Dhirubhai Ambani Institute of Information and Communication Technology
Gandhinagar, Gujarat 382007

EL464 VLSI VALIDATION AND TESTING

Course Instructor: Sreeja Rajendran

TESTABILITY ANALYSIS

BY

Name	**StudentID**
Ashish Parmar	202101174
Nikita Shah	202101209
Rohan Mistry	202101231

Testability Measures:

An attempt to quantify testability by Goldstein and Grason resulted in two testability measures:

- **Controllability:**
  - The measure of difficulty of setting a particular logic signal to 0 or 1.
- **Observability:**
  - The measure of difficulty in propagating the value at a particular signal to primary output (i.e. difficulty of observing the logic 0 or 1 at any signal).

Out of several methods available to find testability measurers, hers we follow:
SCOAP: (Scandia Controlability Observability Analysis Program)

Introduction:

The project focused on assessing both combinational and sequential testability measures in VLSI circuits. Combinational testability involves evaluating the ease with which faults in the combinational logic of the circuit can be detected, while sequential testability concerns the effectiveness of testing methods for sequential elements like flip-flops. By analyzing controllability and observability metrics, the project aimed to identify critical points for testing and optimize testing strategies for better fault coverage. The goal was to develop techniques that ensure thorough testing of both the combinatorial and sequential components of the circuit, ultimately enhancing its reliability and performance.

Netlist conversion:

Netlist conversion from Verilog involves translating a Verilog file(.v), which describes a digital circuit, into a netlist(.txt). A netlist is a textual representation of a circuit composed of interconnected electronic components such as gates, wires, etc. The netlist describes how these components are connected to each other.

Code Description:

Let's break down the provided code and explain how it accomplishes netlist conversion from Verilog:

1. **Struct Definitions (Gate and Module):**
   The code defines structures Gate and Module to represent gates and modules in the Verilog file, respectively. Each gate contains information such as its type, name, inputs, outputs, and level.

2. **Verilog Parsing(parseVerilog():**
   The parseVerilog function reads a Verilog file and extracts information about inputs, outputs, and gates. It uses regular expressions to identify Verilog constructs such as inputs, outputs, and various gate types (e.g., NAND, AND, XOR). It then populates the Module structure with this information.

3. **Netlist Generation (generateNetlist()):**
   The generateNetlist function takes the parsed Verilog module and generates a netlist file. It starts by writing the inputs and outputs to the file. Then, it processes each gate in the module, converting it into a corresponding line in the netlist file. Each line represents a gate with its type, outputs, and inputs.

4. **Levelization (levelize():**
   The levelize function assigns levels to gates based on their de-

pendencies. Gates with no input dependencies are assigned level 0. Gates with inputs depend on the levels of those inputs, and their own level is set to the maximum input level plus one. This process ensures that gates are ordered correctly in the netlist to maintain logical consistency.

5. **Main Function(main())**
   The main function serves as the entry point. It prompts the user to enter the filename of the Verilog file to be converted. It then calls the parseVerilog function to parse the Verilog file, generates a netlist using generateNetlist, displays gate details using displayGates, and finally outputs a message indicating successful netlist generation.

Combinational Testability Measure in VLSI:

Combinational testability is a critical aspect of VLSI (Very Large Scale Integration) design that assesses the ease with which a digital circuit can be tested for correct functionality. It primarily focuses on how efficiently faults within combinational logic can be detected. One common metric used to evaluate combinational testability is the controllability and observability of signals within the circuit.

Description:

In the provided code, combinational testability is assessed through the calculation of controllability and observability metrics for each signal in the circuit. Controllability measures the ability to control the value of a signal, while observability measures the ease of observing the signal's value. These metrics are calculated recursively based on the gate-level logic and fanout structure of the circuit.

Explanation of Code:

1. `read_file()`: Reads the netlist description of the circuit from a file and initializes data structures.
2. `calc0()`: Initializes the controllability metrics for primary inputs.
3. `calc1()`: Calculates the controllability metrics for internal nodes based on gate-level logic and fanout.
4. `gate_in_list()`: Generates a sorted list of gate inputs to facilitate observability calculation.
5. `calc2()`: Initializes the observability metrics for all wires in the circuit.
6. `calc3()`: Calculates the observability metrics for internal nodes based on gate-level logic and fanout.
7. `print_ctrl()`: Prints the calculated controllability metrics for each wire.
8. `print_obs()`: Prints the calculated observability metrics for each wire.

