Digital Logic Design Assignment: Memory Protection & Interface Enhancement

Overview

In this assignment, you will analyze, understand, and extend a memory circuit implemented in Logisim. You'll apply your knowledge of digital logic and floating inputs while incorporating several new features to enhance the circuit's functionality and user interface. This assignment extends on what we have been talking about in class with respect to digital logic and ALU design. It will prepare us for future lectures where we talk about the entire RISC-V datapath as memory is a critical component. This assignment contains a starter circuit that has a rom component to simulate real firmware, and a ram component to simulate the way RAM chips are actually constructed - specifically by using a collection of smaller RAM chips to create a holistic larger memory space.

Learning Objectives

By completing this assignment, you will:

- Demonstrate understanding of digital logic circuits and memory interfaces
- Apply knowledge of floating inputs and pull-up/pull-down resistors
- Implement memory protection features using control logic
- Create user interfaces for digital systems using hex displays
- Develop error detection for improper configurations

Prerequisites

- Logisim-evolution v3.9.0 or compatible version
- Understanding of basic digital logic concepts
- Familiarity with the attached study guide on floating inputs

Getting Started with the Circuit

Before implementing new features, it's important to understand how the existing circuit operates. Follow these steps to explore the circuit behavior:

Manual Clock Operation

1. Open the provided "memlab.circ" file in Logisim-evolution
2. Locate the clock signal in the main circuit (labeled "clk")

3. Initially, explore the circuit by manually clicking on the clock to advance one cycle at a time:
    - Right-click on the clock and select "Tick" or use the keyboard shortcut
    - Observe how the address register increments and memory operations occur
    - Note how the sweep_enable input affects the address generation
    - Toggle the DIP switches to observe different operation modes

Using Auto-Tick Simulation

After understanding the basic operation, use Logisim's auto-tick feature to run the simulation continuously:

1. Configure the simulation settings:

    - Go to Simulate → Tick Frequency in the menu
    - Start with a low frequency (4Hz or 8Hz) to easily observe the behavior
    - For faster operation, you can increase up to 256Hz or 512Hz maximum

2. Enable auto-ticking:

    - Go to Simulate → Ticks Enabled or press Ctrl+K
    - The circuit will now run continuously at the specified frequency
    - Observe the address counter incrementing through the memory range

3. Experiment with different configurations:

    - Try various combinations of DIP switch settings
    - Toggle the sweep_enable input to control address generation
    - Press the reset and mem_clear buttons to observe their effects

4. When debugging your implementation:

    - Use slower tick frequencies (4-8Hz) to observe detailed behavior
    - Temporarily disable auto-ticking when making modifications

Note: Operating at frequencies higher than 512Hz may make it difficult to observe circuit behavior and could cause Logisim to become less responsive on some systems. For best results, stay within the recommended frequency range.

Background

The provided circuit ("memlab.circ") contains a memory system with address generation, read/write control, and a 1K memory implementation. The system uses a concept called

"floating inputs" for control, which is explained in detail in the study guide. Before beginning your modifications, take time to thoroughly understand the existing circuit components.

Configuration DIP Switches

The circuit includes a 3-position DIP switch that configures the operation mode:

1. **First Switch (read_mode)**: When ON, enables memory read operations
2. **Second Switch (write_mode)**: When ON, enables memory write operations
3. **Third Switch (mem_protect)**: When ON, enables memory protection features

Note that read_mode and write_mode should not both be ON simultaneously, as this creates an invalid configuration. Your implementation will need to detect and handle this condition.

Implementation Location

Important: All your new features should be implemented in the main circuit at the bottom of the design. Do not modify the subcircuits (oneKram, firmware) unless specifically instructed to do so. Your solution should include:

- Hex displays for address and value
- LEDs for status indicators
- Logic for memory protection and configuration error detection

Part 1: Circuit Analysis (20 points)

Study the provided circuit and answer the following questions:

1. Identify the main components of the circuit and their functions:

    - What is the purpose of the firmware subcircuit?
    - How does the oneKram subcircuit work?
    - What controls the address generation in the main circuit?

