LAB 7

Real-World Interfacing with a Nintendo Controller

Value: 300 points (Plus up to 20 points of extra credit)

1 Introduction

The purpose of this lab is to learn how to design, develop and implement a sequential digital circuit whose purpose is to interface with an actual hardware device.

2 Preparation

- Read Chapter 7 of Wakerly, especially Section 7.6 on state machine synthesis.
- Reading section 2.16 of Wakerly is suggested as it provides some context for this assignment.

3 Design specification

3.1 Overview

The design task of this lab is to develop an interface to a Nintendo-16 controller. Your design will cause different LEDs to light up depending upon which button on the controller has been pressed. In addition, you are to design a state machine that observes the keys pressed on the controller. When a certain combination is observed, one of the 7-segment LEDs will light up.

3.2 Nintendo-16 Controller

The Nintendo controller has eight buttons: left, right, up, down, A, B, select and start. The controller is connected to the Nintendo Entertainment System (called *NES*) by seven wires. One wire is ground, one Vcc, and two are unused. The other three wires are called "Latch", "Pulse" and "Data". When the controller is connected to the NES, the NES asks it to send the current state of its buttons about 60 times a second (60 Hertz). The NES does this in two stages. First it sends a pulse on the *Latch* wire. One the rising edge of this pulse, the controller samples all eight buttons. The NES then sends pulses on the *Pulse* wire. For each rising edge on the Pulse wire, the controller sends information about a different button on the *Data* wire. For more detailed information, see http://web.mit.edu/tarvizo/web/nes-controller.html. A figure from that page can be found below.

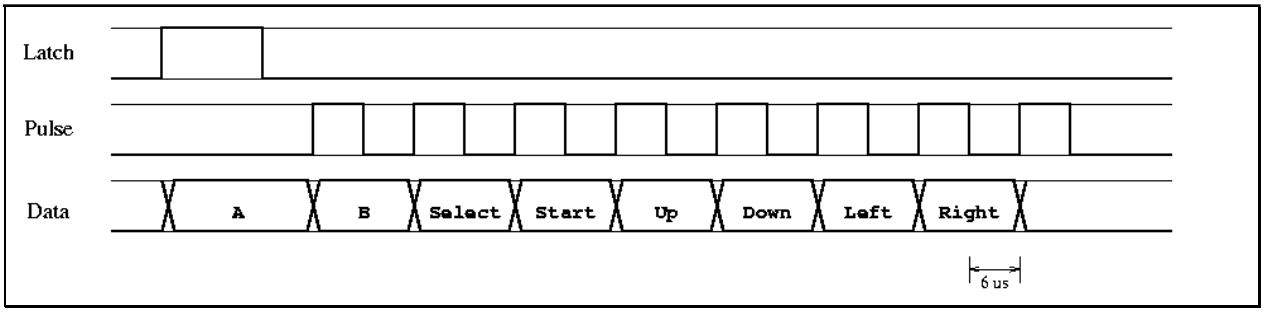

Figure 1: A controller transaction

3.2.1 Connecting the NES controller to the Xilinx board

Each lab station should already have a controller connected to the Xilinx board. Because all of the pins on the FPGA are already connected to some type of I/O device (LEDs, switches, buttons, etc.), we have elected to connect the controller to the same pins used by the first three dipswitches. *It is imperative that the first three dipswitches be in the **off** position throughout this lab. If they are in the on position the wires will be kept low at all times, and you won't be able to communicate with the controller!*

- The *Latch* wire is connected to DIPSW2 (pin 8) (pin 7 on the board)
- The *Pulse* wire is connected to DIPSW3 (pin 9) (pin 11 on the board)
- The *Data* wire is connected to DIPSW1 (pin 7) (pin 6 on the board)

In the event that the wires become disconnected, ask your GSI or instructor to reconnect them for you. For the generic controllers, the brown, yellow, red, white and blue wires are connected to ground, VDD, pin 6, pin 7 and pin 11, respectively. For the Nintendo® brand controllers, the brown, white, yellow, orange and red wires are connected to ground, VDD, pin 6, pin 7 and pin 11, respectively.

3.2.2 NES/controller communication[1]

The NES uses a serialized polling mechanism to query the state of the buttons 60 times a second. First, the NES sends a 12µs high signal to the Latch wire, telling the controller to latch the state of all buttons internally. 6 µs later, the NES sends 8 high pulses on the Pulse wire, each with a period of 12µs per full cycle and a 50% duty cycle.

After detecting the rising edge on Latch, the controller sends the status of the A button on the Data wire; Data is low if the A button is pressed, high if it is not. For each pulse on the Pulse wire, the controller drives the Data wire low if the button corresponding to that pulse was pressed. (The button states on Data are thus active low.) The button order is always the same: A, B, Select, Start, Up, Down, Left, Right. Again, see Figure 1 for more information.