How it Works:

1. The circuit's netlist is parsed to identify gates, inputs, outputs, and fanout connections.
2. Controllability metrics are initialized for primary inputs (`input_list`).
3. Controllability metrics for internal nodes (gate outputs) are calculated recursively based on gate types (AND, OR, etc.).

4. Fanout connections are considered to propagate controllability metrics.
5. Similarly, observability metrics are initialized and calculated for internal nodes based on gate types.
6. Finally, the calculated controllability and observability metrics are printed for analysis.

This approach provides insights into how efficiently signals can be controlled and observed within the circuit, aiding in the design and testing process.

Sequential Testability Measure in VLSI:

Sequential testability measure in VLSI refers to the ability to efficiently test sequential circuits, which consist of memory elements such as flip-flops. It involves determining the controllability and observability of signals within the circuit, which are crucial for effective testing.

Description:

In the provided code, the sequential testability measure is implemented through the functions `calc1()` and `calc2()`. These functions calculate the controllability and observability of signals, respectively, for each wire in the circuit. The controllability (`CC0` and `CC1`) and observability (`SC0` and `SC1`) values are computed iteratively until convergence.

Explanation of Code:

1. **Reading File:** The `read_file()` function reads the netlist from a file, which describes the logic gates and connections in the circuit. It initializes data structures and extracts relevant

information such as input and output wires, flip-flops, gates, etc.

2. **Initialization:** The `ini()` function initializes the controllability and observability values for input wires and flip-flops. It sets `CC0` and `CC1` values to 1 for input wires and flip-flops' input (D) signals, while setting the observability values (`SC0` and `SC1`) to 0.

3. **Controllability Calculation (`calc1()`):** This function iteratively calculates the controllability of output wires by considering the controllability of input wires and gates. It updates the `CC0` and `CC1` values for each wire until convergence is achieved.

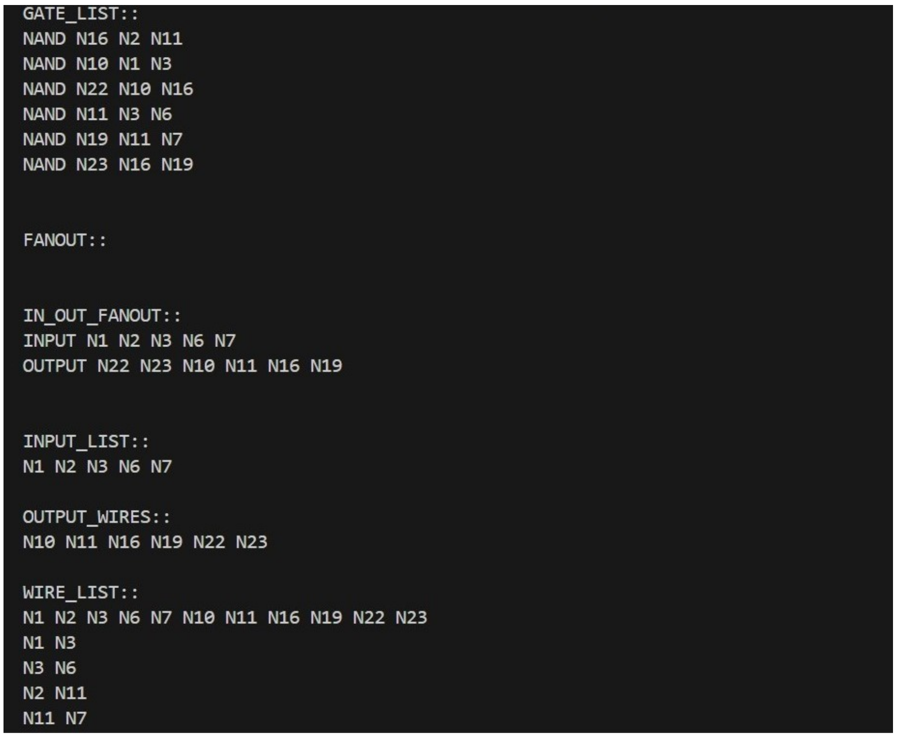
4. **Observability Calculation (`calc2()`):** Similar to `calc1()`, this function iteratively calculates the observability of output wires by considering the observability of input wires and gates. It updates the `SC0` and `SC1` values for each wire until convergence is achieved.

