Introduction to Sequential Logic (cont.)

Antoine Lavault[1,2]

[1]Apeira Technologies

[2]UMR CNRS 9912 STMS, IRCAM, Sorbonne Université

January 19, 2024

Contents

Contents

Outline

Shift registers

There are a few different types that are made by chaining D latches.

- Serial-in to Parallel-out (SIPO)
- Serial-in to Serial-out (SISO)
- Parallel-in to Serial-out (PISO)
- Parallel-in to Parallel-out (PIPO)

Which can be used for :

- SIPO/PISO: serial communication, i.e., UART, modems...
- SISO: Digital delay line, i.e., a building block of digital filters
- PIPO: usage for bit-shifting operation (exponentiation by 2)

SIPO shift register

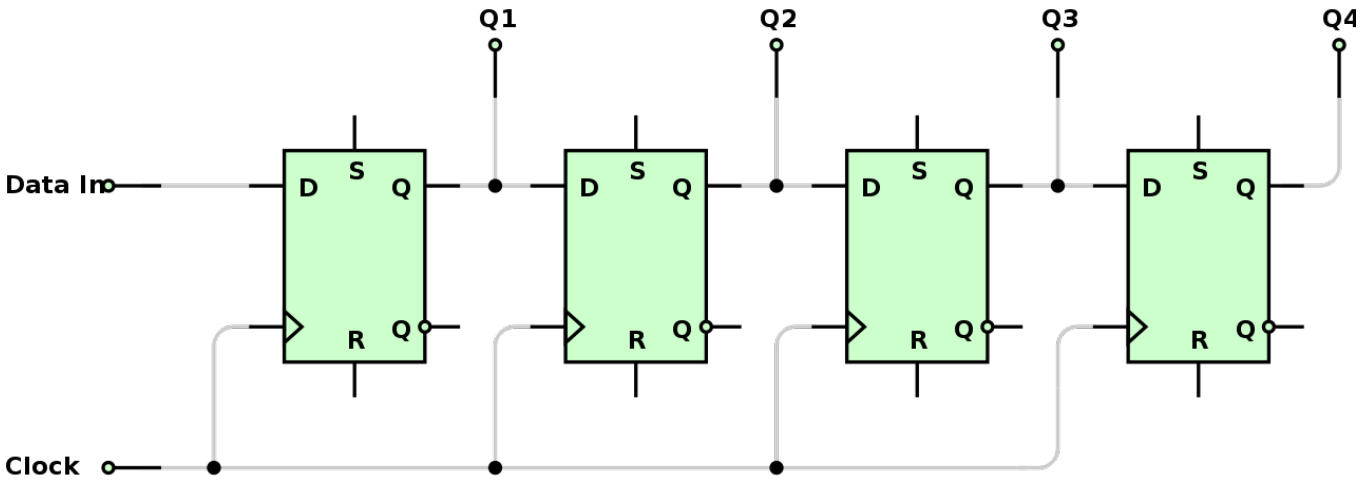


Figure: Caption

Outline

Virtual Memory: abstraction of the hardware!

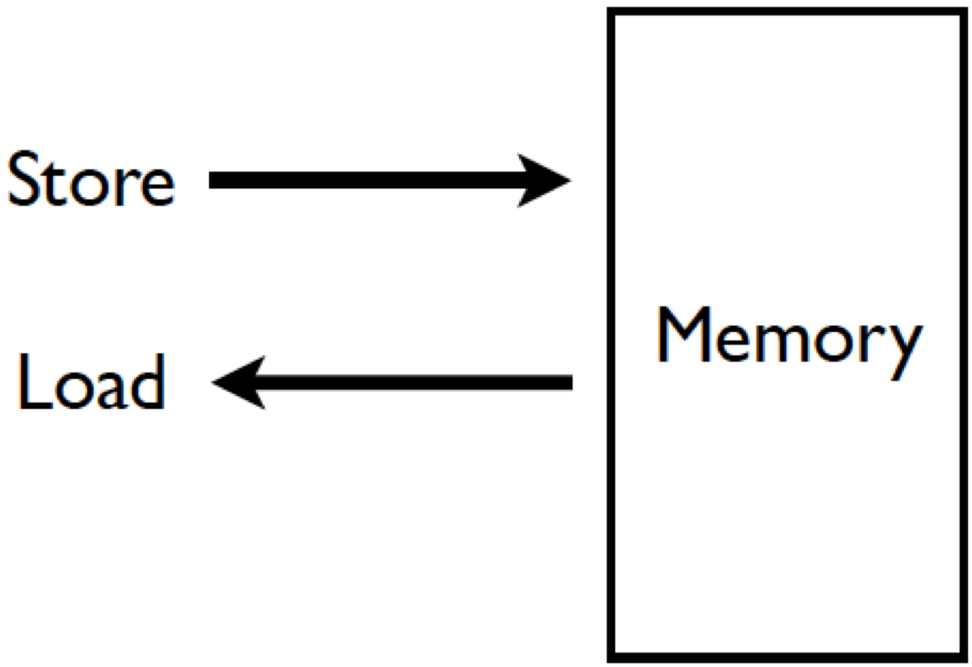


Figure: Virtual Memory

Memory for programmers

- When programming, a programmer sees an abstraction: virtual memory. Which can be assumed to be "infinite."
- In reality, everything is finite
- In computers, the system (i.e., hardware + software) maps virtual addresses to real addresses.
- But on programmable logic, abstractions are not that common... Viva bare metal!

Overview of memories

- Flip-flop/registers :
  - very fast, parallel access
  - Expensive (in terms of hardware)
- Static RAM:
  - rather fast,
  - Expensive (in terms of hardware), one data word at a time
- Dynamic RAM:
  - Cheap (in terms of hardware)
  - Slower, refresh needed, 1 word at a time
- Non-volatile storages:
  - Non-volatile, cheap (in terms of hardware and cost)
  - Much slower, longer access times

Memory Array

A way to store data efficiently:

