ECE 2220 Final Design Project

Burke Libbey
John Ermitanio
Crystal Kotlar

December 3, 2007

Contents

1 Introduction **2**

2 Design and Implementation **2**
2.1 Overview . . . 2
2.2 Input . . . 3
2.2.1 Keypad . . . 3
2.2.2 Pushbutton . . . 3
2.3 Logic . . . 3
2.3.1 Selector . . . 3
2.3.2 Comparator . . . 4
2.3.3 Zero-overflow Detection . . . 5
2.4 Output . . . 5
2.4.1 LEDs . . . 5
2.4.2 Seven Segment Displays . . . 5

3 Unimplemented Ideas **5**
3.1 BCD Zero Counter . . . 5

4 Problems we Encountered **6**
4.1 Manual Input . . . 6
4.2 Quartus Quirks . . . 6
4.3 D Flip-flop Initial Condition . . . 6

5 Simulations **7**

6 Conclusion **7**

List of Figures

1 Overview of Project Architecture . . . 2
2 Selected encoded sequences . . . 3
3 Selector . . . 4
4 Comparator . . . 4
5 State diagram for zero detection . . . 5
6 Possible SSD output values . . . 6
7 Simulation of Zero-detection Finite State Machine . . . 7
8 Simulation of Comparator . . . 7
9 Full schematic of final circuit . . . 8

1 Introduction

Aircraft often use Non-Directional Beacons (NDBs) to determine their position. These beacons transmit a morse code callsign, the strength of which is used to calculate the distance from the beacon to the aircraft. It is very important that these beacons continue to function properly, so an autonomous beacon monitoring system is necessary.

Using an Altera FPGA and Quartus II software, our team designed a prototype system to autonomously monitor the output transmission of a selected aircraft navigation beacon.

2 Design and Implementation

2.1 Overview

Two distinct two-character sequences are stored in a series of shift registers. A keypad is used to select one of the two sequences, which is then sequentially unloaded into the comparator circuit, which compares the stored sequence against the incoming morse code input from a pushbutton. If the two signals differ on a negative clock edge, an error condition is raised and the system waits for a reset. If the signal matches successfully, a green LED is lit and the system waits for keypad input to begin listening for another sequence.

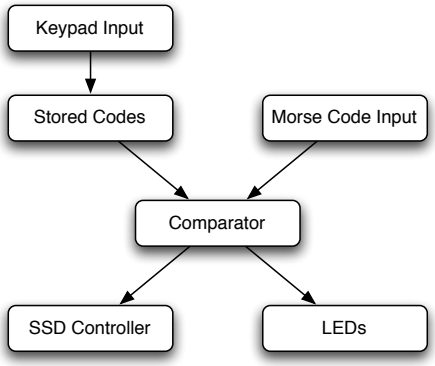


Figure 1: Overview of Project Architecture

2.2 Input

2.2.1 Keypad

The '0' (AB) and '1' (CD) buttons on the project board keypad are used to select one of two stored sequences. These sequences are stored using a modified morse code, defined by the following rules:

- Dashes convert to three consecutive '1's.
- Dots convert to a single '1'.
- A single '0' is inserted between each dot or dash.
- Three consecutive '0's are inserted between letters.
- Seven '0's are inserted after each word.

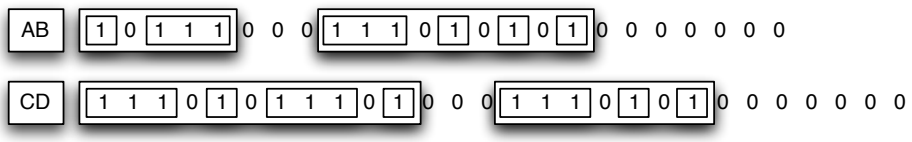


Figure 2: Selected encoded sequences

When one of the defined buttons on the keypad is pressed, both shift registers begin simultaneously unloading into the *selector* (§2.3.1) on the positive clock edge.

2.2.2 Pushbutton

A pushbutton is used as the morse code input (analogous to the transmitted signal from the beacon), for comparison against the selected stored input. This input is passed directly to the *comparator* (§2.3.2).

2.3 Logic

2.3.1 Selector

When the '1' key is pressed, a D flip-flop is set to High. It is reset to Low when '0' is pressed. This flip-flop serves as the select input for a multiplexer fed both of the now-unloading shift registers. The net effect is to output 'AB' or 'CD' from the multiplexer, depending on whether '0' or '1' is pressed.

