Timing Analysis

Nachiket Kapre
nachiket@uwaterloo.ca

Outline

- Measuring performance
  - CPU model
- Composing Clock Period
- Understanding Timing reports
- Static Timing Analysis
  - Longest Path algorithm
  - False Path

Analyzing software performance

- How many cycles are needed to execute this code? Assume each instruction is one cycle

```
void poly(int x, int a, int b, int c, int *y)
{
 *y=a*x*x+b*x+c;
}
```

  - Must analyze each input transition separately

- How many cycles are needed to execute this code? Assume condition checks take one cycle, jumps are free

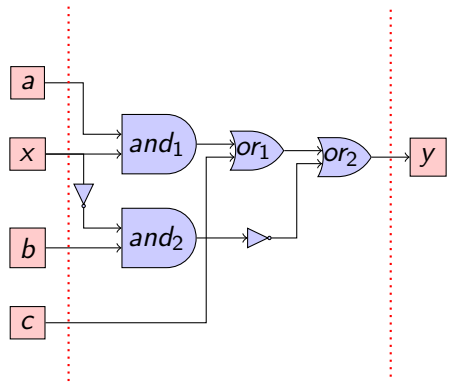
```
void poly(int x, int a, int b, int c, int *y)
{
 if(a>0)
  *y = a*x*x

  *y = *y + bx

 if(a==0)
  *y = *y + c
}
```

  - Must analyze each input transition separately, and ...
  - Different paths possible from input to output

Timing Measurement

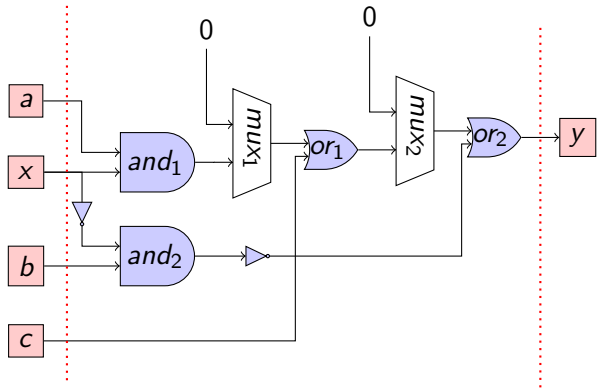
- Worst-case execution time is essential in real-time system design → critical systems
- Average-case execution time (1/throughput) is better suited for shared systems → best-effort systems
- For CPUs, you need to analyze various conditions (if/then) statements to figure out input→output path
- For hardware pipelines, we also need to consider which inputs are changing

Input→Output Paths



- Various paths possible from input to output
- Must enumerate all, as we do not know gate conditions offline

Input→Output Paths (Conditions)

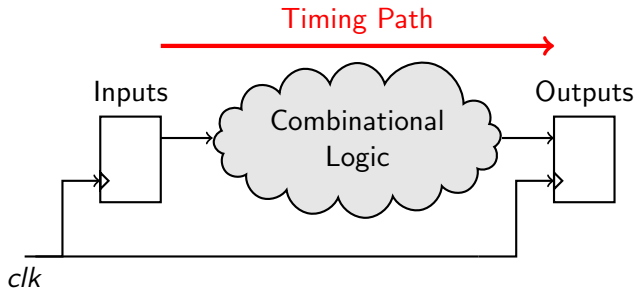


- Some conditions may preempt certain segments

Understanding Timing Budget in Hardware

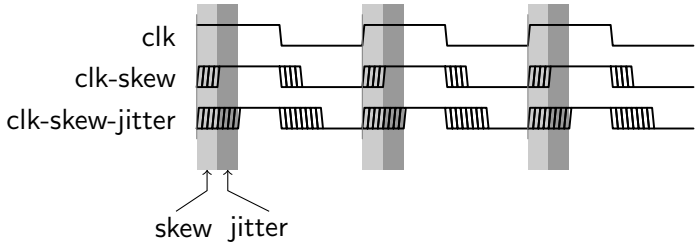
- **Goal**: Make sure the **worst-case** timing path meets user constraints
- Timing paths measured between registers
- Time composed of various components:
  - Clock Skew, Clock Jitter (somewhat out of your control)
  - Setup and Hold time (Register properties also dictated to you)
  - Logic delay $\rightarrow$ you can optimize this!