- A memory array to store data
- Address selection logic (select a row)
- Read the row
- A M-bit value can be read or written at each unique N-bit address
  - All values can be accessed, but only M-bits at a time
  - Access restriction allows a more compact organization

ion

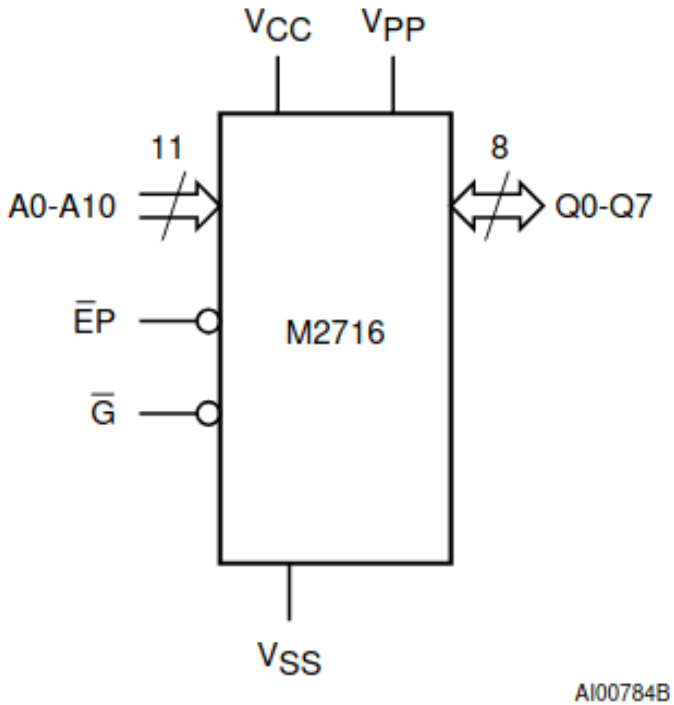


Figure: ROM M2716, how many bits inside ?

Memory array cont.

- In a memory array, a cell stores 1-bit
- In a memory array (like the M2716) with N-address bits and M data bits:
  - 2^N rows of M columns each
  - Depth: number of rows/number of words
  - Width: number of columns/size of the words
  - Array size : $2^N \times M$

Example

For the M2716, we have 11 address bits ($2^11 = 2048$ rows) and an 8-bit output. Which means:

- The depth is 2048
- The width is 8
- The size is 16384 bits in total

Example

Address	Value
00000000	EB
00000001	21
00000002	67
00000003	20
00000004	06
00000005	00
00000006	7E
00000007	BB
...	...

Table: 8-bit ROM dump (M2732)

Diving inside a memory array

- Storage nodes in one column connected to one-bit line
- Address decoder activates only ONE-word line/row
- Content of one line of storage available at the output

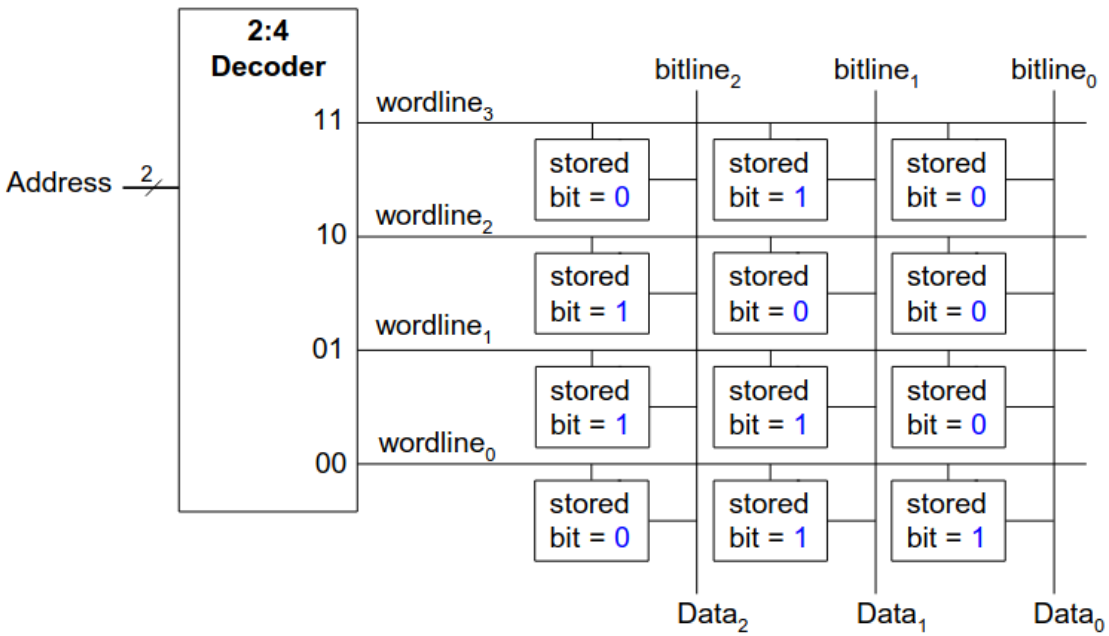


Figure: Illustration of memory array addressing

Two ways to store a bit

- Access transistors configured as switches connect the bit storage to the bit line
- Access controlled by the word line

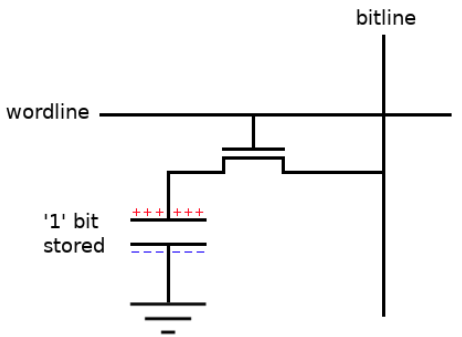


Figure: Capacitor-based memory element

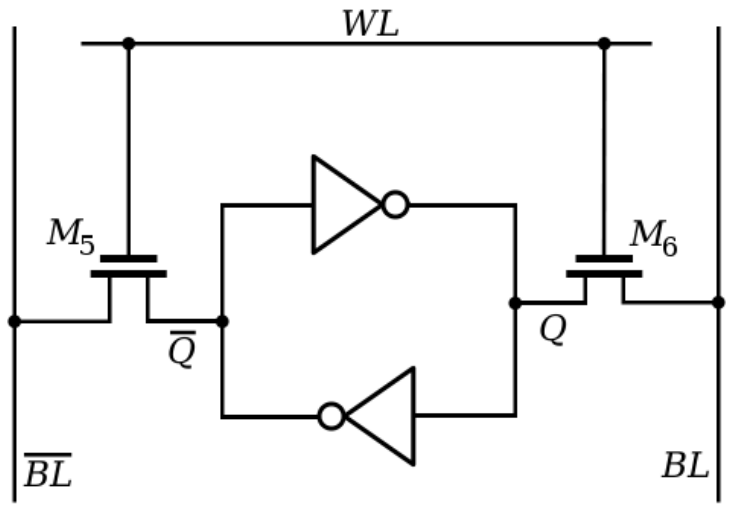


Figure: Bistable-based memory element

Building Larger Memories

- M2716 = 16kbits of memory... So small...
- Larger memories $\Rightarrow$ slow
- Idea: Divide the memory into smaller arrays and interconnect the arrays to input/output buses
  - Large memories are hierarchical array structures
  - DRAM: Channel $\rightarrow$ Rank $\rightarrow$ Bank $\rightarrow$ Subarrays $\rightarrow$ Mats