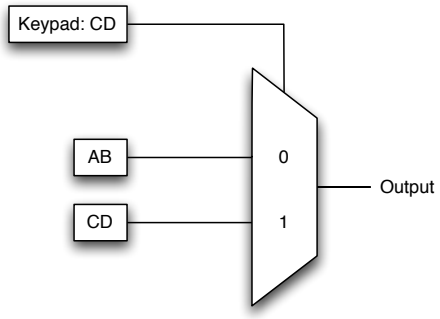


Figure 3: Selector

2.3.2 Comparator

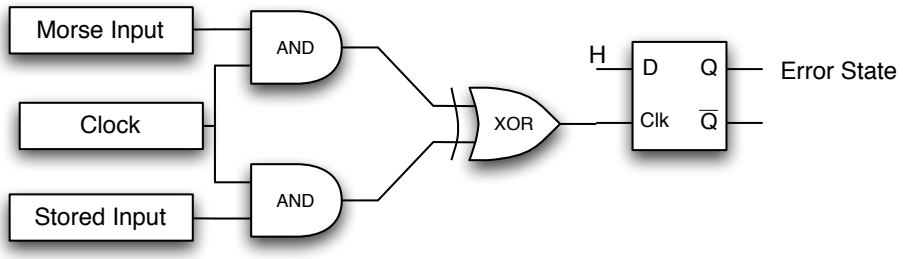


Figure 4: Comparator

The comparator is used to compare the selected stored input and the incoming morse signal. Each of the inputs is `AND`ed with the 1Hz clock and fed to an `XNOR` gate. This gate will output High when there is a difference in the incoming data. This output is passed to a D flip-flop such that the `Q` output of the flip-flop is initially Low, then goes permanently High the first time the Data line (`XNOR` output) goes High. The system then waits for a manual reset, as an error has been encountered.

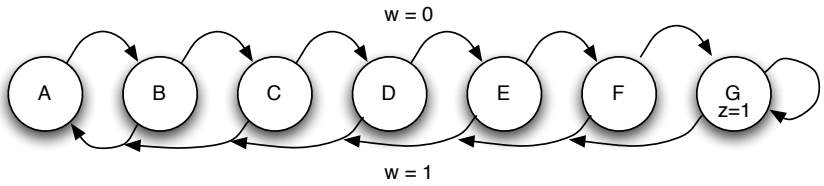


Figure 5: State diagram for zero detection

2.3.3 Zero-overflow Detection

This subsystem runs parallel to the comparator. When a sequence of seven sequential zeroes is encountered and no errors have been raised, the output is verified. If a sequence of 8 or more zeroes is encountered, a timeout has occurred and an error is raised. The zero-overflow detection circuit was implemented with a finite state machine.

2.4 Output

2.4.1 LEDs

Three LEDs are used:

- **Green**: Indicates that the previous sequence was correct.
- **Yellow**: Indicates that a sequence is currently in transmission.
- **Red**: Indicates that an error occurred and the system must be reset.

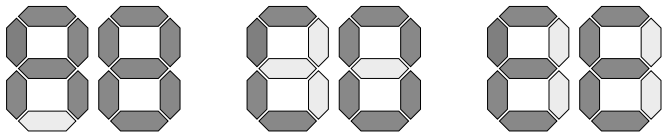
2.4.2 Seven Segment Displays

The Seven-Segment Displays on the project board are used to display the sequence currently being transmitted, the sequence successfully matched, or 'EE' in case of an error condition. *(Fig. 6)*

3 Unimplemented Ideas

3.1 BCD Zero Counter

The idea: To use a BCD counter to detect 7 and 8 sequential zeroes on the morse code input line (§2.3.3). Given the outputs $\{A, B, C, D\}$, a sequence of seven zeroes is given by ABC, and eight by D.



(a) AB (b) CD (c) EE

Figure 6: Possible SSD output values

Why it wasn't implemented: The project requirements specified that we must use at least one finite state machine. We felt we had found better solutions for every opportunity, and this was the one we felt least strongly about, hence, it was implemented with a FSM rather than a BCD.

4 Problems we Encountered

4.1 Manual Input

The first significant problem we had related to the timing of the shift registers that stored the encoded morse code. Originally, the shift register enable was tied to a switch (S_{6a}). The switch was initially low, then set high when the registers should start unloading. This meant that to start unloading the shift register at the appropriate time, one would have to press the keypad button and flip a switch at the same time. This problem was corrected by passing our two keypad inputs each into a D flip-flop to store whether they had been pressed yet. The `OR` of these two flip-flops was used in place of the switch we had used initially.

4.2 Quartus Quirks

The single most common error we encountered was in copying and pasting parts of our schematic between *Block Diagram Files*. Lines would often accidentally be crossed in very inappropriate places. This was often caught with a compile-time error, but on a couple occasions, we had to carefully scan our circuit for unintentionally crossed lines.

4.3 D Flip-flop Initial Condition

We ran into trouble several times trying to use D flip-flops as latches. Our intention was to have Q stay low until the first high was encountered on

D, then to have Q stay permanently high. We tried several configurations, each with very limited or no success. In the end, we succeeded by feeding the inverted output line back to the PRN.