Visualizing Timing paths



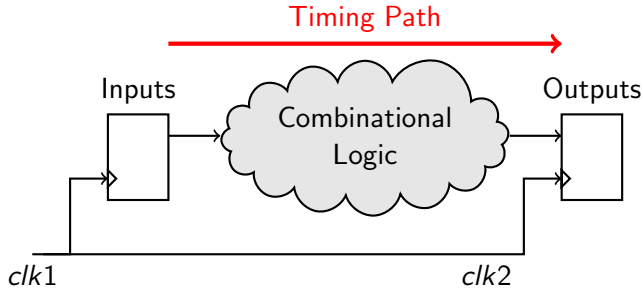
- A timing path starts at the output of a register and ends at the input of a register.
- A circuit will have many such paths. Clock period is defined by the **worst-case** (slowest) path.

Understanding clock margins

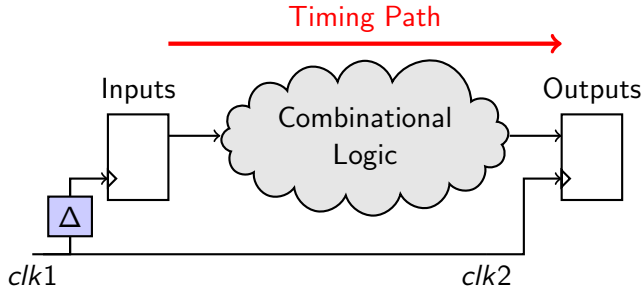


- Clock to two different registers may be offset from each other due to different wire distances, parasitics of the wires that distribute clock to FFs. This is **clock skew**. This is a property of the clock distribution network.
- Clock period to a single register itself may deviate by a small amount periodically. This is **clock jitter**. This is a property of the clock generator and distribution network.
- During timing analysis, we must account for this clock uncertainty *i.e.* subtract both these terms from the clk period.

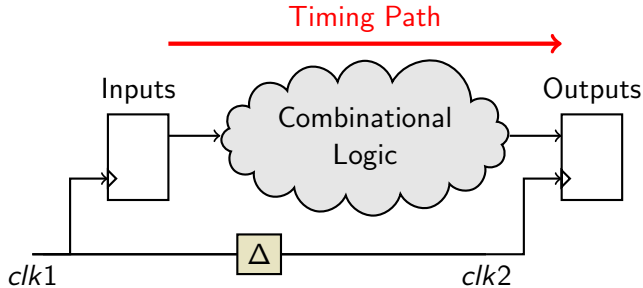
Visualizing Timing paths – Clock margins



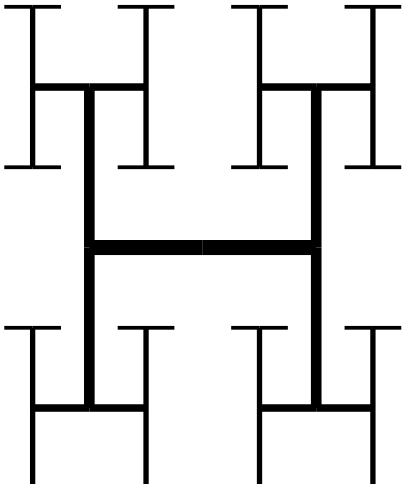
Visualizing Timing paths – Clock margins



Visualizing Timing paths – Clock margins

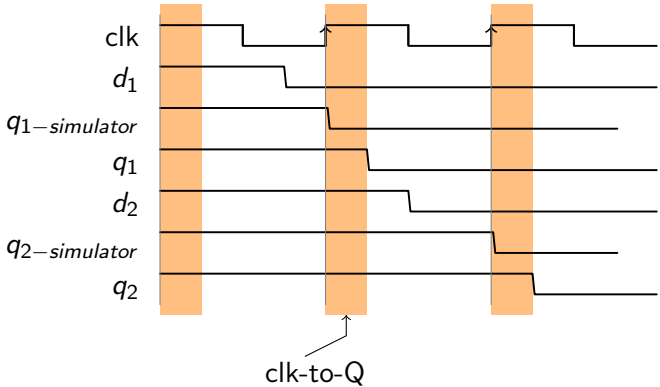


H-Tree for Clock Distribution



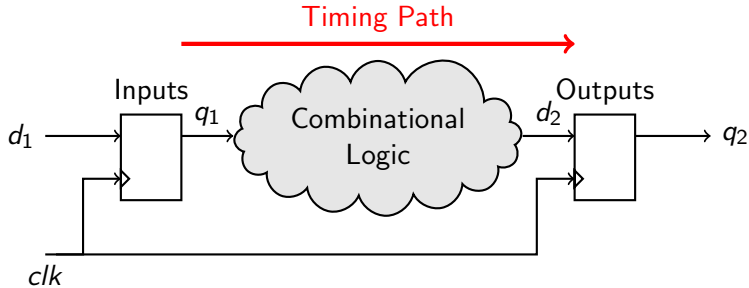
- Only recti-linear wires allowed inside chips
- Most efficient topology to distribute clock + reset is a balanced H-tree
- Wires higher up in the tree carry more current and must be sized accordingly
- However, perfect balance is practically difficult to achieve, hence we need to introduce skew + jitter.