General Principle- Interleaving (Banking)

- Problem: a single monolithic large memory array takes a long time to access and does not enable multiple accesses in parallel
- Goal: Reduce the latency of memory array access and enable multiple accesses in parallel
- Divide a large array into multiple banks that can be accessed independently (in the same cycle or in consecutive cycles)
  - Each bank is smaller than the entire memory storage
  - Accesses to different banks can be overlapped
- Key Issue: How do you map data to different banks? (i.e., how do you interleave data across banks?)

Outline

Dynamic random access memory

- That's a DRAM
- Capacitor charge state indicates stored value
  - Whether the capacitor is charged or discharged indicates storage of 1 or 0
  - 1 capacitor
  - 1 access transistor
- Capacitor leaks (there is an RC path somewhere)
  - a DRAM cell loses charge over time
  - a DRAM cell needs to be refreshed to hold its data

Static random access memory

- That's an SRAM
- Two cross-coupled inverters store a single-bit
  - Feedback path enables the stored value to persist in the "cell."
  - 4 transistors for storage
  - 2 transistors for access

Control on hardware level

Well... That's the fun part...
Critical timing, bus clearance, asynchronous vs. synchronous...
Maybe later?

Contents

Outline

Circuit Timing

- Until now, we investigated logical functionality
- What about timing?
  - How fast is a circuit?
  - How can we make a circuit faster?
  - What happens if we run a circuit too fast?
- a logically correct design can still fail because of real-world implementation issues!

Combination circuit delay

- Outputs do not change instantaneously with inputs
- Transistors take a finite amount of time to switch
- Gate outputs are delayed with respect to inputs

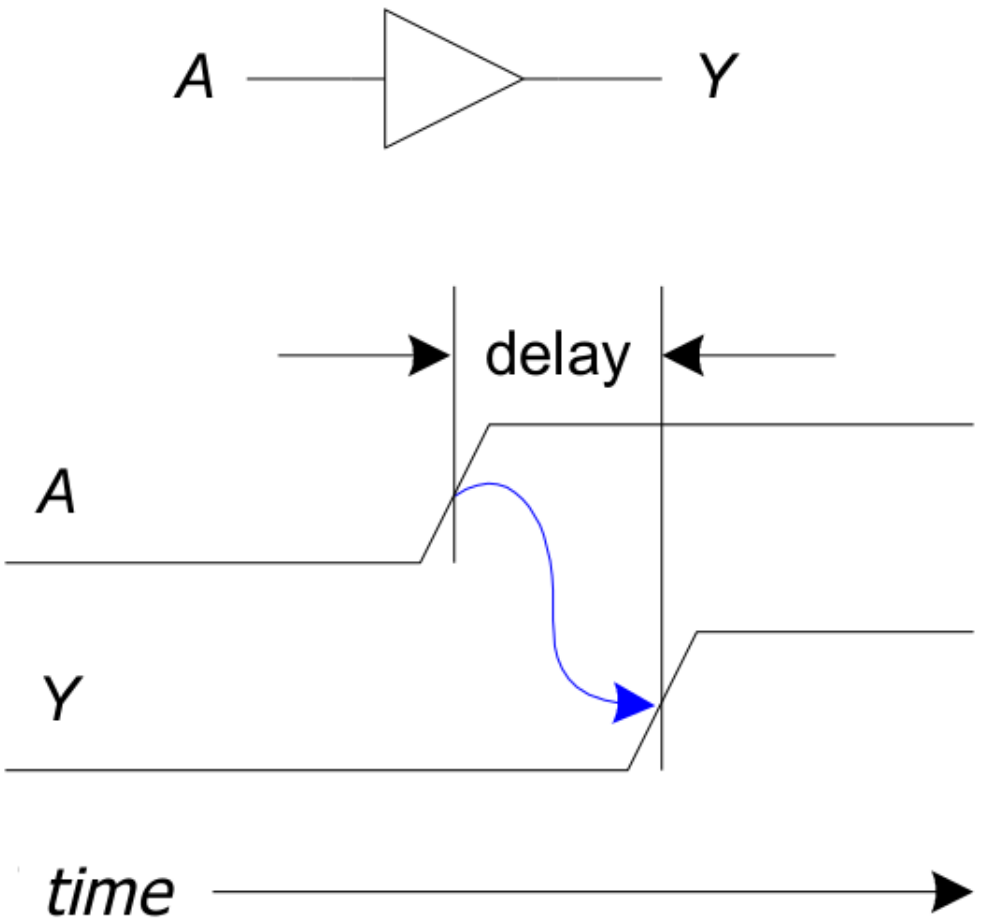


Figure: Caption

Circuit delay variation

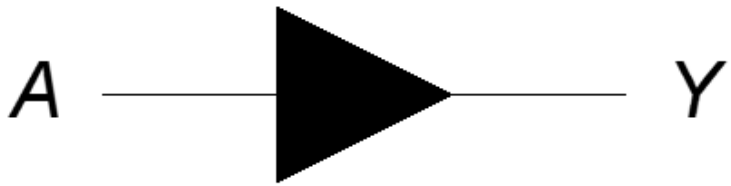
Unfortunately, this is a crude view of circuit delays.

