ASP Timer

STA Engine Technical Report

1

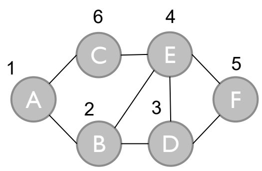
Outline

- Prerequisites
  - Graphs Theory Prerequisites
    - Depth First Search (DFS)
    - Breadth First Search (BFS)
    - Topological Sort
  - Interpolation & Extrapolation
    - Linear
    - Bilinear
    - Examples
  - Longest/Shortest Path Algorithm
- Longest Path
  - Pseudocode
  - Levelisation
- STA Theory
  - Timing Arcs
  - Unateness
  - Transitions

2

Graph Theory: DFS

- Depth First Search (DFS)
  - The DFS traversal starts from root nodes and explores as far as possible along each branch following the neighbors (successors) of the nodes.
  - A Graph may contain cycles, therefore newly visited nodes will have to be marked so that they won't be revisited
  - When the DFS cannot proceed any further across a branch it continues exploring from the first unmarked successor closest to the root.



Example of a valid DFS traversal: A, B, D, E , F, C

Here the traversal began from node A.

The successors that were chosen to be visited were random.

Graph Theory: DFS

- Depth First Search (DFS)
  - Recursive DFS traversal pseudocode.

```
procedure DFS(G,v):
    label v as discovered
    for all edges from v to w in G.adjacentEdges(v) do
        if vertex w is not labeled as discovered then
            recursively call DFS(G,w)
```

  - Iterative DFS traversal pseudocode

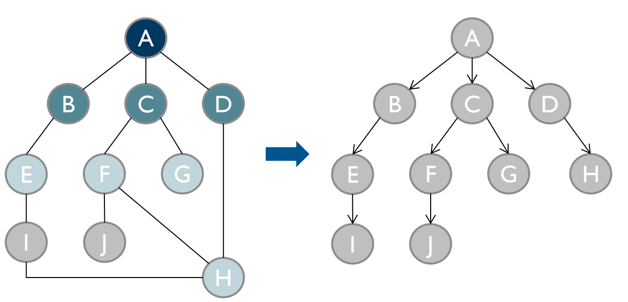
```
procedure DFS-iterative(G,v):
    let S be a stack
    S.push(v)
    while S is not empty
        v = S.pop()
        if v is not labeled as discovered:
            label v as discovered
            for all edges from v to w in G.adjacentEdges(v) do
                S.push(w)
```

Graph Theory: BFS

- Breadth First Search (BFS)
  - The BFS traversal starts from the root nodes and explores all of the neighbor nodes at the present depth prior to moving on to the nodes at the next depth level.
  - BFS variants can be used to compute the shortest/longest levelisation of a graph as well as the delay propagation of a circuit.
    - For instance, an alternate BFS on a directed graph, in order to proceed to a node, might require all of its predecessors visited.

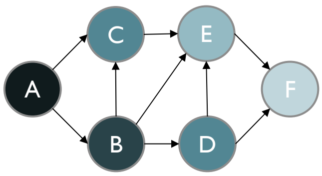
Graph Theory: BFS

- Breadth First Search (BFS)
  - Example of BFS on an undirected graph.
    - Visit order: A, B, C, D, E, F, G, H, I , J
    - Note that H is visited as a 3rd level node as it is connected to D



Graph Theory: Topological Sort

- BFS and Topological Sort
  - We can perform topological sort by using a BFS variant on the DAG
    - During BFS we will visit a node if and only if all of its predecessors are discovered



In this example
- We cannot visit C if we haven't visited B first
- We cannot visit E if we haven't visited B, C, D
- We cannot visit F if we haven't visited E and D

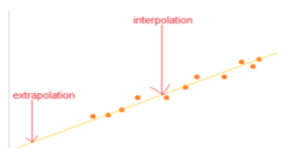
The nodes will be visited in the following order:
A, B, C, D, E, F

8

Interpolation & Extrapolation

- Extrapolation is an estimation of a value based on extending a known sequence of values beyond the area that is known.

- Interpolation is an estimation of a value within two known values in a sequence of values.



- Linear extrapolation/interpolation formula:
  - Let $x_1, x_2, \ldots x_n$ being points where $y(x)$ is known.
  - Let x_* being the point that we desire to estimate y
  - Let x_k and x_{k-1} being the nearest points to x^* where the value of y is known.
  - We can estimate $y(x^*)$ with the following linear formula:

$$y(x_*) = y_{k-1} + \frac{x_* - x_{k-1}}{x_k - x_{k-1}}(y_k - y_{k-1}).$$

if $x_{k-1} < x_* < x_k$ we have interpolation

9

Interpolation & Extrapolation

- Bilinear interpolation is an extension of linear interpolation for interpolating functions of two variables (e.g., x and y) on a rectilinear 2D grid. (respectively for extrapolation)

- Let $z = f(x, y)$ and we want to estimate z_* value at point (x_*, y_*)
  - Let $x_1 < x_* < x_2$ and $y_1 < y_* < y_2$ (interpolation)
  - Let $Q_{11} = (x_1, y_1)$, $Q_{12} = (x_1, y_2)$, $Q_{21} = (x_2, y_1)$ and $Q_{22} = (x_2, y_2)$ (z is known)

  - In order to estimate z_*:

    1) We first perform linear interpolation in the x-direction for each $y = \{y1, y2\}$:

    $$z_i = z(x_*, y_i) = z_{1x} + \frac{x - x_1}{x_2 - x_1}(z_{2x} - z_{1x})$$

    where $z_{1x} = z(x_1, y_i)$, $z_{2x} = z(x_2, y_i)$ for $i = \{1, 2\}$

    2) Interpolate in the y direction using the conputed z_1 and z_2

    $$z_* = z(x_*, y_*) = z_1 + \frac{y - y_1}{y_2 - y_1}(z_2 - z_1)$$

Interpolation & Extrapolation

- **Interpolation/Extrapolation on Arrays:**
  - Let a one-dimensional 4x1 vertical array
  - The estimation of z at point y, $z(x_1, y) = z(y)$ is:

$$z_1 - (y_1 - y)\frac{z_2 - z_1}{y_2 - y_1} \quad \text{if } y < y_1$$

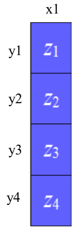
$$z_1 + (y - y_1)\frac{z_2 - z_1}{y_2 - y_1} \quad \text{if } y_1 < y < y_2$$

$$z_2 + (y - y_2)\frac{z_3 - z_2}{y_3 - y_2} \quad \text{if } y_2 < y < y_3$$

$$z_3 + (y - y_3)\frac{z_4 - z_3}{y_4 - y_3} \quad \text{if } y_3 < y < y_4$$

$$z_4 + (y - y_4)\frac{z_4 - z_3}{y_4 - y_3} \quad \text{if } y > y_4$$

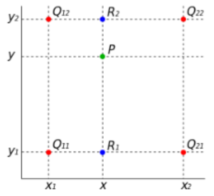
z_i if $y = y_i$ for $i = \{1, 2, 3, 4\}$



  - The same stands for horizontal arrays with the difference that we will be using the x indices instead of y.

Interpolation & Extrapolation

- Interpolation/Extrapolation on Arrays:
  - Let a two-dimensional 4x3 array
  - Let $x_2 < x < x_3$ and $y_2 < y < y_3$
    - Determine z_{first} by linear x-interpolation on z_{22} and z_{32}
    - Determine z_{second} by linear x-interpolation on z_{23} and z_{33}
    - Determine z by linear y-interpolation using z_{first} and z_{second}



	x1	x2	x3
y1	z_{11}	z_{21}	z_{31}
y2	z_{12}	z_{22}	z_{32}
y3	z_{13}	z_{23}	z_{33}
y4	z_{14}	z_{24}	z_{34}

Classic Longest Path Algorithm

```
Algorithm: Longest Path
Input: directed graph G(V,E)
Output: AATs of all nodes v ∈ V based on worst-case (longest) paths
foreach (node v ∈ V)
  AAT[v] = -∞ // all AATs are by default unknown
  AAT[source] = 0 // except source, which is 0
Q = TOPO_SORT(V) // topological order
while (Q != Ø)
  u = FIRST_ELEMENT(Q) // u is the first element in Q
  foreach (neighboring node v of u)
    AAT[v] = MAX(AAT[v],AAT[u]+t[u][v]) // t[u][v] is the (u,v) edge delay
  REMOVE(Q,u) // remove u from Q
```

STA Longest Path Algorithm

```
Algorithm: STA Longest_Path
Input:(Graph(V, E), L, I, spec)
n = |V|; m = |E|; q = |I|;
for(v in V) {
 dist[v] := 0 ;
 D_v= | →v| ;
}
Q = I;
while(Q != 0) {
 v = DEQUEUE(Q);
 foreach(a in v→){
   dist[a] = max(dist[a], (dist[v] + L(v, a)));
   D_a= D_a−1;
   if(D_a== 0)
     QUEUE(Q, a);
 }
maxdist = max_v in _v(dist[v]);
maxv= SELECT1(V, maxdist);
critical_path= BACK_TRACE(V, E, L, dist[], maxv, (spec -maxdist));
return(critical_path, dist[]);
```


14

STA Longest Path Algorithm


```
Algorithm: STA Longest_Path
Input:(Graph(V, E), L, I, spec)
n = |V|; m = |E|; q = |I|;
for(v in V) {
 dist[v] := 0 ;
 D_v= | →v| ;
}
Q = I;
while(Q != 0) {
 v = DEQUEUE(Q);
 foreach(a in v→){
   dist[a] = max(dist[a], (dist[v] + L(v, a)));
   D_a= D_a−1;
   if(D_a== 0)
     QUEUE(Q, a);
 }
maxdist = max_v in _v(dist[v]);
maxv= SELECT1(V, maxdist);
critical_path= BACK_TRACE(V, E, L, dist[], maxv, (spec -maxdist));
return(critical_path, dist[]);
```


Legend:

- L(v, u) is the edge length.
- dist[v] is an iteratively increasing lower bound on the longest path length from the Primary Inputs to node v.
- D_v is the number of incoming edges to node v in V.

15

STA Longest Path Algorithm

```
Algorithm: STA Longest_Path
Input:(Graph(V, E), L, I, spec)
n = |V|; m = |E|; q = |I|;
for(v in V) {
 dist[v] := 0 ;
 Dv= | →v| ;
}
Q = I;
while(Q != 0) {
 v = DEQUEUE(Q);
 foreach(a in v→){
   dist[a] = max(dist[a], (dist[v] + L(v, a)));
   Da= Da-1;
   if(Da== 0)
     QUEUE(Q, a);
 }
maxdist = maxv in v(dist[v]);
maxv= SELECT1(V, maxdist);
critical_path= BACK_TRACE(V, E, L, dist[], maxv, (spec -maxdist));
return(critical_path, dist[]);
```

Legend:

- v→ are the successors of v.
- → v are the predecessors of v.
- The length of the longest path to any node maxdist is computed and passed to select one node, whereby dist[v]=maxdist.
- spec is the Required Arrival Time (RAT).

STA Longest Path Algorithm

```
Algorithm: STA Longest_Path
Input:(Graph(V, E), L, I, spec)
n = |V|; m = |E|; q = |I|;
for(v in V) {
 dist[v] := 0 ;
 Dv= | →v| ;
}
Q = I;
while(Q != 0) {
 v = DEQUEUE(Q);
 foreach(a in v→){
   dist[a] = max(dist[a], (dist[v] + L(v, a)));
   Da= Da-1;
   if(Da== 0)
     QUEUE(Q, a);
 }
maxdist = maxv in v(dist[v]);
maxv= SELECT1(V, maxdist);
critical_path= BACK_TRACE(V, E, L, dist[], maxv, (spec -maxdist));
return(critical_path, dist[]);
```

Legend:

- (spec-maxdist) indicates path slack

Back-tracing

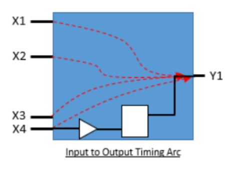
```
Algorithm: BACK_TRACE
Input: Graph(V, E), L, maxdist, maxv, Rslack)
foreach(v in V) slack[v] = maxdist;
slack[maxv] = Rslack;
critical_path= {maxv};
QUEUE(Q, maxdist);
while(Q != 0) {
 v = DEQUEUE(Q);
 foreach(a in v→){
  slack[a] = slack[v] + (dist[v] -(dist[a] + La,v));
  if (slack[a] == Rslack) {
     QUEUE(Q, a);
     critical_path= {a} U critical_path;
     break;
   }
 }
}
return (critical_path, slack[]);
```


Dijkstra’s Shortest Path Algorithm

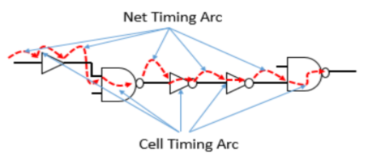
```
Algorithm: DIJKSTRA's Shortest Path
Input: (Graph(V, E), source)
for each vertex v in Graph:       // Initializations
dist[v] := infinity ;             // Unknown distance function from source to v
previous[v] := undefined ;        // Previous node in optimal path
end for                           // from source
dist[source] := 0 ;               // Distance from source to source
Q := the set of all nodes in Graph ; // All nodes in the graph are unoptimized
                                     // thus are in Q
while Q is not empty:             // the main loop
 u := vertex in Q with smallest distance in dist[] ; // Source node in first case
 remove u from Q ;
 If dist[u] = infinity:
   break;                         // all remaining vertices are
 end if                           // inaccessible from source
 for each neighbor v of u:        // where v has not yet been removed from Q.
   alt := dist[u] + dist_between(u, v) ;
   if alt < dist[v]: // Relax (u,v,a)
     dist[v] := alt ;
     previous[v] := u ;           // Store Shortest Path
     decrease-key v in Q;         // Reorder v in the Queue
   end if
 end for
end while
return dist;
```

Timing Arcs

- **Timing Arcs** represent the timing relationship between 2 pins of an element/component, block or any type of boundaries.
  - Each timing arc has a Start-point and an Endpoint
    - The start-point can be either an input or an inout pin.
    - The endpoint is **usually** an output or an inout pin.
      - If the endpoint is an input pin we have a constrained timing arc (Setup, Hold, Recovery or Removal constraint.



