BRADLEY UNIVERSITY
DEPARTMENT OF ELECTRICAL ENGINEERING
SUBMITTED BY: 270968 & 261302

VGA DISPLAY: Pong Game

Purpose

In this lab:

- The creation of a complete design of pong game, interactive system
- Further exploration of processes and behavioral description in VHDL
- Additional techniques for integrating human input with a digital circuit and implementing it on a FPGA board.

Background Information

The game *Pong* is one of the first complete, standalone digital arcade games to reach mainstream popularity. *Pong*, released in 1972, is a simple tennis-like game that pits two players against each other on an almost-empty screen. A small, square ball bounces back and forth between paddles controlled by two players, who attempt to bounce it back at their opponent. If either player fails to hit the ball (it goes off the edge of their screen), that player loses the round, and their opponent gets a point.

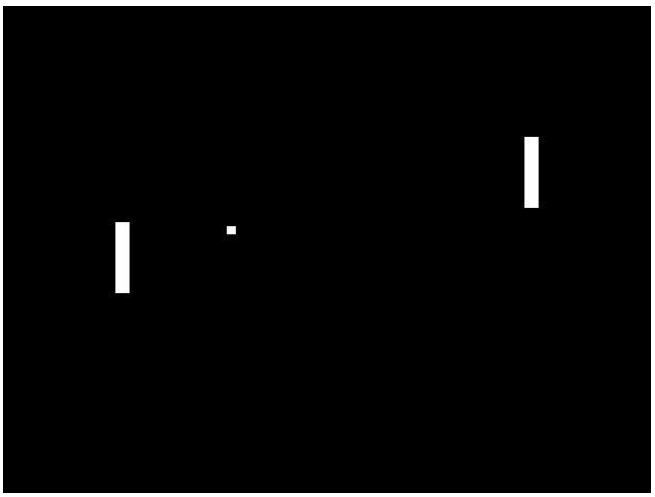

Figure 1: Image of *Pong*

The game of *Pong* consists of three main "entities": two paddles and the ball. You will design a datapath to control each of these entities and manage the flow of the game. Inside this datapath, we will implement a module for each entity. Each module will track the X and Y coordinates of its entity. Because the game "screen" will be 640 by 480, all X coordinates will be represented in ***10 bits*** (values from 0 to 1023, though only 0 to 639 should be considered valid). To simplify the design, we will also use ***10 bits*** to represent Y coordinates. As with the VGA controller, the ***top left corner*** is the origin (X=0, Y=0). Higher values of X move right, while higher values of Y move down.

DESIGN OF THE PADDLES AND THE BALL:

Each paddle has a fixed X-coordinate on the screen, and the only thing that will change is the Y-coordinate as we move the paddle up and down. The *paddle* module will take two one-bit inputs for suppose take "U"-, telling the paddle to move up, and "D", to move down. If the U input is 1, the paddle will move up by 4 (***decrease*** Y by 4). If the D input is '1', the paddle will move down by 4 (***increase*** Y by 4). If both are '1', the paddle should stay in position.

We need to prevent the paddle from going off the screen. To handle this, you will perform the following check: if Y < 100 don't decrease Y, and if Y >= 440 don't increase Y.

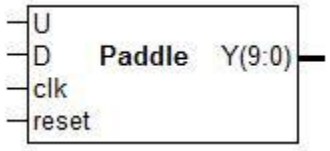

Figure 2: Interface to the Paddle Module

The *ball*, which will bounce back and forth from one player's side to the other. For the ball, we will track the current direction it is traveling in separate X and Y-components, or its *velocity*. We will store the velocity using two flip-flops, one for X and the other for Y. One flip-flop will tell us whether the ball is moving in the positive-X direction (1, ***right***) or the negative-X direction (0, ***left***), and the other will do the same for Y (1 is ***down***, 0 is ***up***). These values are recapped in Table 1 and Figure 3.

X	Y	X direction	Y direction
0	0	Left	Up
0	1	Left	Down
1	0	Right	Up
1	1	Right	Down

Table 1: Ball Direction Encodings

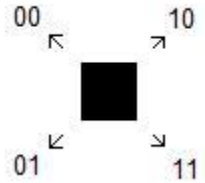

Figure 3: Ball Direction Encodings (X = MSB, Y = LSB)

When the ball collides with a paddle or the border of the screen, the ball will bounce away from it. This will be done by reversing direction away from the collision. Using the velocity encoding table above, this can clearly be accomplished by flipping either the X or the Y bit.

Whenever we want to change direction, we will pass a '1' to the corresponding flip flop. All of the possible collisions in the *Pong* game are shown in Figure 4, along with the flip-flop that will toggle in each case.