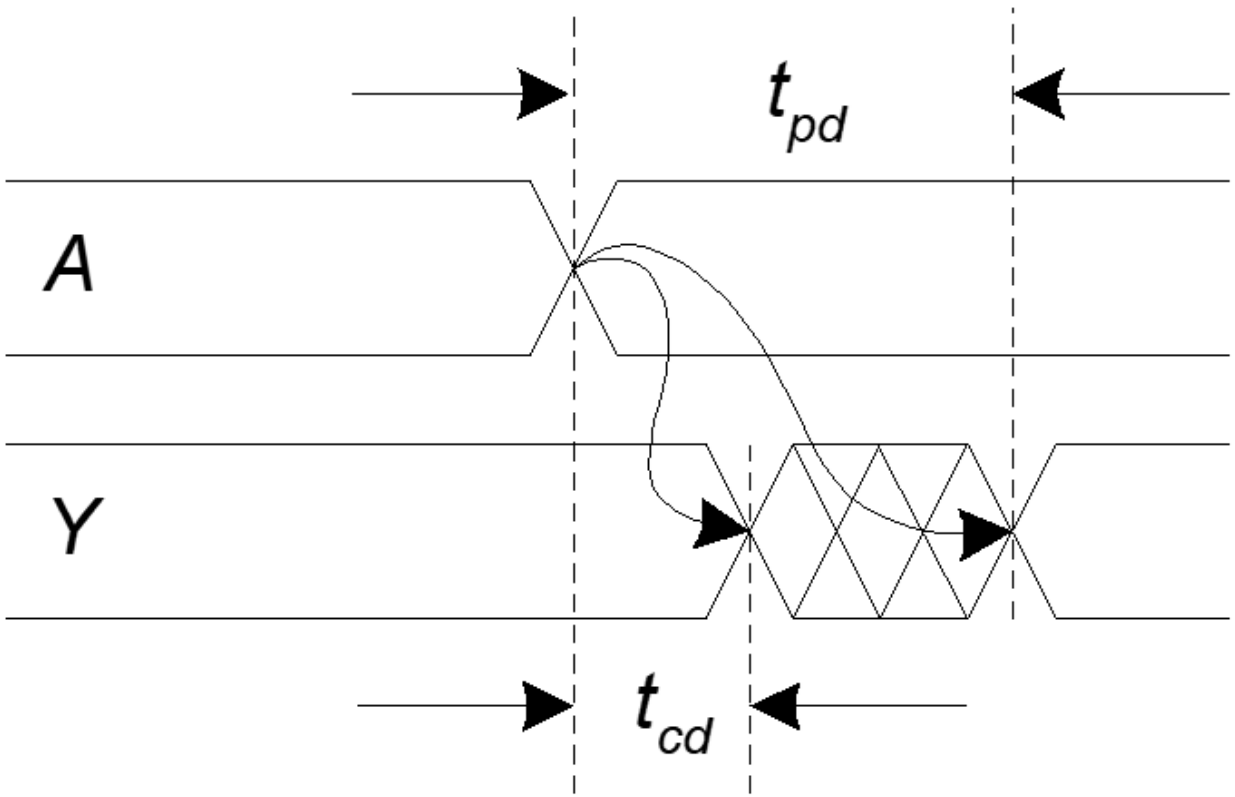
- Delay is fundamentally caused by
  - q Capacitance and resistance in a circuit
  - Finite speed of light (not so fast on a nanosecond scale!)
- Anything affecting these quantities can change delay:
  - Rising (i.e., 0 -> 1) vs. falling (i.e., 1 -> 0) inputs
  - Different inputs have different delays
  - Changes in the environment (e.g., temperature)
- We have a range of possible delays from input to output

Delays from input to output

- Contamination delay t_{cd}: minimum delay
- Propagation delay t_{pd}: maximum delay

Note: cross-hatching means the value is changing. This is not a stable state.s





Calculating path-length

For the following circuit:

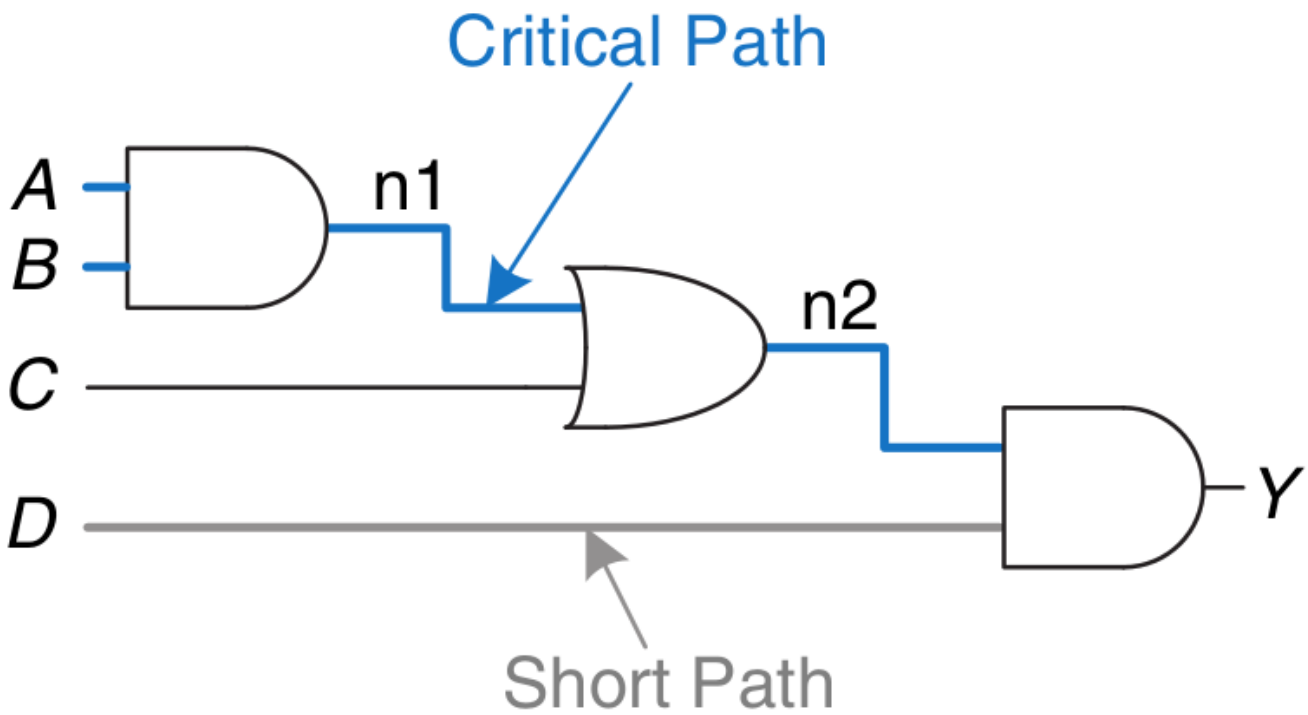


Figure: Path length in a digital circuit

- Critical/Longest path : $t_{pd} = 2t_{pd_{AND}} + t_{pd_{OR}}$
- Shortest path : $t_{cd} = t_{cd_{AND}}$

A real NAND gate

Heavy dependence on voltage and temperature!