5. **Gate Functions:** Various functions such as `c_controllability_output_of_g` and `s_controllability_output_of_gate()` are defined to compute the controllability and observability of gate outputs based on their input signals.

6. **Helper Functions:** Several helper functions are provided to assist in finding gates, gate inputs, setting fanouts, printing controllability, etc.

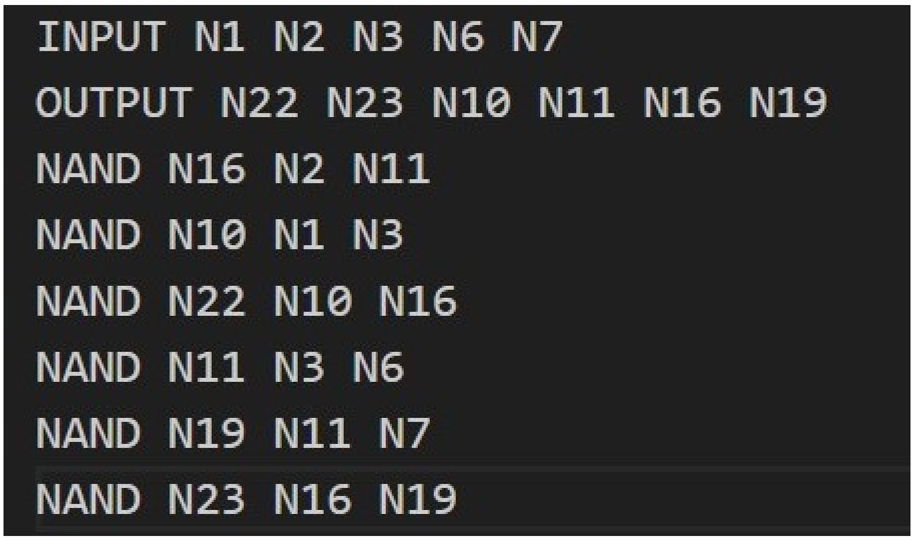
Example Code:

```
module c17 (N1,N2,N3,N6,N7,N22,N23);

input N1,N2,N3,N6,N7;

output N22,N23;

wire N10,N11,N16,N19;

nand NAND2_3 (N16, N2, N11);
nand NAND2_1 (N10, N1, N3);
nand NAND2_5 (N22, N10, N16);
nand NAND2_2 (N11, N3, N6);
nand NAND2_4 (N19, N11, N7);
nand NAND2_6 (N23, N16, N19);
endmodule
```



