Lab1: Review of Logic Circuit Design

Introduction

The purpose of this lab is to review digital circuit design concepts from the ECE241 Digital Systems course, including Verilog, finite state machines, and good design practices like control/datapath separation. It will also introduce you to the SystemVerilog language extensions to Verilog, which helps make your design experience more productive and less error-prone. See the accompanying *SystemVerilog Tutorial* before attempting this lab. We will use the *Quartus Prime* design software with which you became familiar in ECE241. Make sure you allocate enough time to review tutorials on how to use the software.

Part I

You will design a "guess the number" game and simulate the game using Modelsim: the circuit chooses a random 8-bit number, and the player needs to guess it. The player enters their guess and uses "enter" signal to confirm. The game, then tells the player whether their guess is under, over, or equal to the correct value. In the initial version of the game, the player has unlimited tries and the game stops responding when the number is guessed correctly. Random number generation is achieved by simply incrementing a counter continuously until the user provides the first "enter" signal. The number at which the counter stops counting might as well be random.

Figure 1 is a block diagram of the overall system. For Part 1, the source code is provided to you in the starter kit, and you will only need to modify the `control` module. Another purpose of this first lab is to (re)familiarize you with good hardware design practices, which include good use of module hierarchy and the naming and purpose of signals. To make the grading process simpler, the actual number that the user needs to guess is outputted, but this should not be the case for designing the game in reality.

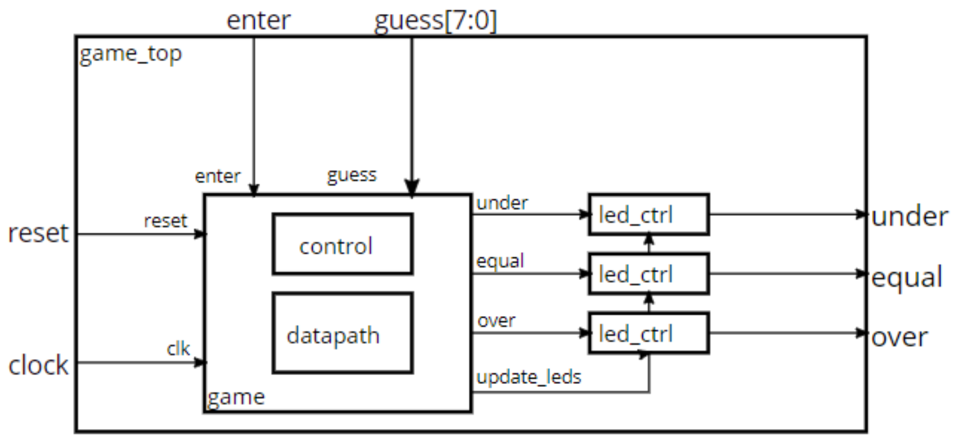


Figure 1: Top-level block diagram of the guess-the-number game

Note that game contains a separate control and datapath module, which will be discussed below.

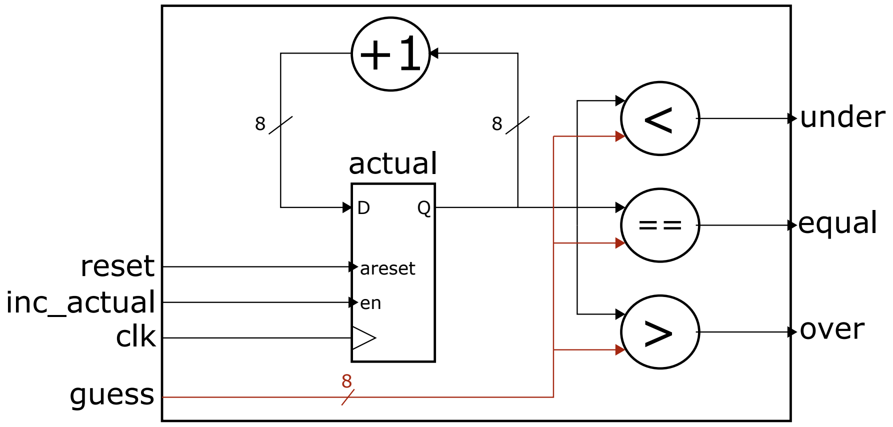


Figure 2: Game datapath

Figure 2 shows the datapath module for the first version of the game. Your control logic should initially keep the `inc_actual` signal high until the user enters their first guess. This will have the 8-bit `actual` register land on an effectively-random value, and it should stay constant throughout the rest of the game.

