Logic Circuits Simulator

Introduction

The Logic Circuits Simulator project aimed to develop an event-driven logic circuit simulator that allows users to create virtual representations of digital circuits and observe their behavior under different conditions. The simulator accepts inputs in the form of library files, circuit files, and stimuli files, simulates the circuit's behavior, and generates a simulation file as output.

Data Structures

1. **unordered_map<string, Gate> for Gates Storage:**

Purpose: Stores gate objects parsed from the circuit file. Each gate has a unique string identifier.

Usage: Facilitates quick access to gate details using the gate's name, crucial for updating gate states during simulation.

2. **unordered_map<string, vector<string>> for Signal to Gates Mapping:**

Purpose: Maps signal names to lists of gates that are dependent on these signals as inputs. This mapping is essential for determining which gates to evaluate when a signal's state changes.

Usage: Used to quickly find and update all gates affected by a change in signal state, ensuring the simulation reflects the propagation of signals through the circuit.

3. **unordered_map<string, int> for Signal States:**

Purpose: Keeps track of the current state (value) of each signal in the circuit. Signals may represent input, output, or internal connections between gates.

Usage: Accessed to retrieve and update signal values during the simulation, especially when processing events that change signal states.

4. **priority_queue<Event, vector<Event>, decltype(eventCompare)> for Event Management:**

Purpose: Manages simulation events in a time-ordered fashion, ensuring that events are processed in chronological order.

Usage: Events representing changes in signal states at specific times are added to this priority queue. The simulation processes events from the queue in order, simulating the passage of time within the circuit.

5. **vector<string> and stack<string> for Expression Evaluation:**

Purpose: Used in the conversion of gate output expressions from infix to postfix notation and for evaluating these expressions, respectively.

Usage: The vector holds the postfix form of the expression, while the stack is used during the evaluation process to apply logical operators correctly and determine the resulting output signal state.

Algorithms

1. **File Parsing (Library, Circuit, Stimuli Files):**
   Description: Custom parsing algorithms are implemented for reading and interpreting the contents of library, circuit, and stimuli files. These algorithms extract relevant data such as gate definitions, circuit connections, and timing information for stimuli.

   Purpose: To initialize the simulation environment with the correct configuration of gates, signals, and initial events based on external definitions.

2. **Expression Adaptation and Evaluation:**
   Description: The simulator includes an algorithm for adapting gate output expressions based on the specific inputs of instances in the circuit file. Another algorithm converts these expressions to postfix notation and evaluates them to determine gate outputs.

   Purpose: To dynamically calculate gate outputs during the simulation, reflecting the logical behavior specified by the gate definitions in the library file**.**

3. **Event-Driven Simulation:**
   Description: The core simulation loop processes events from the priority queue in time order. Each event may trigger the evaluation of one or more gates, potentially generating new events if gate outputs change.

   Purpose: To simulate the dynamic behavior of the circuit over time, accurately modeling how changes propagate through the circuit in response to input stimuli.

Testing

The testing phase involved creating and simulating five test circuit file sets manually. This manual testing helped verify the correctness and functionality of the simulator under various circuit configurations and input stimuli.

Challenges

1. **Refactoring for Modularity and Readability**

   Challenge: The original code was more procedural and less modular, making it difficult to maintain or extend. For example, the Component and Circuit classes were tightly coupled with the simulation logic.

   Solution: In the new code, functionalities are encapsulated into more granular and cohesive classes (Gate, Event, etc.), improving modularity. This encapsulation makes the code easier to understand and modify, as each class has a clear responsibility.

2. **Improving Data Structure Utilization**

   Challenge: The initial implementation utilized basic data structures without fully leveraging their capabilities, leading to inefficiencies in data management and access.

   Solution: The new version employs more appropriate data structures like unordered_map for fast lookups and priority_queue for event management, significantly optimizing performance. The choice of data structures is aligned with their intended use cases, such as quick access to gate and signal information or ordered event processing.

