SOFTWARE ASSIGNMENT 2

In this assignment, we are required to work with cicuits containing D Flip Flops. The assignment consists of 2 parts.

PART-1

We are required to compute the maximum delay of all paths. A DFF has one input and one output. The output signal of the DFF is 0.

Algorithm

We have stored each of primary inputs, internal signals and primary outputs in vectors. We map the gates with their respective smallest delays. We define an adjacency list in which we map each signal to the gate it asires from and the input signals of that gate. We initialize the primary inputs with delay 0, internal signals with delay -1 and primary outputs with delay -1. We then define a function to calculate the delays of internal signals and primary outputs. This is a recurrsive function that finds the delay by adding the delay of the gate to maximum of the inputs of the gate. If the gate is a DFF, we consider this signal as 0 delay. We pass each of the internal signals and primary outputs to this fuction and calculate its delay. Then we find the longest delay by iterating over all these internal signals and primary outputs. We then simply write it in the longest_delay file.

Implementation

```
double calc_del(map<string,double>& gate_del,map<string,double>& delays,map<string,vector<string>>& adj_list,string& signal){
    if(delays[signal] != -1){
        return delays[signal];
    }
    else{
        if(adj_list[signal].size() == 3){
                double i1 = calc_del(gate_del,delays,adj_list,adj_list[signal][1]);
                double i2 = calc_del(gate_del,delays,adj_list,adj_list[signal][2]);
                return max(i1,i2)+gate_del[adj_list[signal][0]];
        }
        else{
            if(adj_list[signal][0]=="DFF"){
                return 0.0;
            }
            else{
                return calc_del(gate_del,delays,adj_list,adj_list[signal][1]) + gate_del[adj_list[signal][0]];
            }
        }

    }
}
```

Calculating function

```
for(auto i:internals){
    delays[i] = calc_del(gate_del,delays,adj_list,i);
}

for(auto i:outputs){
    delays[i] = calc_del(gate_del,delays,adj_list,i);
}
```

Calculation of the delays of signals

```
  for (auto in : internals) {
      longest = max(delays[in],longest);
  }

  for (auto out : outputs) {
      longest = max(delays[out],longest);
  }
```

Finfing the longest delay

The calculating function has an amortized time complexity of O(logn). We run this function n times(for each signal). Therefore, amortized time complexity of the the above algorithm is O(nlogn).

Testing Stratergy

We created a test case to test if the program runs correctly if the signals and gates are not organiPart -zed in ordered. We also used the same test case to test if it runs correctly for circuits in which an output is used for inputs for multiple gates.
In the second test case we are trying to check the correctness of the program if the circuit contains loops.

PART-1

We are required to compute the optimized minimum area for a bounded longest time delay and take care of both time and area.

Algorithm

We start with the maximum delay of each gate. We calculate the longest delay of the circuit. If this delay is less than the constraint, we return the area. If not, we check for the longest delay and change its ancestor gates one by one and do this until the longest delay comes under the value of constraint delay. If this doesn't help we the start decreasing the other gates in a similar pattern and do the same checks. We maintain a minimum current area variable. Anytime, we get a longest delay lower than the constraint delay, we compare it with the minimum and keep doing till a satisfactory point. Then we write the current minimum area in the file.

Implementation

```
void minimise(string signal,double allowed){
        if(delays2[signal]==0){
                return;
        }
        string type = adj_list2[signal][0] ;
        if(type == "DFF"){
                return;
        }
        string imp = adj_list2[signal].back();
        int i = stoi(imp);
        if(i<2 && delays2[signal] > allowed){
                i++;
                delays2[signal] = -1;
                adj_list2[signal][adj_list2[signal].size()-1] = to_string(i);
                for(auto j:all_sig){
                        delays2[j] = calc_del2(j);
                }
                minimise(signal,allowed);
        }
        else if(i == 2 && delays2[signal] >= allowed){
                double p = delays2[signal];
                delays2[signal] = -1;
                string max_sig = max_signal(signal);
                minimise(max_sig,allowed - gate_del2[adj_list2[signal][0]][i]);
                for(auto j:all_sig){
                        delays2[j] = calc_del2(j);
                }
        }
        else{
                return;
        }
}
```

```
double calc_area() {
    double ret = 0.0;
    for(auto signal:all_sig){
        string type = adj_list2[signal][0];
        if(type !="DFF"){
                string imp = adj_list2[signal][adj_list2[signal].size()-1];
                int t_i = stoi(imp);
                ret = ret + areas[adj_list2[signal][0]][t_i];
                }
    }
    return ret;
}
```

```
infile1.close();
for(auto i:primary2){
        delays2[i] = 0;
}
for(auto i:internals2){
        delays2[i] = -1;
}
for(auto i:outputs2){
        delays2[i] = -1;
}
for(auto i:all_sig){
        delays2[i] = calc_del2(i);

}
for(auto i:all_sig){
        minimise(i,max_allowed);
}
double area = calc_area();

ofstream outputFile(argv[5]);
outputFile << area << endl;
outputFile.close();
```

Aditya Jha
Harshal Kumar