- There are two types of timing arcs:
  - Net Arcs
  - Cell Arcs
    - Combinational Cells (delay arc)
    - Sequential Cells (constraint arc)



Unateness

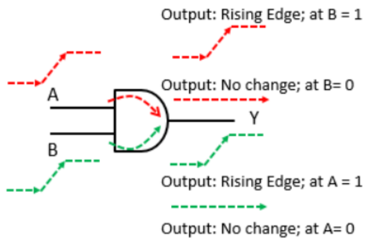
- Timing arcs have a property called **unateness**.
  - Unateness (or "timing sense") of an arc if defined as the sense of traversal from the input to the output pin of the arc.
- Unate timing arc types:
  - Positive Unate Timing Arcs
    - The rise transition at the input pin causes rise transition (if at all) at the output pin and vice-versa. (e.g: AND, OR, Buffer)
  - Negative Unate Timing Arcs
    - The rise transition at the input pin causes fall transition (if at all) at the output pin and vice-versa. (e.g: NAND, NOR, Inverter)
  - Non Unate Timing Arcs
    - If the arc is neither positive nor negative unate then it is said to be non-unate (e.g: XOR, XNOR)

Unateness

- **Examples**
  - AND Gate:

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

AND Gate Truth Table



Having A = 0 and B (0→1): Y does not change (constant at 0)
Having A = 1 and B (0 →1): Y changes from 0 to 1
Having B = 0 and A (0→1): Y does not change (constant at 0)
Having B = 1 and A (0 →1): Y changes from 0 to 1

AND is Positive Unate

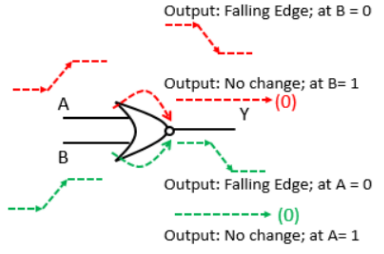
22

Unateness

- **Examples**
  - NOR Gate:

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

NOR Gate Truth Table



Having A = 0 and B (1→0): Y changes from 0 to 1
Having A = 1 and B (1 →0): Y does not change (constant at 0)
Having B = 0 and A (1→0): Y changes from 0 to 1
Having B = 1 and A (1 →0): Y does not change (constant at 0)

NOR is Negative Unate

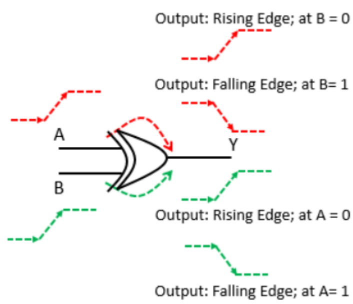
23

Unateness

- **Examples**
  - XOR Gate:

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

XOR Gate Truth Table

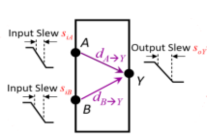


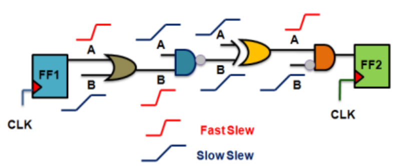
Having A = 0 and B (1→0): Y changes from 1 to 0
Having A = 1 and B (1 →0): Y changes from 0 to 1
Having B = 0 and A (1→0): Y changes from 1 to 0
Having B = 1 and A (1 →0): Y changes from 0 to 1

XOR is Non-Unate

Transitions

- Signal transitions are characterised by their input and output slew.
  - **Slew** (aka transition time) is defined as the amount of time required for a signal to transition from high-to-low and low-to-high.

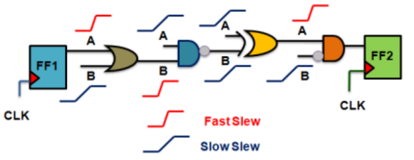




- Valid Transitions:
  - For each timing arc, delay and output slew values **will propagate only for <u>transitions that exist</u>.**
  - For instance we cannot have a Fall→Rise transition (low-to-high) at the input of the inverter and get a Fall→Rise signal transition at the output.

Graph-Based vs Path-Based STA

- Graph-Based Analysis
  - Graph based analysis computes worst case delays considering worst slew as the standard case for **all** inputs of a gate.
  - E.g. The worst case delay up to the first OR gate, after FF1, would be including the signal slew at B and not at A.
- Path-Based Analysis
  - Path based analysis takes into account the actual slew of **each** timing arc.
  - E.g. The worst case delay up to the first OR gate, after FF1, would be including the signal slew at A as it belongs to the path we are interested in.



Graph-Based vs Path-Based STA

Graph-Based

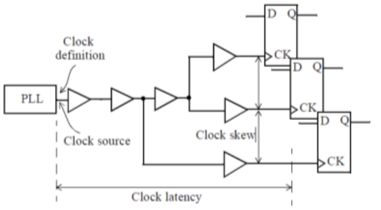
- Pros:
  - Reasonable algorithms' complexity.
  - No need to be extra accurate for most of the design's cells.
- Cons:
  - Extra pessimism inflicted, which can lead to a slower design due to over-pessimistic calculation of the delays on the top k critical paths.
  - Not every single path is sensitisable.

Path-Based

- Pros:
  - Accurate results.
  - No pessimism inflicted.
- Cons:
  - Run-time cost makes this approach non-feasible. The number of paths is usually exponential to the number of nodes.

Clock Skew

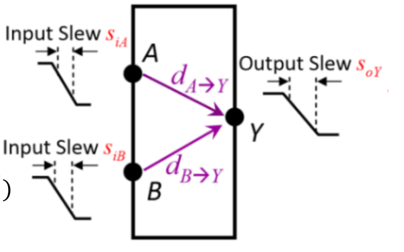
- **Skew** is the difference in timing between two or more signals.
- **Clock skew** is the difference in arrival times at the end points of the clock tree.
- **Clock latency** is the total time it takes from the clock source to an end point.



Actual Arrival Time

- Starting from the Primary Inputs, arrival times are computed by adding delays across a path and performing the minimum or maximum of such accumulated times at a convergence point, in early mode (hold analysis) and late mode (setup analysis) respectively.
- For example, let at_A^{early} , at_B^{early} to be the early arrival times at pins A and B. Then the **early mode arrival time** at the output pin Y will be :

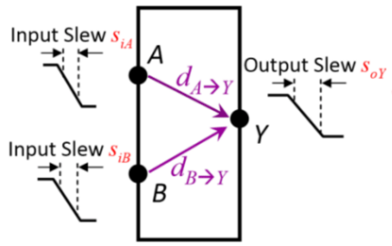
$$at_Y^{early} = \min(at_A^{early} + d_{A\to Y}^{early}, at_B^{early} + d_{B\to Y}^{early})$$



Actual Arrival Time

- Following the same example, let at_A^{late} , at_B^{late} to be the late arrival times at pins A and B. Then the **late mode arrival time** at the output pin Y will be :

$$at_Y^{late} = \max(at_A^{late} + d_{A\to Y}^{late}, at_B^{late} + d_{B\to Y}^{late})$$



Required Arrival Time

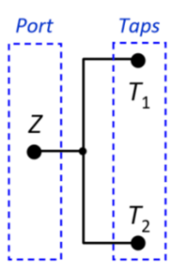
- Starting from the Primary Outputs, required arrival times are computed by subtracting the delays across a path and performing the maximum or minimum of such accumulated times at a convergence point, in early mode (hold analysis) and late mode (setup analysis) respectively.

- For example, the **early mode required arrival time** at the input pin Z will be :

$$rat_Z^{early} = \max(rat_{T1}^{early} - d_{Z\to T1}^{early}, rat_{T2}^{early} - d_{Z\to T2}^{early})$$

- For the same example, the **late mode required arrival time** at the input pin Z will be :

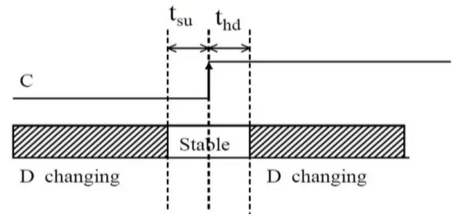
$$rat_Z^{late} = \min(rat_{T1}^{late} - d_{Z\to T1}^{late}, rat_{T2}^{late} - d_{Z\to T2}^{late})$$



Setup & Hold Constraint

- Proper operation of a flip-flop requires the logic value of the input data pin to be stable for a specific period of time **before** the capturing clock edge. This period of time is designated by the **setup time** (t_{setup}) .

$t_{setup} = RAT - AAT$



Whereas, $t_{su} = t_{setup}$ and $t_{hd} = t_{hold}$

- Additionally, the logic value of the input data pin must also be stable for a specific period of time **after** the capturing clock edge. This period of time is designated by the **hold time** (t_{hold}) .

$t_{hold} = AAT - RAT$

Setup & Hold Constraint

- A common misconception is that by increasing the clock period, a designer can face any violation and solve it. This works perfectly and can erase any setup violation, but that's not the case for hold violations.

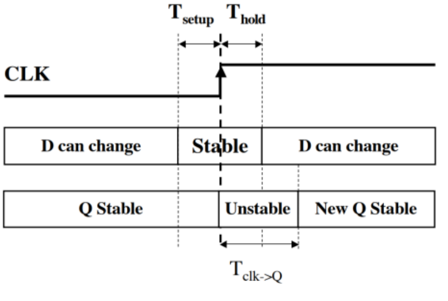
- Hold violations occur when the delay between a launching and a capturing flip-flop is so small that the launching flip-flop's hold time is smaller or equal to the sum of the delay between the flip-flop's plus the minimum Clock to Q time of the launching flip-flop.

Setup & Hold Constraint

- Hold violation may cause data hazard, whereas setup violation may cause a race condition to rise.

$T_{clk\rightarrow Q}$

- $T_{clk\rightarrow Q}$: the amount of time you have to wait after the clock (CLK) before the flip-flop's output (Q) is valid.
- If you try to use the output before this you will get inconsistent results depending on if Q changes.



Slack

- For proper circuit operation, the following conditions must hold:
  - $at_{early} \geq rat_{early}$
  - $at_{late} \leq rat_{late}$

- To quantify how well timing constraints are met at each circuit node, slacks can be computed based on the above conditions. That is, slacks are positive when the required times are met and negative otherwise.
  - $slack_{early} = at_{early} - rat_{early}$
  - $slack_{late} = rat_{late} - at_{late}$

Constrained and related pins

- The related pin is commonly the clock input pin.

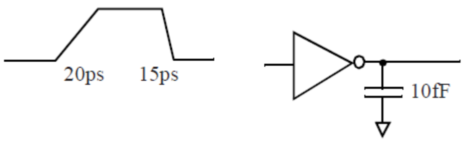
- The constrained pin is commonly the data input pin.
  - We can identify the clock pin by the sequential group clocked_on attribute.

- In a more formal sense, the related pin's edge is used as a reference point to measure the time that the constrained pin's value must be stable, in order to prevent violations.

.Lib Lookup Tables

- An inverter cell with an NLDM model has the following lookup tables:
  - Rise delay
  - Fall delay
  - Rise transition
  - Fall transition

- For example, the **rise delay** is obtained from the *cell_rise* table for 15ps input transition time (falling) and 10fF load and the **fall delay** is obtained from the *cell_fall* table for 20ps input transition time (rising) and 10fF load.



LUT Interpolation

- In most cases the input transition time (index_1) and output capacitance (index_2) does not correspond to the values given in the lookup table.
- Given the following Lookup Table:

```
fall_transition(delay_template_3x3) {
 index_1 ("0.1, 0.3 . . .");
 index_2 (". . . 0.35, 1.43");
 values ( \
   ". . . 0.1937, 0.7280", \
   ". . . 0.2327, 0.7676"
   . . .
```

- For example, we want to find the value for the input transition time of 0.15ns and output capacitance of 1.16pF.

LUT Interpolation

- The values of *index_1* and *index_2* are denoted as x_i, y_i and the corresponding table values are denoted as T_{ij} .
- If the table lookup is required for (x_0 , y_0), the value T_{00} is given by the following:

$$T_{00} = x_{20} * y_{20} * T_{11} + x_{20} * y_{01} * T_{12} + x_{01} * y_{20} * T_{21} + x_{01} * y_{01} * T_{22}$$

- where:

$$x_{01} = \frac{(x_0 - x_1)}{(x_2 - x_1)}$$

$$x_{20} = \frac{(x_2 - x_0)}{(x_2 - x_1)}$$

$$y_{01} = \frac{(y_0 - y_1)}{(y_2 - y_1)}$$

$$y_{20} = \frac{(y_2 - y_0)}{(y_2 - y_1)}$$

Propagate Delay Longest BFS Pseudocode

```
// STEP 1: Initialise Structures

initialise levels_gatepin_longest structure
initialise delay_gatepin_longest structure

// empty arrays
current_output_pins = {}
successor_output_pins = {}

// STEP 2: Local all Path Startpoints

for each PI in Circuit {
  insert_PIN_to_successor_output_pins(PI)
}

for each component_gatepin in Circuit {
  if (component_gatepin == FF output pin)
    insert_PIN_to_successor_output_pins(component_gatepin)
}

// replace
current_output_pins = successor_output_pins
successor_output_pins = {}


// STEP 3: Visit available nodes
while current_output_pins not empty {
  …
}
```

- Step 1 performs the initialization of the structures.
  - Levelisation and delay information is contained in separate structures (levels_gatepin_longest, delay_gatepin_longest)
  - Each entry of the data structures is referring to a specific gatepin of component.
  - All longest path levels of each pin are initialised to 0.
  - All delaygatepin's fields are initialized to zero except RAT which is set to Infinity

Propagate Delay Longest BFS Pseudocode

```
// STEP 1: Initialise Structures

initialise levels_gatepin_longest structure
initialise delay_gatepin_longest structure

// empty arrays
current_output_pins = {}
successor_output_pins = {}

// STEP 2: Local all Path Startpoints

for each PI in Circuit {
  insert_PIN_to_successor_output_pins(PI)
}

for each component_gatepin in Circuit {
  if (component_gatepin == FF output pin)
    insert_PIN_to_successor_output_pins(component_gatepin)
}

// replace
current_output_pins = successor_output_pins
successor_output_pins = {}


// STEP 3: Visit available nodes
while current_output_pins not empty {
  …
}
```

- Step 1 performs the initialization of the structures.
  - Levelisation and delay information is contained in separate structures (levels_gatepin_longest, delay_gatepin_longest)
  - Each entry of the data structures is referring to a specific gatepin of component.
- Step 2 inserts all of the Path Startpoints (PIs and FF output pins) into the arrays.
  - The algorithm propagates the delay by using a BFS variant which, in order to annotate the delay on a component (output pin), requires all of its predecessors visited.
    - In that way the nodes are visited in topological sort order.