Symbol	Parameter	Conditions	25 °C			–40 °C to +125 °C		Unit
			Min	Typ	Max	Max (85 °C)	Max (125 °C)	
74HC00								
t_{pd}	propagation delay	nA, nB to nY; see Figure 6 [1]						
		V_{CC} = 2.0 V	-	25	-	115	135	ns
		V_{CC} = 4.5 V	-	9	-	23	27	ns
		V_{CC} = 5.0 V; C_L = 15 pF	-	7	-	-	-	ns
		V_{CC} = 6.0 V	-	7	-	20	23	ns
t_t	transition time	see Figure 6 [2]						
		V_{CC} = 2.0 V	-	19	-	95	110	ns
		V_{CC} = 4.5 V	-	7	-	19	22	ns
		V_{CC} = 6.0 V	-	6	-	16	19	ns

Figure: Datasheet for the 74HC00

Example - Propagation delay calculation

Two different implementations of a 4:1 multiplexer:

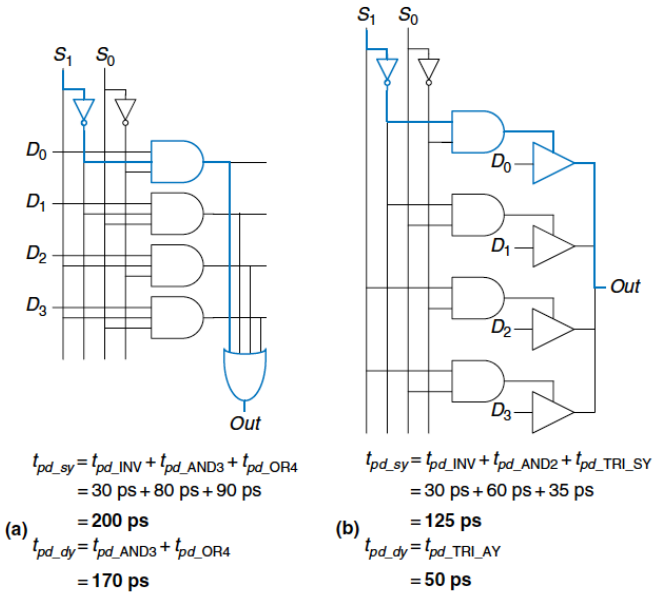


Figure 2.73 4:1 propagation del
(a) two-level lo
(b) tristate

Figure: Two multiplexer architectures

Table 2.7 Timing specifications for multiplexer circuit elements

Gate	t_{pd} (ps)
NOT	30
2-input AND	60
3-input AND	80
4-input OR	90
tristate (A to Y)	50
tristate (enable to Y)	35

Figure: Propagation delays for some gates

Calculating path-length cont.

- It's not always easy to determine the long/short paths!
  - Not all input transitions affect the output
  - Can have multiple different paths from input to output
- In reality, circuits are not all built equally
  - Different instances of the same gate have different delays
  - Wires have a nonzero delay (increasing with length, 30cm/ns)
  - Temperature/voltage affects circuit speeds
  - Not all circuit elements are affected in the same way
  - Can even change the critical path!
- When designing, assume the "worst-case" conditions and run many statistical simulations to balance yield/performance

Output glitches

Definition

Glitch: one input transition causes multiple output transitions

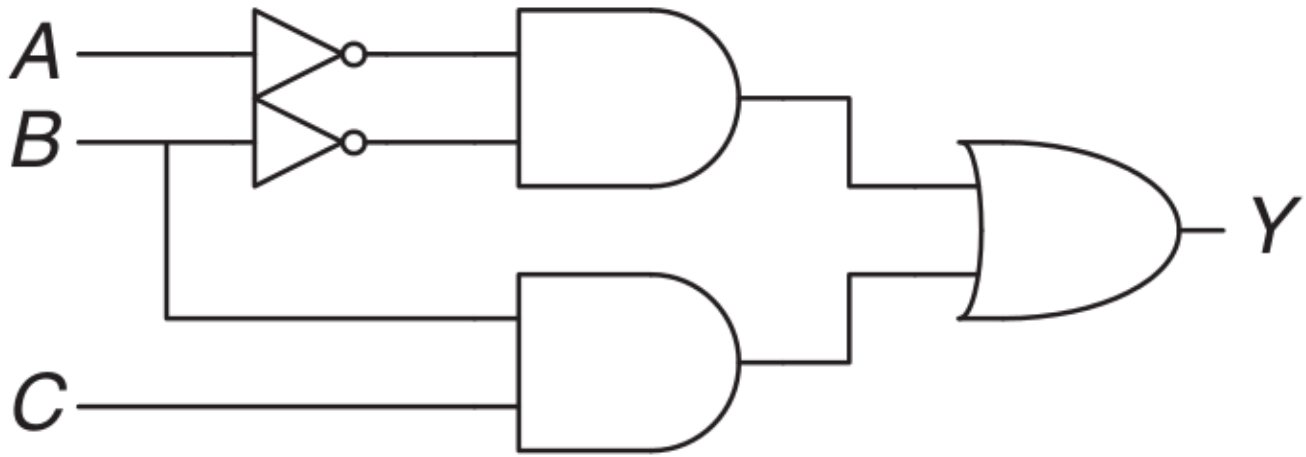


Figure: Example circuit

- Two paths: short (2 gates) and long (3 gates)
- What happens if B goes from high (1) to low (0)?

