LOGIC LEVELIZATION

We first make a map to store the logic levels of the gates in it.

First we assign level 0 to the primary inputs.

for (auto it : inputs)

    {

        log[it] = 0;

    }

Then for the gates, we first check whether the gate is dummy or not.

If it’s a dummy gate, the gate and the output line get the same logic level as it’s input line.

 if (ge[k].type == "dummy")

        {

            log[ge[k].g\_name] = log[ge[k].inputs[0]];

            log[ge[k].outputs[0]] = log[ge[k].inputs[0]];

        }

If it’s a logic gate, we apply the algorithm by checking the maximum level among it’s inputs, incrementing it by 1 and assigning it to the gate and the output line.

int maxof = 0;

            for (int l = 0; l < ge[k].inputs.size(); l++)

            {

                maxof = max(log[ge[k].inputs[l]], maxof);

            }

            log[ge[k].g\_name] = maxof + 1;

            log[ge[k].outputs[0]] = maxof + 1;

BRANCHING

We have 2 “map” here, one which maps a gate to it’s indexing and other which maps gates to their outputs.

    map<string, vector<string>> mp;

    map<string, int> gate\_mp;

  int ind = 0;

    for (auto it : module.gates)

    {

        gate\_mp[it.g\_name] = ind;

        ind++;

    }

    for (auto it : module.gates)

    {

        for (auto it2 : it.inputs)

        {

            mp[it2].push\_back(it.g\_name);

        }

    }

Then, we check if the number of output gates in the map of a gate is more than one, which means that the same output line is branched into multiple gates. Then, we apply the branching algorithm.

* For each gate connected to the current input, it retrieves the gate's information.
* It generates a unique string **ch** based on the input's name and a character corresponding to the index of the gate.
* It updates the input of the gate to be the generated unique string **ch** and adds it to the **module**'s outputs, when the input of the gate is the one which is being branched into multiple input lines.

 if (it.second.size() >= 2)

        {

            int ind = 0;

            for (auto it2 : it.second)

            {

                Gate prev = module.gates[gate\_mp[it2]];

                vector<string> prev\_inp = prev.inputs;

                string ch = it.first + (char)((int)'a' + ind);

                for (int i = 0; i < prev\_inp.size(); i++)

                {

                    if (prev\_inp[i] == it.first)

                    {

                        module.gates[gate\_mp[it2]].inputs[i] = ch;

                        module.outputs.push\_back(ch);

                        break;

                    }

                }

* It creates a dummy gate (**g**) with a unique name, type "dummy", and input as the current input name and output as the generated unique string **ch**.
* It inserts the dummy gate into the **module**'s gates vector at the position of the current gate being processed.
* It then updates the **gate\_mp** map with the new indices of gates in the **module**.

 Gate g;

                g.g\_name = "branch" + to\_string(name++);

                g.type = "dummy";

                g.inputs.push\_back(it.first);

                g.outputs.push\_back(ch);

                auto pos = module.gates.begin() + gate\_mp[it2];

                module.gates.insert(pos, g);

connectNodes

iterating through each primary input, we create a new node, give it a name and type. We assign its index and thereby increase the indexing by 1 to be added to another input.

for (auto it : module.inputs)

    {

        Node \*ne = new Node;

        ne->name = it;

        ne->type = "INPUT";

        ne->next = NULL;

        // nodes.push\_back(ne);

        mp[it] = ind++;

        // cout << it << " " << mp[it] << " " << ind << endl;

    }

iterating through each output gate in “module.output”, we create a new node. We assign its index and thereby increase the indexing by 1 to be added to another output.

for (auto it : module.outputs)

    {

        Node \*ne = new Node;

        ne->name = it;

        ne->next = NULL;

        // nodes.push\_back(ne);

        mp[it] = ind++;

        // cout << it << " " << mp[it] << " " << ind << endl;

    }

At the end of it, all the input and output gates of the circuits are indexed accordingly and we call the BRANCHING function, explain above.

    branching(module);