Propagate Delay Longest BFS Pseudocode

```
// STEP 3: Visit available nodes/gatepins
while current_output_pins not empty {
  // STEP 3.1: Visit next gatepin (BFS)
  curr_pin_index = current_output_pins.pop()
  if (curr_pin_index == NOT_FOUND) {
    current_output_pins = successor_output_pins
    successor_output_pins = {}

    curr_pin_index = current_output_pins.pop()
    if (curr_pin_index == NOT_FOUND) {
      break;
    }
  }
  // get the current output gatepin's level
  Current_level = get_level(curr_pin_index)

  // get the output gatepin's successor components
  Successor_list = get_successors(curr_pin_index)

  // get the total output net capacitance
  total_capacitance = get_total_output_cap(successor_list)

  // calculate/propagate delay to the current output pin
  current_pin = &delay_gatepins_longest[curr_pin_index]
  calculate_delay_output_pin(current_pin)

  // STEP 4: Act on each successor and update BFS arrays
  if (successor_list not empty) {
    …
  }
```

- Step 3.1 “visits” the next output gatepin
  - If there are no output gatepins left to visit we proceed with the next level of gatepins which are the valid successors of the current level.
    - If there are no successors this means that we have discovered all of the nodes of the circuit and we must terminate the execution

Propagate Delay Longest BFS Pseudocode

```
// STEP 3: Visit available nodes/gatepins
while current_output_pins not empty {

  // STEP 3.1: Visit next gatepin (BFS)
  curr_pin_index = current_output_pins.pop()
  if (curr_pin_index == NOT_FOUND) {
    current_output_pins = successor_output_pins
    successor_output_pins = {}

    curr_pin_index = current_output_pins.pop()
    if (curr_pin_index == NOT_FOUND) {
      break;
    }
  }

  // get the current output gatepin's level
  Current_level = get_level(curr_pin_index)

  // get the output gatepin's successor components
  Successor_list = get_successors(curr_pin_index)

  // get the total output net capacitance
  total_capacitance = get_total_output_cap(successor_list)

  // calculate/propagate delay to the current output pin
  current_pin = &delay_gatepins_longest[curr_pin_index]
  calculate_delay_output_pin(current_pin)

  // STEP 4: Act on each successor and update BFS arrays
  if (successor_list not empty) {
    …
  }
```

- Step 3.1 "visits" the next output gatepin
  - If there are no output gatepins left to visit we proceed with the next level of gatepins which are the valid successors of the current level.
    - If there are no successors this means that we have discovered all of the nodes of the circuit and we must terminate the execution
- After acquiring the output gatepin:
  - We retrieve its level which will be used later in successor levelisation.
  - We retrieve the gatepin's successors.
  - We calculate the output net capacitance which is used by *calculate_delay_output_pin* for the delay annotation.
  - We calculate/annotate the delay of the current output pin (current_pin)

Propagate Delay Longest BFS Pseudocode

```
// STEP 4: Act on successors
 if (successor_list not empty) {

   // delay_gatepins_longest[i].visited = 1
   set all successor' input pins visited flag to 0;

   fanoutnum = 0

   for each successor in successor_list {

     succ_pin_index = get_successor_pin_index(successor)

     // bypass successor if its pin is visited
     successor_pin = delay_gatepin_longest[succ_pin_index]
     if successor_pin.visited == 1 {
       continue
     }

     successor_pin.visited = 1

     // STEP 4.1 update levelisation information
     newlevel = current_level + 1
     if (newlevel > maxlevel) {
       maxlevel = newlevel
       maxlevel_point = successor
     }
     lsucc_pin = &level_gatepin_longest[succ_pin_index]
     lsucc_pin->level = newlevel
     lsucc_pin->previous = curr_gatepin_index
     …
```

- In step 4 iterates though the successor components of the previously visited output gatepin. (curr_pin_index)
  - Initially, we set all of the successor gatepins' visited flag to zero
  - For each visited successor we will be incrementing *fanoutnum.*
- **NOTE**: Each successor might be multiple times present in the successor_list, one time **for each** of its input pins that is connected to the current_gatepin.

Propagate Delay Longest BFS Pseudocode

```
// STEP 4: Act on successors
if (successor_list not empty) {

  // delay_gatepins_longest[i].visited = 1
  set all successor' input pins visited flag to 0;

  fanoutnum = 0

  for each successor in successor_list {

    succ_pin_index = get_successor_pin_index(successor)

    // bypass successor if its pin is visited
    successor_pin = delay_gatepin_longest[succ_pin_index]
    if successor_pin.visited == 1 {
      continue
    }

    successor_pin.visited = 1

    // STEP 4.1 update levelisation information
    newlevel = current_level + 1
    if (newlevel > maxlevel) {
      maxlevel = newlevel
      maxlevel_point = successor
    }
    lsucc_pin = &level_gatepin_longest[succ_pin_index]
    lsucc_pin->level = newlevel
    lsucc_pin->previous = curr_gatepin_index
    …
```

- In step 4 iterates though the successor components of the previously visited output gatepin. (curr_pin_index)
  - Initially, we set all of the successor gatepins' visited flag to zero
  - For each visited successor we will be incrementing *fanoutnum.*
- In step 4.1 the levelisation is performed
  - Each successor input pin gets the level of the previous output pin incremented by 1.

Propagate Delay Longest BFS Pseudocode

```
    // acting on each successor (cont.)
    fanoutnum++;

     // STEP 4.2 Successor input pin delay annotation
    succ_pin = &delay_gatepins_longest[succ_pin_index];
    if successor component is sequential {
      calculate_delay_input_pin(succ_pin, SEQUENTIAL)
      continue
    }
    else {
       calculate_delay_input_pin(succ_pin, COMBINATIONAL)
    }

    // STEP 4.3 Count successor input pins
    total_successor_inputs = 0

    for each input_pin_index in successor.input_pins {
      input_pin = &delay_gatepins_longest[input_pin_index]

      // bypass unconnected input pins
      if (input_pin->connection == UNCONNECTED) {
        continue
      }
      total_successor_inputs++;
    }
```

- We continue in step 4.2 by annotating the delay on the successor.
  - The successor input pin simply inherits the fields (transitions, delays etc) of the output gatepin.
- If the current successor is a sequential element we do not proceed any further and continue with the BFS

Propagate Delay Longest BFS Pseudocode

```
// acting on each successor (cont.)
fanoutnum++;


 // STEP 4.2 Successor input pin delay annotation
succ_pin = &delay_gatepins_longest[succ_pin_index];
if successor component is sequential {
  calculate_delay_input_pin(succ_pin, SEQUENTIAL)
  continue
}
else {
   calculate_delay_input_pin(succ_pin, COMBINATIONAL)
}


// STEP 4.3 Count successor input pins
total_successor_inputs = 0

for each input_pin_index in successor.input_pins {
  input_pin = &delay_gatepins_longest[input_pin_index]

  // bypass unconnected input pins
  if (input_pin->connection == UNCONNECTED) {
    continue
  }
  total_successor_inputs++;
}
```

- We continue in step 4.2 by annotating the delay on the successor.
  - The successor input pin simply inherits the fields (transitions, delays etc) of the output gatepin.
- In step 4.3 we count the total and connected input pins of the successor.
  - This will be used later on, as we also count the **visited** input pins of the successor.
    - If the successor's total input pins are equal to the visited input pins, we know that the next level of BFS will contain the successor output gatepin.

Propagate Delay Longest BFS Pseudocode

```
    // Step 4.4: Insert valid output pins to the
    // successor arrays
    for each output_pin_index in successor.output_pins {

     out_pin = &delay_gatepins_longest[output_pin_index]

     if (out_pin->total_prev_pins == 0) {
       out_pin->total_prev_pins = total_successor_inputs;
     }

     // gets incremented for each successor
     out_pin->visited_prev_pins++; // initialised to 0

     total_inputs = out_pin->total_prev_pins
     visited_inputs = out_pin->visited_prev_pins

     // Not all predecessors discovered…
     if (total_inputs != total_visited_inputs) {
       continue
     }

     // Step 4.4: Update output pin levelisation
     out_pin.level = newlevel
     out_pin.previous = succ_pin_index
     insert_PIN_to_successor_output_pins(PI)

   }
 } // end of foreach successor
 current_pin->total_next_pins = fanoutnum
 current_pin->visited_next_pins = 0
}
}
```

- In step 4.4 we insert output pins in the successor pins array (which contains the next level of the BFS traversal)
  - We increment visited pins by one for each successor pin.
  - For a successor's output pin we check if the number of the total input pins are equal to the number of the visited input pins .
    - If yes, then the successor has all of its predecessors discovered and therefore will be in the next level of the BFS traversal

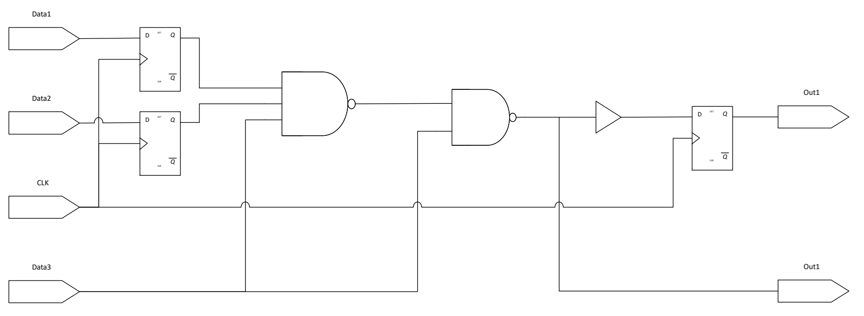
Propagate Delay Longest BFS Pseudocode

```
    // Step 4.4: Insert valid output pins to the
    // successor arrays
    for each output_pin_index in successor.output_pins {

     out_pin = &delay_gatepins_longest[output_pin_index]

     if (out_pin->total_prev_pins == 0) {
       out_pin->total_prev_pins = total_successor_inputs;
     }

     // gets incremented for each successor
     out_pin->visited_prev_pins++; // initialised to 0

     total_inputs = out_pin->total_prev_pins
     visited_inputs = out_pin->visited_prev_pins

     // Not all predecessors discovered…
     if (total_inputs != total_visited_inputs) {
       continue
     }

     // Step 4.4: Update output pin levelisation
     out_pin.level = newlevel
     out_pin.previous = succ_pin_index
     insert_PIN_to_successor_output_pins(PI)

    }
  } // end of foreach successor
  current_pin->total_next_pins = fanoutnum
  current_pin->visited_next_pins = 0
 }
}
```

- In step 4.4 we insert output pins in the successor pins array (which contains the next level of the BFS traversal)
  - We increment visited pins by one for each successor pin.
  - For a successor's output pin we check if the number of the total input pins are equal to the number of the visited input pins .
    - If yes, then the successor has all of its predecessors discovered and therefore will be in the next level of the BFS traversal
- The levelisation information regarding an output pin is updated if and only if all of the previous input pins are visited.
  - In that way, due to the topological order, the longest path levelisation will be correct.

Create Clock SDC Example (w/o Buffering)

```
create_clock CLK -name clock1 -period 2
Search moduleports to find clock source gatepin.
For each clock source gatepin's successor component pin:
    Mark gatepin's isconstrained field with -1 (indicates clock pin) and point their
    previous indices to clock source. Also store clock source's timing info (r/f
    transition and delay
    Loop through FF pins and mark their iscontrained field with 1 (indicates data pin).
    Also when on output point the previous gatepin indices to clock gatepin
```

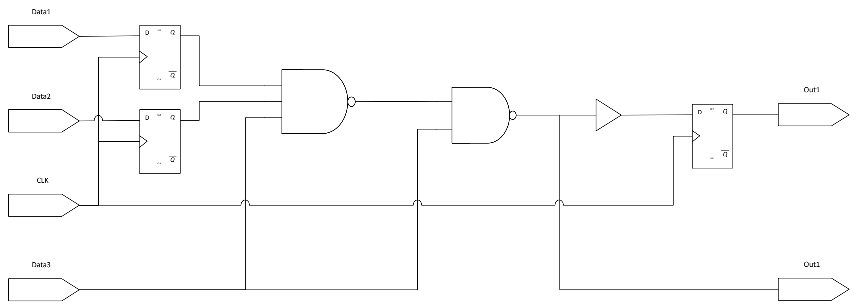


Create Clock SDC Example (w/o Buffering)

```
create_clock CLK -name clock1 -period 2
```

- Search moduleports to find clock source gatepin.

```
For each clock source gatepin's successor component pin:
      Mark gatepin's isconstrained field with -1 (indicates clock pin) and point their
      previous indices to clock source. Also store clock source's timing info (r/f
      transition and delay
      Loop through FF pins and mark their iscontrained field with 1 (indicates data pin).
      Also when on output point the previous gatepin indices to clock gatepin
```



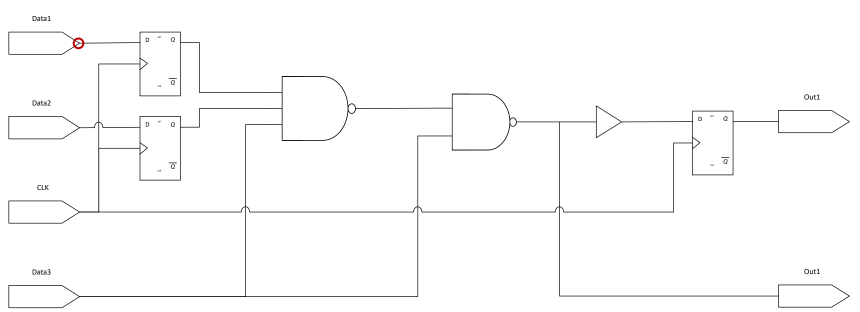
52

Create Clock SDC Example (w/o Buffering)

```
create_clock CLK -name clock1 -period 2
```

- Search moduleports to find clock source gatepin.

```
For each clock source gatepin's successor component pin:
      Mark gatepin's isconstrained field with -1 (indicates clock pin) and point their
      previous indices to clock source. Also store clock source's timing info (r/f
      transition and delay
      Loop through FF pins and mark their iscontrained field with 1 (indicates data pin).
      Also when on output point the previous gatepin indices to clock gatepin
```



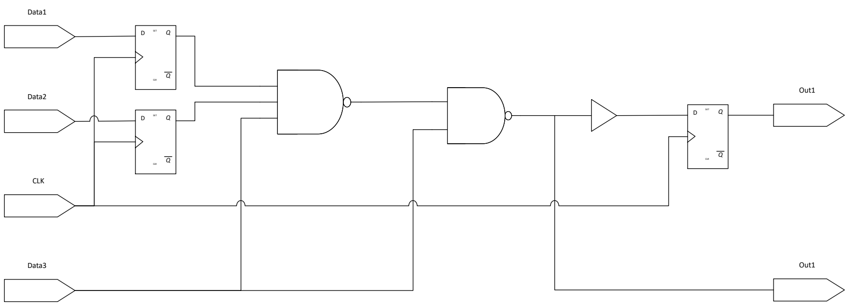
53

Create Clock SDC Example (w/o Buffering)

```
create_clock CLK -name clock1 -period 2
```

- Search moduleports to find clock source gatepin.

```
For each clock source gatepin's successor component pin:
      Mark gatepin's isconstrained field with -1 (indicates clock pin) and point their
      previous indices to clock source. Also store clock source's timing info (r/f
      transition and delay
      Loop through FF pins and mark their iscontrained field with 1 (indicates data pin).
      Also when on output point the previous gatepin indices to clock gatepin
```



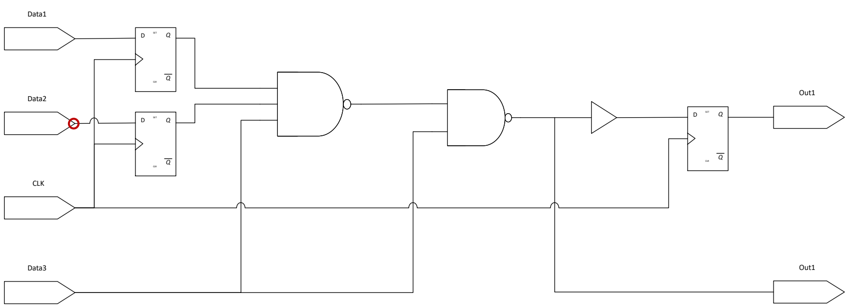
54

Create Clock SDC Example (w/o Buffering)

```
create_clock CLK -name clock1 -period 2
```

- Search moduleports to find clock source gatepin.

```
For each clock source gatepin's successor component pin:
      Mark gatepin's isconstrained field with -1 (indicates clock pin) and point their
      previous indices to clock source. Also store clock source's timing info (r/f
      transition and delay
      Loop through FF pins and mark their iscontrained field with 1 (indicates data pin).
      Also when on output point the previous gatepin indices to clock gatepin
```



