Software Assignment 2

Daksh Dhaker-2022CS51264 ,Umesh Kumar-2022CS11115
COL215:- Digital Logic & System Design
Group-3

1 Algorithm

1.1 Structure of Node

We have created a Node structure using classes. Each node represents either **a primary input** or **an internal signal** or **primary output**. Since each primary output or internal signal has gate associated with it whose output is that primary output or internal signal, therefore **Each Node represents a Gate or a primary input where Node is mapped to the name of primary output or internal signal or it is mapped to the name of primary input**. A Node has four attributes:

- **delay_val** : For **Primary Outputs** and **Internal signals** : It is the delay value of the gate whose output signal is this Primary output or internal signal.
  For **Primary inputs** : It is zero.
- **inputs** : **For Primary Outputs and Internal signals** : Basically Node here represents a gate whose output is that primary Output or internal signal to which it is mapped with. So this input vector contains all the Nodes(gates outputs or primary inputs) which are input to this gate.**(Basically it is the adjacency list)**
  For Primary Inputs : It is empty since there are no Nodes input to this.
- **outputs** : **For Primary Inputs and Internal signals** : It contains all Nodes to which this signal goes as input.
  For Primary Outputs : This is empty since there are no such Nodes(or gates) to which this goes as input.
- **type** : It represents the type of gate which this Node represents (string type). (It can take values like AND2, OR2, NOR2 etc).

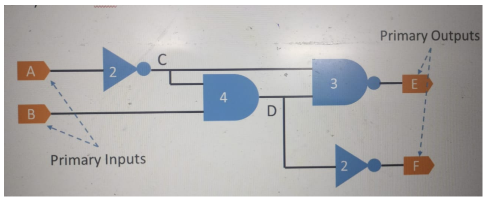


Image 1.1

For example ,for above : (Note inputs and outputs vectors **contains addresses of Nodes not their names as strings**)
Node A = {delay_val =0 , inputs = [ ], outputs = [C]}
Node B = {delay_val =0, inputs =[ ], outputs = [D]}
Node C = {delay_val = 2, inputs = [A], outputs = [D,E]}
Node D = {delay_val = 4, inputs = [B,C], outputs = [E,F]}
Node E = {delay_val = 3, inputs = [C,D], outputs = [ ]}
Node F = {delay_val = 2,inputs = [D], outputs = [ ]}

1.2 Structure of gate_type

We have created this structure "gate_type" which stores pair of delay value and value of area of a particular gate. This structure has two attributes:

- **delay** : this contains the delay value of a gate.
- **area** : this contains the value of the area of that gate. (for eg. In NAND2_1 gate, delay = 3.5 and area = 11.2 (let's assume). So, this entire NAND2_1 gate is represented by this structure gate_type).

1.3 Structure of gate_cluster

We have also created a structure called "gate_cluster" which contains three gate_types attributes :

- first : (type – gate_type) This attribute represents the Fastest type of a particular type of gate.
- second : (type – gate_type) This attribute represents the Moderate type of a gate.
- third : (type – gate_type) This attribute represents the Slowest type of a gate

Maps or Vectors used in the code:
Here unordered maps are used because of O(1) time in accessing any elements using a key to it.

- **delays (Unordered Map)** :
  key = Name of gate(type-string)(like AND2, NAND3 etc)
  value = value by which it delays it inputs (type - double)

- **node (Unordered Map)** :
  key = Name of Primary input or Internal signal or Primary Output which this gate represents.(type-string)(eg: A, B,C)
  value = address of Node which it represents (type – Node*)

- **str (Unordered Map)** :
  key = address of Node which it represents (type – Node*)
  value = Name of that Primary input or Internal signal or Primary Output which it represents (type – string).

- **answer (Unordered Map)** : **key** = address of Node

(For answer of Part-A) value = The delay till that Node (type- double)
This stores the delays of internal_signals, primary_inputs and primary_outputs. Final answer is stored in it for part-A.

- **internal_sig(Unordered Map)** :
  It tells us those internal signals which are not primary outputs and not primary inputs. Sometimes, in some testcases primary_outputs and internal_signals can be given same in circuit.txt (as it was given in Software Assignment-1 it handles this)
  Key = string name of internal_signals (type – string)
  Value = 1 if it is internal signal and not a primary_output and primary_input ,else it is zero(type – int)

- **primary_outputs(UnorderedMap)** :
  key = Name of gate(type-string)(like AND2, NAND3 etc)
  value = 1 if it is a primary_output else 0

- **primary_inputs(UnorderedMap)** :
  key = Name of gate(type-string)(like AND2, NAND3 etc)
  value = 1 if it is a primary_input else 0 (type - double)

- **delay_calculate(vector<string>)** :
  It contains all names of all primary_outputs and DFF inputs whose delays have to be calculated and their maximum has to be taken.

