DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
UNIVERSITY OF BRITISH COLUMBIA
CPEN 311 – Digital Systems Design
2015/2016 Term 1

Lab 4: More Algorithm Design

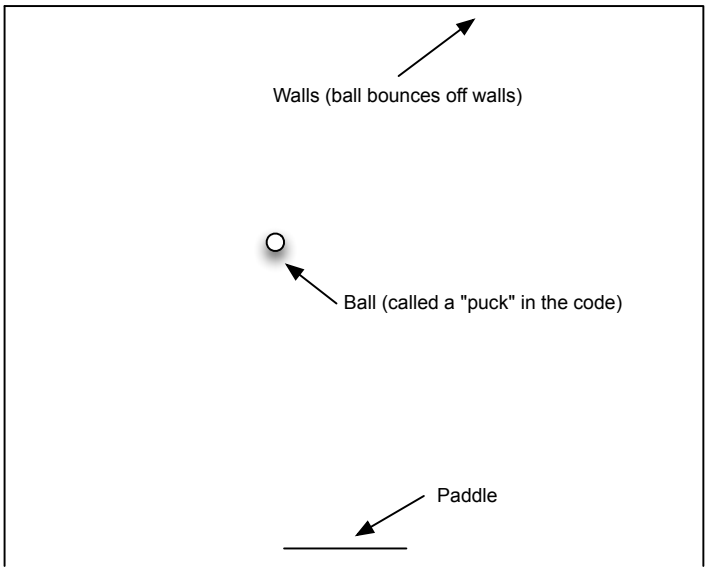
Working week: October 27/29, 2015
Marking week: November 3/5, 2015

In this lab, you will learn how to work with fixed-point numbers and arithmetic. You will also be introduced to an important debugging tool, SignalTap II. Finally, you will get some more experience working with non-trivial datapath circuits. Unlike Labs 1 to 3 where you wrote most of the code yourself, here you will be given a working VHDL implementation that you will enhance in several ways. Although this means that the *amount* of VHDL that you write in this lab is less than other labs, it also means you will need to spend some time understanding someone else's code. When you go to industry, rarely will you get to write your own code from scratch; usually projects are done by modifying an existing code base. Thus, the ability to *read* and *understand* existing code is as important as being able to write your own code.

Task 1: Download and Run the code

In this task, you should download the existing code base from Connect, compile it using Quartus II (remember your pin assignments!), download it to the DE2 board, and test it. The files to be downloaded can be found on Connect. The top-level file is called **lab4.vhd**. A package is included called **lab4_pkg.vhd** (you may not have seen packages before, but you should be able to figure it out by looking at the code). The other files contain the VGA core that you used in Lab 3.

When you download the design, and connect your DE2 board to the VGA monitor, the game should run. A ball (called a "puck" in the code) bounces off the walls at the top and right/left of the playing area. You can control a paddle at the bottom of the screen using KEY0 and KEY1. If the ball reaches the bottom line without hitting your paddle, the game is over and the system resets. At any time, you can use KEY3 to reset the game. This is a pretty simple game, and it shouldn't take long before you get bored with the game and are ready to proceed to Task 2.



Task 2: Understand the Code by Reading

Have a look at the VHDL code, in particular, **lab4_pkg.vhd** and **lab4.vhd.** The other files contain the VGA core that you used in lab 3; you don't have to read through those files.

Notice that **lab4.vhd** is designed using the design style we talked about in Slide Set 8. In this style, the state machine and state transitions are explicit, however the structure of the datapath is implicitly defined within the state machine process. I would suggest drawing (on paper) the state machine bubble diagram, indicating the condition that causes state transitions; this will help you understand how this circuit works.

Some questions you should be able to answer at this point (you do not have to hand in the answers, although your TA might ask you, and you should be able to answer):

1. Why is the IDLE state required? It seems like it does nothing. Is it true that we are in the IDLE state whenever the user is not pressing KEY0 and KEY1? (hint: no, that is not true; you should be able to explain).
2. What is the difference between DRAW_PADDLE_ENTER and DRAW_PADDLE_LOOP? Why do we have two different states? How many cycles would the machine stay in DRAW_PADDLE_ENTER? How many cycles would it stay in DRAW_PADDLE_LOOP?
3. In what state does the circuit check if the user is pressing KEY0 or KEY1? In what state does the circuit check if the ball has hit the wall or a paddle?
4. Why is each velocity component a signed value, while the ball location (called a "puck" in the code) is an unsigned value? It seems that when the ball hits a wall or paddle, the sign of the velocity is flipped (+1 or -1). Why does it do this?
5. If you wanted to draw another line on the screen, where would you add code to do this? How would you do it? Would you need another state?