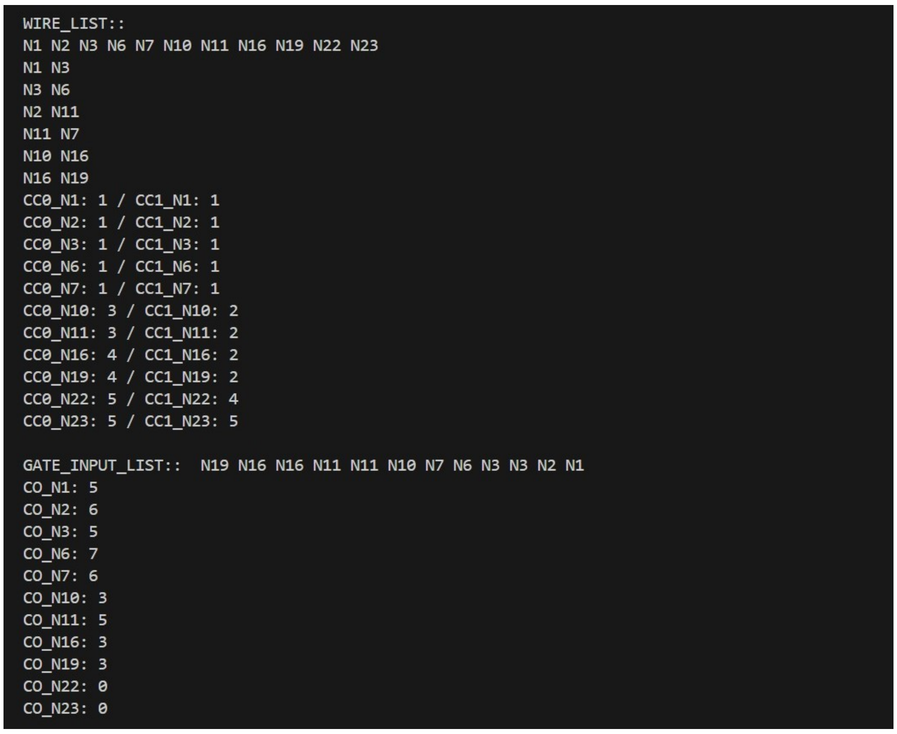
```
GATE_LIST::
NAND N16 N2 N11
NAND N10 N1 N3
NAND N22 N10 N16
NAND N11 N3 N6
NAND N19 N11 N7
NAND N23 N16 N19


FANOUT::


IN_OUT_FANOUT::
INPUT N1 N2 N3 N6 N7
OUTPUT N22 N23 N10 N11 N16 N19


INPUT_LIST::
N1 N2 N3 N6 N7

OUTPUT_WIRES::
N10 N11 N16 N19 N22 N23

WIRE_LIST::
N1 N2 N3 N6 N7 N10 N11 N16 N19 N22 N23
N1 N3
N3 N6
N2 N11
N11 N7
```

Figure 1: Netlist Conversion



```
INPUT N1 N2 N3 N6 N7
OUTPUT N22 N23 N10 N11 N16 N19
NAND N16 N2 N11
NAND N10 N1 N3
NAND N22 N10 N16
NAND N11 N3 N6
NAND N19 N11 N7
NAND N23 N16 N19
```

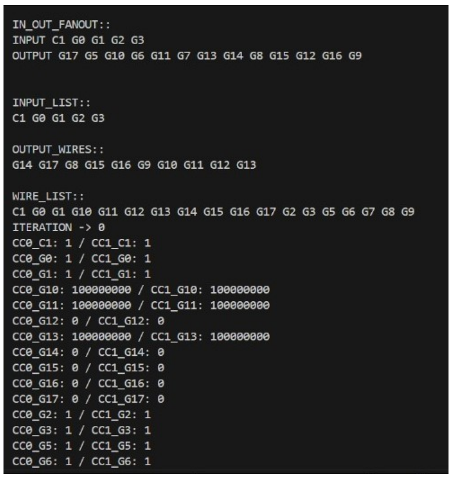
Figure 2: Netlist Output File



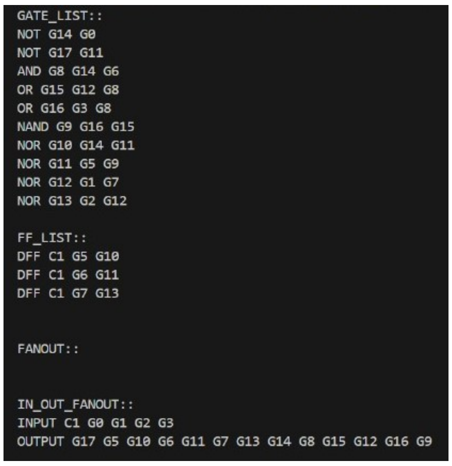
```
WIRE_LIST::
N1 N2 N3 N6 N7 N10 N11 N16 N19 N22 N23
N1 N3
N3 N6
N2 N11
N11 N7
N10 N16
N16 N19
CC0_N1: 1 / CC1_N1: 1
CC0_N2: 1 / CC1_N2: 1
CC0_N3: 1 / CC1_N3: 1
CC0_N6: 1 / CC1_N6: 1
CC0_N7: 1 / CC1_N7: 1
CC0_N10: 3 / CC1_N10: 2
CC0_N11: 3 / CC1_N11: 2
CC0_N16: 4 / CC1_N16: 2
CC0_N19: 4 / CC1_N19: 2
CC0_N22: 5 / CC1_N22: 4
CC0_N23: 5 / CC1_N23: 5

GATE_INPUT_LIST::  N19 N16 N16 N11 N11 N10 N7 N6 N3 N3 N2 N1
CO_N1: 5
CO_N2: 6
CO_N3: 5
CO_N6: 7
CO_N7: 6
CO_N10: 3
CO_N11: 5
CO_N16: 3
CO_N19: 3
CO_N22: 0
CO_N23: 0
```