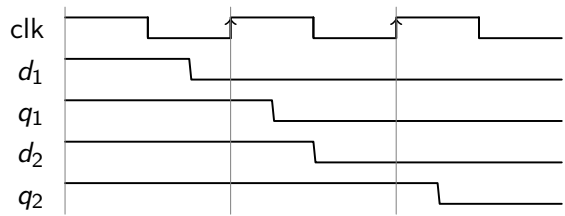
Understanding Clk-to-Q



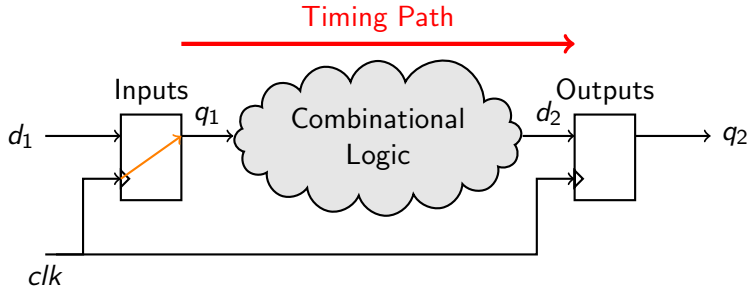
- **Clock-to-Q** delay is the time required for the output Q of a register to be stable after sampling D input on **posedge** `clk` condition.
  - Remember, simulator used an artificial timestep to account for non-blocking assignments. Physical manifestation as $T_{clk \rightarrow Q}$