- **gate_num(UnorderedMap)** :
  key = Name of output of a gate(type-string)
  value = 0,1 or 2 .(0 if fastest, 1 if moderate and 2 if slowest).

- **gates (vector<string>)** :
  It contains name of the output of each gate used in circuit.

1.4 Algorithm for Part-A (Recursion)(Pseudo Code of original code is given below)

```
actual_delay (output_node , answer, gate_num, delays, str)
    if( output_node →inputs.size()==0) // **(means this is a primary output)**
            return answer[output_node] =0; //**( return 0 and store it in answer and thus it will be marked visited)**
    else if(answer[output_node] !=0) **( this node is previously visited so its value is present in answer already)**
            return answer[output_node];
    else
            int ad = actual_delay(output_node→input[0]); // **(this loop calculates max of actual_delay of all nodes in inputs of current node**
            for(int i=0;i<output_node →input.size();i++){
                    ad = max(ad, actual_delay( output_node→input[i],answer));
            }
             if(gate_num[str[output_node]]==0) { // **This implies this is fastest gate so we take delay value as fastest value**
             delay_value = delays[output_node]→first →delay;
             else if(gate_num[str[output_node]] =1) { // **This implies that this gate is moderate type so we take delay value as moderate value**
                    delay_value = delays[output_node]→first→delays;
            }
             else{ // **This means this is slowest gate so we take delay value as slowest value**
                     delay_value = delays[output_node]→first→delays;}
            ad = ad+output_node→delay_val; // **in addition to delay value of current node**
```

```
        answer[output_node] = ad;
        return ad;
```

The second last line answer[output_node] = ad marks this node as visited by changing its value from zero in answer map.

Explanation : The delay value of a Node is basically the max(delay_values of all its inputs) + (delay_val corresponding to that gate which this Node represents). This delay_val depends on the type of gate which can be slow, fast or moderate. So, for this purpose we have made an unordered map gate_num which has a value (0 or 1 or 2), which means which type of gate it is; 0 means fastest, 1 means moderate and 2 means slowest. This code is implemented above using concepts of recursion and Dynamic Programming, where each Node is visited once to evaluate its delay value. So basically to solve a circuit we are solving sub-circuits, then using the outputs of these sub-circuits to solve this bigger circuit.

Recursion used:
actual_delay in current node = (max of actual_delay of all nodes in inputs of curr. Node) + delay_val)
Base case (of recursion) : actual_delay of a primary input Node = 0.

Time Complexity : In the above algorithm each node is visited. If it is not visited till now then the algorithm will visit all node, whose value are required to calculate the value of current Node. But then in future, if value of any one of these Nodes will be required for some another calculation, then these nodes will not be evaluated again since they are already visited, their actual_delay values will be directly accessed from map (answer) in O(1) time. So each Node is visited exactly once and then their value are directly used from the map. So to calculate delays in all outputs , all nodes must be visited at least once . Thus the time complexity of our algorithm is O(V+E) where V is the total number of Vertices and E is the number of edges.
Here n = number of Edges, because if a Node A is not visited yet, and to calculate its value ,we need to calculate values of Node B and Node C. let C is not visited yet but B was visited earlier in some other calculation, but it was visited through a different edge not the edge joining A and B. so now the value of B will be directly used after visiting this edge joining A and B. So like this we have to travel all the edges of the graph.
Hence the time complexity = O(V+E) : E = number of edges and V = number of vertices

In worst case in a connected graph E = O(V^2)
In worst case time complexity = O(V^2)

1.5 Algorithm for Part-B

Algorithm : Mainly, in this algorithm basically first we calculate the threshold values of all internal signals, primary inputs and primary outputs. Threshold value is basically the greatest value at which it must be present so that longest combinational delay is bounded by the given delay constraint. We store these value in a map called answer2. These threshold values are calculated with gates operating at their fastest mode because it will maximise the threshold values. Then we are applying time_avail function which calculates the minimum area. It calculates it in the way, assume we are at certain gate we first apply this function at its output it will return the time at which this output must be ready and also updates the total area that should be taken for this gate in order to minimize the area, such that the value which it return is smaller than the threshold value of the signal. In this way, this function will applied on all gates (using dynamic programming and recursion) and corresponding area will be calculated in the temp variable which is passed into it by reference. So the final answer will be stored in temp variable.