Figure 3: Testability Measures

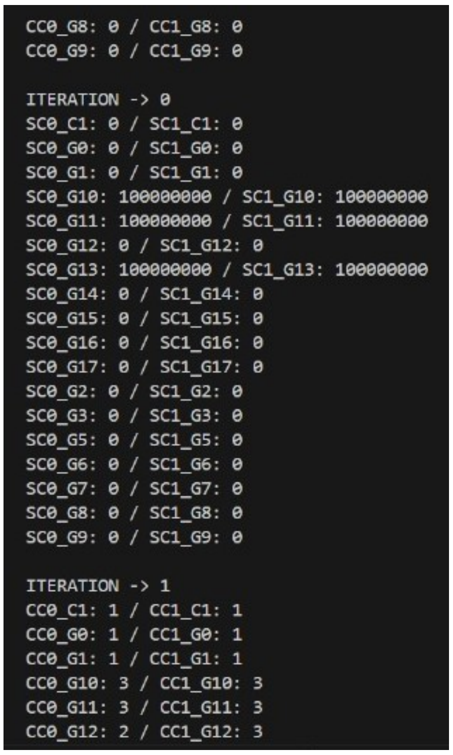
Example s27.v



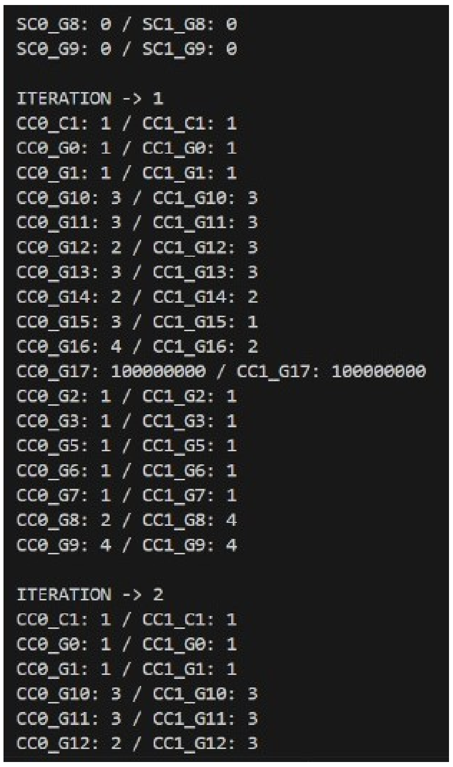
```
IN_OUT_FANOUT::
INPUT C1 G0 G1 G2 G3
OUTPUT G17 G5 G10 G6 G11 G7 G13 G14 G8 G15 G12 G16 G9


INPUT_LIST::
C1 G0 G1 G2 G3

OUTPUT_WIRES::
G14 G17 G8 G15 G16 G9 G10 G11 G12 G13

WIRE_LIST::
C1 G0 G1 G10 G11 G12 G13 G14 G15 G16 G17 G2 G3 G5 G6 G7 G8 G9
ITERATION -> 0
CC0_C1: 1 / CC1_C1: 1
CC0_G0: 1 / CC1_G0: 1
CC0_G1: 1 / CC1_G1: 1
CC0_G10: 100000000 / CC1_G10: 100000000
CC0_G11: 100000000 / CC1_G11: 100000000
CC0_G12: 0 / CC1_G12: 0
CC0_G13: 100000000 / CC1_G13: 100000000
CC0_G14: 0 / CC1_G14: 0
CC0_G15: 0 / CC1_G15: 0
CC0_G16: 0 / CC1_G16: 0
CC0_G17: 0 / CC1_G17: 0
CC0_G2: 1 / CC1_G2: 1
CC0_G3: 1 / CC1_G3: 1
CC0_G5: 1 / CC1_G5: 1
CC0_G6: 1 / CC1_G6: 1
```



