Class Report 1: Rotating Square Circuit

Josie D'Acquisto
ECE, Baylor University

September 11, 2025

Contents

1 **Introduction** 1

2 **Background** 1

3 **Problem Statement** 2

4 **Solution Methodology** 2

5 **Experimental Results** 2

6 **Discussion** 2

1 Introduction

This report describes the design and implementation of a rotating square circuit on a four-digit seven-segment LED display using SystemVerilog. The project involved creating a finite-state machine (FSM) that cycles through specific square patterns depending on two input signals: `en` (enable) and `cw` (clockwise). A combination of hardware description language techniques, testbench simulation, and FPGA prototyping were used to develop and verify the circuit.

GitHub repo: `https://github.com/josietwirls/SoC.git`
Oral presentation: `https://youtube.com/shorts/Gxc0wPiHQPU?feature=share`

2 Background

Seven-segment displays are a common way to represent numeric or symbolic patterns in digital circuits. Each digit is composed of segments labeled `a--g` (plus a decimal point), which can be enabled individually. Complex patterns are created by selectively turning segments on and off. In this problem, two distinct square-like patterns were identified: one formed with `a, b, f, g` and another with `c, d, e, g`.

A key idea from the textbook *FPGA Programming by SystemVerilog Examples* is the use of counters and finite-state machines (FSMs) to generate sequential behavior. By using the most significant bits of a counter as the "state," one can naturally cycle through output patterns at a human-visible refresh rate. Listing 4.15 in the textbook demonstrates how to use counter bits as control signals, which guided this project's design.

3 Problem Statement

In a seven-segment LED display, a square pattern can be created by enabling the `a, b, f, g` segments or the `c, d, e, g` segments. We want to design a circuit that circulates the square patterns in the four-digit seven-segment LED display. The circuit should have an input `en`, which enables or pauses the circulation, and an input `cw`, which specifies the direction (i.e., clockwise or counterclockwise) of the circulation.

4 Solution Methodology

To solve this problem, I first sketched the design on a whiteboard to map out the square circulation. I recognized that each of the four digits would require its own state, and that the circuit would need to change state depending on the values of `en` and `cw`.

My approach was to use a counter whose top three bits (`q_reg[N-1:N-3]`) serve as the state register. This decision simplified the FSM because the states naturally cycled with the counter. The `en` signal controls whether the counter advances, while `cw` determines whether the counter increments or decrements. This structure matched the example provided in Listing 4.15 of the textbook, which reinforced the design strategy.

After defining the states, I mapped each state to the correct digit and segment pattern. I then implemented the module in SystemVerilog, followed by a testbench to verify functionality before deploying to the FPGA board.

5 Experimental Results

The simulation testbench successfully demonstrated correct behavior. The waveforms confirmed that when `en` was low, the state held constant, and when it was high, the states advanced or reversed depending on the `cw` input. Reset behaved as expected, clearing the counter.

When programmed onto the FPGA board, the circuit produced the desired square rotation across the four-digit display. The rotation paused when `en` was disabled and reversed when `cw` was toggled. Overall, the experimental results aligned with the design objectives.

6 Discussion

This project showed how FSM principles and counter-based state encoding can be applied to control seven-segment displays. The design is simple yet effective because it uses the counter's high-order bits as the state, avoiding the need for a separate FSM controller. The testbench proved valuable in verifying the behavior prior to hardware testing, reducing debugging time on the board.

Future improvements could include adjusting the counter's bit width to fine-tune the speed of rotation, or extending the design to include additional patterns beyond squares. This exercise reinforced both theoretical understanding from the textbook and practical FPGA design skills.