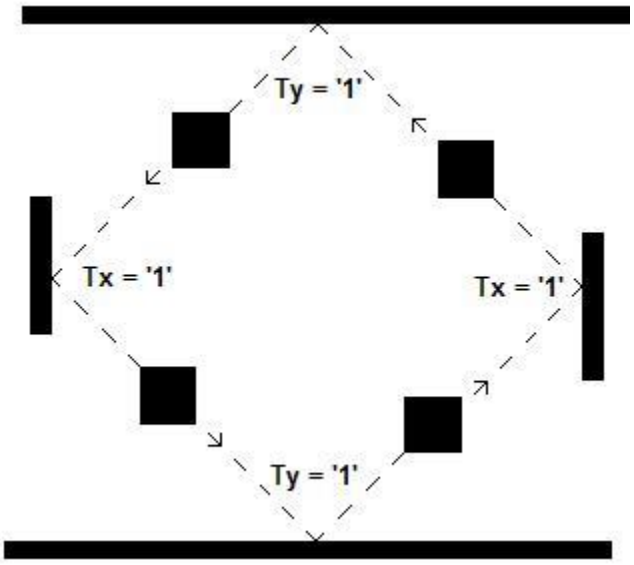

Figure 4: Possible Collisions in *Pong*

From Figure 4, we can see that there are only four possibilities:

1. The ball collides with the top border (Ty = '1')
2. The ball collides with the left paddle (Tx = '1')
3. The ball collides with the bottom border (Ty = '1')
4. The ball collides with the right paddle (Tx = '1')

This module will take, as input, the X and Y coordinates of the ball, as well as the Y coordinates of both paddles. If there is a collision that would require the ball to change its velocity in the X-direction, **Cx** will be '1'. Likewise, if a collision requires the ball to change its velocity in the Y-direction, **Cy** will be '1'

In this module we also need the movement of the ball's position as the game progresses. This module will have four direction inputs: XU, XD, YU, and YD. These correspond to moving the X or Y coordinate of the ball either up (U) or down (D). Additionally, because this module has sequential logic, it will require clock and reset signals, unlike the paddle module, the ball module does not need to wrap (check bounds) for the X and Y coordinates, because this will happen naturally because of the collision module.

If the above logic is evaluated every clock cycle at 25 MHz, the ball will move around the screen too quickly for the users to see it. Thus, we use an additional clock and reset to have a single output. The OPPC module will ensure that when the input **Di** becomes high, the output **Do** will become high for a single clock cycle. This serves to limit the rate at which our datapath runs.

VGA INTERFACING:

The VGA interface was Standardized by IBM in 1987 with signal requirements to display color images on a Cathode Ray Tube computer Monitor or (CRT) for short. When LCD monitors were later released, they adopted the same VGA signal standard in order to function with the current computer infrastructure out in the world. Therefore LCD Monitors have an internal translator circuit that converts VGA Signals to the required Liquid Crystal display control Signals. This all happens automatically inside the LCD Monitors circuitry.

Basically there are 5 separate signal requirements for the VGA interface to function. These are H-SYNC & V-SYNC resolution control signals and the RED, GREEN and BLUE color display signals.

The H-SYNC & V-SYNC signals you send to the monitor, dictate which Resolution setting you are attempting to use. Depending on the monitor, it may only support a specific resolution, where other Monitors may support more than one. The Monitor stores reference signal Patterns for all of the available resolution settings, within an EEPROM memory chip that is part of the monitors internal circuitry.

The RED, GREEN and BLUE color control signal lines, each use Analog Voltage levels between (0.0 to 0.7 Volts DC) to determine the amount of color intensity used for each Pixel. As you know, all colors are made up from mixing various amounts of RED, GREEN and BLUE together.

The completed datapath will have two one-bit inputs: **clock** (25 MHz) and **reset**, two two-bit buttons **btn_1 and btn_2**(which are the user buttons for moving both paddles up and down), hsync, vsync and rgb as outputs.

The *Pong* game will be displayed on a monitor using the VGA port of the Spartan 3 board. This module takes as input a 25 MHz clock, a reset signal, the 10-bit X and Y coordinates of the ball, and the 10-bit Y coordinate of both paddles. This module outputs the VGA control signals (HSYNC, VSYNC, video_on, p_tick, pixel_x and pixel_y) and RGB color signals.

Summary and Top-Level Schematic

To summarize, we have designed and built the following modules (with sub-components):

1. Designing a VGA sync module which includes:
    a. mod-2 circuit to generate 25 MHz enable tick
    b. mod-800 horizontal sync counter and
    c. mod-525 vertical sync counter
2. Pong datapath
    a. Paddle module
    b. Collision detection module
    c. Ball position module which includes trapping the ball within the boundaries
    d. RGB multiplexing circuit
3. Top module
    a. instantiate VGA sync
    b. instantiate graphic generator
    c. rgb buffer

SPECIFICATIONS ACHIEVED:

Changed the left wall to a left paddle, which is now controlled by a slide switch.

When the ball hits one of paddles, the color of paddle changes.

Every time when the ball hits one of paddles, the velocity of reflected ball changes. The increment or decrement of velocity is defined.

The ball is trapped in the display screen. When it hits the boundary of display screen, it bounces back.