- **required_delay function** : In this part first, we are using the required_delay function which returns the values of all internal signals and primary inputs , such that this value is the greatest value at which this internal signal must be ready for longest combinational delay to be bounded by the given delay constraint (This is the same function used in part-B of Software Assignment-1).
  (Time Complexity = O(E+V), the explanation is same as it was in part A as we are visiting all edges and nodes exactly once because as we encounter an already visited node that instead of calculating its value again we simply return its value from the map(answer2)).

- **time_avail function** : This function as described above basically works in similar way as actual_delay function(in part-1) except that it takes that area such that area is minimized and the delay at output is less than the threshold calculated by required_delay function. So this function is applied on all gates of this graph and final answer is stored in temp variable which is passed by reference into it.

Time complexity = O(E+V) the explanation is same as for required_delay function.

- **random_min function** : This function is just created to check whether it is possible to get the area which is less than the area returned by time_avail function, by just randomly allocating the delay_values to the gates (at most 10000 times so that time limit does not exceed). This function just checks whether we can find a circuit with longest delay less than delay constraint and check we apply 10000 times. This is just an optimization. This random allocation is done by giving random value 0,1,2 to the gates in gate_num map.
- **optimized_area** : This function is divided in two parts:-
  - **if(nodes <=10)** : In this case exponential solution also works because $3^{10} < 10^7$ and 10^7 operations roughly takes 1 second so if nodes <=10 (here nodes represents number of gates) then exponential solution also works very fast. So for nodes <=10 we have used this method.
  - **If(nodes >10)** : in This case we use three algorithm in order to try to find the minimum area such that delay constraint is met.
    - Firstly, we apply the time_avail function after calculating the answer2(threshold values using required_delay function). This time_avail gives us answer, we say ans2 (This is the best area we can find by our algorithm This computation works in O(V+E) time as described above.)
    - Secondly, we find the answer by random_min function. This is an attempt to find whether we can find an area better than computed in part-1 satisfying the delay constraint. Call this answer as ans1.
    - Third time, we apply time_avail function (at max 10000/nodes times). Note that we have taken the delay_values as the fastest values in calculating threshold values. This time we randomly allocate the delay_values to gates for calculating threshold values using required_delays function. This random allocation is done using gate_num unordered map. Call this answer as ans4.

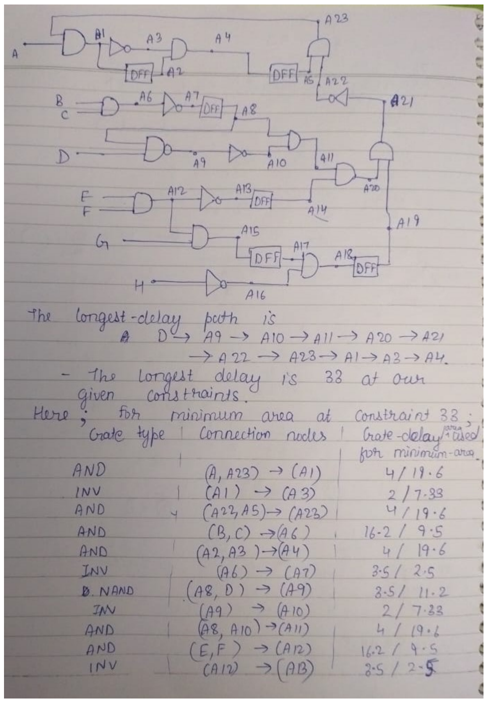
  Then we take the minimum of all the areas we get in these three parts to report the final minimum area.

Time Complexity : For the first answer ans2, time taken is O(E+V) because the time_avail function and required_delay function both work in O(E+V). Secondly ans1 takes time O((10000/V)∗(V+E)); this 10000 factor is taken to make sure that the time limit does not exceed the given value. For ans4 again it takes O((10000/V) ∗ (V+E)) similarly as we explained for part2.
This is so if nodes <=10 ,then maximum time is 3^{10} ∗(E+V). This can be computed very fast if nodes > 10 and then maximum time taken will be O(E+V) as described above. Then we can say maximum time is bounded by O(E+V).
So actually maximum time = O((E+V) ∗ 10000), but 10000 is a constant, so we can say time complexity is O(E+V).