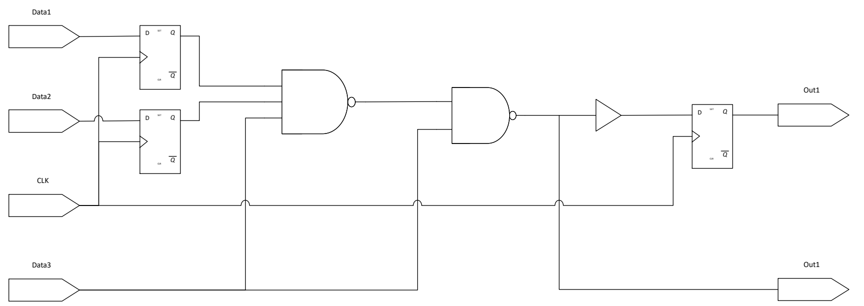
55

Create Clock SDC Example (w/o Buffering)

```
create_clock CLK -name clock1 -period 2
```

▶ Search moduleports to find clock source gatepin.

```
For each clock source gatepin's successor component pin:
      Mark gatepin's isconstrained field with -1 (indicates clock pin) and point their
      previous indices to clock source. Also store clock source's timing info (r/f
      transition and delay
      Loop through FF pins and mark their iscontrained field with 1 (indicates data pin).
      Also when on output point the previous gatepin indices to clock gatepin
```



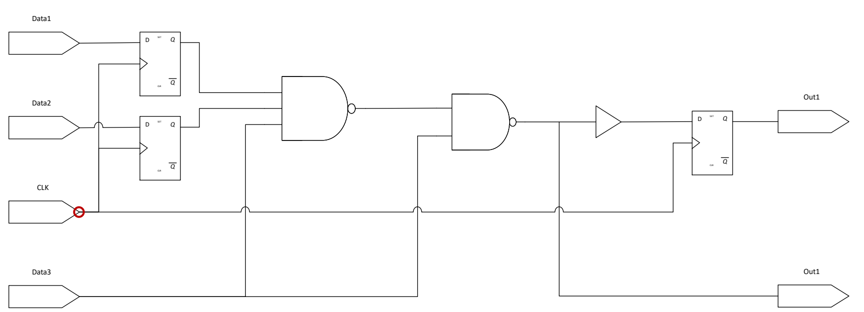
56

Create Clock SDC Example (w/o Buffering)

```
create_clock CLK -name clock1 -period 2
```

▶ Search moduleports to find clock source gatepin.

```
For each clock source gatepin's successor component pin:
      Mark gatepin's isconstrained field with -1 (indicates clock pin) and point their
      previous indices to clock source. Also store clock source's timing info (r/f
      transition and delay
      Loop through FF pins and mark their iscontrained field with 1 (indicates data pin).
      Also when on output point the previous gatepin indices to clock gatepin
```



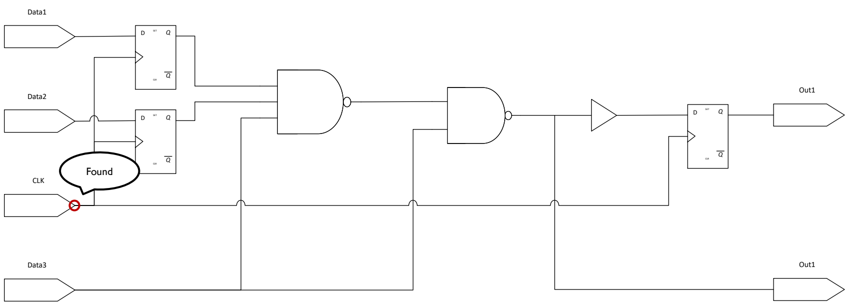
57

Create Clock SDC Example (w/o Buffering)

```
create_clock CLK -name clock1 -period 2
```

▶ Search moduleports to find clock source gatepin.

```
For each clock source gatepin's successor component pin:
     Mark gatepin's isconstrained field with -1 (indicates clock pin) and point their
     previous indices to clock source. Also store clock source's timing info (r/f
     transition and delay
     Loop through FF pins and mark their iscontrained field with 1 (indicates data pin).
     Also when on output point the previous gatepin indices to clock gatepin
```



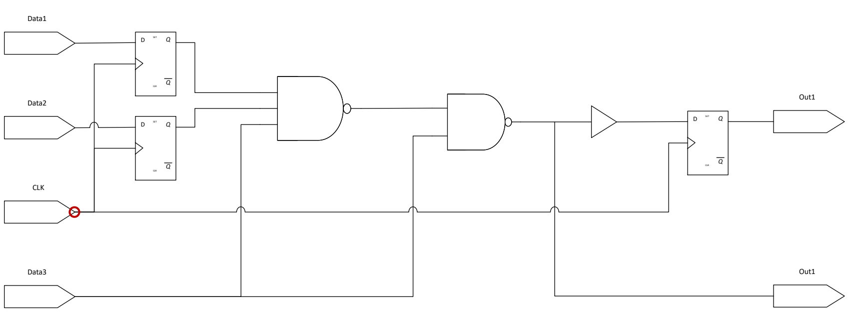
58

Create Clock SDC Example (w/o Buffering)

```
create_clock CLK -name clock1 -period 2
```

▶ Search moduleports to find clock source gatepin.

```
For each clock source gatepin's successor component pin:
     Mark gatepin's isconstrained field with -1 (indicates clock pin) and point their
     previous indices to clock source. Also store clock source's timing info (r/f
     transition and delay
     Loop through FF pins and mark their iscontrained field with 1 (indicates data pin).
     Also when on output point the previous gatepin indices to clock gatepin
```



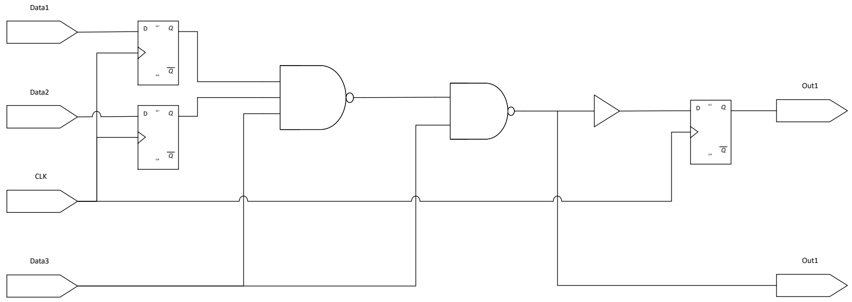
59

Create Clock SDC Example (w/o Buffering)

```
create_clock CLK -name clock1 -period 2
```

- Search moduleports to find clock source gatepin.

```
For each clock source gatepin's successor component pin:
      Mark gatepin's isconstrained field with -1 (indicates clock pin) and point their
      previous indices to clock source. Also store clock source's timing info (r/f
      transition and delay
      Loop through FF pins and mark their iscontrained field with 1 (indicates data pin).
      Also when on output point the previous gatepin indices to clock gatepin
```



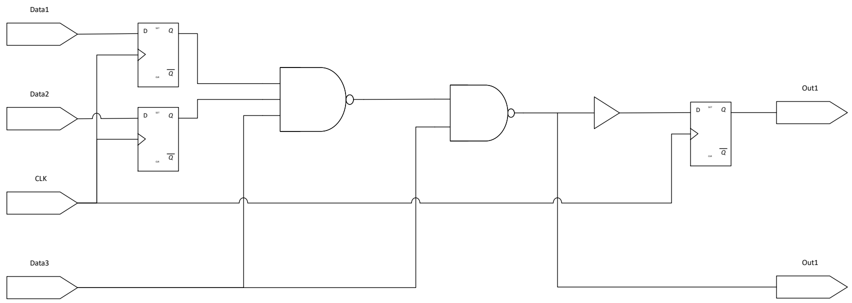
60

Create Clock SDC Example (w/o Buffering)

```
create_clock CLK -name clock1 -period 2
```

Search moduleports to find clock source gatepin.

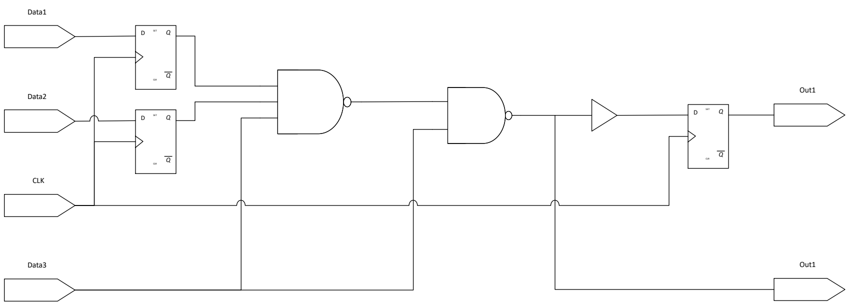
```
For each clock source gatepin's successor component pin:
      Mark gatepin's isconstrained field with -1 (indicates clock pin) and point their
      previous indices to clock source. Also store clock source's timing info (r/f
      transition and delay
      Loop through FF pins and mark their iscontrained field with 1 (indicates data pin).
      Also when on output point the previous gatepin indices to clock gatepin
```



61

Create Clock SDC Example (w/o Buffering)

```
create_clock CLK -name clock1 -period 2
Search moduleports to find clock source gatepin.
For each clock source gatepin's successor component pin:
      Mark gatepin's isconstrained field with -1 (indicates clock pin) and point their
      previous indices to clock source. Also store clock source's timing info (r/f
      transition and delay
      Loop through FF pins and mark their iscontrained field with 1 (indicates data pin).
      Also when on output point the previous gatepin indices to clock gatepin
```

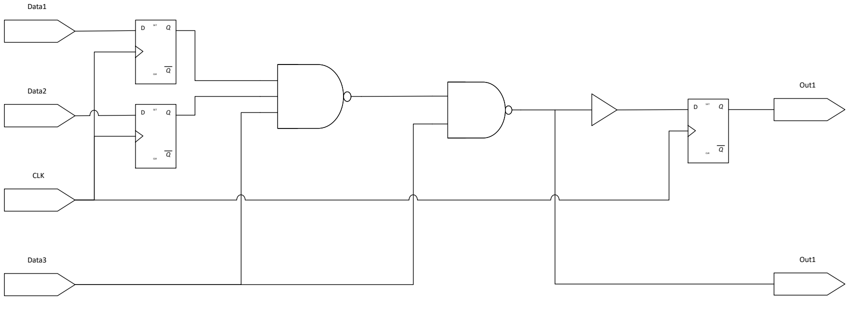


62 ASP Timer 31/3/2025

62

Create Clock SDC Example (w/o Buffering)

```
create_clock CLK -name clock1 -period 2
Search moduleports to find clock source gatepin.
For each clock source gatepin's successor component pin:
      Mark gatepin's isconstrained field with -1 (indicates clock pin) and point their
      previous indices to clock source. Also store clock source's timing info (r/f
      transition and delay
      Loop through FF pins and mark their iscontrained field with 1 (indicates data pin).
      Also when on output point the previous gatepin indices to clock gatepin
```

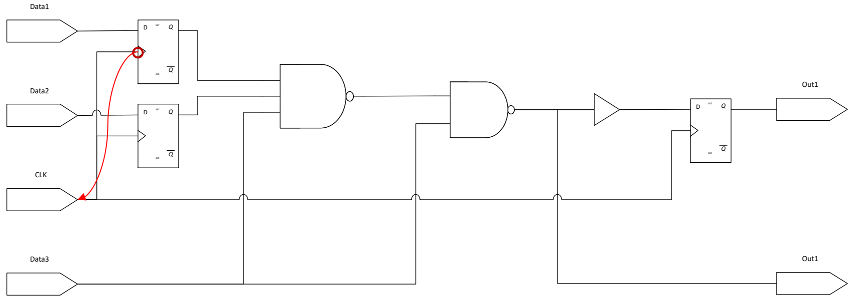
63 ASP Timer 31/3/2025

63

Create Clock SDC Example (w/o Buffering)

```
create_clock CLK -name clock1 -period 2
```

```
Search moduleports to find clock source gatepin.
For each clock source gatepin's successor component pin:
      Mark gatepin's isconstrained field with -1 (indicates clock pin) and point their
      previous indices to clock source. Also store clock source's timing info (r/f
      transition and delay
      Loop through FF pins and mark their iscontrained field with 1 (indicates data pin).
      Also when on output point the previous gatepin indices to clock gatepin
```

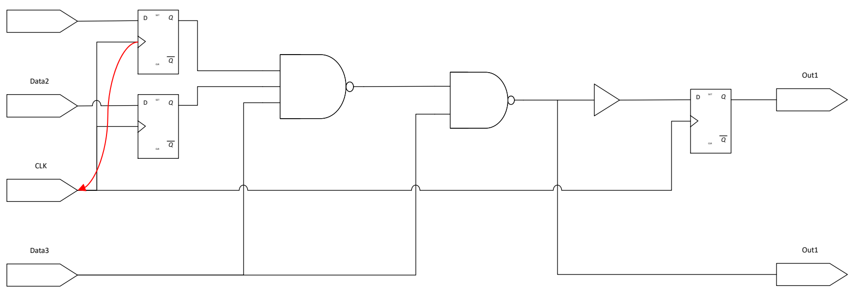


64

Create Clock SDC Example (w/o Buffering)

```
create_clock CLK -name clock1 -period 2
```

```
Search moduleports to find clock source gatepin.
For each clock source gatepin's successor component pin:
      Mark gatepin's isconstrained field with -1 (indicates clock pin) and point their
      previous indices to clock source. Also store clock source's timing info (r/f
      transition and delay
      Loop through FF pins and mark their iscontrained field with 1 (indicates data pin).
      Also when on output point the previous gatepin indices to clock gatepin
```

65

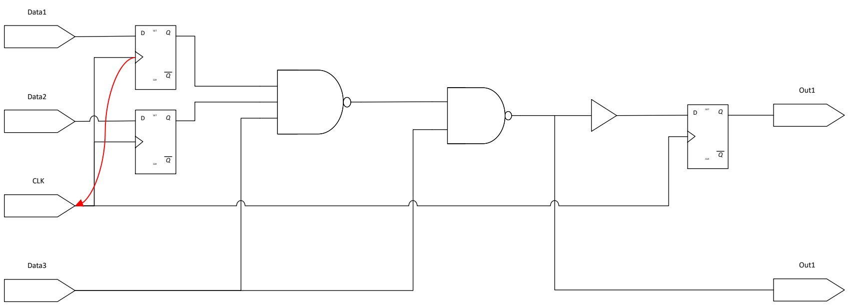
Create Clock SDC Example (w/o Buffering)

```
create_clock CLK -name clock1 -period 2
```

Search moduleports to find clock source gatepin.