3.2.3 Timing issues

The Xilinx chip has an internal 12MHz clock. It is connected to pin 13 and you can use it like any other input (add the appropriate line to your ucf file, have an IPAD and IBUF, etc.). In your design, you need not use exactly the same timing used by the NES. In general, the pulse widths for the Pulse signal should not be less than 6µs, and the pulse width for your Latch signal should not be less than 12µs. Also, you at a rate near 60Hz, but sampling as slow as 30Hz will work just fine for this project. Finally, depending upon your implementation, it is probably best to have the

[1] Portions of this section are taken from http://web.mit.edu/tarvizo/web/nes-controller.html.

whole polling time (the time from the rising edge of Latch to the last falling edge of Pulse) be no more than 5% of the time between Latches (so the controller is idle 95% of the time). See Figure 2 for an illustration of this idea.

You will need to divide the clock frequency to a speed that is appropriate. You will need to be very sure that your clock does not "glitch." Use the techniques discussed in class, latching the output or using T-flip-flops, to avoid this glitching. Note that you have two main timing requirements to meet—the sampling rate (i.e., the time between Latch pulses) and the pulse widths and periods for Latch and Pulse—so you may need multiple divided clocks.

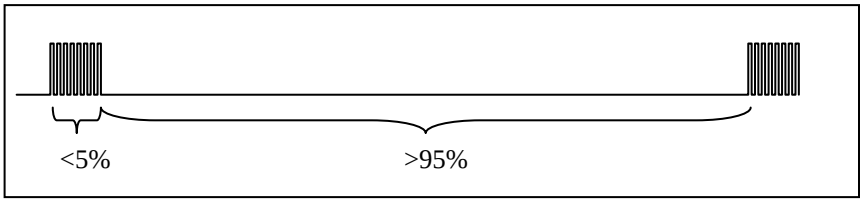

Figure 2: Pulse timing

3.2.4 Generating Latch and Pulse

You will need to generate the Latch and Pulse signals so that the controller responds correctly. Assuming you have managed to divide the clock to an appropriate frequency, you are left with a number of options. You could design a state machine which generates Latch and Pulse. This state machine would restart every time you desired to sample the controller, and its outputs would be the Latch and Pulse signals.

Another option is to use a counter (which is of course a state machine itself) to generate Latch and Pulse. Again, you would need to be sure you are only generating these signals when sampling the controller.

3.2.5 Reading Data

Once you correctly generate the Latch and Pulse signals, the controller will transmit the button status on the Data wire. You will want to use a shift register to read each bit from the Data wire. You should give careful consideration to how to clock the shift register.

3.2.6 Displaying data to the LEDs

You are to map the button data to the bar LEDs so that when a given button is pushed some corresponding LED is lit. You may map the buttons to the bar LEDs in any way you see fit so long as each button lights a unique LED.

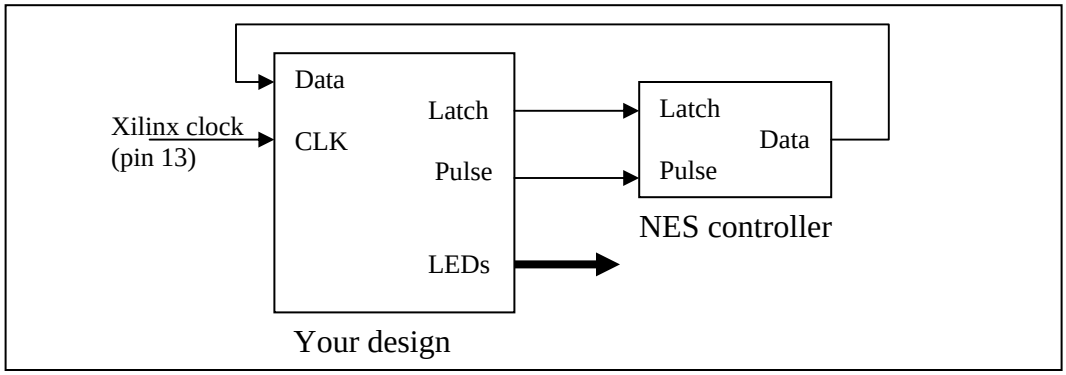

Figure 3: How the NES controller interacts with your design

3.3 Easter Egg

3.3.1 Background

> ***Easter egg:***
> *Programmers sometimes put pieces of code into an application that produce unexpected surprises. Users usually won't see them unless they press a certain key combination. They're usually found by word of mouth and rarely by accident.*
> www.pccomputernotes.com/pcterms/glossarye.htm

Many games have secret codes that cause the game to behave differently. These "Easter Eggs" are activated by pressing a certain sequence of keys. Most NES games have Easter Eggs of one type or another in them.

3.3.2 Requirements

Each student will be assigned a different 4-key code. When that code is entered into the NES controller, a single segment of one of the 7-segment LEDs should be lit—you choose which one. The user must press the right buttons in order and press no other buttons between correct presses for the entry of the code sequence to be considered correct. The segment on the 7-segment LED should remain lit until the start button on the controller is pressed. In all cases, the bar LEDs should continue to function as before. Once you have successfully implemented the required functionality, you can improve upon this Easter Egg feature to earn a few extra credit points. Information is available in the Deliverables section of this lab.

