Digital Logic Design Project 2

Zidane Karim

November 18, 2024

Contents

1 Introduction

The assignment was to design a sequential logic circuit that can both write and read a byte into a RAM chip, to be displayed on an array of LED outputs.

In other words, the circuit should be able to take in an 8-bit input and store it in memory, and then be able to read that memory and display it on the LED array.

For example, if the input is 10101000, the LED array should display the following configuration: ↓

Figure 1: LED Output for 10101000

The heart of the problem is manipulating the data around the RAM chip, as it was a 1024×4 RAM chip, meaning it had 1024 memory locations, each with 4 bits of data.

However, recall that the input is 8 bits, so we need to split the input into two 4-bit chunks to store in the RAM chip.

The circuit should be able to write the first 4 bits into the RAM chip, and then write the second 4 bits into the RAM chip, and then read the data from the RAM chip and display it on the LED array.

This manipulation of data is the main challenge of the project, which is why we turn to a sequential finite state machine to solve the problem.

1.1 Outline

1.1.1 Methodology

The circuit is divided into three main parts: the input, the RAM chip, and the output. The essential issue in each was:

- Input: Automatically splitting the 8-bit input into two 4-bit chunks to write to the RAM chip in sequence
- RAM Chip: Keeping the data in memory and being able to read it back out without losing it/moments of gibberish
- Output: Continously displaying the data on the LED array

1.1.2 Implementation

The vast majority of the circuit was implemented using TTL chips. The only exception was the bidirectional register used to hold the data which was a CMOS chip, requiring additional buffering.

2 Methods

2.0.1 Theory

The circuit was designed as two finite state machines, one for read and one for write:

- The **write FSM** would take in the 8-bit input and split it into two 4-bit chunks, writing them to the RAM chip in sequence.
- The **read FSM** would read the data from the RAM chip and display it on the LED array.

Why do we use a finite state machine?

Because the circuit needs to be able to remember the state it is in, and then change states based on the input. This is the essence of a finite state machine. The practical relation to the circuit is that the circuit needs to know whether it is in the write state or the read state, and then change addresses in the RAM chip accordingly. based on the details of the state.

This comes from the fact that the RAM chip has a limited address width (10 bits for 1024 locations), which requires control over address selection to store and retrieve the data correctly.

In addition to addresses, the RAM chip requires a write-enable signal to write data to memory. This signal is controlled by the write FSM, which is responsible for writing the data to the RAM chip in the correct sequence.

The design of the RAM chip has certain time-intervals where between address changes, the data written to memory is invalid/incorect. Because of this, the write FSM must be designed to wait for the correct time to write the data to memory, and then change the address to write the next data.

However, the timing intervals on the 2114 TTL RAM chip is measured in nanoseconds, which our clocks generated from 555-timers cannot feasibly reach due to equipment availability. Thus, this timing was not of much concern considering our RAM chip would never be able to reach the speeds required to cause a problem. Because of this, we do not really care about an idle state in both FSMs as the RAM chip will never be able to write/read data fast enough to cause a problem. But in my original thinking, I designed the states with an idle in mind.

To summarize, the finite state machines are responsible for selecting the correct address and enabling the read or write operation as required by the process:

Why do we use external registers?

The RAM chip has a 4-bit data width, which means it can only store 4 bits of data at a time. However, the input is 8 bits, which means we need to split the input into two 4-bit chunks to store in memory.

To do this, we use two 4-bit registers to hold the data before writing it to memory. The first register holds the first 4 bits of data, and the second register holds the second 4 bits of data. Why do we use registers instead of reading from RAM directly?

This is because the RAM chip has a limited address width, which means we need to write the data to memory in the correct sequence. If we read the data from memory directly, we would not be able to control the sequence in which the data is written to memory. In other words, we want to view all the data at once, not just the 4-bits at a time. The external registers also store the values inside so we can change the values on the dip switches, yet the values on the LED array remain the same since we have not written to the RAM.

2.0.2 State Transitions

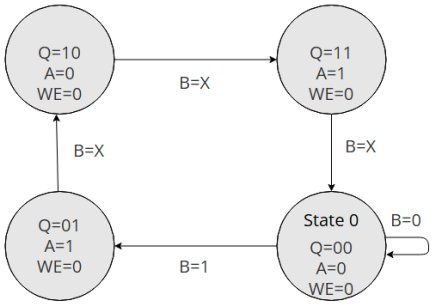

Figure 2: Read State Transition Diagram

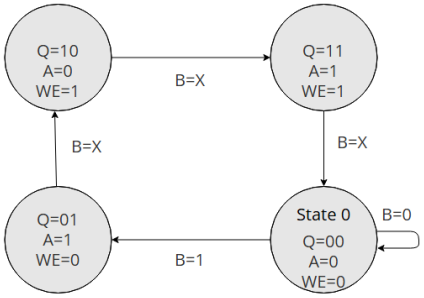

Figure 3: Write State Transition Diagram

Above are the two state transiton diagrams for read and write respectively.