For each clock source gatepin's successor component pin:

- Mark gatepin's isconstrained field with -1 (indicates clock pin) and point their previous indices to clock source. Also store clock source's timing info (r/f transition and delay
- Loop through FF pins and mark their iscontrained field with 1 (indicates data pin). Also when on output point the previous gatepin indices to clock gatepin



66

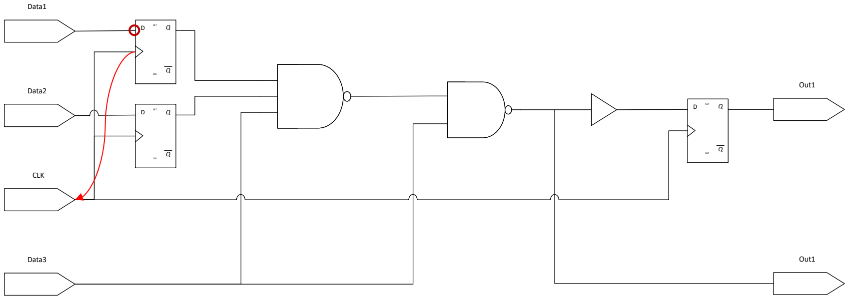
Create Clock SDC Example (w/o Buffering)

```
create_clock CLK -name clock1 -period 2
```

Search moduleports to find clock source gatepin.

For each clock source gatepin's successor component pin:

- Mark gatepin's isconstrained field with -1 (indicates clock pin) and point their previous indices to clock source. Also store clock source's timing info (r/f transition and delay
- Loop through FF pins and mark their iscontrained field with 1 (indicates data pin). Also when on output point the previous gatepin indices to clock gatepin

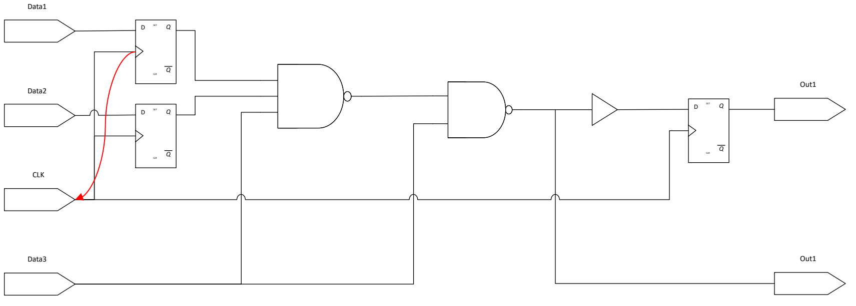


67

Create Clock SDC Example (w/o Buffering)

```
create_clock CLK -name clock1 -period 2
```

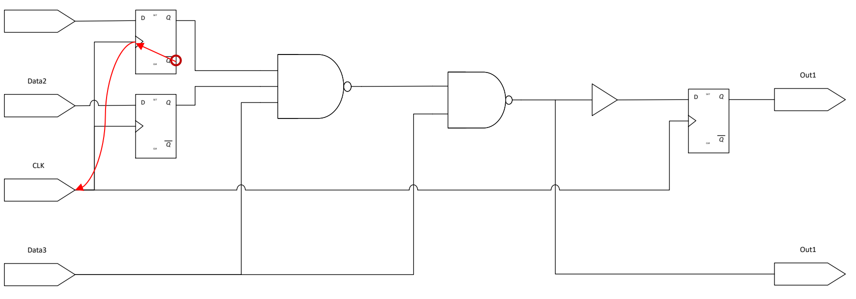
Search moduleports to find clock source gatepin.
For each clock source gatepin's successor component pin:
- Mark gatepin's isconstrained field with -1 (indicates clock pin) and point their previous indices to clock source. Also store clock source's timing info (r/f transition and delay
- Loop through FF pins and mark their iscontrained field with 1 (indicates data pin). Also when on output point the previous gatepin indices to clock gatepin



Create Clock SDC Example (w/o Buffering)

```
create_clock CLK -name clock1 -period 2
```

Search moduleports to find clock source gatepin.
For each clock source gatepin's successor component pin:
- Mark gatepin's isconstrained field with -1 (indicates clock pin) and point their previous indices to clock source. Also store clock source's timing info (r/f transition and delay
- Loop through FF pins and mark their iscontrained field with 1 (indicates data pin). Also when on output point the previous gatepin indices to clock gatepin

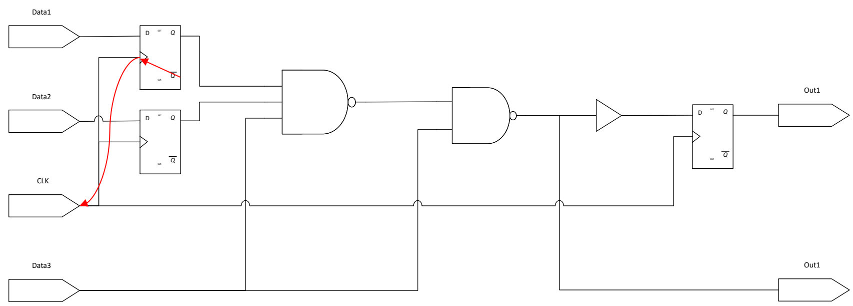


Create Clock SDC Example (w/o Buffering)

create_clock CLK -name clock1 -period 2

Search moduleports to find clock source gatepin.

For each clock source gatepin's successor component pin:
- Mark gatepin's isconstrained field with -1 (indicates clock pin) and point their previous indices to clock source. Also store clock source's timing info (r/f transition and delay
- Loop through FF pins and mark their iscontrained field with 1 (indicates data pin). Also when on output point the previous gatepin indices to clock gatepin



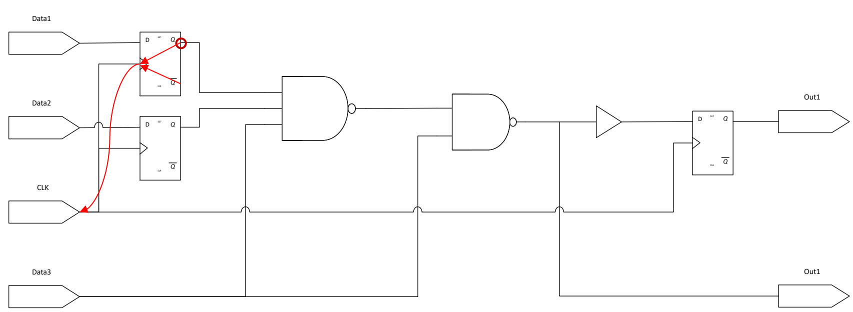
70

Create Clock SDC Example (w/o Buffering)

create_clock CLK -name clock1 -period 2

Search moduleports to find clock source gatepin.

For each clock source gatepin's successor component pin:
- Mark gatepin's isconstrained field with -1 (indicates clock pin) and point their previous indices to clock source. Also store clock source's timing info (r/f transition and delay
- Loop through FF pins and mark their iscontrained field with 1 (indicates data pin). Also when on output point the previous gatepin indices to clock gatepin



71

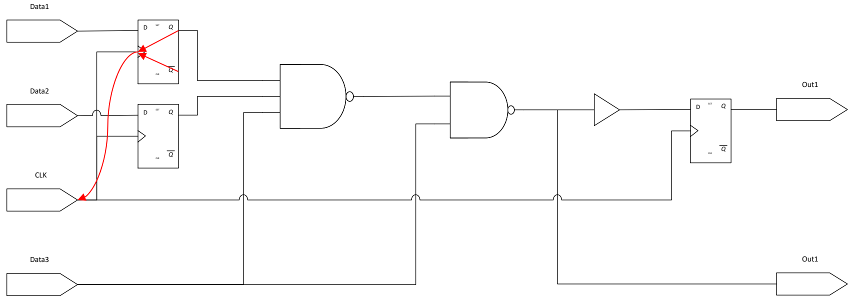
Create Clock SDC Example (w/o Buffering)

```
create_clock CLK -name clock1 -period 2
```

Search moduleports to find clock source gatepin.

For each clock source gatepin's successor component pin:

- Mark gatepin's isconstrained field with -1 (indicates clock pin) and point their previous indices to clock source. Also store clock source's timing info (r/f transition and delay
- Loop through FF pins and mark their iscontrained field with 1 (indicates data pin). Also when on output point the previous gatepin indices to clock gatepin



72

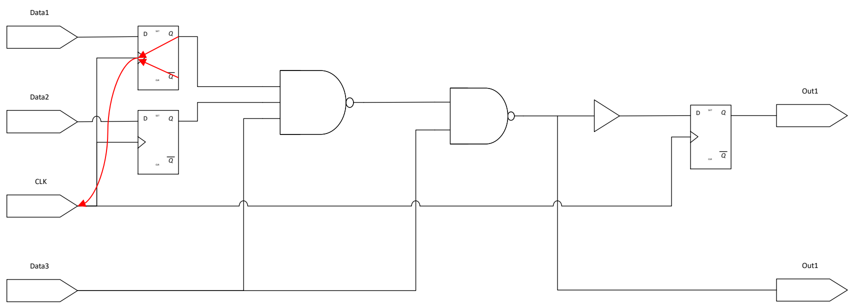
Create Clock SDC Example (w/o Buffering)

```
create_clock CLK -name clock1 -period 2
```

Search moduleports to find clock source gatepin.

For each clock source gatepin's successor component pin:

- Mark gatepin's isconstrained field with -1 (indicates clock pin) and point their previous indices to clock source. Also store clock source's timing info (r/f transition and delay
- Loop through FF pins and mark their iscontrained field with 1 (indicates data pin). Also when on output point the previous gatepin indices to clock gatepin



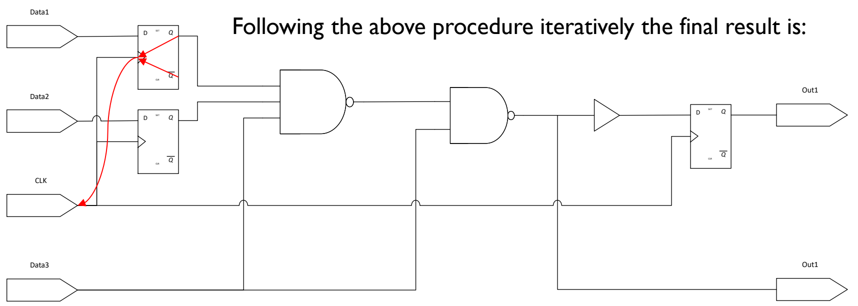
73

Create Clock SDC Example (w/o Buffering)

`create_clock CLK -name clock1 -period 2`

```
Search moduleports to find clock source gatepin.
For each clock source gatepin's successor component pin:
     Mark gatepin's isconstrained field with -1 (indicates clock pin) and point their
     previous indices to clock source. Also store clock source's timing info (r/f
     transition and delay
     Loop through FF pins and mark their iscontrained field with 1 (indicates data pin).
     Also when on output point the previous gatepin indices to clock gatepin
```

Following the above procedure iteratively the final result is:

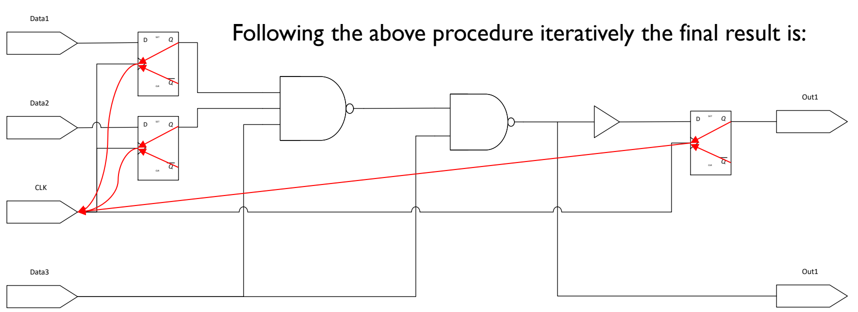


Create Clock SDC Example (w/o Buffering)

`create_clock CLK -name clock1 -period 2`

```
Search moduleports to find clock source gatepin.
For each clock source gatepin's successor component pin:
     Mark gatepin's isconstrained field with -1 (indicates clock pin) and point their
     previous indices to clock source. Also store clock source's timing info (r/f
     transition and delay
     Loop through FF pins and mark their iscontrained field with 1 (indicates data pin).
     Also when on output point the previous gatepin indices to clock gatepin
```

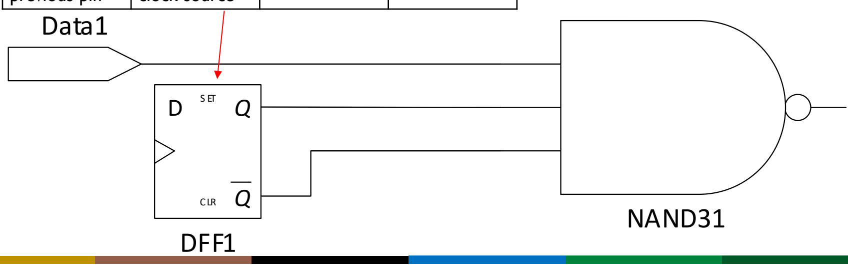
Following the above procedure iteratively the final result is:



Timing Annotation Longest Path (Sequential)

	CLK	Q	Q’
fall_delay	0		
rise_delay	0		
AAT	0		
local_cell_rise			
local_cell_fall			
fall_transition	0		
rise_transition	0		
previous pin	clock source		

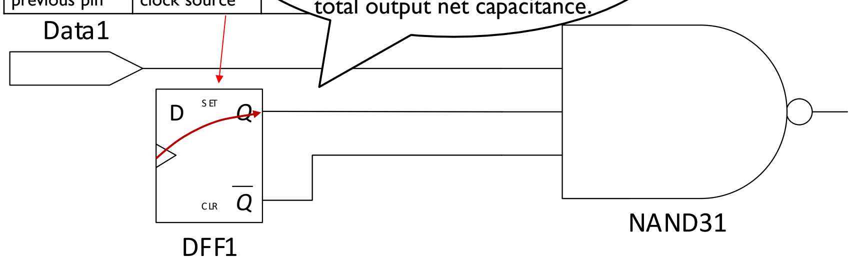
Assume Ideal Clock



Timing Annotation Longest Path (Sequential)

	CLK	Q	Q’
fall_delay	0		
rise_delay	0		
AAT	0		
local_cell_rise			
local_cell_fall			
fall_transition	0		
rise_transition	0		
previous pin	clock source		

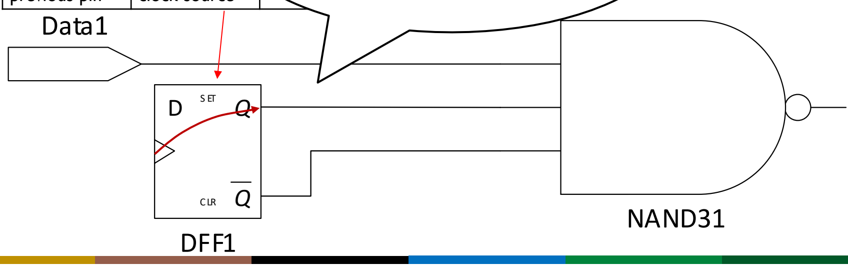
We discover the arcs one by one. The delay of an arc is calculated by interpolation or extrapolation , given the previous pin transition and the total output net capacitance.



Timing Annotation Longest Path (Sequential)

	CLK	Q	Q'
fall_delay	0		
rise_delay	0		
AAT	0		
local_cell_rise			
local_cell_fall			
fall_transition	0		
rise_transition	0		
previous pin	clock source		

In case of a FF the previous pin's transition is stored in the clock pin by the SDC or by the clock network delay propagation function.