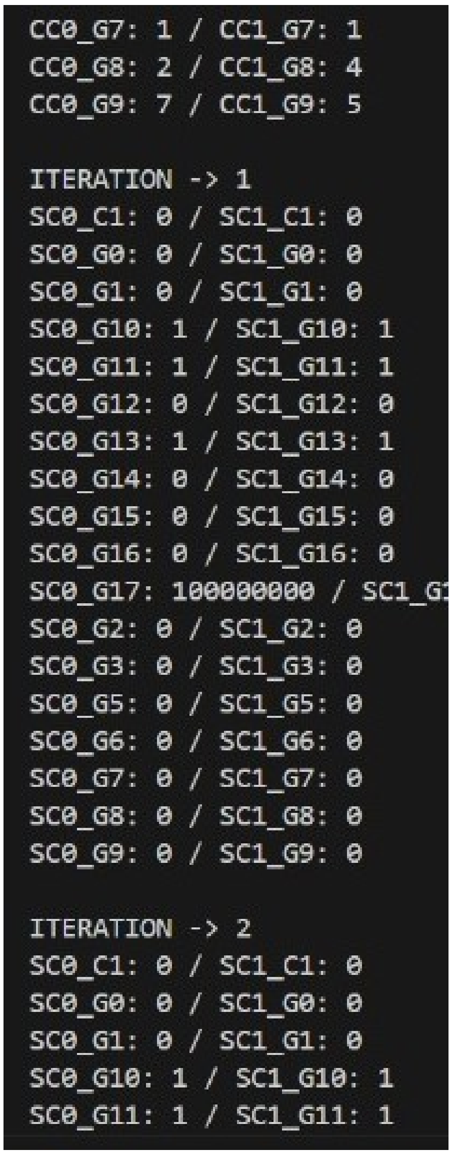
```
GATE_LIST::
NOT G14 G0
NOT G17 G11
AND G8 G14 G6
OR G15 G12 G8
OR G16 G3 G8
NAND G9 G16 G15
NOR G10 G14 G11
NOR G11 G5 G9
NOR G12 G1 G7
NOR G13 G2 G12

FF_LIST::
DFF C1 G5 G10
DFF C1 G6 G11
DFF C1 G7 G13


FANOUT::


IN_OUT_FANOUT::
INPUT C1 G0 G1 G2 G3
OUTPUT G17 G5 G10 G6 G11 G7 G13 G14 G8 G15 G12 G16 G9
```



