Combinational Logic Simulator – Algorithm Summary

by Tatipaka Aditya EE24B071

1. Overview

The simulation process follows four main steps:

1. Parse the input circuit → parse_circuit()
2. Evaluate logic gates → evaluate_gate()
3. Run the simulation over all stimuli → run_simulation()
4. Convert results to JSON format → to_wavedrom()

The main function (main) manages this pipeline — it reads the input .net file, calls each processing function, and saves the resulting WaveDrom JSON waveform description.

---

2. Circuit Parsing (parse_circuit)

This function extracts structured circuit information from plain text.

Algorithm Steps:

- The input file (text_data) is cleaned — comments (lines starting with #) and blank lines are skipped.
- The helper function locate_section(section_name) finds the index of each required section header:
  INPUTS, OUTPUTS, GATES, and STIMULUS.
  If any are missing, a ValueError is raised.
- input_signals and output_signals are parsed from their respective section lines.
- Gate definitions (between GATES and STIMULUS) follow the form:
  OUT = GATE(IN1, IN2)
  Each gate is stored as a tuple: (out_sig, gate_kind, params) in logic_blocks.
- Stimulus lines define time steps and input values, stored as (timestamp, values) in stimuli_data, ensuring timestamps are strictly increasing and each line matches the number of input signals.

Output:
A dictionary with keys:

```
{
  "inputs": [...],
  "outputs": [...],
  "gates": [...],
  "stimulus": [...]
}
```

3. Gate Evaluation (evaluate_gate)

Implements basic Boolean logic behavior:

Gate	**: Operation**
"AND"	a & b
"OR"	`a
"XOR"	a ^ b
"NOT"	1 - a

It receives the gate type (**gate_kind**) and argument values (**args**), returning the result.
If an invalid gate is found, a ValueError is raised.

---

4. Simulation Engine (run_simulation)

Performs logic propagation over time using the parsed circuit.

Algorithm Steps:

1. Initialize waveform_map — a dictionary storing logic waveforms of all signals (inputs, outputs, and internal nodes).
2. For each stimulus (timestamp, input_values):
    - Create signal_env: maps input signal names to their logic values.
    - Evaluate gates iteratively until all signals are computed:
        - For each gate, if all its input signals are known, compute the output using evaluate_gate() and update signal_env.
3. Append the computed logic values for all input and output signals to waveform_map.

Output:
A dictionary mapping each signal to its logic value sequence over time.

---

5. Waveform Conversion (to_wavedrom)

Converts simulation results into a WaveDrom JSON for visualization.

Algorithm Steps:

- Combine inputs and outputs into one list → all_signals.
- For each signal, create a JSON object with:

{ "name": "A", "wave": "0101" }

Main Function (main)

Handles command-line interaction and file operations.

Process:

- Uses argparse to read:
    - .net file path → args.net_file
    - Optional output path → args.out
- Reads the circuit file and calls:

1. parse_circuit()
2. run_simulation()
3. to_wavedrom()

- Writes the resulting JSON to output_path and prints the path on completion.

---

7. Algorithmic Highlights

- Structured Parsing: Uses simple, deterministic section searching and splitting for clarity.
- Iterative Evaluation: Gates are resolved dynamically without explicit topological sorting.
- Compact Output: Waveform strings use direct concatenation for efficiency.
- Error Handling: Ensures correctness — checks for missing sections, mismatched stimuli, and non-increasing timestamps.