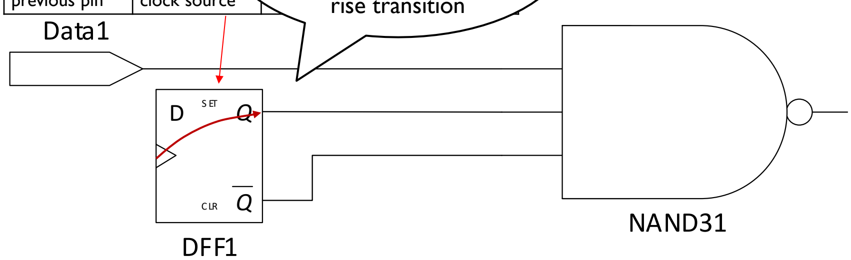


78

Timing Annotation Longest Path (Sequential)

	CLK	Q	Q'
fall_delay	0		
rise_delay	0		
AAT	0		
local_cell_rise			
local_cell_fall			
fall_transition	0		
rise_transition	0		
previous pin	clock source		

Note that we have a positive edge triggered FF so previous transition is always the rise transition

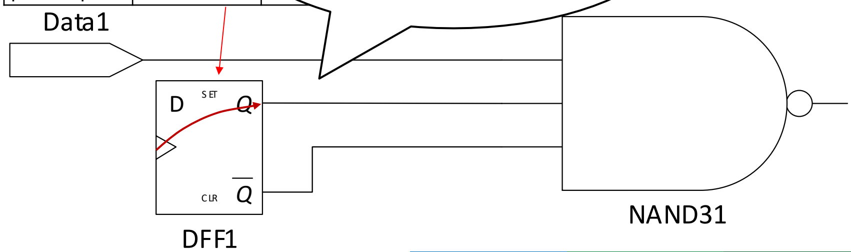


79

Timing Annotation Longest Path (Sequential)

	CLK	Q	Q'
fall_delay	0		
rise_delay	0		
AAT	0		
local_cell_rise			
local_cell_fall			
fall_transition	0		
rise_transition	0		
previous pin	clock source		

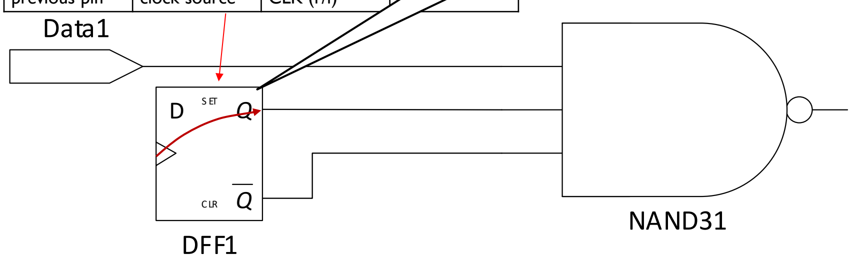
Generally, the total output net capacitance is the sum of the worst case capacitance on successor inputs plus the capacitance of the wire.



Timing Annotation Longest Path (Sequential)

	CLK	Q	Q'
fall_delay	0		
rise_delay	0	1	
AAT	0		
local_cell_rise		1	
local_cell_fall			
fall_transition	0		
rise_transition	0		
previous pin	clock source	CLK (r/f)	

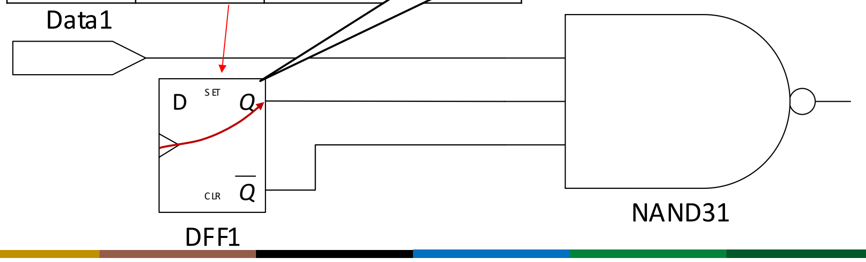
Assume that the discovered arc's (CLK→Q) cell_rise delay, obtained by the interpolation function is 1.



Timing Annotation Longest Path (Sequential)

	CLK	Q	Q'
fall_delay	0	0.5	
rise_delay	0	1	
AAT	0		
local_cell_rise		1	
local_cell_fall		0.5	
fall_transition	0		
rise_transition	0		
previous pin	clock source	CLK (r/f)	

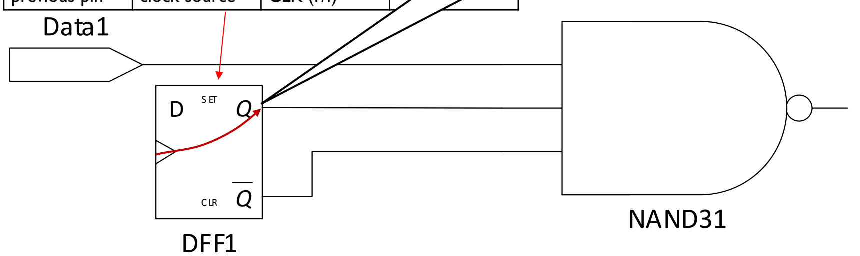
Assume that the discovered arc's (CLK→Q) cell_fall delay, obtained by the interpolation function is 0.5.



Timing Annotation Longest Path (Sequential)

	CLK	Q	Q'
fall_delay	0	0.5	
rise_delay	0	1	
AAT	0		
local_cell_rise		1	
local_cell_fall		0.5	
fall_transition	0	0.1	
rise_transition	0		
previous pin	clock source	CLK (r/f)	

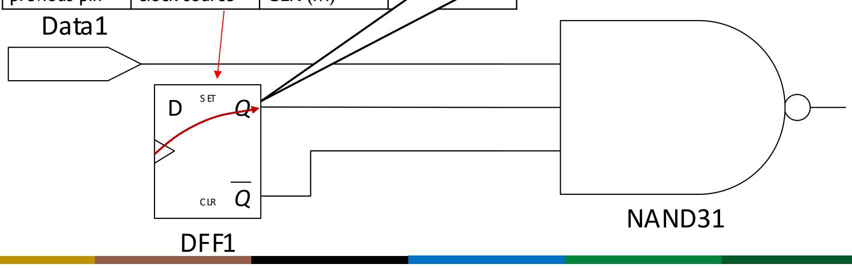
Assume that the discovered arc's (CLK→Q) fall_transition delay, obtained by the interpolation function is 0.1.



Timing Annotation Longest Path (Sequential)

	CLK	Q	Q'
fall_delay	0	0.5	
rise_delay	0	1	
AAT	0		
local_cell_rise		1	
local_cell_fall		0.5	
fall_transition	0	0.1	
rise_transition	0	0.15	
previous pin	clock source	CLK (r/f)	

Assume that the discovered arc's (CLK→Q) rise_transition delay, obtained by the interpolation function is 0.15.

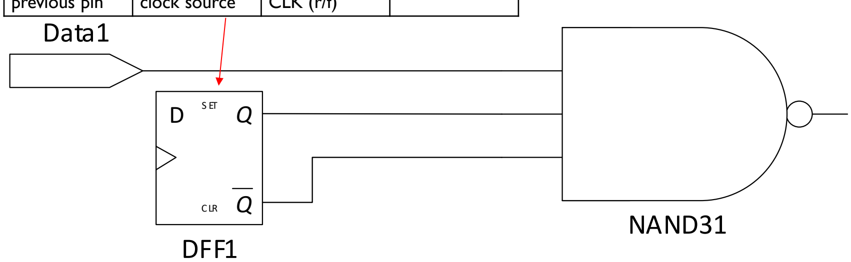


84

Timing Annotation Longest Path (Sequential)

	CLK	Q	Q'
fall_delay	0	0.5	
rise_delay	0	1	
AAT	0	1	
local_cell_rise		1	
local_cell_fall		0.5	
fall_transition	0	0.1	
rise_transition	0	0.15	
previous pin	clock source	CLK (r/f)	

After discovering every arc we know that the AAT on DFF1/Q is max(cell_rise, cell_fall)

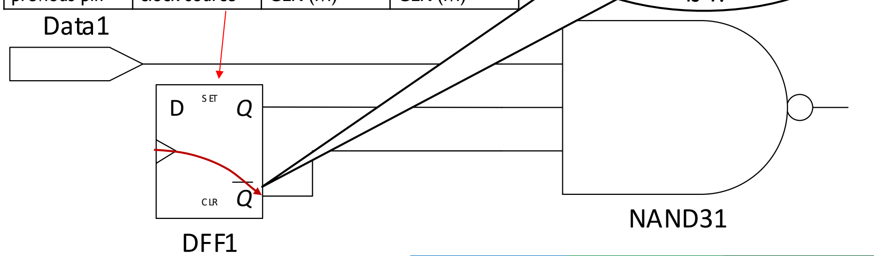


85

Timing Annotation Longest Path (Sequential)

	CLK	Q	Q'
fall_delay	0	0.5	
rise_delay	0	1	1
AAT	0	1	
local_cell_rise		1	1
local_cell_fall		0.5	
fall_transition	0	0.1	
rise_transition	0	0.15	
previous pin	clock source	CLK (r/f)	CLK (r/f)

Assume that the discovered arc's (CLK→Q') cell_rise delay, obtained by the interpolation function is 1.

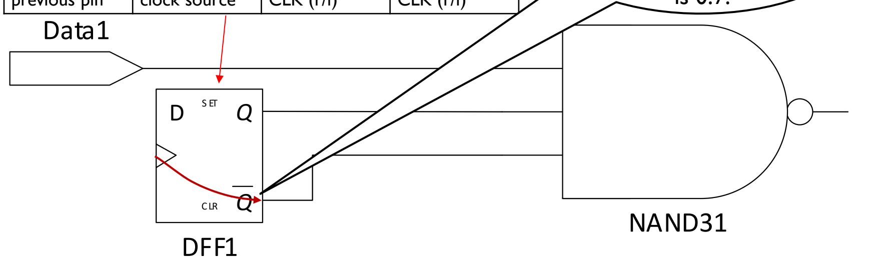


86

Timing Annotation Longest Path (Sequential)

	CLK	Q	Q'
fall_delay	0	0.5	0.7
rise_delay	0	1	1
AAT	0	1	
local_cell_rise		1	1
local_cell_fall		0.5	0.7
fall_transition	0	0.1	
rise_transition	0	0.15	
previous pin	clock source	CLK (r/f)	CLK (r/f)

Assume that the discovered arc's (CLK→Q') cell_fall delay, obtained by the interpolation function is 0.7.

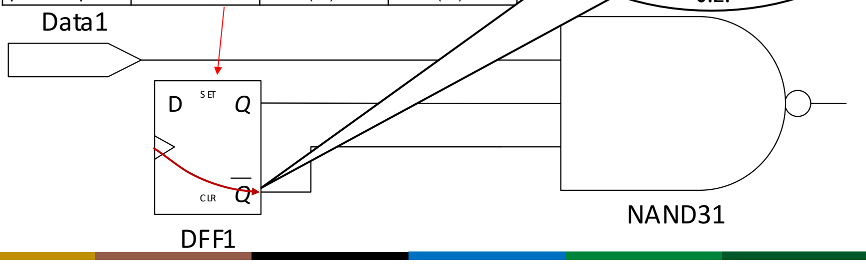


87

Timing Annotation Longest Path (Sequential)

	CLK	Q	Q'
fall_delay	0	0.5	0.7
rise_delay	0	1	1
AAT	0	1	
local_cell_rise		1	1
local_cell_fall		0.5	0.7
fall_transition	0	0.1	0.2
rise_transition	0	0.15	
previous pin	clock source	CLK (r/f)	CLK (r/f)

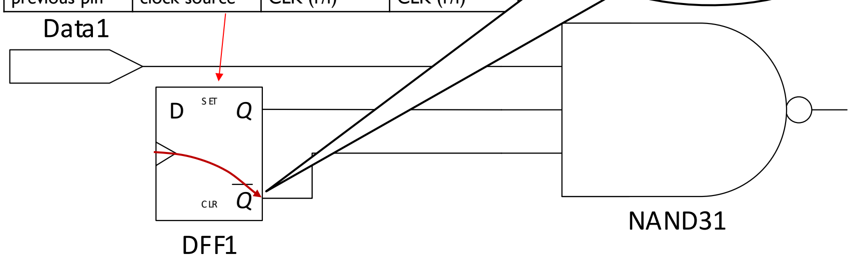
Assume that the discovered arc's (CLK→ Q') fall_transition delay, obtained by the interpolation function is 0.2.



Timing Annotation Longest Path (Sequential)

	CLK	Q	Q'
fall_delay	0	0.5	0.7
rise_delay	0	1	1
AAT	0	1	
local_cell_rise		1	1
local_cell_fall		0.5	0.7
fall_transition	0	0.1	0.2
rise_transition	0	0.15	0.1
previous pin	clock source	CLK (r/f)	CLK (r/f)

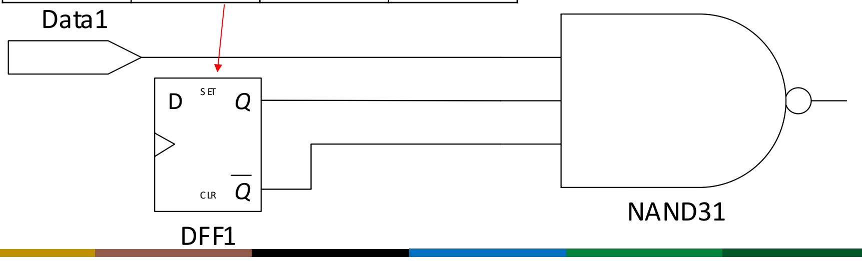
Assume that the discovered arc's (CLK→ Q') rise_transition delay, obtained by the interpolation function is 0.1.