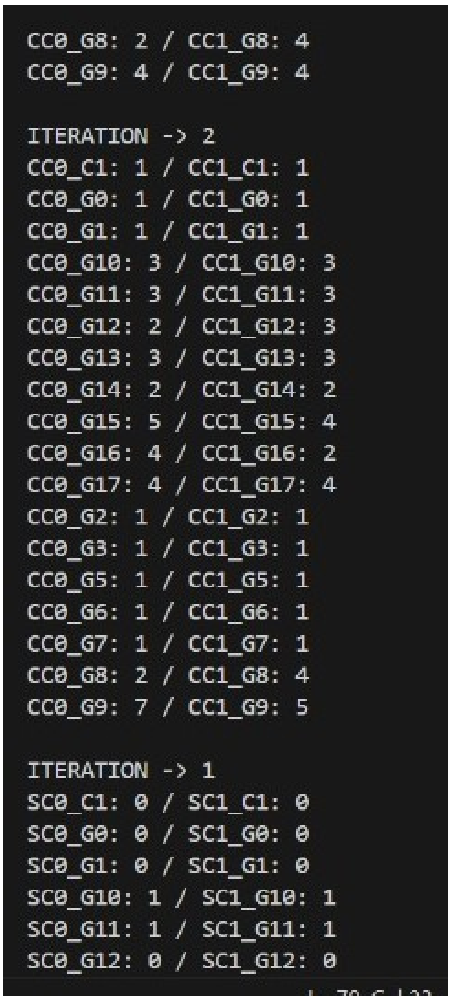
```
CC0_G8: 0 / CC1_G8: 0
CC0_G9: 0 / CC1_G9: 0

ITERATION -> 0
SC0_C1: 0 / SC1_C1: 0
SC0_G0: 0 / SC1_G0: 0
SC0_G1: 0 / SC1_G1: 0
SC0_G10: 100000000 / SC1_G10: 100000000
SC0_G11: 100000000 / SC1_G11: 100000000
SC0_G12: 0 / SC1_G12: 0
SC0_G13: 100000000 / SC1_G13: 100000000
SC0_G14: 0 / SC1_G14: 0
SC0_G15: 0 / SC1_G15: 0
SC0_G16: 0 / SC1_G16: 0
SC0_G17: 0 / SC1_G17: 0
SC0_G2: 0 / SC1_G2: 0
SC0_G3: 0 / SC1_G3: 0
SC0_G5: 0 / SC1_G5: 0
SC0_G6: 0 / SC1_G6: 0
SC0_G7: 0 / SC1_G7: 0
SC0_G8: 0 / SC1_G8: 0
SC0_G9: 0 / SC1_G9: 0

ITERATION -> 1
CC0_C1: 1 / CC1_C1: 1
CC0_G0: 1 / CC1_G0: 1
CC0_G1: 1 / CC1_G1: 1
CC0_G10: 3 / CC1_G10: 3
CC0_G11: 3 / CC1_G11: 3
CC0_G12: 2 / CC1_G12: 3
```