2. Explain the role of floating inputs in the oneKram subcircuit:

    - How are floating inputs detected and utilized?
    - Which components implement the floating input detection?
    - How does this relate to the concepts in the study guide?

3. Trace the signal flow through the circuit:

- When the sweep_enable input is activated, what happens to the memory addresses?
- How does the circuit distinguish between read and write operations?
- What purpose do the configuration switches serve?

Part 2: Circuit Enhancement (60 points)

Modify the circuit to add the following features:

2.1: Memory Protection for Low Addresses (20 points)

Implement a feature that protects memory addresses below 0x0FF when the mem_protect configuration switch is on. Protected addresses should not be writable but should remain readable.

Requirements:

- Use the existing mem_protect switch (third DIP switch) to enable/disable this feature
- When protected, attempts to write to addresses 0x000-0x0FF should be blocked
- Reading from protected addresses should still be allowed
- The protection should not affect addresses 0x100 and above
- Add a green "mem_fault" LED to the main circuit that illuminates when an attempt is made to write to a protected memory address
- Implement this feature in the main circuit at the bottom of the design

2.2: Configuration Error Detection (20 points)

Implement logic to detect and indicate when the configuration switches are set to an invalid state (i.e., both read_mode and write_mode are active simultaneously).

Requirements:

- Create a "mode_error" indicator using a green LED in the main circuit
- The LED should be green when the configuration is valid (either read or write, but not both)
- The LED should turn off (or implement as a separate red LED) when both read_mode and write_mode are enabled simultaneously
- When the error condition is detected, all memory operations should be halted
- Implement this feature in the main circuit at the bottom of the design

2.3: Address and Data Display Interface (20 points)

Create a visual interface to display the current memory address and data value using Hex displays.

Requirements:

- Use the Hex Display component from Logisim's I/O library for both address and data values
- Configure the displays to show the current address (3 hex digits) with blue color
- Configure the displays to show the current data value (2 hex digits) with red color
- Include appropriate labels for the displays
- The displays should update in real-time as the address and data values change
- Implement all display components in the main circuit at the bottom of the design

Part 3: Testing and Documentation (20 points)

3.1: Test Your Implementation

Create a testing procedure that demonstrates your enhancements:

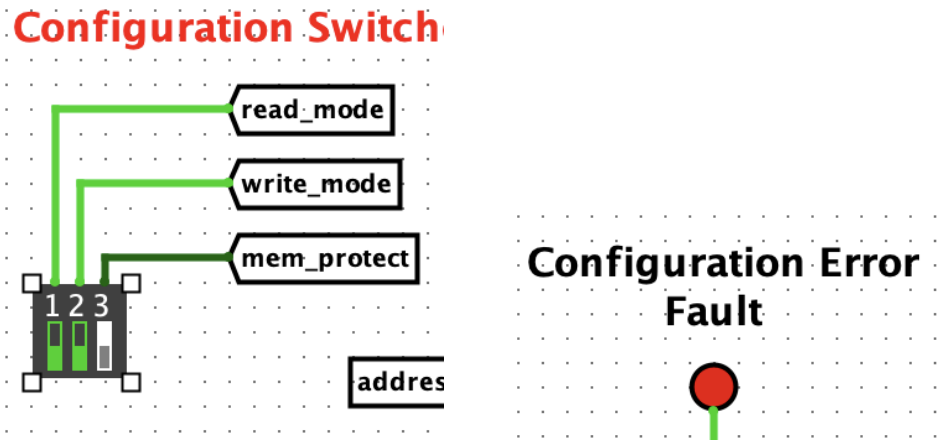
- Show that the memory protection works for addresses below 0x0FF
- Demonstrate the error detection for invalid switch configurations
- Verify that the displays correctly show address and data values

3.2: Documentation

Provide a written explanation of your implementation:

- Describe your approach to implementing each feature
- Explain any design decisions you made
- Identify any challenges you encountered and how you resolved them
- Reference concepts from the study guide where applicable
- Place screen shots in your submission document along with a few sentences showing your testing outcomes.