Timing Annotation Longest Path (Sequential)

	CLK	Q	Q'
fall_delay	0	0.5	0.7
rise_delay	0	1	1
AAT	0	1	1
local_cell_rise		1	1
local_cell_fall		0.5	0.7
fall_transition	0	0.1	0.2
rise_transition	0	0.15	0.1
previous pin	clock source	CLK (r/f)	CLK (r/f)

After discovering every arc we know that the AAT on DFF1/Q' is max(cell_rise, cell_fall)

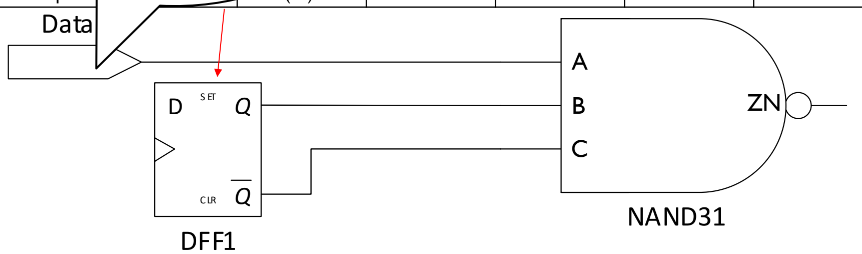


90

Timing Annotation Longest Path (Successor Gatepins)

	Q	Q'	Data1	A	B	C
fall_delay	0.5	0.7	0			
rise_delay			0			
AAT			0			
loc			0			
lo			0			
fall			0			
rise_tr			0			
previous pin		CLK (r/f)	NULL			

Assume Data1 is an unconstrained Primary Input

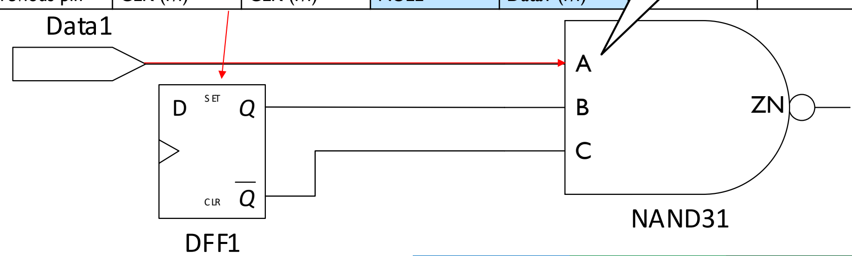


91

Timing Annotation Longest Path (Successor Gatepins)

	Q	Q'	Data1	A	B	
fall_delay	0.5	0.7	0	0		
rise_delay	1	1	0	0		
AAT	1	1	0	0		
local_cell_rise	1	1	0	0		
local_cell_fall	0.5	0.7	0	0		
fall_transition	0.1	0.2	0	0		
rise_transition	0.15	0.1	0	0		
previous pin	CLK (r/f)	CLK (r/f)	NULL	Data1 (r/f)		

Data1 is unconstrained so A's isconstrained field will be set to 0

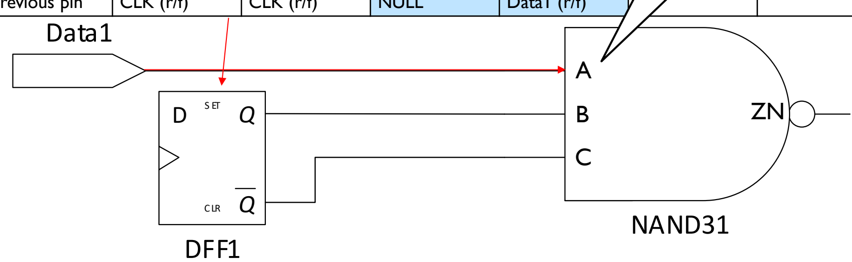


Timing Annotation Longest Path (Successor Gatepins)

			Data1	A	B	
fall_delay			0	0		
rise_delay			0	0		
AAT			0	0		
local_cell_rise		1	0	0		
local_cell_fall	0.5	0.7	0	0		
fall_transition	0.1	0.2	0	0		
rise_transition	0.15	0.1	0	0		
previous pin	CLK (r/f)	CLK (r/f)	NULL	Data1 (r/f)		

The blue color indicates that these pins were visited.

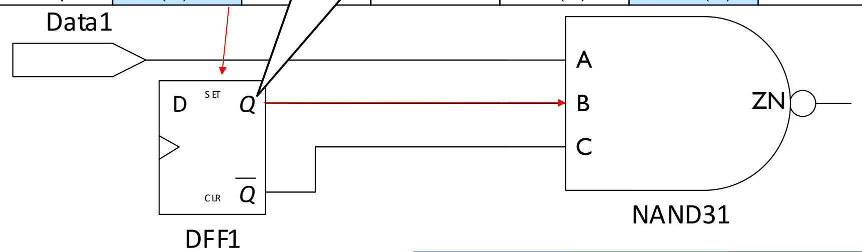
Data1 is unconstrained so A's isconstrained field will be set to 0



Timing Annotation Longest Path (Successor Gatepins)

	Q	Q'	Data1	A	B	C
fall_delay	0.5	0.7		0	0.5	
rise_delay	1	1		0	1	
AAT	1				1	
local_cell_rise	1				1	
local_cell_fall	0.5				0.5	
fall_transition	0.1	0.		0	0.1	
rise_transition	0.15	0.1		0	0.15	
previous pin	CLK (r/f)	CLK (	NULL	Data1 (r/f)	DFF1/Q (r/f)	

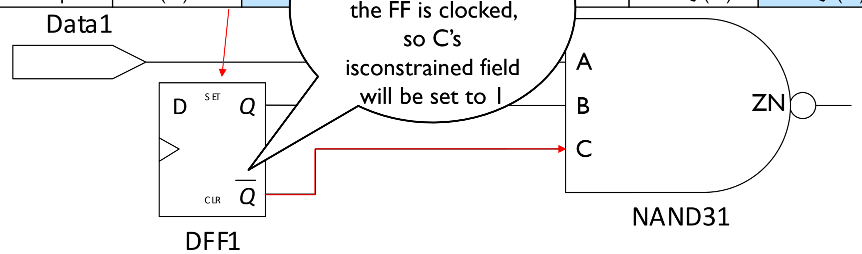
DFF1/Q is a constrained FF output, because the FF is clocked, so B's isconstrained field will be set to 1



Timing Annotation Longest Path (Successor Gatepins)

	Q	Q'	Data1	A	B	C
fall_delay	0.5	0.7	0	0	0.5	0.7
rise_delay	1	1	0	0	1	1
AAT	1	1	0	0	1	1
local_cell_rise	1	1	0	0	1	1
local_cell_fall	0.5	0.7		0	0.5	0.7
fall_transition	0.1	0.2			0.1	0.2
rise_transition	0.15	0.1			0.15	0.1
previous pin	CLK (r/f)	CLK			DFF1/Q (r/f)	DFF1/Q' (r/f)

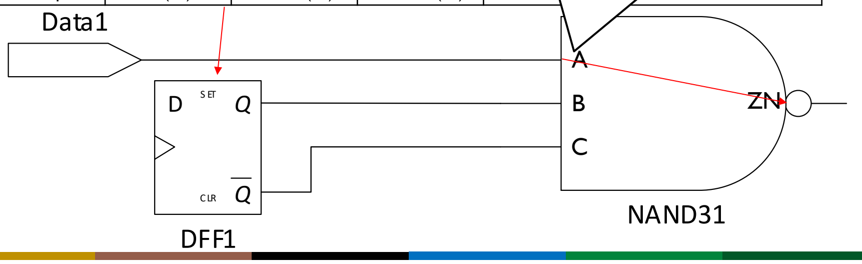
DFF1/Q' is a constrained FF output, because the FF is clocked, so C's isconstrained field will be set to 1



Timing Annotation Longest Path (Successor Gatepins)

	A	B	C	ZN
fall_delay	0	0.5	0.7	
rise_delay	0	1	1	
AAT	0	1	1	
local_cell_rise	0	1	1	
local_cell_fall	0	0.5	0.7	
fall_transition	0	0.1	0.2	
rise_transition	0	0.15	0.1	
previous pin	Data1 (r/f)	DFF1/Q (r/f)	DFF1/Q' (r/f)	

A is unconstrained so A's cell_rise and cell_fall arcs will be ignored and A can not be the previous pin of ZN, though we need to take into account the A's r/f transition times.

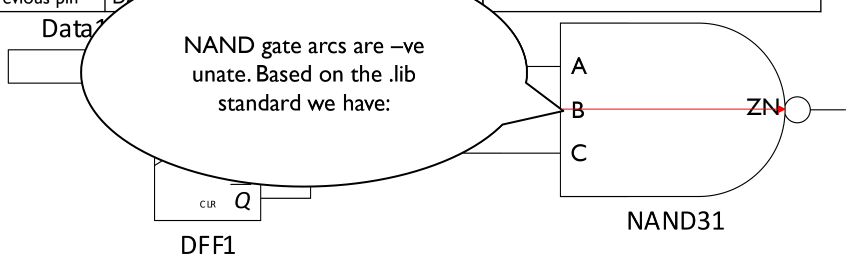


96

Timing Annotation Longest Path (Successor Gatepins)

	A	B	C	ZN
fall_delay	0	0.5	0.7	
rise_delay	0	1	1	
AAT	0	1	1	
local_cell_rise	0	1	1	
local_cell_fall	0	0.5	0.7	
fall_transition	0	0.1	0.2	
rise_transition	0			
previous pin	Da			

NAND gate arcs are –ve unate. Based on the .lib standard we have:

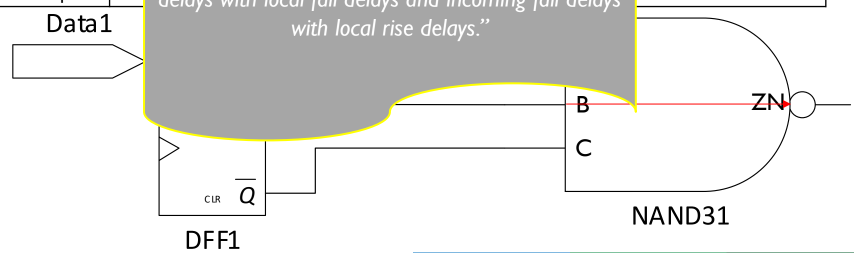


97

Timing Annotation Longest Path (Successor Gatepins)

	A	B	C	ZN
fall_delay	0	0.5	0.7	
rise_delay	0	1	1	
AAT	0	1	1	
local_cell_rise	0	1	1	
local_cell_fall	0	0.5		
fall_transition	0			
rise_transition	0			
previous pin	Dat			

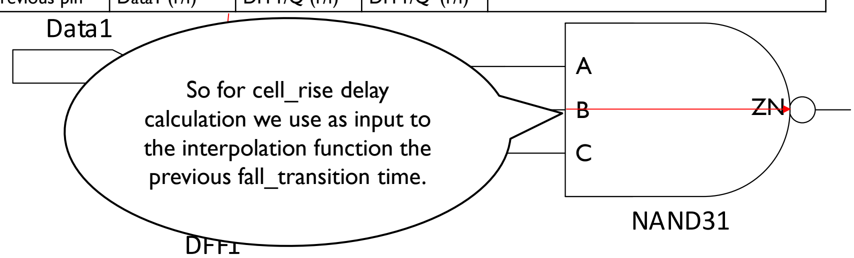
"A negative_unate arc combines incoming rise delays with local fall delays and incoming fall delays with local rise delays."



Timing Annotation Longest Path (Successor Gatepins)

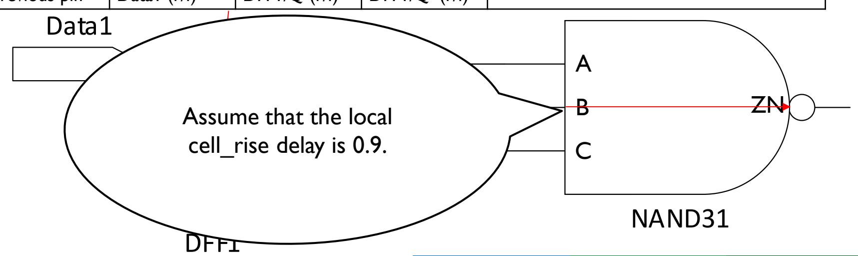
	A	B	C	ZN
fall_delay	0	0.5	0.7	
rise_delay	0	1	1	
AAT	0	1	1	
local_cell_rise	0	1	1	
local_cell_fall	0	0.5	0.7	
fall_transition	0	0.1	0.2	
rise_transition	0	0.15	0.1	
previous pin	Data1 (r/f)	DFF1/Q (r/f)	DFF1/Q' (r/f)	

So for cell_rise delay calculation we use as input to the interpolation function the previous fall_transition time.



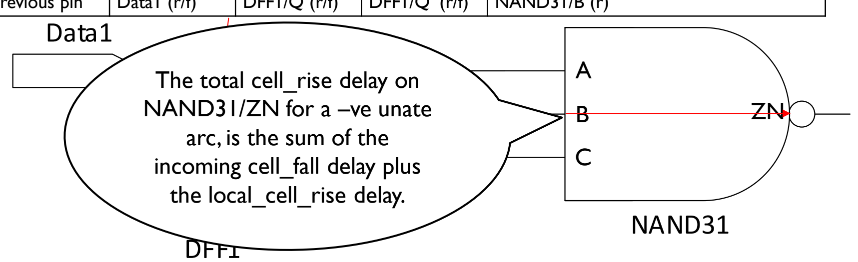
Timing Annotation Longest Path (Successor Gatepins)

	A	B	C	ZN
fall_delay	0	0.5	0.7	
rise_delay	0	I	I	
AAT	0	I	I	
local_cell_rise	0	I	I	0.9
local_cell_fall	0	0.5	0.7	
fall_transition	0	0.1	0.2	
rise_transition	0	0.15	0.1	
previous pin	Data1 (r/f)	DFF1/Q (r/f)	DFF1/Q' (r/f)	



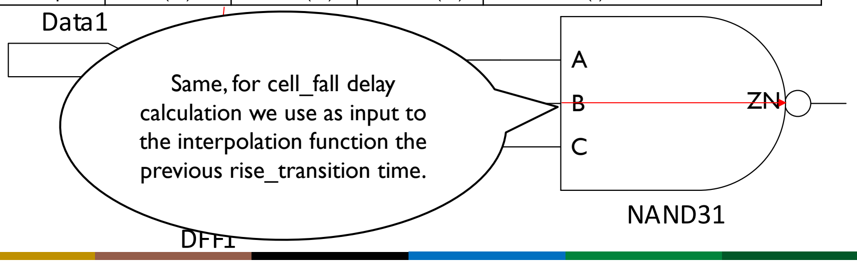
Timing Annotation Longest Path (Successor Gatepins)

	A	B	C	ZN
fall_delay	0	0.5	0.7	
rise_delay	0	I	I	1.4
AAT	0	I	I	
local_cell_rise	0	I	I	0.9
local_cell_fall	0	0.5	0.7	
fall_transition	0	0.1	0.2	
rise_transition	0	0.15	0.1	
previous pin	Data1 (r/f)	DFF1/Q (r/f)	DFF1/Q' (r/f)	NAND31/B (r)



Timing Annotation Longest Path (Successor Gatepins)

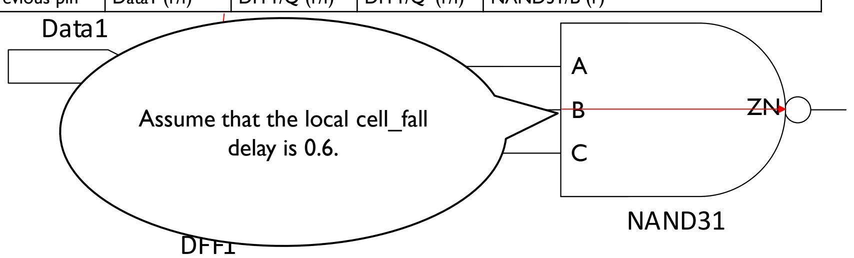
	A	B	C	ZN
fall_delay	0	0.5	0.7	
rise_delay	0	1	1	1.4
AAT	0	1	1	
local_cell_rise	0	1	1	0.9
local_cell_fall	0	0.5	0.7	
fall_transition	0	0.1	0.2	
rise_transition	0	0.15	0.1	
previous pin	Data1 (r/f)	DFF1/Q (r/f)	DFF1/Q' (r/f)	NAND31/B (r)



102

Timing Annotation Longest Path (Successor Gatepins)