Time Complexity = O(E+V)

2 Testing Strategy (test case-1 and test case-2 are solved manually using above algorithm for understanding)

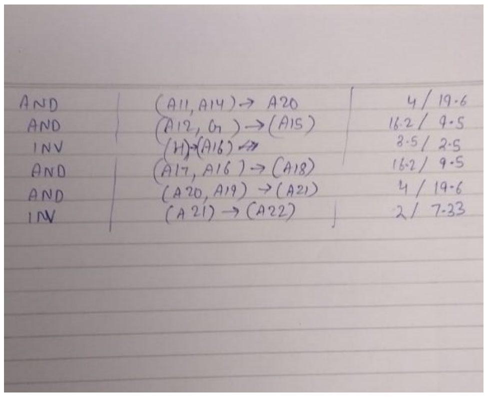
- **Test Case-1**



The longest-delay path is
~~A~~ D → A9 → A10 → A11 → A20 → A21
→ A22 → A23 → A1 → A3 → A4.
- The longest delay is 33 at our given constraints.
Here; for minimum area at constraint 33;

Gate type	Connection nodes	Gate-delay/area used for minimum-area
AND	(A, A23) → (A1)	4 / 19·6
INV	(A1) → (A3)	2 / 7·33
AND	(A22, A5) → (A23)	4 / 19·6
AND	(B, C) → (A6)	16·2 / 9·5
AND	(A2, A3) → (A4)	4 / 19·6
INV	(A6) → (A7)	3·5 / 2·5
~~B~~. NAND	(A8, D) → (A9)	3·5 / 11·2
INV	(A9) → (A10)	2 / 7·33
AND	(A8, A10) → (A11)	4 / 19·6
AND	(E, F) → (A12)	16·2 / 9·5
INV	(A12) → (A13)	3·5 / 2·5

Image 2.1.1

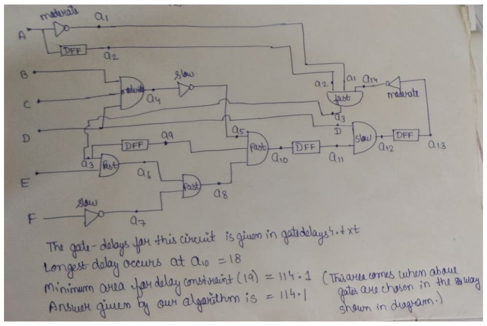


AND	(A11, A14) → A20	4 / 19·6
AND	(A12, G) → (A15)	16·2 / 9·5
INV	(H) → (A16)	3·5 / 2·5
AND	(A17, A16) → (A18)	16·2 / 9·5
AND	(A20, A19) → (A21)	4 / 19·6
INV	(A21) → (A22)	2 / 7·33

Image 2.1.2

We chose this test case because in our circuits, cycles are also possible. So to check that our algorithm works correctly we made a test case with large number of cycles and also there are no primary outputs. So the largest combinational delay has to be at the input of some DFF. To check the correctness of our algorithm over large test cases including cycles we made this circuit.

- **Test case-2**

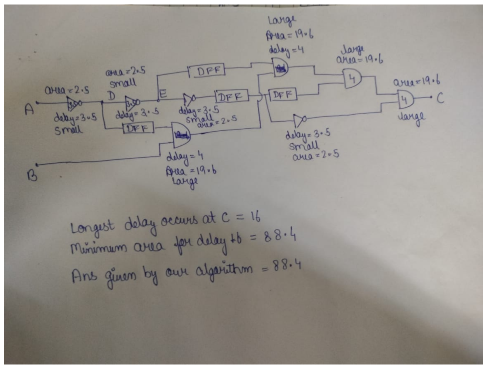


The gate-delays for this circuit is given in gatedelays4.txt
Longest delay occurs at a_{10} = 18
Minimum area for delay constraint (19) = 114.1 (This area comes when above gates are chosen in the way shown in diagram.)
Answer given by our algorithm is = 114.1

Image 2.2

This circuit also tests the same thing the correctness of algorithm in case of cycles. The basic difference in this case is that in this test case nodes <= 10 whereas in above test case nodes >= 10 so above test case tests part-2 of our algorithm which gives an approximate answer close to our original answer whereas part-1 is tested by this test case which gives the exact answer using the algorithm as defined above.

