainder of the project.

A PS/2 keyboard sends 11-bit words serially; the second through ninth bits are a one-byte scancode, LSB first. The data are clocked into a shift register by a clock provided by the keyboard. The receiver keeps count of the number of bits received; when a full 11 bits are in the shift register, a scancode-received line is pulsed. This sets into motion a scancode-decoder state machine, necessary to interpret multi-byte sequences.

The keyboard sends one-byte keycodes for keys W/S (left paddle up/down) and O/L (right paddle up/down), and sends the same keycodes prefixed by 0xF0 on key-up. The state machine interprets these sequences to provide four outputs, each of which is high precisely when the corresponding key is depressed. These lines provide control input to the game logic.

Game Logic

With paddle up/down lines provided, and the display hardware needing only the positions of the paddles and ball, all that remains is to create a simple state machine that moves and bounces the ball.

The ball itself has two pieces of state: position (X/Y) and velocity. To simplify the game hardware, velocity always has X and Y components that are either +1 or -1; thus, only four directions are possible. On each clock to the state machine, the ball moves in the specified direction by one pixel.

Collision detection consists of a set of comparators that determine whether the ball has hit any of the screen edges; and, if the left or right edge, if the ball hit the paddle. These results are provided to the bounce logic. This logic inverts the X or Y velocity if the appropriate bounce-detect has occurred; it also sets a flag in this case so that bounce detection resumes only once the ball has moved clear of the wall (otherwise it would continually flip direction and remain motionless). In the case of a left or right bounce without a paddle, the game-over flag is set, and this flag gates the motion state-machine clock so that all motion stops.

The entire motion unit is driven by a slow motion clock produced by dividing the 50 MHz system clock by 2^18.

Verilog

Attached.