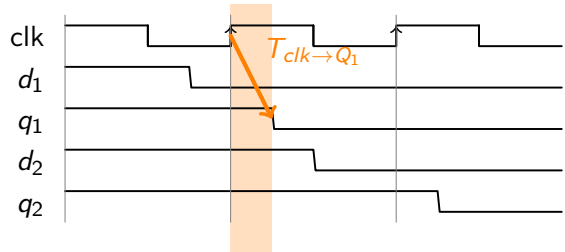
Visualizing Timing paths – Clock-to-Q



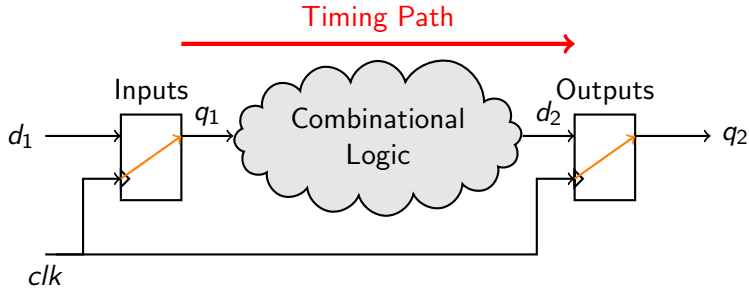


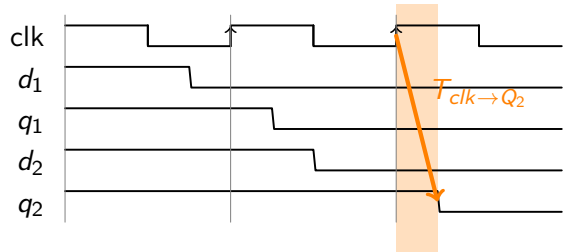
Visualizing Timing paths – Clock-to-Q



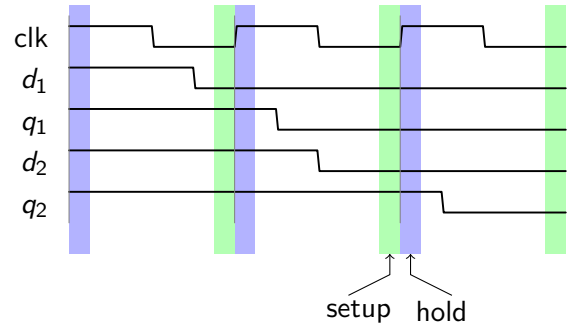


Visualizing Timing paths – Clock-to-Q



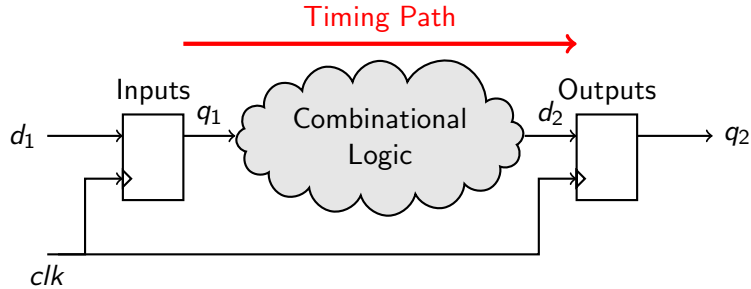


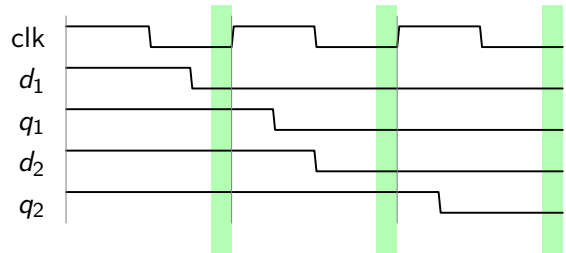
Understanding Setup and Hold time



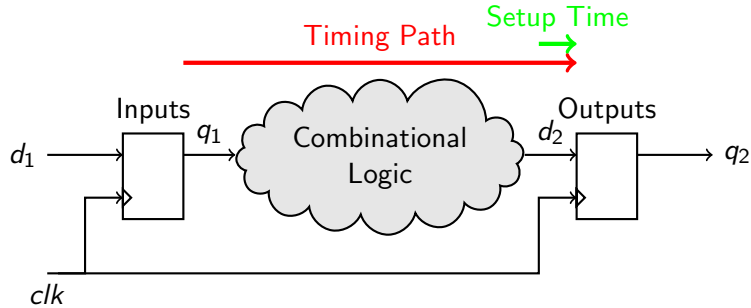
- **Setup time** is the time that the D input to a register must be stable *before* the clock edge hits.
- **Hold time** is the time that the D input to a register must be stable *after* the clock edge hits.
  - Hold time requires that our circuit not work too quickly.
- Within the setup+hold window around the clock edge, the D inputs of any register are not allowed to change. If they do, results are unstable (metastable).

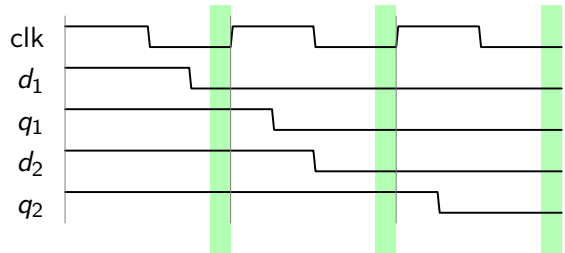
Visualizing Timing paths – Setup Time



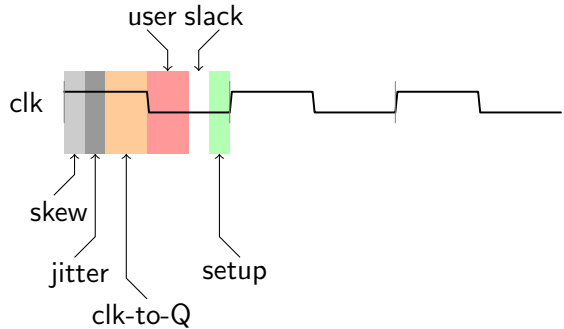


Visualizing Timing paths – Setup Time



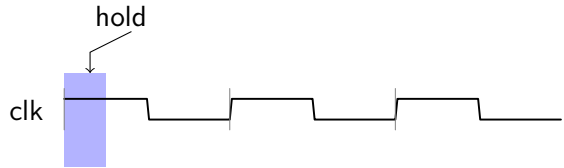


Putting it all together



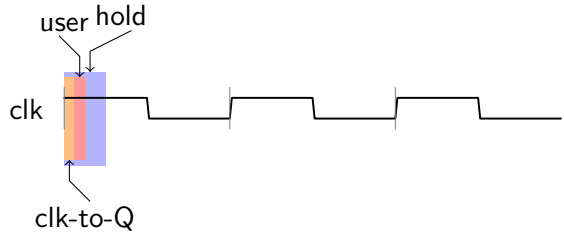
- A user wants to run at frequency $F \rightarrow$ clock period $T_{clk} = \frac{1}{F}$.
- For a circuit to work at a given frequency, user logic delay (worst case) must not exceed T_{clk}
- Physical realities reduce the budget available for user logic.
  $T_{clk} = (T_{clk \rightarrow Q} + T_{skew} + T_{jitter} + T_{setup}) + T_{user} + T_{slack}$
- Final leftover is slack T_{slack}. **Goal**: make sure slack ≥ 0

Putting it all together



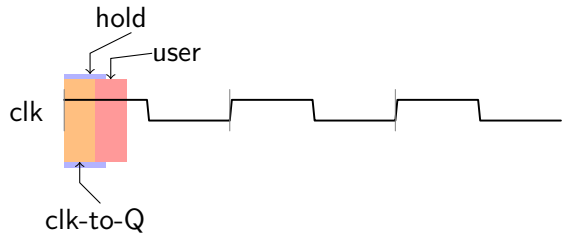
- Hold time constraint $T_{clk \to Q} + T_{user} \geq T_{hold} \rightarrow$ not *too* fast
- As mentioned before, circuit cannot work too quickly to violate next stage register's hold time constraint.
- Optimistically assume there is no skew or jitter on clock edge (cannot rely on skew+jitter to provide enough margin).

