# ECE 484 – Pong Laboratory Assignment

Last Updated: 1 March 2013

## Lab Overview

In this lab, you will build upon the modules created in the VGA laboratory assignment to implement a simplified version of the classic Pong video game. This laboratory assignment is based on a [FSM lab](http://web.mit.edu/6.111/www/s2009/LABS/LAB4/lab4.pdf) from the 6.111 course at MIT.

## Game Overview

The final video game implementation must have the features listed below. A sample screenshot is provided in Figure 1.

1. Background image of the “AF” logo
2. User-controllable paddle position on the left side of the screen
3. “Ball” that moves smoothly across the screen
   1. Bounces when it hits the top, right, or bottom walls.
   2. The ball is hidden when it is moving across the logo letters.
   3. When the game is over (i.e., the ball hits the left wall), the ball freezes in position.
4. The game can be restarted by pressing the “reset” button.



Figure : Sample screen shot of this lab's simplified version of the Pong video game.

## Prelab Assignment

On the first day of the lab, turn in a typed hard copy the following items:

1. Draw the state-transition diagram for your ball movement in pong\_control module.
2. Describe how you will implement bounds checking. Provide an example combinational statement that will be used as an input (e.g., has\_hit\_paddle) to your FSM for state-transition determination.
3. Write the register/combinational equations used to update the paddle position.

## Hardware Implementation

You must use a FSM to implement the pong game logic. The instructor implemented the game logic for this lab using a single FSM contained within your pong\_control module. The new entity definitions used by your instructor for this lab are shown on the next-to-last page of this document. There are two significant changes to the VGA lab assignment:

1. This implementation includes the game-logic module with user inputs for “up” and “down”.
2. The pixel generator has the current ball and paddle positions as inputs.

Using this implementation, the pong\_control FSM stays in an idle state until the current screen refresh is complete. It leaves the idle state when v\_completed is pulsed high for one clock cycle. It then moves through a variety of states to update the ball positions before returning to the idle state. The paddle position can be updated with a register and simple combinational logic.

The method described above is what was used by the instructor to implement this lab. However, there are more elegant, modular, and resource efficient ways to implement this game.

Using a more modular hardware design would simplify the addition of game logic – for example, expanding our game to implement a rudimentary version of [Breakout](http://en.wikipedia.org/wiki/Breakout_(video_game)) or adding a game menu system for high scores, difficulty level, and etcetera. This is highly desirable if you plan to build on this logic for your final project.

## Lab Hints

* Use a package header to define global constants (e.g, size of paddle\_width, ball\_radius; the states common to both FSMs, etc.)
* When checking the boundary conditions within a state, use only one if statement with elsif/else clauses. *Using* *multiple if statements in parallel can have unintended consequences.* Even better, you can do all your bounds checking with combinational statements. This is less likely to have unintended consequences.
* Make sure you don’t infer any latches in your design! In the past, this was the cause of most hardware implement issues.

## Extra Credit

Add additional features to this lab for extra credit. Here are a few ideas, but you can come up with your own as well:

* Change the speed of the ball in real-time based on the switch configuration.
* Creating different “hot” zones on the paddle. This would result in the ball bouncing off the paddle at different angles based on where it hits.

|  |
| --- |
| **entity** pong\_pixel\_gen **is**  **port** **(** row **:** **in** unsigned**(**10 **downto** 0**);**  column **:** **in** unsigned**(**10 **downto** 0**);**  blank **:** **in** std\_logic**;**  ball\_x **:** **in** unsigned**(**10 **downto** 0**);**  ball\_y **:** **in** unsigned**(**10 **downto** 0**);**  paddle\_y **:** **in** unsigned**(**10 **downto** 0**);**  rgb **:** **out** std\_logic\_vector**(**7 **downto** 0**)**  **);**  **end** pong\_pixel\_gen**;**  **entity** pong\_control **is**  **port** **(** clk **:** **in** std\_logic**;**  reset **:** **in** std\_logic**;**  up **:** **in** std\_logic**;**  down **:** **in** std\_logic**;**  v\_completed **:** **in** std\_logic**;**  ball\_x **:** **out** unsigned**(**10 **downto** 0**);**  ball\_y **:** **out** unsigned**(**10 **downto** 0**);**  paddle\_y **:** **out** unsigned**(**10 **downto** 0**)**  **);**  **end** pong\_control**;**  **entity** atlys\_lab\_video **is**  **port** **(**  clk **:** **in** std\_logic**;** -- 100 MHz  reset **:** **in** std\_logic**;**  up **:** **in** std\_logic**;**  down **:** **in** std\_logic**;**  tmds **:** **out** std\_logic\_vector**(**3 **downto** 0**);**  tmdsb **:** **out** std\_logic\_vector**(**3 **downto** 0**)**  **);**  **end** atlys\_lab\_video**;** |

Code Listing - Entity templates for the lab to ensure consistency between student designs.

|  |  |
| --- | --- |
| Pong Cut Sheet | **Name:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **Instructor:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **Section:** \_\_\_\_\_\_\_\_ |

**Number of hours spent on this lab:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (no points associated with this unless you leave it blank)

**Suggestions to improve this lab in future years:** (use blank space below)