Example:
The figure below demonstrates that when the switches are placed in an illegal configuration the configuration error fault LED displays an error. Show these types of things to highlight everything you tested (a few screen shots and a description).



Implementation Guidelines

Components to Consider Using:

1. For Memory Protection:

    - Comparators to check address ranges (< 0x0FF)
    - AND/OR gates to combine the protection signal with write control
    - Consider using a Multiplexer to selectively block write operations
    - Green LED component for the memory fault indicator

2. For Configuration Error Detection:

    - XOR gates can detect when exactly one option is selected
    - Use AND gates to detect when both modes are active
    - Connect the error signal to the halt line to prevent operations
    - Green LED that stays lit only when the configuration is valid

3. For Hex Display:

    - Splitters to separate address/data bits into individual hex digits
    - Hex Display components from the I/O library
    - Tunnels to route address and data signals to the display
    - Configure the display color properties to match the requirements (blue for address, red for data)

Where to Make Changes:

1. Memory Protection:

- Modify the main circuit to intercept write operations based on address value
- You'll need to connect the mem_protect switch to your new protection logic
- Consider placing your logic between the write_mode signal and the oneKram circuit

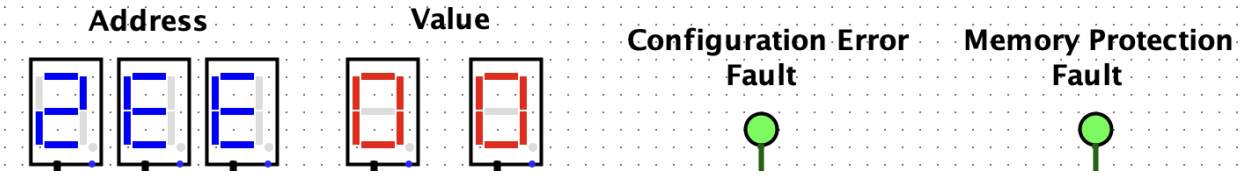
2. Configuration Error Detection:

    - Add logic in the main circuit that monitors the read_mode and write_mode signals
    - Connect your detection logic to a new LED component
    - Integrate with the halt signal to prevent memory operations

3. Hex Display:

    - Place the Hex Display components directly in the main circuit
    - Create tunnels or direct connections to route the address and data signals
    - Configure displays with appropriate colors (blue for address, red for data)
    - Arrange the displays in a clean, organized layout with labels

For your reference here is a picture of the solution I created with respect to what the outputs should look like. I am using 3 hex display components for the address, 2 hex display components for the memory value, and 2 LEDs configured to show green if no error conditions are found, and these will change to red if an error condition is calculated.



Submission Requirements

Submit your completed assignment as a Logisim circuit file (.circ) along with a MS Word or PDF document containing:

- Your answers to the analysis questions from Part 1
- Documentation of your implementation as described in Part 3.2
- Screenshots showing your working circuit with the new features

Reference Material

- Refer to the attached study guide on floating inputs for understanding the memory control mechanism
- Review the Logisim documentation for information on specific components

Grading Criteria

Your assignment will be evaluated based on:

- Correctness of implementation (does each feature work as specified?)
- Quality of design (is the solution elegant and efficient?)
- Clarity of documentation (is your explanation clear and complete?)
- Understanding of concepts (do you demonstrate knowledge of the underlying principles?)

Getting Help

I will try to monitor discord the best I can while I am away next week. Our TAs will monitor discord as well and they have a lot of time available via office hours during the week.

YOU ARE SUPPOSED TO BE IN CLASS FOR 2 HOURS and 40 MIN OF LECTURE, ALONG WITH 1 HOUR and 50 MINUTES OF LAB TIME NEXT WEEK. THAT IS A TOTAL OF 4 ½ HOURS. THIS ASSIGNMENT WILL NOT TAKE LONGER THAN THAT - NOWHERE NEAR IT. DO NOT WAIT UNTIL THE END OF THE WEEK TO GET STARTED. THERE WILL BE NO ADJUSTMENTS TO THE DUE DATE, NOR CAN LATE DAYS BE USED FOR THIS ASSIGNMENT.

Good luck, and remember to test your circuit thoroughly!