5 Simulations

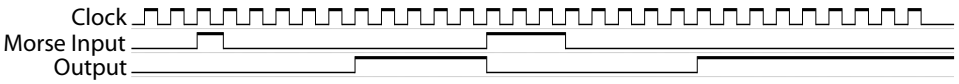


Figure 7: Simulation of Zero-detection Finite State Machine

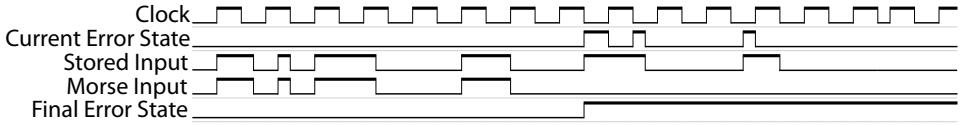


Figure 8: Simulation of Comparator

6 Conclusion

This project was an exercise in debugging. Most of our problems arose from not testing component circuits thoroughly before combining parts. Much effort could have been saved if we had done more testing along the way. In future projects, we would run tests much more frequently.

Additionally, we discovered that partitioning a poorly-understood project into poorly-understood component tasks does not work well. In the future, we would spend more time as a group trying to fully understand the scope of the project.

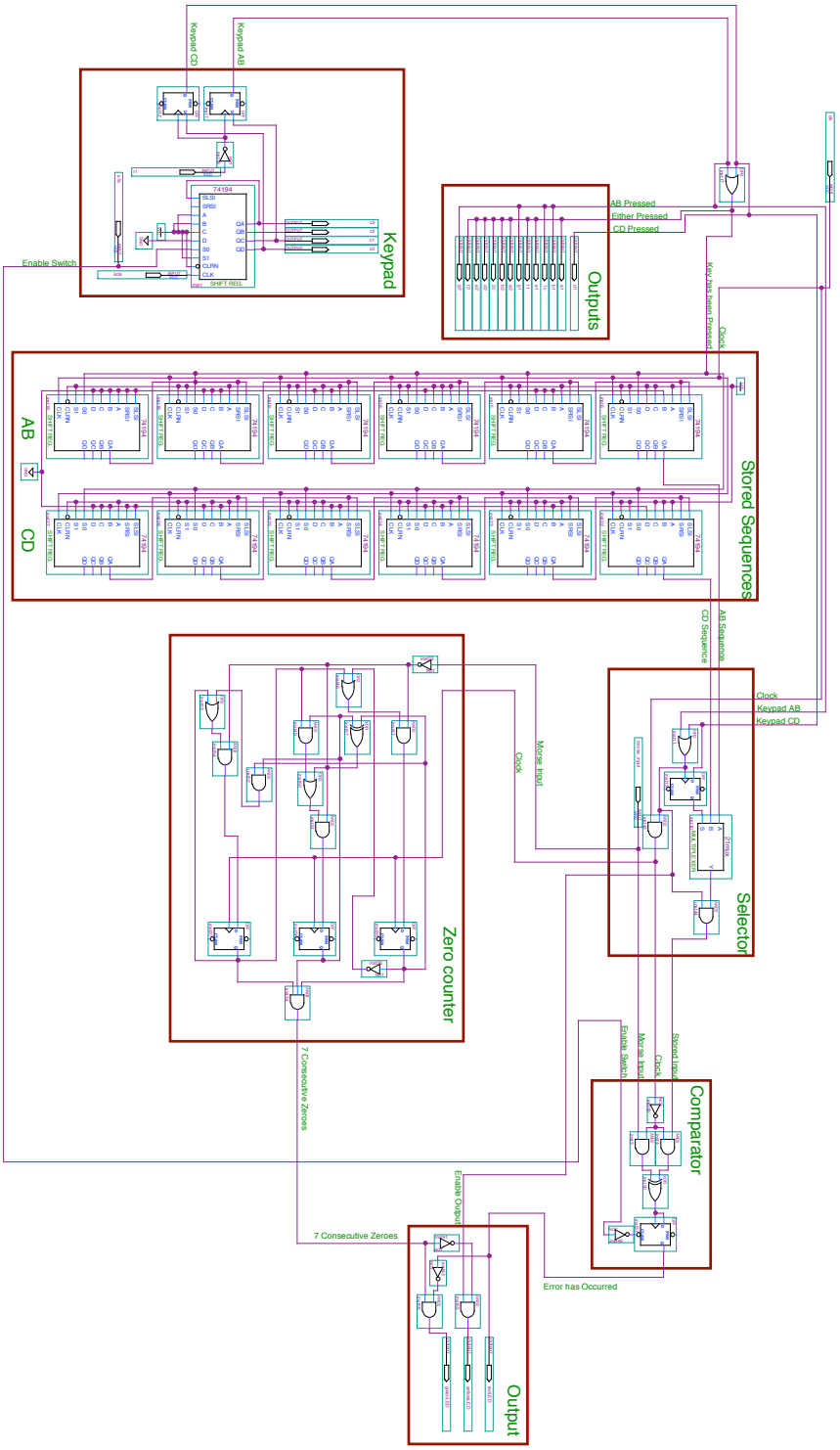


Figure 9: Full schematic of final circuit