- **Test case-3**

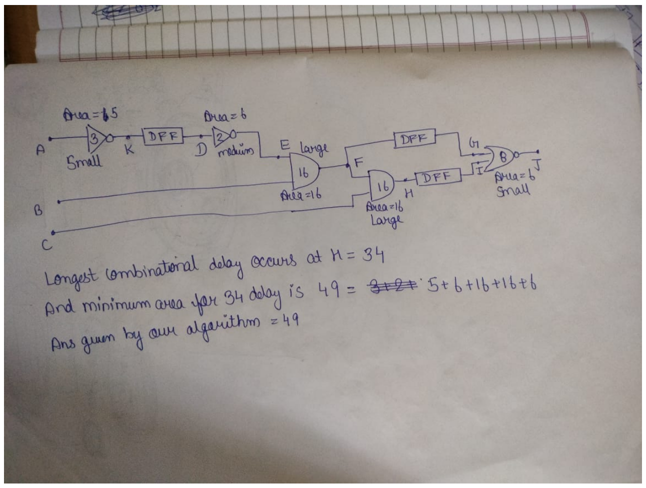


Longest delay occurs at C = 16
Minimum area for delay 16 = 88.4
Ans given by our algorithm = 88.4

Image 2.3

A smaller test case was made to make sure that Part-1 of our algorithm gives the correct answer (as it is easy to check for smaller circuits that the answer is exact or approximate). So, we know ,in this nodes <=10 so our algorithm must give exact answer.

- **Test case-4**



Longest combinational delay occurs at H = 34
And minimum area for 34 delay is 49 = ~~3+2+~~ 5+6+16+16+6
Ans given by our algorithm = 49

Image 2.4

One more smaller test case was created to test Part-1 of our algorithm. It was also run to check how close the answer of part-1 and part-2 comes when we run them simultaneously. This was also done for test case-3. So these test case helped to check that our algorithm gives answer close to exact answer or not.

- **Test case-5**

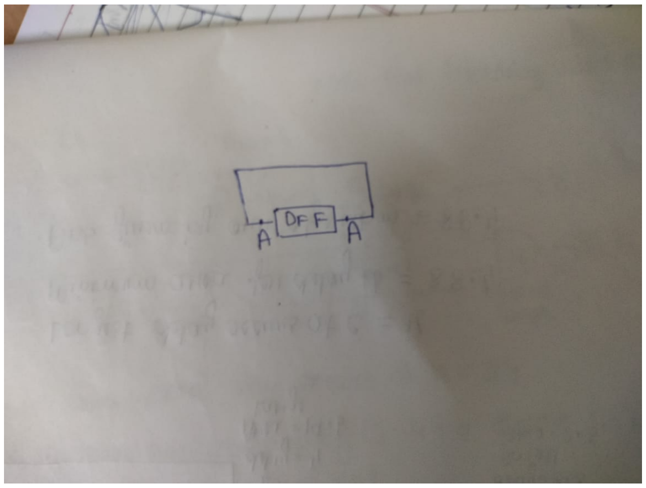
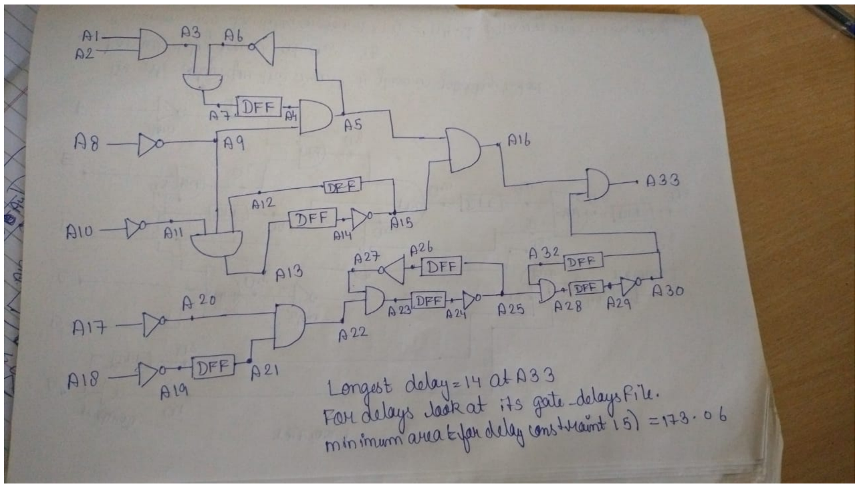


Image 2.5

This test case is added to show triviality of our algorithm (It works on cases as trivial as this.

A few more test cases:-

- **Test case-6**



Longest delay = 14 at A33
For delays look at its gate_delays file.
minimum area (for delay constraint 15) = 173.06

Image 2.6

This is a general test case.

END OF REPORT