Putting it all together



- Hold time constraint $T_{clk \to Q} + T_{user} \geq T_{hold} \rightarrow$ not *too* fast
- As mentioned before, circuit cannot work too quickly to violate next stage register's hold time constraint.
- Optimistically assume there is no skew or jitter on clock edge (cannot rely on skew+jitter to provide enough margin).

Putting it all together

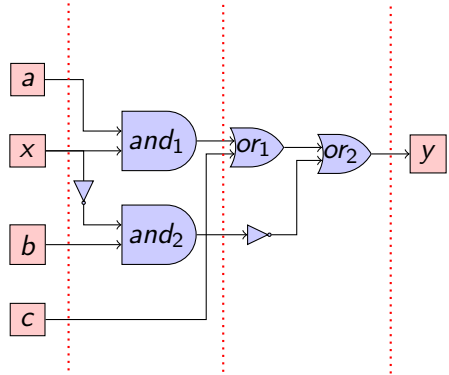


- Hold time constraint $T_{clk \to Q} + T_{user} \geq T_{hold} \rightarrow$ not *too* fast
- As mentioned before, circuit cannot work too quickly to violate next stage register's hold time constraint.
- Optimistically assume there is no skew or jitter on clock edge (cannot rely on skew+jitter to provide enough margin).

Timing Analysis Equations

- $T_{clk} \geq (T_{clk \to Q} + T_{skew} + T_{jitter} + T_{setup}) + T_{user}$
  - Encourages designs to minimize the T terms.
- $T_{clk \to Q} + T_{user} \geq T_{hold}$
  - Forces designs to maintain a minimum T
  - Conflicts with previous optimization

How to design with clock constraints



- 1 cyc: $T_{user1} = max(T_{and} + 2 \cdot T_{or}, T_{and} + T_{or} + 2 \times T_{not}$
  - $T_{clk} = (T_{skew} + T_{jitter} + T_{clk \to Q} + T_{setup}) + T_{user1}$
- 2 cyc: $T_{user2} = max(T_{not} + T_{and}, max(2 \cdot T_{or}, T_{not} + T_{or}))$
  - $T_{clk} = (T_{skew} + T_{jitter} + T_{clk \to Q} + T_{setup}) + T_{user2}$
- Clock tax must be paid in each pipeline stage

Understanding Timing Report

- FPGA CAD tools need to be told how fast you want your circuit to run. For Xilinx FPGAs, this information is often supplied via an XDC constraints file using TCL syntax.
- RTL Pipelining is still manual – RTL designer must chop logic into register stages, take care to meet latency constraints
- CAD tools can help pack gates into LUTs based on your desired frequency
- If it doesn't meet timing, go back and try a different pipelining strategy (add more stages)
- Better strategy is to do paper-pencil evaluation first, try to predict how fast your code will synthesize and choose pipelining properly upfront
- Tools report WNS (worst-case negative slack) and TNS (total negative slack) values of slack
  - Clock period dictated by WNS alone
  - How far away you are from timing closure is captured in TNS

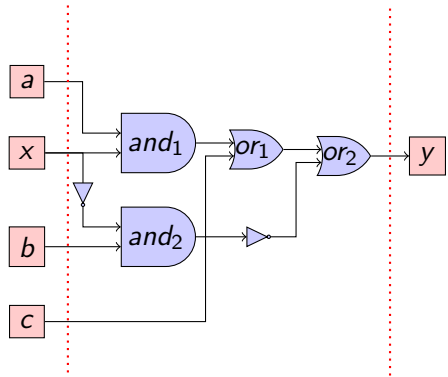
Critical Paths

- A **critical path** is the input→output path between registers that defines maximum frequency
- The slowest path (longest path) in your design is your critical path
- Slowest path either goes through a lot of gates (LUTs), or generally lots of wires

