7 Project

7.1 Abstract

The last experiment in this course is a moderate-sized project that will give you a chance to build something non-trivial and fun. Your task is to design and implement a 2-player version of the game *Simon*. You will implement the game as a state machine on the FPGA board. The supporting hardware involves 4 LEDs and is interfaced to the computer. Game controls and the user interface are programmed in C.

7.2 Introduction

For those not familiar with the original version, the objective is to repeat a sequence of lights which becomes longer each time it is entered correctly. There are 4 LEDs to indicate the sequence.

Player 1 starts the game by entering entering the first combination. The game then replays the combination after a short pause and waits for player 2. Player 2 has to repeat the previous sequence and then adds one combination at the end. The game replays the current sequence and it is player 1's turn again. This cycle repeats as long as both players correctly repeat (and extend) the sequence. The game is over when the pattern is repeated incorrectly or a player takes too much time (time-out).

In the conventional *S*imon game, a valid input consisted of exactly one of the four lights. Here, we introduce two difficulty levels. In *easy* mode, the input is exactly as before. However, in *difficult* mode, one or more lights can be entered as a new combination in the sequence.

7.3 State Machine

Program the game *Simon* as a state machine in Verilog. The state machine must implement the following functions:

- Two difficulty levels (easy and difficult).
- Storing and extending the current sequence upon player input. For our purpose, a maximum sequence length of 32 should be sufficient.
- Replaying the current sequence.
- Track the current player's input and verify if the sequence is repeated correctly.
- Enforce a time constraint on the player input.
- Indicate when the game is over.

Break your state machine down into several major functions. What are they? Each of those functions should be represented by a state in your state machine. Subfunctions of that, such as replaying or verifying the different combinations in a sequence, can be implemented using counters while remaining in the same state.

The time constraint can be implemented with a counter internal to the state machine. Starting from zero, have the counter count up until an input is detected or the maximum count value is reached. In this lab, the timeout should occur between 2 and 3 seconds. Your clock will probably be much faster (around 30–60Hz), so you need to adjust your maximum count value.

To keep the computer synchronized with the state machine, the FPGA should indicate its state. The software will need to know:

- when the game is over
- if the FPGA is expecting a new input
- the FPGA should signal when a valid input combination has been received
- when the FPGA is replaying the sequence

Here is a list of suggested input and output pins on the FPGA. Note that this is not necessarily complete and that you might need more signals than that:

- CLK: for clocking your state machine from the computer
- RESET: to reset the state machine and start the game from scratch
- LEVEL: to indicate the difficulty level at the start of a new game.
- A, B, C, D: four input lines corresponding to the four lights that the player can enter.
- LEDA, LEDB, LEDC, LEDD: four output lines to the LEDs.
- GAME: to indicate that the game is over.
- REPLAY: to signal the computer that the state machine is currently replaying the sequence and not expecting input.
- STEP: signal from the FPGA to the computer that a correct input has been received and that the user can enter the next combination. This signal should only be high for one clock cycle, when the state machine starts waiting for a new input, and go low again afterwards.

7.4 Hardware

The hardware for this circuit is relatively simple. It consists of the FPGA board, a setup for 4 LEDs and the I/O-board interface to the computer.

Plug the FPGA board into the breadboard and establish a common ground connection by wiring the FPGA board's ground pin to the power supply ground. Be careful not to short-circuit any power supply pins. A good way to protect your FPGA board is to insert

buffers between the board and your hardware for all signals. If something goes wrong, the buffers will burn and save the FPGA (and you money).

Put the 50-pin connector into the breadboard and also connect it to the common ground. As a precaution, use buffers for all signals between the I/O connector and the FPGA board.

The four LEDs should be driven by the FPGA. You can either use positive logic ("1" lights an LED) by connecting one LED terminal to the buffer line from the FPGA and the other terminal to ground; or you can use a negative logic setup ("0" lights an LED) by using inverting buffers (or inverted FPGA outputs) and connecting the other LED terminal to ground.

7.5 C/C++ Program

The C/C++ program should initialize your hardware and provide all necessary input and control signals during operation. It also operates as an input terminal to the game; i.e. it should indicate the game's status and difficulty level, take keyboard inputs to indicate which of the four lights have been pressed etc. The core functionality should be implemented in the state machine, so your program should not store data or implement any parts of the state machine.

The demo program that you have from experiment #3 shows how to generate a clock. There are 60 or more ticks per second, so you can easily generate a clock that is fast enough.

Note that on the computer, at most one key is entered at any given point in time. Combinations of regular keys, like you will have to enter them in the difficult game mode, will in fact appear as two separate key strokes. Your program will have to handle that condition accordingly. A fast clock actually comes in handy here. If you design your state machine to accept partially correct inputs (i.e. wait for the remainder of the input), then the problem of sequential key strokes is easily solved.

Your program has to check the status of the state machine. It needs to reset the inputs to the FPGA when the state machine signals a change in state.

7.6 Report

Part of your grade for the project is based on a brief report. This report should include the following items:

- A description of your design. Discuss your key design choices/decisions that you made and explain why.
- A state diagram of your state machine.
- A description of the interface between the PC/software and the FPGA board.