Do not go onto Task 3 until you are relatively comfortable with the code.

Task 3: Use SignalTap II to observe the circuit in action (2 marks)

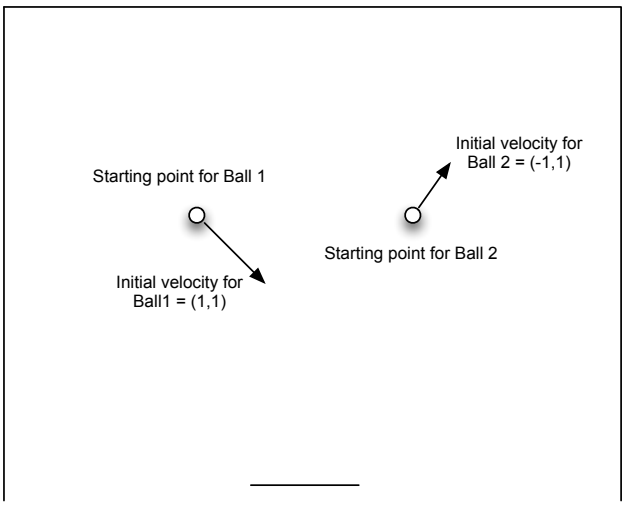
Download the tutorial **signal_tap.pdf** from the Connect Site and work through it. This document will show you how to use the SignalTap II tool that is part of Quartus II. You can think of this code as an embedded logic analyzer that records the behaviour of selected signals while the chip is running.

Take a screenshot or photo (with your phone) of your waveform at the end, and show this to the TA during your demo session. There are 2 marks for doing this, and if you don't have evidence that you completed Task 3, you won't get those two marks (even if the rest of the design is working).

Task 4: Easy Changes (1.5 marks)

The game we have given you is boring. You are now going to make it a bit more fun. In this task, we will get you going, by suggesting two very simple changes.

1. Everything is black and white. Choose a new colour scheme. The ball and paddles should be different colours (not white). This is a trivial change. (0.5 marks)
2. Playing with only one ball on the screen is too easy. Add a second ball (so there are two balls moving on the screen at the same time). Set the start positions of the two balls so they are not on-top of each other. One of the balls should start moving up and to the right at 45 degrees, while the other ball should start by moving to the right and down at 45 degrees, as illustrated in the following diagram. Hint: this is fairly easy, and likely involves adding extra states to the existing state machine. If you find yourself adding a second state machine, you are probably making it much harder than it has to be. (1 mark)



Task 5: Shrinking Paddle (2 marks)

In the original circuit, the paddle is always 10 pixels wide (in the x dimension). Modify the circuit so that, every twenty seconds, the paddle gets 1 pixel shorter (when it gets to 4 pixels, it doesn't get any shorter than that).

Task 6: Fixed Point (2 marks)

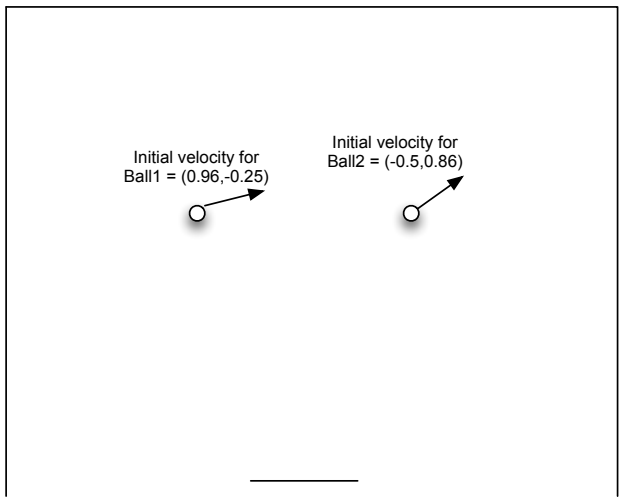
In the original circuit, the balls moves at 45 degrees only. However, a more realistic game would allow the balls to move at different angles. In this case, the two components of the velocity vector would not be limited to -1 or 1, but may take on fractional values. In this task, you are to generalize the angles at which the balls move.