Output glitches

- Two paths: short (2 gates) and long (3 gates)
- Let's assume $A = 0$, $C = 1$
- What happens if B goes from high (1) to low (0)?
- n_1 output of the upper AND gate
- n_2 output of the lower AND gate

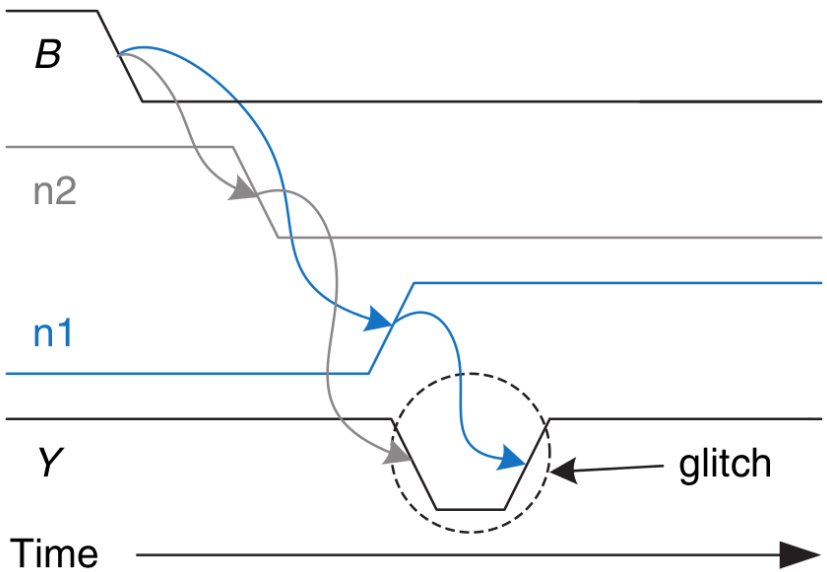


Figure: Glitch chronogram

Karnaugh maps to the rescue

Glitches are visible in K-maps :

- Recall: K-maps show the results of a change in a single input by construction (Gray Code, etc.)
- A glitch occurs when moving between prime implicants (i.e., when groupings are separated)

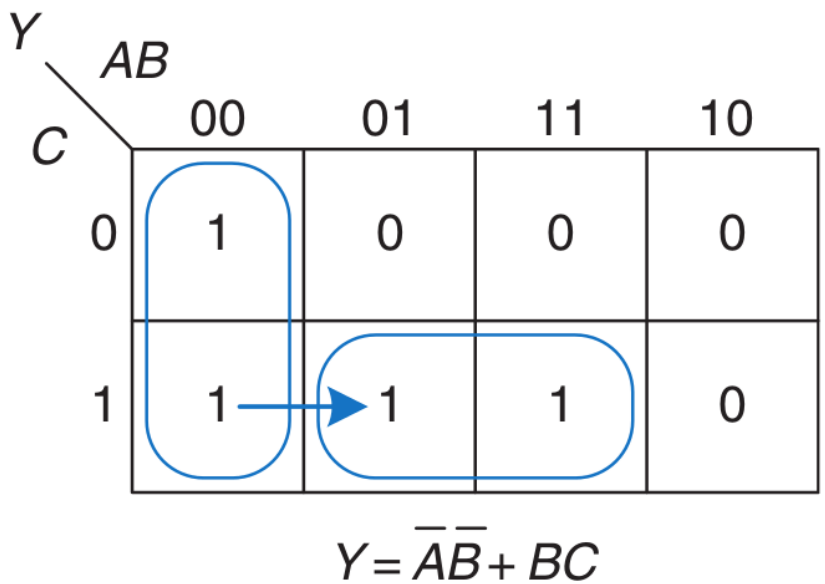


Figure: K-map and prime implicants

Fixing a glitch

Glitches are visible in K-maps :

- We can fix the issue by adding the consensus term
- In general, ensures no transition between different prime implicants

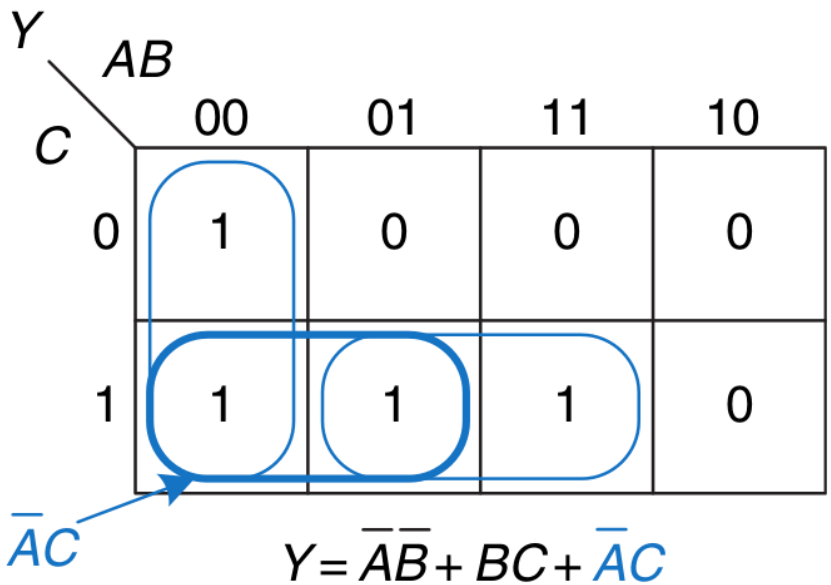


Figure: K-map without prime implicants transitions

Final words on glitches

- Question: Do we always care about glitches?
  - Fixing glitches is undesirable
    - More chip area (more gates)
    - More power consumption (more silicon)
    - More design effort (more work)
  - The circuit is eventually guaranteed to converge to the right value regardless of "glitchiness."
- Answer: No, not always! If we only care about the long-term steady-state output, we can safely ignore glitches

Outline

Back to the basics 2 - D flip-flop

- Flip-flop samples D at the active clock edge
- It outputs the sampled value to Q
- It "stores" the sampled value until the next active clock edge
- The D flip-flop is made from combinational elements
- D, Q, and CLK all have timing requirements!