Path Analysis

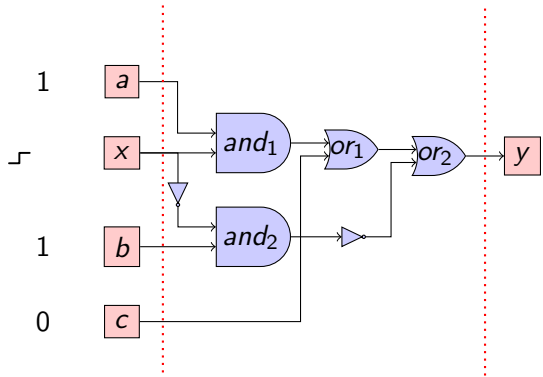
- Timing analysis is done in terms of **transitions** of input that **propagate** to the output
- A circuit with n inputs, each input can be $\rightarrow$ 0, 1, ⎺⎽, ⎽⎺
- Like a truth table, build a **delay table** for $4^n - 2^n$ entries
  - n inputs with 4 choices each $= 4^n$ combinations
  - exclude truth table, as that does not directly define delay $= 2^n$
- For simple analysis, possible to just consider $2 * n$ combinations $\rightarrow$ rising or falling edge on only one input at a time (no real-world tools do this though)
- Smarter tools are able to extract exact input conditions that **excite** specific paths in your circuit $\rightarrow$ reduces the delay table size substantially

Transition Analysis



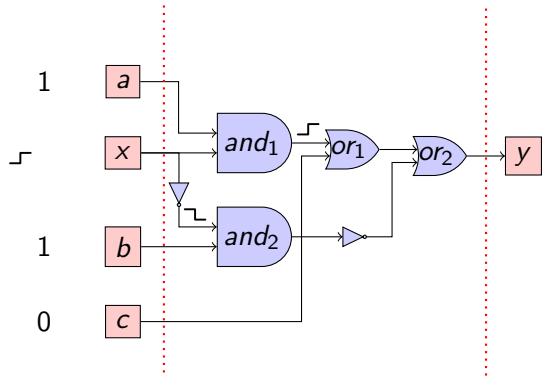
- If a=b=1, c=0 as constants, analyze impact of ⌐¯ and ¯⌐ on x
- Two paths from x to y. Both cause output to change state.
- Depending on specific delays of the gates, one of the paths will `win` the race to the output.

Transition Analysis



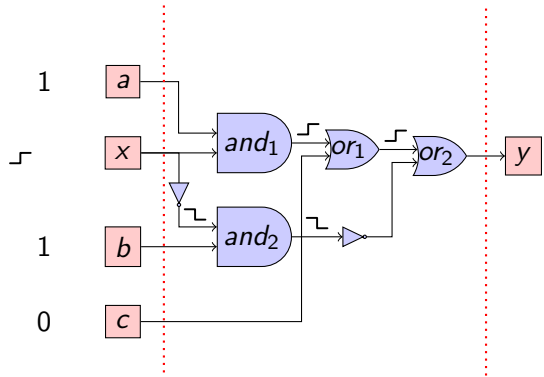
- If a=b=1, c=0 as constants, analyze impact of ⌐¬ and ¬⌐ on x
- Two paths from x to y. Both cause output to change state.
- Depending on specific delays of the gates, one of the paths will `win` the race to the output.

Transition Analysis



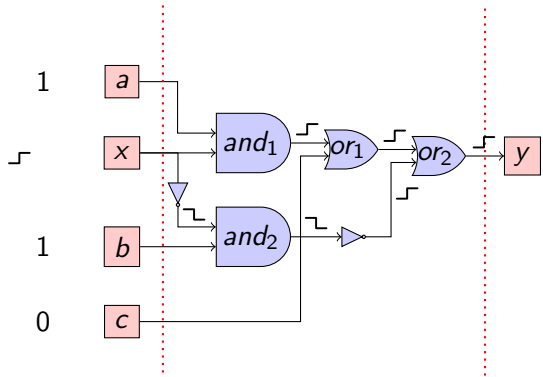
- If a=b=1, c=0 as constants, analyze impact of ⎍⎾ and ⎾⎍ on x
- Two paths from x to y. Both cause output to change state.
- Depending on specific delays of the gates, one of the paths will `win` the race to the output.

Transition Analysis



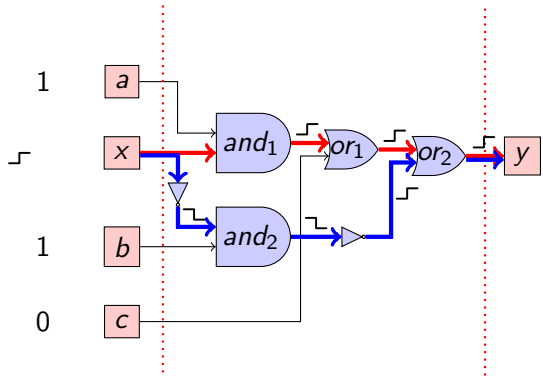
- If a=b=1, c=0 as constants, analyze impact of ⌐¬ and ¬⌐ on x
- Two paths from x to y. Both cause output to change state.
- Depending on specific delays of the gates, one of the paths will win the race to the output.