The simplest way to implement this is to use a fixed-point representation for the ball position *and* the ball velocity. For each, I suggest using 8 bits to the right of the decimal point (this will mean adding bits to each register defined in **lab4_pkg.vhd**. Eight bits might sound like more precision than you need, but it will make Task 7 possible. You will likely need to make changes in both **lab4.vhd** and **lab4_pkg.vhd** to complete this task. Be sure to check out the slides related to fixed-point arithmetic that we will go through in class.

In the fixed-point implementation, when the ball collides with a wall or paddle, it should exit the bounce at the correct angle (so if it comes into the wall at 30 degrees, it should exit the wall at 30 degrees). Note this isn't as hard as it sounds: as an example, if the original velocity vector is ($\Delta X=0.7$, $\Delta Y=0.5$), then after a collision with the upper wall, the new velocity vector is ($\Delta X=0.7$, $\Delta Y=-0.5$). Note that the new direction can be computed by simply inverting the sign of ΔY, just as before!

Once you have your implementation, test your design with several initial velocity vectors for each ball. To get ready for Task 6, set your initial velocity vectors as shown in this diagram (this corresponds to 30 degrees for Ball 1 and 15 degrees for Ball 2).

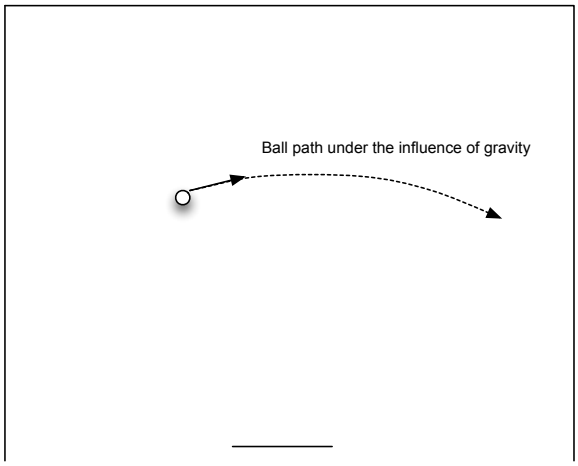
Hint: If you find yourself writing a lot of code for this task, you might be on the wrong track.



Task 7: Gravity (1.5 marks)

In the current version of the game, the ball travels in a straight line until it hits a wall. The game would be much more exciting if gravity pulled the ball towards the bottom of the screen. In this task, you will add gravity.

Recall that a gravitational force is an acceleration. From your first-year Physics courses, you should be able to recall the relationship between acceleration, velocity, and position. Using this information, add gravity to your game. The diagram below shows the ball path under the influence of gravity (only Ball 1 is shown, but gravity will affect both balls in the same way). Note that you have to be careful: if you add too much gravity, the game quickly becomes unplayable. You need to find a balance such that gravity is clearly evident when you demonstrate your game to the TA, but subtle enough that the game does not become unplayable.



Challenge Task: (1 mark)

Challenge tasks are tasks that you should only perform if you have extra time, are keen, and want to show off a little bit. This challenge task is only worth 1 mark. If you don't demo the challenge task, the maximum score you can get on this lab is 9/10 (which is still an A+).

In this challenge task, you are to be creative and enhance the game in a way of your own choosing. One suggestion might be the addition of some obstacles on the screen that your ball can bounce off of; another suggestion might be a second paddle (allowing two players to play). Don't go crazy here, you have other courses which are also important.

What to demo:

The marks for each task are as indicated in the Task descriptions. If you have completed the whole thing, you can demo the final working circuit, being sure that the TA has a chance to see that each change has been made. If you don't get every part working, demonstrate what you have, and the TA will give marks based on the mark breakdown.

Be sure to remember to submit your files to Connect immediately after your demo.