Parsing ISCAS-85 Netlist in Python and Processing it in C++ Using JSON

Deliverable

1 . Parse an ISCAS-85 netlist in Python, write the parsed data into a JSON file,

2. Define C++ structures to hold data that is parsed

Eg: 1. Gate

2. All gate information in the circuit

3. Read that JSON file in C++ to populate your C++ circuit structure.

This method separates the parsing logic and the processing logic, making it more flexible and easier to debug.

Prerequisites

Before starting, make sure you have the following tools installed:

1. **C++ Compiler**: A modern C++ compiler that supports C++11 or later.
2. **Python**: Python 3.x installed.
3. **JSON for Modern C++**: We will use the nlohmann/json library for handling JSON in C++. You can install it using your preferred package manager, or download the `json.hpp` header file directly from the repository.

Step 0: Download the ISCAS-85 netlist benchmark

Step 1: Python Script to Parse the Netlist and Write to JSON

The first step in this approach is to parse the ISCAS-85 netlist using Python and save the parsed data in a JSON file.

1. Netlist Format

The ISCAS-85 netlist format looks like this:

```
<line_number> <gate_name> <type> <fanout> <fanin> >sa0 >sa1
```

2. Python Script to Parse and Write to JSON

Write is the Python script that reads the netlist and writes it to a JSON file.

Example JSON Output

After running the Python script on a sample netlist, the output `parsed_netlist.json` might look like this:

```
[
        {
                "line_number": 1,
                "identifier": "1gat",
                "gate_type": "inpt",
                "fanout": 6,
                 "fanin": 0,
                 "inputs": [],
                "faults": [">sa0", ">sa1"]
        },
        {
                "line_number": 2,
                "identifier": "1f01",
                "gate_type": "from",
                "fanout": 1,
                "fanin": 1,
                "inputs": [1],
                "faults": [">sa1"]
        }
 ]
```

Step 2: C++ Code to Read the JSON File and Populate the Circuit Structure

Based on the netlist format in the benchmark. Declare structures that can hold information about a gate, and total gates as a circuit.

Step 3: C++ Code to Read the JSON File and Populate the Circuit Structure

Once the data is written to JSON, we can read this file in C++ and populate the circuit structure.

Include the Header File in Your C++ Code: Once you have the `json.hpp` file in your project, you can include it in your code like this:

```
#include <json.hpp>
```

Final output

Print the gates structures onto the terminal