Transition Analysis



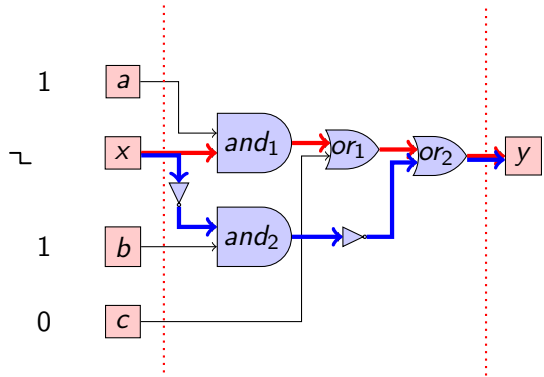
- If a=b=1, c=0 as constants, analyze impact of ⎍ and ⎴ on x
- Two paths from x to y. Both cause output to change state.
- Depending on specific delays of the gates, one of the paths will `win` the race to the output.

Transition Analysis



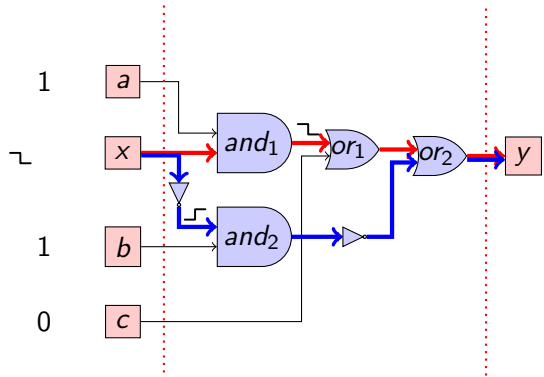
- If a=b=1, c=0 as constants, analyze impact of ⌐¬ and ¬⌐ on x
- Two paths from x to y. Both cause output to change state.
- Depending on specific delays of the gates, one of the paths will win the race to the output.

Transition Analysis



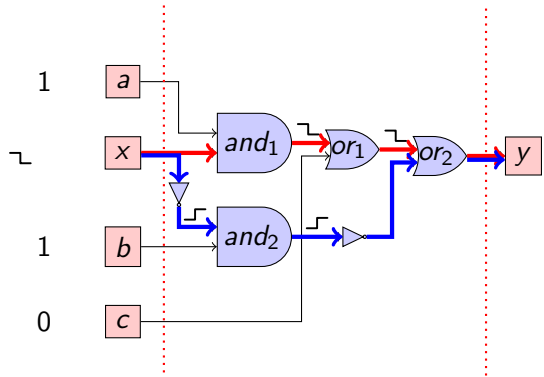
- If a=b=1, c=0 as constants, analyze impact of ⌋⌈ and ⌈⌋ on x
- Two paths from x to y. Both cause output to change state.
- Depending on specific delays of the gates, one of the paths will win the race to the output.

Transition Analysis



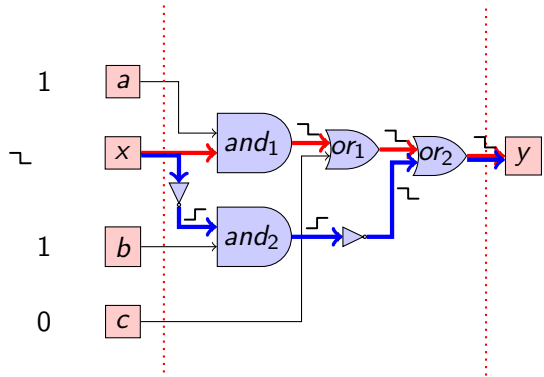
- If a=b=1, c=0 as constants, analyze impact of ⎍⎾ and ⎾⎍ on x
- Two paths from x to y. Both cause output to change state.
- Depending on specific delays of the gates, one of the paths will win the race to the output.