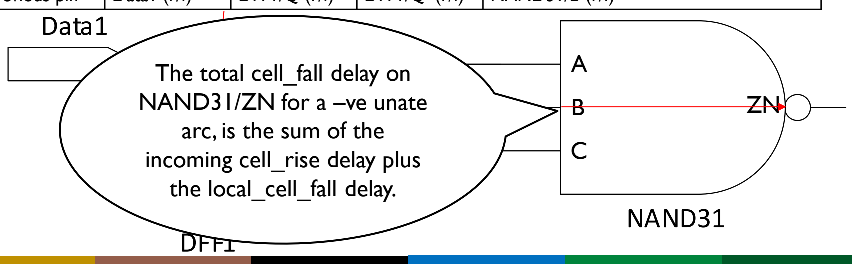
	A	B	C	ZN
fall_delay	0	0.5	0.7	
rise_delay	0	1	1	1.4
AAT	0	1	1	
local_cell_rise	0	1	1	0.9
local_cell_fall	0	0.5	0.7	0.6
fall_transition	0	0.1	0.2	
rise_transition	0	0.15	0.1	
previous pin	Data1 (r/f)	DFF1/Q (r/f)	DFF1/Q' (r/f)	NAND31/B (r)



103

Timing Annotation Longest Path (Successor Gatepins)

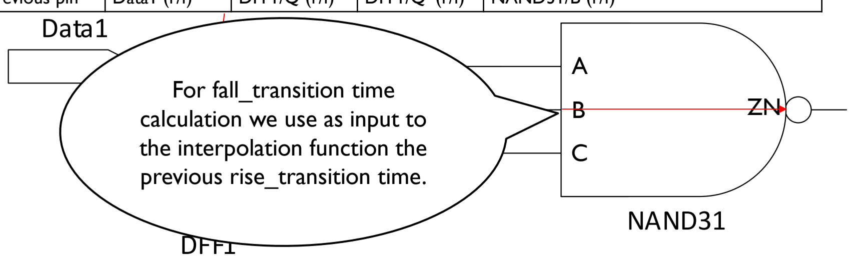
	A	B	C	ZN
fall_delay	0	0.5	0.7	1.6
rise_delay	0	1	1	1.4
AAT	0	1	1	
local_cell_rise	0	1	1	0.9
local_cell_fall	0	0.5	0.7	0.6
fall_transition	0	0.1	0.2	
rise_transition	0	0.15	0.1	
previous pin	Data1 (r/f)	DFF1/Q (r/f)	DFF1/Q' (r/f)	NAND31/B (r/f)



104

Timing Annotation Longest Path (Successor Gatepins)

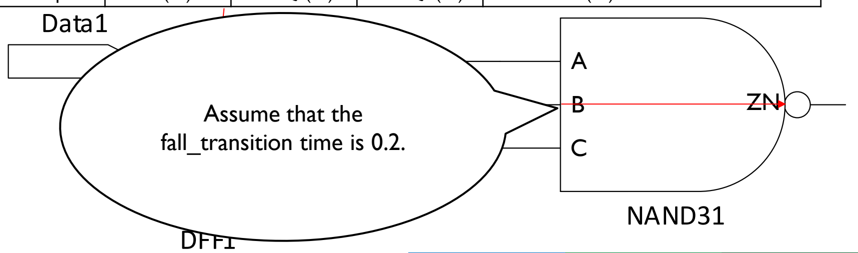
	A	B	C	ZN
fall_delay	0	0.5	0.7	1.6
rise_delay	0	1	1	1.4
AAT	0	1	1	
local_cell_rise	0	1	1	0.9
local_cell_fall	0	0.5	0.7	0.6
fall_transition	0	0.1	0.2	
rise_transition	0	0.15	0.1	
previous pin	Data1 (r/f)	DFF1/Q (r/f)	DFF1/Q' (r/f)	NAND31/B (r/f)



105

Timing Annotation Longest Path (Successor Gatepins)

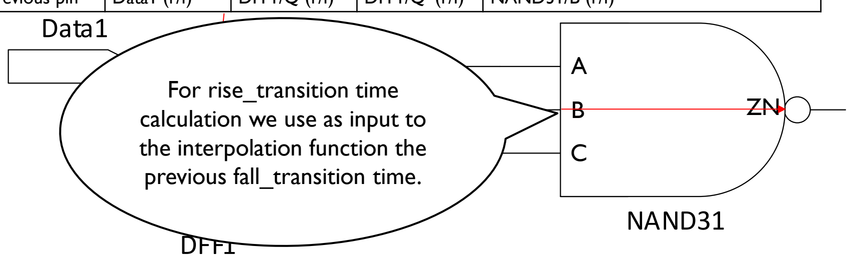
	A	B	C	ZN
fall_delay	0	0.5	0.7	1.6
rise_delay	0	1	1	1.4
AAT	0	1	1	
local_cell_rise	0	1	1	0.9
local_cell_fall	0	0.5	0.7	0.6
fall_transition	0	0.1	0.2	0.2
rise_transition	0	0.15	0.1	
previous pin	Data1 (r/f)	DFF1/Q (r/f)	DFF1/Q' (r/f)	NAND31/B (r/f)



106

Timing Annotation Longest Path (Successor Gatepins)

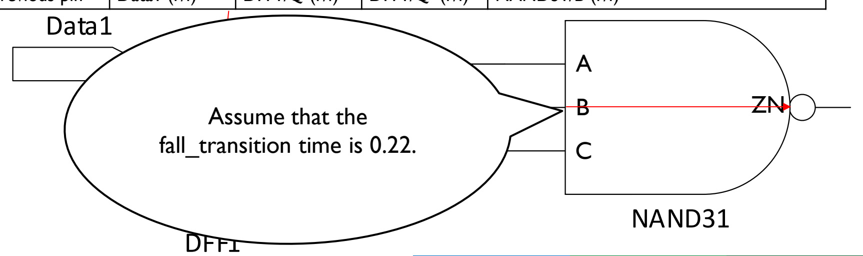
	A	B	C	ZN
fall_delay	0	0.5	0.7	1.6
rise_delay	0	1	1	1.4
AAT	0	1	1	
local_cell_rise	0	1	1	0.9
local_cell_fall	0	0.5	0.7	0.6
fall_transition	0	0.1	0.2	0.2
rise_transition	0	0.15	0.1	
previous pin	Data1 (r/f)	DFF1/Q (r/f)	DFF1/Q' (r/f)	NAND31/B (r/f)



107

Timing Annotation Longest Path (Successor Gatepins)

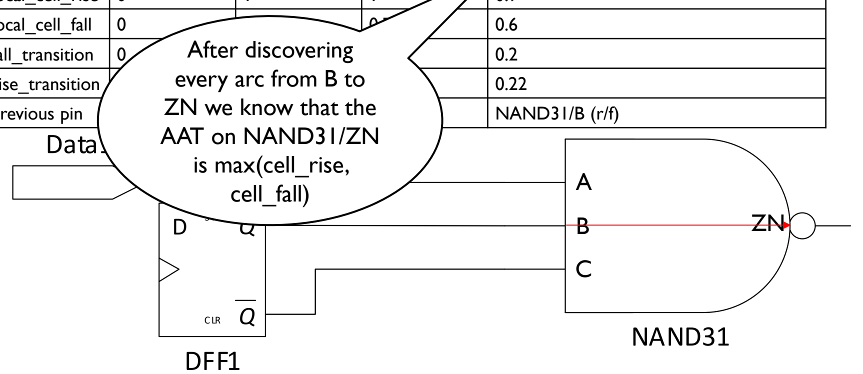
	A	B	C	ZN
fall_delay	0	0.5	0.7	1.6
rise_delay	0	1	1	1.4
AAT	0	1	1	
local_cell_rise	0	1	1	0.9
local_cell_fall	0	0.5	0.7	0.6
fall_transition	0	0.1	0.2	0.2
rise_transition	0	0.15	0.1	0.22
previous pin	Data1 (r/f)	DFF1/Q (r/f)	DFF1/Q' (r/f)	NAND31/B (r/f)



108

Timing Annotation Longest Path (Successor Gatepins)

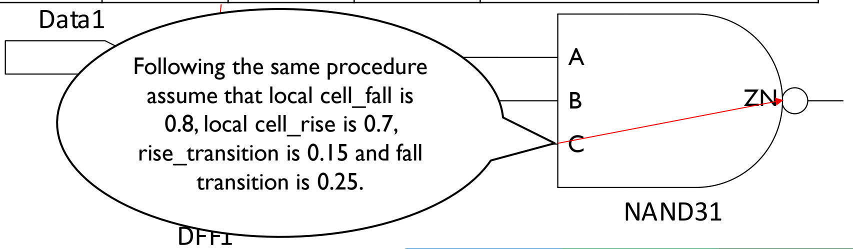
	A	B	C	ZN
fall_delay	0	0.5	0.7	1.6
rise_delay	0	1	1	1.4
AAT	0	1	1	1.6
local_cell_rise	0	1	1	0.9
local_cell_fall	0			0.6
fall_transition	0			0.2
rise_transition				0.22
previous pin				NAND31/B (r/f)



109

Timing Annotation Longest Path (Successor Gatepins)

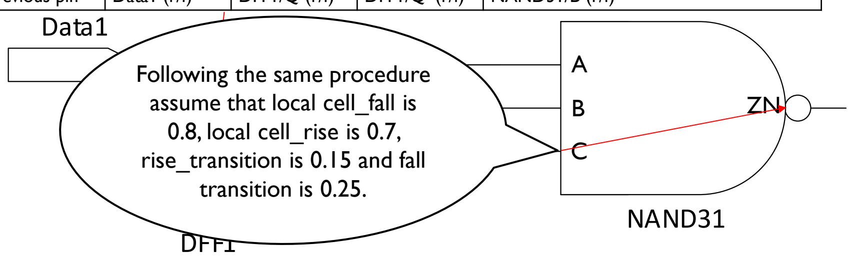
	A	B	C	ZN
fall_delay	0	0.5	0.7	1.6
rise_delay	0	1	1	1.4
AAT	0	1	1	1.6
local_cell_rise	0	1	1	0.9
local_cell_fall	0	0.5	0.7	0.6
fall_transition	0	0.1	0.2	0.2
rise_transition	0	0.15	0.1	0.22
previous pin	Data1 (r/f)	DFF1/Q (r/f)	DFF1/Q' (r/f)	NAND31/B (r/f)



110

Timing Annotation Longest Path (Successor Gatepins)

	A	B	C	ZN
fall_delay	0	0.5	0.7	1.6
rise_delay	0	1	1	1.4
AAT	0	1	1	1.6
local_cell_rise	0	1	1	0.7
local_cell_fall	0	0.5	0.7	0.8
fall_transition	0	0.1	0.2	0.2
rise_transition	0	0.15	0.1	0.22
previous pin	Data1 (r/f)	DFF1/Q (r/f)	DFF1/Q' (r/f)	NAND31/B (r/f)

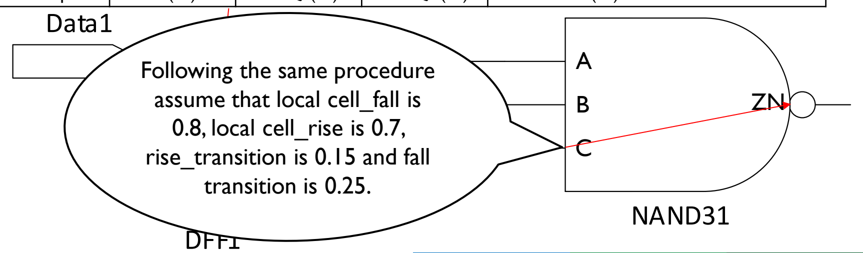


111

Timing Annotation Longest Path (Successor Gatepins)

	A	B	C	ZN
fall_delay	0	0.5	0.7	1.6
rise_delay	0	1	1	1.4
AAT	0	1	1	1.6
local_cell_rise	0	1	1	0.7
local_cell_fall	0	0.5	0.7	0.8
fall_transition	0	0.1	0.2	~~0.2~~ 0.25
rise_transition	0	0.15	0.1	0.22
previous pin	Data1 (r/f)	DFF1/Q (r/f)	DFF1/Q' (r/f)	NAND31/B (r/f)

For every field we keep the max between the previous values and the current ones.

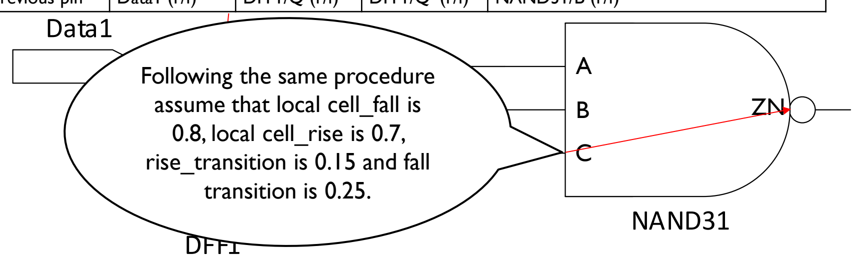


112

Timing Annotation Longest Path (Successor Gatepins)

	A	B	C	ZN
fall_delay	0	0.5	0.7	1.6
rise_delay	0	1	1	1.4
AAT	0	1	1	1.6
local_cell_rise	0	1	1	0.7
local_cell_fall	0	0.5	0.7	0.8
fall_transition	0	0.1	0.2	~~0.2~~ 0.25
rise_transition	0	0.15	0.1	0.22
previous pin	Data1 (r/f)	DFF1/Q (r/f)	DFF1/Q' (r/f)	NAND31/B (r/f)

1.4 == 1.4 so do not replace.

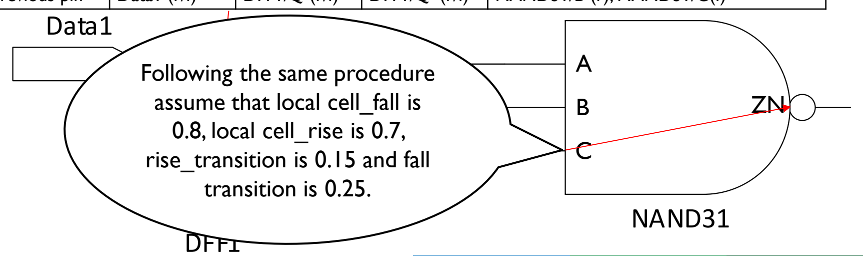


113

Timing Annotation Longest Path (Successor Gatepins)

	A	B	C	ZN
fall_delay	0	0.5	0.7	~~1.6~~ 1.8
rise_delay	0	1	1	1.4
AAT	0	1	1	1.6
local_cell_rise	0	1	1	0.7
local_cell_fall	0	0.5	0.7	0.8
fall_transition	0	0.1	0.2	~~0.2~~ 0.25
rise_transition	0	0.15	0.1	0.22
previous pin	Data1 (r/f)	DFF1/Q (r/f)	DFF1/Q' (r/f)	NAND31/B (r), NAND31/C(f)

1.8 > 1.6 so replace and set C as ZN's previous fall pin.



114

References

- Timing Arcs and Unateness:
  - http://www.vlsi-expert.com
  - http://vlsiuniverse.blogspot.com
- Simulation
  - https://www.swarthmore.edu/NatSci/echeeve1/Ref/mna/MNA2.html

115