3. **Algorithmic Efficiency**

   Challenge: Evaluating logic expressions and managing events were not optimized for performance, potentially leading to slow simulations for large circuits.

   Solution: The introduction of postfix notation for logic expression evaluation and a priority queue for event management streamlined these processes. Converting expressions to postfix notation simplifies their evaluation, and using a priority queue ensures events are processed in the correct chronological order without unnecessary sorting.

4. **Error Handling and Debugging**

   Challenge: The original code lacked robust error handling and debugging mechanisms, making it difficult to diagnose issues or understand the simulator's internal state during execution.

   Solution: Enhanced error handling was introduced, including more descriptive error messages and checks for potential issues (e.g., missing gate types or input signals). Additionally, debugging outputs, such as gate evaluations and signal states, were added to assist in understanding the simulation flow and identifying problems.

5. **Optimizing Simulation Output Management**

Challenge: Managing simulation outputs in a way that is both efficient and easy to analyze was not adequately addressed initially.

Solution: Implementing a method to write, sort, and clean simulation outputs ensures that the final output file is both accurate and free of duplicates, making analysis more straightforward.

Contributions of Each Member

1. **Toqa Mahmoud:** took on the parsing of input files and the management of data structures that store the simulation's state. This includes:

   Parsing the library, circuit, and stimuli files (parseLib, parseCir, and parseStim methods), extracting necessary information to build the simulation environment.

   Managing the data structures that hold gates (gates unordered_map), signal-to-gate mappings (signalToGates unordered_map), and signal states (signalStates unordered_map).

   Ensuring the integrity and accuracy of the simulation's data throughout its execution, including initial state setting and dynamic updates during the simulation.

2. **Adham Gohar**: handled the representation and evaluation of gates, including their logical expressions. Responsibilities include:

   Defining and managing the Gate class, ensuring it accurately represents each gate's characteristics and behaviors.
   Developing the expression evaluation system, including the conversion of expressions from infix to postfix notation (infixToPostfix method) and their subsequent evaluation (evaluateExpression method).
   Adapting gate expressions based on actual signal names (adaptExpression method) and ensuring the correct evaluation of logical operations based on gate types and inputs.

3. **Khadeejah Iraky**: focus on the core simulation loop, event handling, and the simulation's initialization and finalization processes. This includes:

   Managing the priority queue for events (events priority_queue) and the corresponding event handling logic.
   Implementing the startSimulation, processEvents, and initializeOutputs methods, which drive the simulation forward, process each event, and set initial gate outputs, respectively.
   Overseeing the simulation's setup (GateSimulator constructor) and teardown routines (sortAndClean method), ensuring the simulation is correctly initialized and properly concludes with a sorted and clean output.

.

Conclusion

In conclusion, the Logic Circuits Simulator project successfully developed an event-driven simulator capable of modeling digital circuit behavior. Through
collaborative effort and effective utilization of data structures and algorithms, the team overcame challenges and contributed to creating a functional and efficient simulator.

Use of ChatGPT in the Project

During the project development and debugging process, we utilized ChatGPT to assist us in various aspects:

Report Writing:

ChatGPT was instrumental in helping us draft the project report. We used it to brainstorm ideas for structuring the report, organizing the content logically, and ensuring that all required topics were covered comprehensively. ChatGPT provided valuable suggestions and insights that improved the overall quality of our report.

Debugging Assistance:

In the debugging phase, we encountered several challenges related to code logic, event management, and output evaluation. ChatGPT served as a virtual assistant, helping us troubleshoot specific issues by providing alternative approaches, clarifying concepts, and offering debugging strategies. Through interactive sessions with ChatGPT, we gained deeper insights into potential solutions and refined our code implementation.
By leveraging ChatGPT's capabilities, we enhanced our project development process, addressing challenges effectively and improving the overall quality of our work.