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 for the DE1-SoC board: the circuit chooses a random 8-bit number, and the player needs to guess it. The player enters their guess using `SW[7:0]` and uses `KEY[1]` to confirm. The game, via 3 red LEDs, 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 ends when the number is guessed correctly. Random number generation is achieved by simply incrementing a counter continuously until the user provides the first key press. The number at which the counter stops counting might as well be random. When the player chooses a number with the switches, the number they are choosing is displayed as two hexadecimal digits on the HEX displays `HEX1:HEX0`.

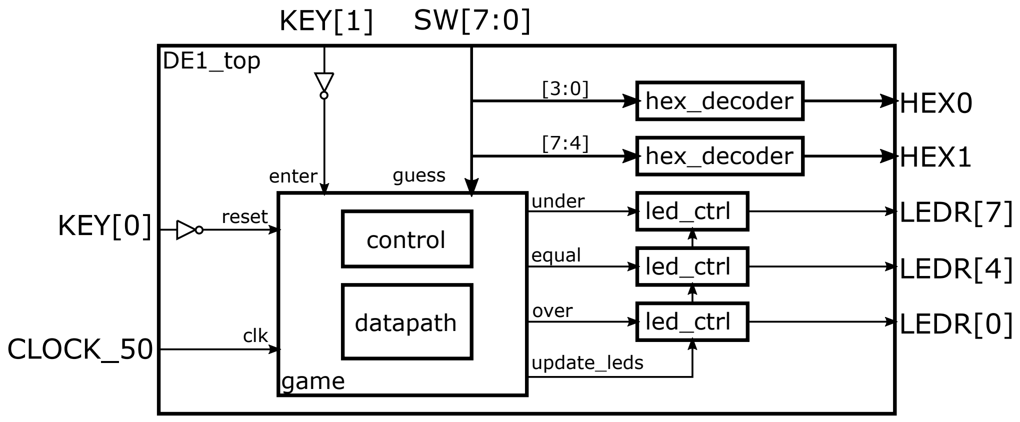

Figure 1: Top-level block diagram of the guess-the-number game

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.

Here are some things you should note about the structure of this example game project:

- The actual functionality of the game is **not** implemented directly in the top-level `DE1_top` module, but rather in an inner `game` module.
- Top-level input and output signals, which are board-specific, are immediately renamed to game-specific signals that relate to their purpose, for example: `KEY[1]` to `enter` and `SW[7:0]` to `guess`.
- Active-low signals coming from the board (the two `KEY` inputs) are immediately converted to active-high, in addition to being renamed.
- The game itself contains a separate control and datapath module, which will be discussed below.

These structural design choices help you with debugging. For example, if you needed to simulate the circuit to track down bugs, you could now simulate the `game` module rather than the entire `DE1_top` module – no need to remember that the reset is active-low (since all signals have been converted to active-high), or remember which `KEY` does what (the signal names make it clear).

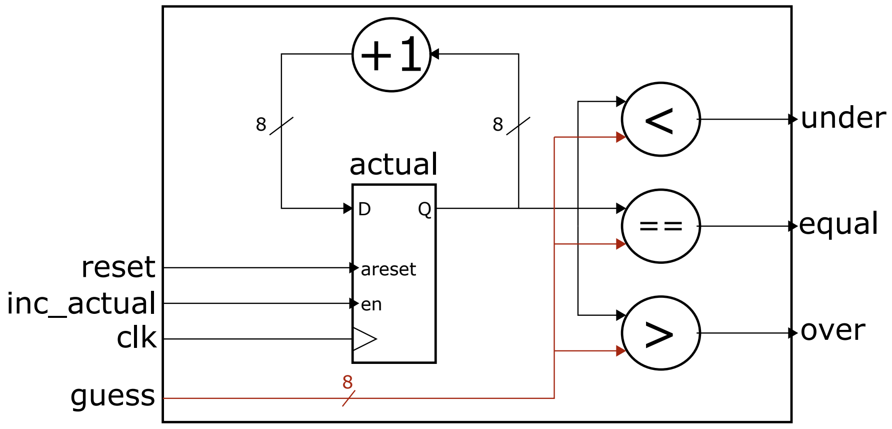

Figure 2: Game datapath

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. Display the remaining number of attempts on `HEX5` (the left-most one on the board). 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.

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.
- Complete the provided SystemVerilog code to implement Part I.
- Enhance the code for Part I to implement Part II.

In-Lab

Demonstrate a working Part II on the DE1-SoC board to your TA and answer their questions. Part I may be demonstrated for partial marks if Part II does not work.

Marking Scheme

- Preparation: Show complete code for Part II (2 marks)
- In-Lab: Demonstrate a working Part II on DE1-SoC board (2 marks)
- In-Lab: Answer TA questions (1 mark)

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.

Q2: When I press a KEY, the counter seems to jump to a 'random' value. The LED also flickers. However, this works fine in ModelSim. What could be the issue?
A2: In ModelSim you are likely proving KEY as an input for a single clock cycle. However, when you press a KEY on the FPGA, you are providing that input for many hundreds of cycles. So your design jumps a lot of states before you see it. One way to avoid this is to have two states to collect an KEY input; one state when the KEY is pressed and then another state for when the KEY is released. This will ensure that you will only see one KEY 'press' on the board.