4 Design Notes, Hints and Restrictions

- Do the pre-lab questions before attempting the lab.
- Getting the Latch and Pulse outputs correct is one of the most difficult portions of this lab—it is easy to underestimate! Deciding between using a counter, a state machine, or some other technique to generate Latch and Pulse is something you should spend some time thinking about (and designing) before you start.
- It is strongly suggested that you test your generation of Latch and Pulse in simulation before trying to interface to the board. In simulation, you will want to use the divided clock as an input rather than the system clock—in other words, you should assign a stimulator to the divided clock. If you have multiple divided clocks, use the fastest one as the simulator input.
- We do have oscilloscopes which can be useful in debugging your design once you actually start trying to communicate with the controller. We will *only* use them with you if you can demonstrate that your design works in simulation!
- When doing your Easter Egg you may find it helpful to drive a different LED depending upon which state you are in so you can see the state changes occur. This can be very helpful in debugging.
- Be sure you keep track of which inputs and outputs are active low!
- *It is imperative that the first three dipswitches be in the **off** position throughout this lab. If they are in the on position the wires will be kept low at all times, and you won't be able to communicate with the controller!*

5 Deliverables

5.1 Pre-lab (110 points)

5.1.1 Questions (20 points)

A) What is the *period* of the system clock? **(2 points)**
B) In order to generate a clock that has a period of 12µs, by how much would you have to divide the system clock? **(3 points)**
C) Now, assume you can only divide the system clock by a power of two. What would be the best value to divide by if you want a clock period as close to 12µs as possible without going under that value? **(4 points)**
D) If you never sent anything on Pulse (i.e., Pulse is always 0), but continued to send high pulses on Latch at a rate of 60 Hz, what would you expect to happen on Data? Explain. **(6 points)**
E) In order to read information from the Data wire, you will need to clock a shift register. How will you generate the clock for the shift register? **(5 points)**

5.1.2 Simulation (80 points)

You are asked to turn in two simulation printouts. During simulation you should be using the *divided* clock as an input (if you have multiple divided clocks, use the fastest one) and not the *system* clock!

1. Turn in a simulation which shows your Latch and Pulse signals being generated correctly to perform a single polling operation of the controller. Your simulation should look much like Figure 1 but will not include the data from the controller. **(20 points)**
2. Turn in a simulation which shows your Latch and Pulse signals being generated correctly to perform two polling operation of the controller. Your simulation should look much like Figure 2 but should show Latch and Pulse rather than just Pulse. **(20 points)**
3. Turn in a simulation which shows: (**40 points)**
   - Your Latch and Pulse signals being generated correctly for a single polling operation.
   - You simulating Data returning all of the directions (Up, Down, Left, Right) not being pressed and all of the other buttons (Start, Select, A, and B) being pressed.
   - The value of your shift register during this time as a binary or hex number.

5.1.3 Supporting documentation (10 points)

Turn in a printout of your top-level schematic, any Verilog modules you wrote or other macros you designed, and your ucf file. These should be printed at scales and orientations that are readable.

5.2 In-lab (110+ points)

Download the bit file for the Nintendo Controller to the XESS board and verify that your implementation is working properly. There are two parts to the in-lab assignment. The first is to demonstrate that your bar LEDs are lighting when they should. **(60 points)** The second is to demonstrate that your Easter Egg works correctly. **(50 points)**

Further, you may earn up to 20 points of extra credit for doing one or more of the following with the Easter Egg. Note that these assignments can be fairly difficult. *Extra credit will not be accepted after the due date for the in-lab.*

- Perform a "light show" of some sort (flashing various 7-segment lights) rather than lighting a single LED when the correct code is entered. Again, the flashing should

stop when start is pressed and should not interfere with the behavior of the bar LEDs. **(5 to 10 points depending upon the quality of the light show (at the GSIs discretion))**

- Require that each button be pressed within one second of the last one being pressed. **(10 points)**
- In addition to responding to the code we gave you, you can have your design react to a different code that involves multiple buttons being pressed at once. This code should involve at least two different multiple button presses. The old code must also continue to work. **(10 points)**

5.3 Post-lab (80 points)

Prepare your lab report as described in the *EECS270 Laboratory Overview* handout. Make sure you complete and include all parts of the report including the *Cover Sheet,* the *Design Narrative* **(30 points)** section, a corrected pre-lab (**20 points)** and the *Design Documentation* **(10 points)** section. Because of the design-intensive nature of this lab, we expect that your Design Narrative will focus primarily on false starts, problems encountered, and design decisions made.

5.3.1 Under the hood (20 points)

Modify your design so that if the SPAREB button is pressed, the clock to all devices is divided by an additional factor of 256 (this should only involve adding a clock divider and a MUX.) Describe the behavior of the bar LEDs when the button is pressed. Explain this behavior.