Timing constraints on the D flip-flop

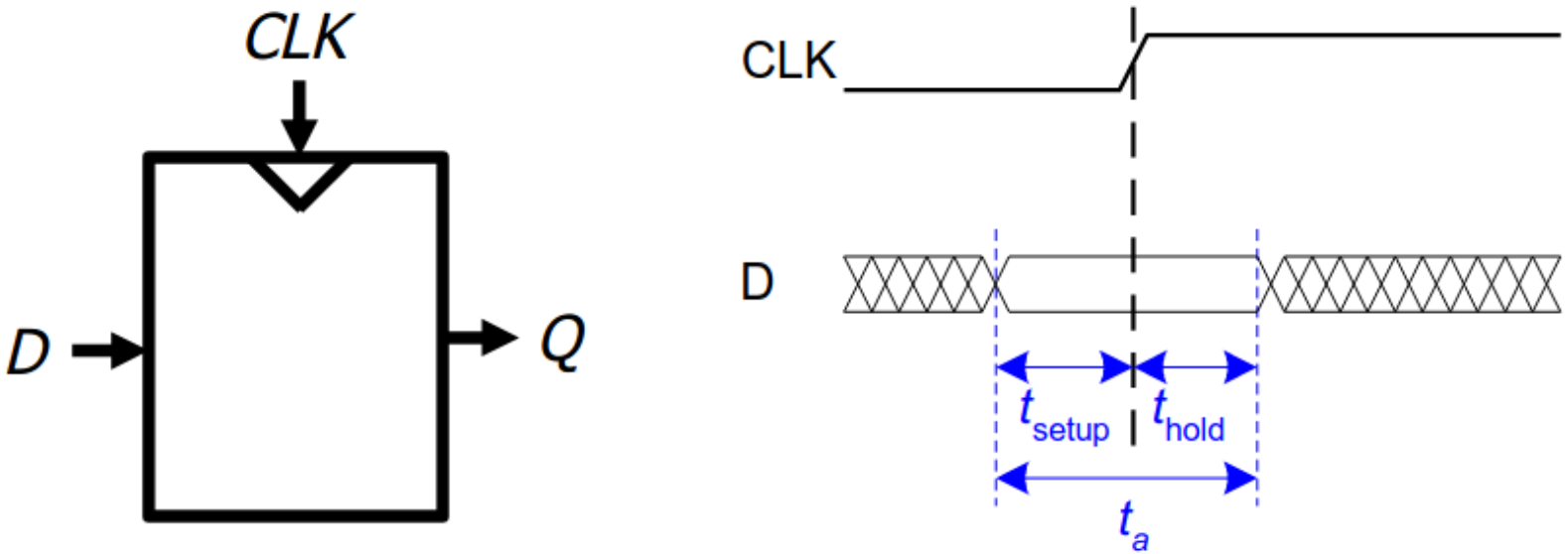


Figure: D flip-flop timing

- The input D must be stable when sampled
- Setup time (t_{setup}): time before the clock edge where D must be stable (i.e. not changing)
- Hold time (t_{hold}): time after the clock edge where D must be stable
- Aperture time (t_a): time around clock edge where D must be stable ($t_a = t_{setup} + t_{hold}$)

Metastability 2 - in colors

If D is changing when sampled, metastability can occur

- Flip-flop output is stuck somewhere between '1' and '0'
- Output eventually settles non-deterministically (i.e., probabilities and stuff)