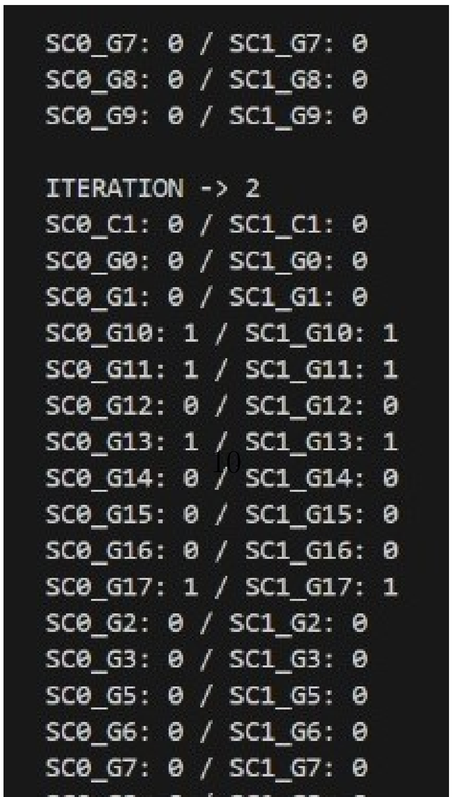
```
SC0_G8: 0 / SC1_G8: 0
SC0_G9: 0 / SC1_G9: 0

ITERATION -> 1
CC0_C1: 1 / CC1_C1: 1
CC0_G0: 1 / CC1_G0: 1
CC0_G1: 1 / CC1_G1: 1
CC0_G10: 3 / CC1_G10: 3
CC0_G11: 3 / CC1_G11: 3
CC0_G12: 2 / CC1_G12: 3
CC0_G13: 3 / CC1_G13: 3
CC0_G14: 2 / CC1_G14: 2
CC0_G15: 3 / CC1_G15: 1
CC0_G16: 4 / CC1_G16: 2
CC0_G17: 100000000 / CC1_G17: 100000000
CC0_G2: 1 / CC1_G2: 1
CC0_G3: 1 / CC1_G3: 1
CC0_G5: 1 / CC1_G5: 1
CC0_G6: 1 / CC1_G6: 1
CC0_G7: 1 / CC1_G7: 1
CC0_G8: 2 / CC1_G8: 4
CC0_G9: 4 / CC1_G9: 4

ITERATION -> 2
CC0_C1: 1 / CC1_C1: 1
CC0_G0: 1 / CC1_G0: 1
CC0_G1: 1 / CC1_G1: 1
CC0_G10: 3 / CC1_G10: 3
CC0_G11: 3 / CC1_G11: 3
CC0_G12: 2 / CC1_G12: 3
```



```
CC0_G7: 1 / CC1_G7: 1
CC0_G8: 2 / CC1_G8: 4
CC0_G9: 7 / CC1_G9: 5

ITERATION -> 1
SC0_C1: 0 / SC1_C1: 0
SC0_G0: 0 / SC1_G0: 0
SC0_G1: 0 / SC1_G1: 0
SC0_G10: 1 / SC1_G10: 1
SC0_G11: 1 / SC1_G11: 1
SC0_G12: 0 / SC1_G12: 0
SC0_G13: 1 / SC1_G13: 1
SC0_G14: 0 / SC1_G14: 0
SC0_G15: 0 / SC1_G15: 0
SC0_G16: 0 / SC1_G16: 0
SC0_G17: 100000000 / SC1_G
SC0_G2: 0 / SC1_G2: 0
SC0_G3: 0 / SC1_G3: 0
SC0_G5: 0 / SC1_G5: 0
SC0_G6: 0 / SC1_G6: 0
SC0_G7: 0 / SC1_G7: 0
SC0_G8: 0 / SC1_G8: 0
SC0_G9: 0 / SC1_G9: 0

ITERATION -> 2
SC0_C1: 0 / SC1_C1: 0
SC0_G0: 0 / SC1_G0: 0
SC0_G1: 0 / SC1_G1: 0
SC0_G10: 1 / SC1_G10: 1
SC0_G11: 1 / SC1_G11: 1
```



```
CC0_G8: 2 / CC1_G8: 4
CC0_G9: 4 / CC1_G9: 4

ITERATION -> 2
CC0_C1: 1 / CC1_C1: 1
CC0_G0: 1 / CC1_G0: 1
CC0_G1: 1 / CC1_G1: 1
CC0_G10: 3 / CC1_G10: 3
CC0_G11: 3 / CC1_G11: 3
CC0_G12: 2 / CC1_G12: 3
CC0_G13: 3 / CC1_G13: 3
CC0_G14: 2 / CC1_G14: 2
CC0_G15: 5 / CC1_G15: 4
CC0_G16: 4 / CC1_G16: 2
CC0_G17: 4 / CC1_G17: 4
CC0_G2: 1 / CC1_G2: 1
CC0_G3: 1 / CC1_G3: 1
CC0_G5: 1 / CC1_G5: 1
CC0_G6: 1 / CC1_G6: 1
CC0_G7: 1 / CC1_G7: 1
CC0_G8: 2 / CC1_G8: 4
CC0_G9: 7 / CC1_G9: 5

ITERATION -> 1
SC0_C1: 0 / SC1_C1: 0
SC0_G0: 0 / SC1_G0: 0
SC0_G1: 0 / SC1_G1: 0
SC0_G10: 1 / SC1_G10: 1
SC0_G11: 1 / SC1_G11: 1
SC0_G12: 0 / SC1_G12: 0
```



```
SC0_G7: 0 / SC1_G7: 0
SC0_G8: 0 / SC1_G8: 0
SC0_G9: 0 / SC1_G9: 0

ITERATION -> 2
SC0_C1: 0 / SC1_C1: 0
SC0_G0: 0 / SC1_G0: 0
SC0_G1: 0 / SC1_G1: 0
SC0_G10: 1 / SC1_G10: 1
SC0_G11: 1 / SC1_G11: 1
SC0_G12: 0 / SC1_G12: 0
SC0_G13: 1 / SC1_G13: 1
SC0_G14: 0 / SC1_G14: 0
SC0_G15: 0 / SC1_G15: 0
SC0_G16: 0 / SC1_G16: 0
SC0_G17: 1 / SC1_G17: 1
SC0_G2: 0 / SC1_G2: 0
SC0_G3: 0 / SC1_G3: 0
SC0_G5: 0 / SC1_G5: 0
SC0_G6: 0 / SC1_G6: 0
SC0_G7: 0 / SC1_G7: 0
```