Logic Circuits Simulator

1. 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.

2. Data Structures and Algorithms Used

Data Structures:

- `unordered_map`:
    - **Purpose**: Used extensively throughout the simulator for efficient storage and retrieval of various entities such as components, inputs, wires, and event scheduling.
    - **Usage**:
        - **Component Storage**: Each component in the circuit is stored in an unordered_map where the key is the component's name and the value is an instance of the Component class. This allows for constant-time access to components during simulation.
        - **Input and Wire Management**: Inputs and wires are stored in unordered_maps, with the key being the input/wire name and the value representing the logic state or delay associated with the input/wire. This facilitates quick lookup and updating of input values and wire states during simulation.
        - **Event Management**: The eventManager priority_queue utilizes unordered_map internally for efficient event lookup and management. Events are scheduled based on timestamps, with each event represented by a pair containing the timestamp and a string detailing the event.
- `priority_queue`:
    - **Purpose**: Used for event management and scheduling in the simulator. Allows for the efficient retrieval and processing of events in chronological order.
    - **Usage**:

        - **Event Scheduling**: Events such as input value changes and component state updates are scheduled using a priority_queue based on their timestamps. This ensures that events are processed in the correct order during simulation, maintaining the integrity of the circuit's behavior.
        - **Event Processing**: During simulation, events are popped from the priority_queue in ascending order of timestamps, and corresponding actions are taken based on the event type. This ensures that events are processed sequentially and the circuit state is updated accurately.
- `Vectors`:
    - **Purpose**: Used for storing multiple inputs associated with a component.
    - **Usage**:
        - **Component Inputs**: Each component in the circuit may have multiple input wires. These input wires are stored in a vector within the Component class. Vectors provide dynamic sizing and efficient traversal, making them suitable for managing variable numbers of inputs for each component.

Algorithms:

- **Event-driven Simulation**:
    - **Purpose**: The event-driven simulation algorithm is the core mechanism used in the Logic Circuits Simulator to model the behavior of digital circuits over time.
    - **Description**:
        - **Event-Based Approach**: The simulator processes events sequentially, where each event represents a significant change in the circuit's state, such as input value changes or component state updates.
        - **Event Scheduling**: Events are scheduled and managed using a priority_queue based on their timestamps. This ensures that events are processed in chronological order, maintaining the temporal integrity of the simulation.
        - **Event Processing**: Upon processing an event, the simulator updates the circuit's state accordingly, evaluating the logic functions of affected components and propagating changes to dependent components and output wires.

    - **Dependency Resolution**: The simulator resolves dependencies between components and wires dynamically, ensuring that changes propagate through the circuit in a consistent and timely manner.
    - **Simulation Termination**: The simulation continues until all scheduled events have been processed, ensuring that the circuit reaches a stable state where no further changes occur.

3. 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.

4. Challenges Faced

Evaluating Output Expression:

One of the main challenges was figuring out how to evaluate the output expressions defined in the library file. This required designing an algorithm to parse and compute bitwise logic expressions involving input variables.

Linking Files Together:

Integrating the logic of parsing library, circuit, and stimuli files together posed a challenge, ensuring that each file's information was utilized correctly in the simulation process.

Event Evaluation and Debugging:

Debugging the code to ensure that all events in the simulation were evaluated until reaching the output state was another challenge, requiring careful tracking of event dependencies and timing.

5. Contributions of Each Member

- **Toqa:** Responsible for parsing the files (library, circuit, stimuli) and ensuring their correct interpretation within the simulator.

- **Adham:** Implemented the Circuit class with simulation and event management functions, handling event scheduling and circuit state updates.
- **Khadeejah:** Developed the component class with the evaluateLogic function, which is responsible for evaluating output expressions and managing component behavior.
- All of us worked together in debugging and combing the code to ensure that it worked well as a whole.

6. 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.