DD_1 PROJECT (1)

Logic Gates Simulator

Adham Ali / 900223243
Omar Saqr / 900223343
Ebram Thabet / 900214496
Digital Design I (Spring 2024)
The American University in Cairo

Contents

1 Introduction

Digital circuits are the building blocks of all electronic devices, from calculators to computers. Our project aims to make a logic gate simulator. In this simulator, you can choose a circuit you want, and it will give you the output with its propagation delay. However, you must make three files: lib, cir and stim. The lib file is responsible for the propagation delay of the gates, as well as the expressions and names of the gates. The CIR file shows the inputs and where every gate's inputs are. The last stim file shows how the inputs vary over time. Moreover, our program can identify any unstandardised gate. How? The program reads the expression that is present in the lib file.

2 Used Data Structures and Algorithms

1. **Unordered_map**
   Usage: This data structure is extensively used to store and access variables and their values, component functions, and propagation delays associated with wires.
   Key Features: It offers average constant-time complexity for insertions, deletions, and searches. The use of hashing allows for fast access to element values using keys.

2. **Vector**
   Usage: This data structure stores sequences of elements, such as inputs, components, and operands for logical expressions. It allows dynamic resizing and direct access to elements.
   Key Features: Provides random access to elements, efficient insertion/removal at the end, and the ability to change size dynamically.

3. **Stack**
   Usage: Used in the LogicalExpressionEvaluator class to implement the algorithm for evaluating infix logical expressions.
   Key Features: Provides Last-In-First-Out (LIFO) access to elements, supporting operations like push, pop, and top.

4. **Tuple**
   Usage: To store grouped elements of different types together. It's particularly used for storing readings from a file with multiple value types.
   Key Features: Allows storage of a fixed-size collection of heterogeneous elements.

5. **Struct Component**
   Usage: Represents the properties of a component in the circuit library.
   Structure: Contains fields such as num_inputs (number of inputs to the compo-

nent), delay_ps (propagation delay in picoseconds), and logic (the logical operation performed by the component).

6. **Queue**
   Usage: Utilized in the function funccall to maintain a queue of gates that need to be evaluated based on changes in input values.
   Key Features: Provides First-In-First-Out (FIFO) access to elements, ensuring that the gates are evaluated in the order they were enqueued.

3 Challenges

One of the core challenges was accurately simulating propagation delays, a fundamental aspect of digital circuits where signals take time to travel through components. Achieving realistic simulations of these delays was critical, as they can significantly affect the behaviour and timing of circuit outputs. We tackled this by integrating delay parameters into our logic functions and developing a scheduling system that could manage and execute events based on their timing. Ensuring that this system accurately reflected a real circuit's sequential and concurrent operations required extensive testing and refinement. Moreover, the logic of the unstandardised gates makes us change the algorithm. Our previous algorithm only read the gate name from the lib file and performed the logical operation as the C++ language has operations that perform the gate output. Unfortunately, c++ has only the operation of the standard gates, but if the lib file has a gate called (wert), which is an unknown gate, it will not read it, so we make the code read the expression that is in the lib file, and we cancelled the idea of the operations of c++. The last challenge was the continuous integration; we needed more time to do it, as we knew the meaning of CI before the deadline by 2 days.

4 Testing

We developed comprehensive tests using predefined circuits with known outcomes to ensure our simulator's accuracy. This process involved comparing the simulator's output against expected results for various input sequences, allowing us to identify and correct discrepancies. Moreover, we tested our simulator by getting some functions that have an error, to handle this part in our code.

5 Contributions

- **Adham Ali**

1. Calculating the Propagation delay at the base case (when all inputs initialised with 0) by event-driven behaviour.
2. Organise the code by adding comments for every function to describe it.
3. Writing the Report.
4. Debugging some errors in the algorithm of read/write files, like the parse_cir_file function and the readFromFile function.
5. Drawing the graphs for the propagation delay for the five test cases.
6. Evaluating the gates only affected by changing the inputs using the queue.
7. Running code from the Terminal.
8. The option of opening a simulation file in the user interface and the design of the interface (using Python).
9. Organising the Repository on GitHub by making a folder for each circuit and erasing all the unnecessary files.

- **Omar Saqr**

  1. Reading the .stim, .lib, and .cir files. Dealing with missing spaces or misplaced spaces. Also, it deals with the case when there are 2 gates on the same line.
  2. Creating a logic evaluator that takes the logic expression from the library file and evaluates its input using modified infix evaluation. This function only gets called for gates directly or indirectly connected to the changed input.
  3. Calculating the Propagation delay so that it only includes gates that have changed, not all the gates, to simulate event-driven behaviour.
  4. Debugging the errors in the code in different stages of development, including selecting only gates connected to the inputs and the cir files design.
  5. Making the user interface, particularly the Python code that runs commands from the terminal, establishes the connection between the front end in Python and the back end in C++.

- **Ebram Thabet**

  1. Visualize the simulation output graphically in terms of waveforms.
  2. Designing the circuits and making the lib, cir and stim files.
  3. Making the user interface by uploading the files from the computer to the program.
  4. Handling the corner cases that may encounter the code.

5. Debugging the code to enhance the output (by creating a template function that prints lines of codes to track the errors).
6. Reviewing and double-checking the report in terms of the content and format.