Transition Analysis



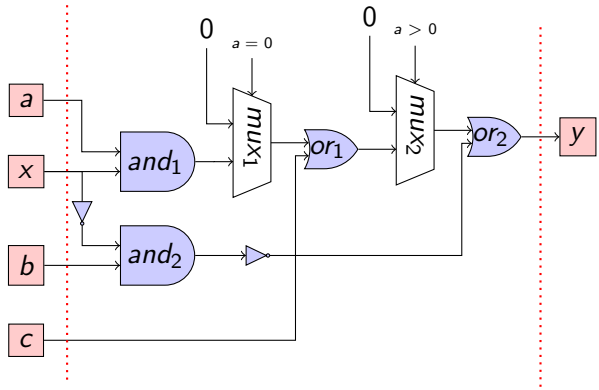
- If a=b=1, c=0 as constants, analyze impact of ⎍⎾ and ⎾⎍ on x
- Two paths from x to y. Both cause output to change state.
- Depending on specific delays of the gates, one of the paths will `win` the race to the output.

Transition Analysis



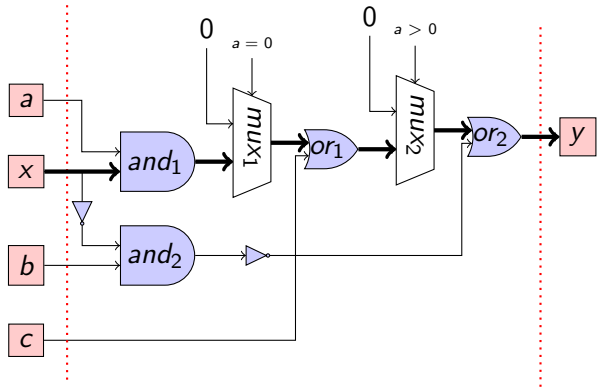
- If a=b=1, c=0 as constants, analyze impact of ⎍⎾ and ⎾⎍ on x
- Two paths from x to y. Both cause output to change state.
- Depending on specific delays of the gates, one of the paths will `win` the race to the output.

Path Analysis (Conditions)



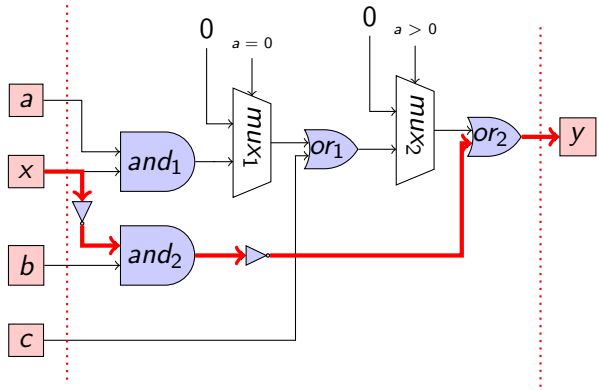
- Longest path connects x to y through 1 AND + 2 OR + 2 MUX. This is a **false path** as can prove non-activation (black)
- One path includes two NOTs, one AND, one OR (red)
- Another path includes two ORs, two MUXes (blue)

Path Analysis (Conditions)



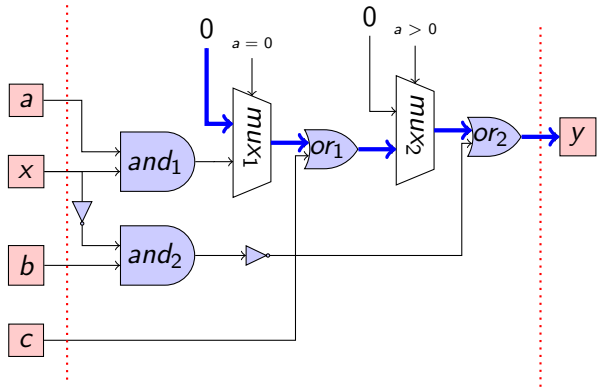
- Longest path connects x to y through 1 AND + 2 OR + 2 MUX. This is a **false path** as can prove non-activation (black)
- One path includes two NOTs, one AND, one OR (red)
- Another path includes two ORs, two MUXes (blue)

Path Analysis (Conditions)



- Longest path connects x to y through 1 AND + 2 OR + 2 MUX. This is a **false path** as can prove non-activation (black)
- One path includes two NOTs, one AND, one OR (red)
- Another path includes two ORs, two MUXes (blue)

Path Analysis (Conditions)



- Longest path connects x to y through 1 AND + 2 OR + 2 MUX. This is a **false path** as can prove non-activation (black)
- One path includes two NOTs, one AND, one OR (red)
- Another path includes two ORs, two MUXes (blue)

Wrapup

- Timing Analysis crucial aspect of digital design $\rightarrow$ allows you to design hardware that satisfies speed constraints
- Clock Period budget eaten away by skew, jitter, and setup times on registers. User circuit must ensure positive slack
- Hold time violations are due to circuits that work too fast
- **Critical paths** are the longest paths in a circuit that carry transitions from input to output
- **False paths** are potential longest paths that **do not** carry transitions the entire path. They cannot be critical paths and should be excluded from analysis.