The `control` module is a finite state machine responsible for implementing the game logic. The file provided in the starter kit is a bare template to remind you of the generic structure of every FSM you will make, with the necessary input and output signals already declared. You will need to add states and functionality to make the game work.

Part II

Modify the game such that the player only gets a maximum of 7 attempts to guess the number. The remaining number of tries should be an output of the game as shown in Figure 3. The game ends when the player either runs out of attempts, or guesses the number correctly, at which point the game stops responding and must be reset. These new game features will require you to understand and modify the control, datapath, and top-level modules.

```
module top
(
    // Clock pins
    input                   clk,
    input                   reset,
    input                   enter,
    input             [7:0] guess,
    output                  dp_over,
    output                  dp_under,
    output                  dp_equal,
    output            [3:0] dp_tries,
    output            [7:0] actual
);
```

Figure 3: Part 2 top module declaration

Instructions

Preparation

- Re-familiarize yourself with the Quartus Prime software by reading the *Quartus Prime Introduction Using Verilog Designs* document, as well as the *SystemVerilog Tutorial*.
- Download the starter kit for this lab. Using lab1.sv as the template for part1 and part2
- Complete the provided SystemVerilog code to implement Part I. Name the finished code part1.sv
- Enhance the code for Part I to implement Part II. Name the finished code part2.sv

In-Lab

Go through the provided tester for both part1.sv and part2.sv to make sure that your code is able to be recognized by the automarker

Submission

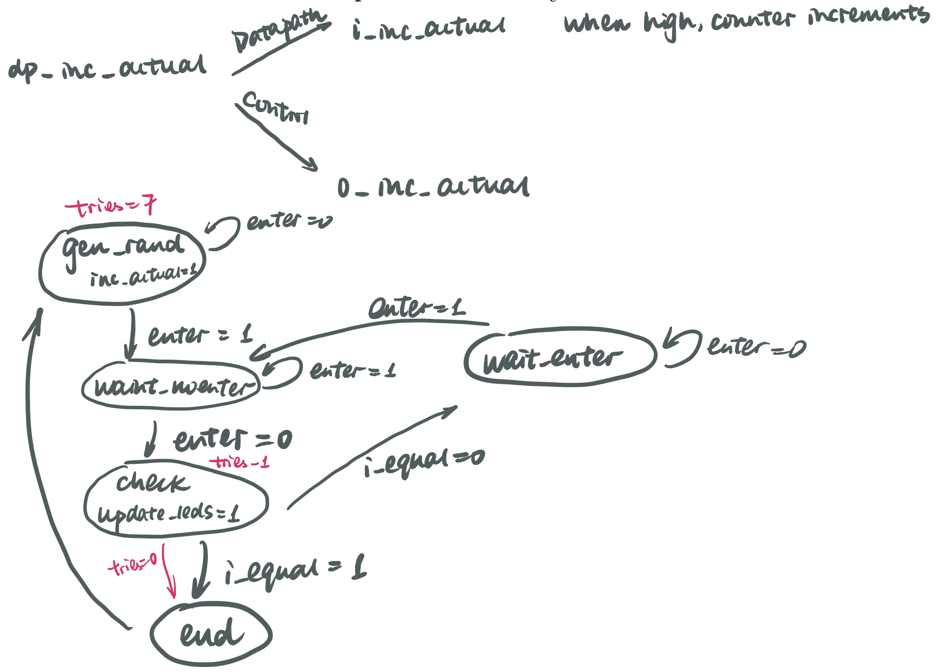
Your should submit part1.sv and part2.sv

Frequently Asked Questions

In this section, questions from previous years have been collected and answered. Please check here first before posting a question on Quercus.

Q1: Does the game need to be reset after 'winning' to continue playing?

A1: Yes, as it is designed now the game needs to be reset after winning to play another round. This was done for ease of implementation only.



check guess does not change after ends