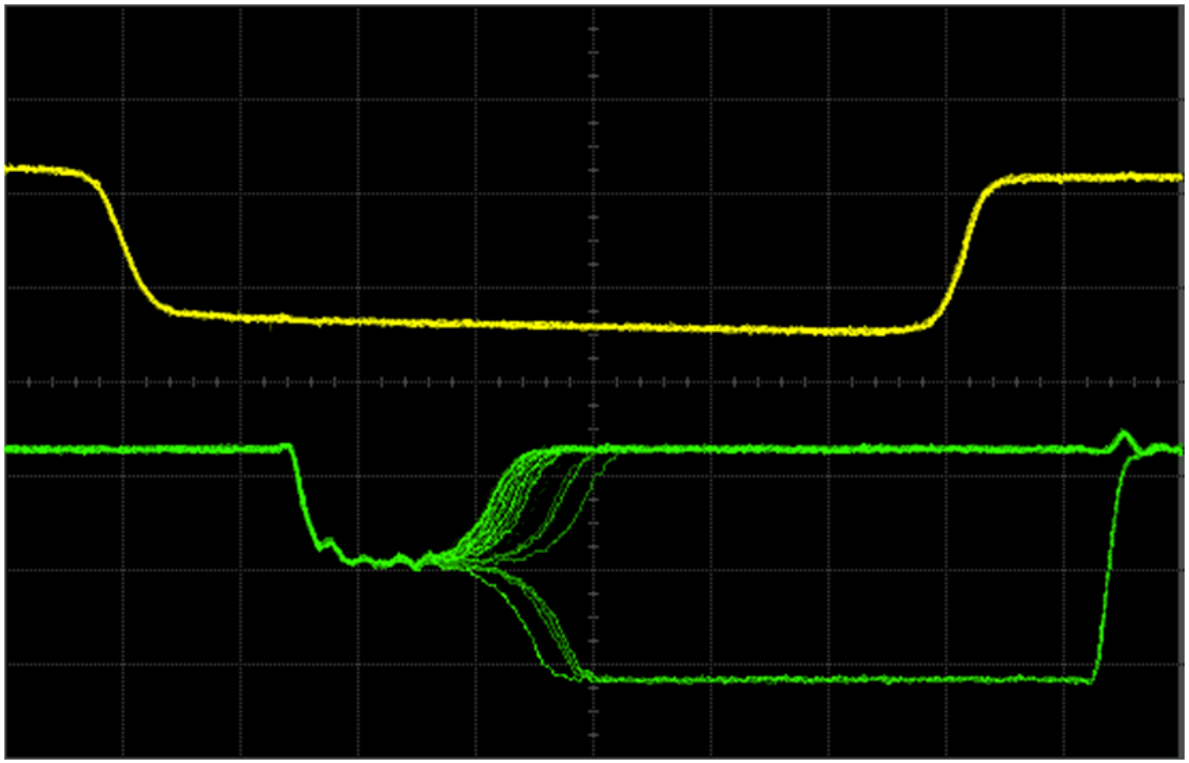


Figure: Metastability and timing violation in an SR latch

D flip-flop 3 - Output timing constraints

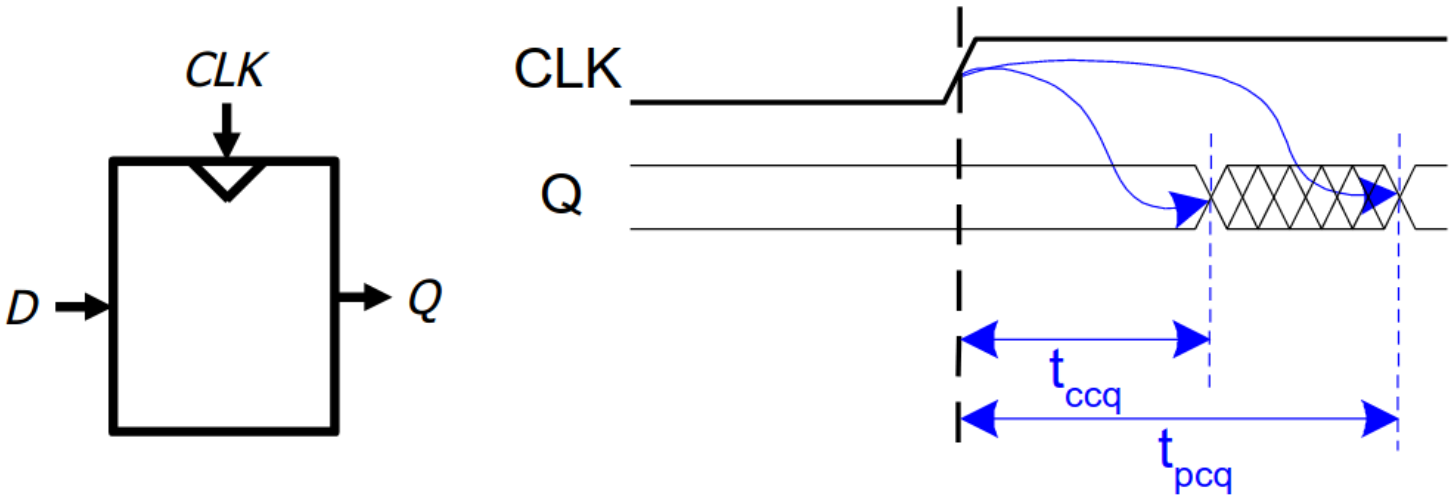


Figure: D flip-flop output timing

- Contamination delay clock-to-q (t_{cqq}): earliest time after the clock edge that Q starts to change (i.e., is unstable)
- Propagation delay clock-to-q (t_{pcq}): latest time after the clock edge that Q stops changing (i.e., is stable)

Ensuring Correct Sequential Operation

- We Need to ensure correct input timing on R2
- Specifically, D2 must be stable:
  - at least t_{setup} before the clock edge
  - at least until t_{hold} after the clock edge

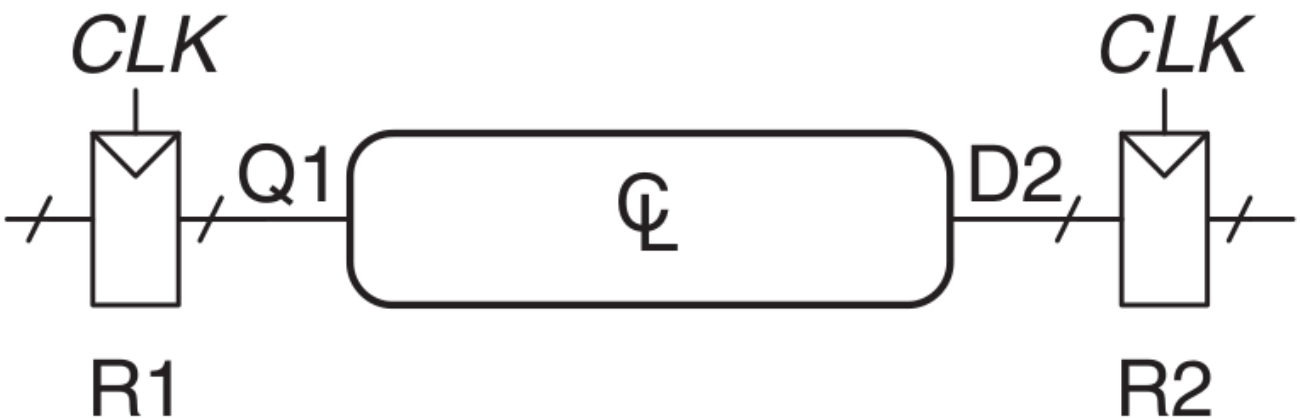


Figure: Sequential circuit

Sequential timing

This means there is both a minimum and maximum delay between two flip-flops

- Comb. Logic too fast $\rightarrow$ R2 t_{hold} violation
- Comb. Logic too slow -> R2 t_{setup} violation

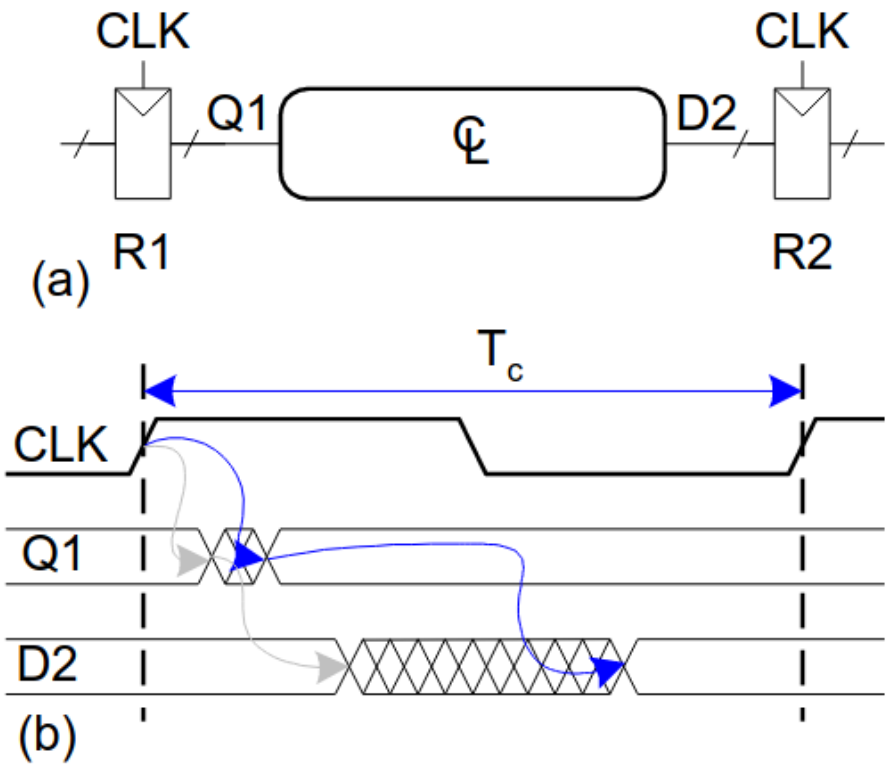


Figure: Sequential circuit flip-flop timing

Setup Time Constraint

- Depends on the maximum delay from R1 to R2
- the input to R2 must be stable at least t_{setup} before the clock edge.
- Sequencing overhead: the amount of time wasted each cycle due to sequencing element timing requirements

$$T_c \geq \overbrace{t_{pcq}}^{\text{wasted}} + \overbrace{t_{pd}}^{\text{useful}} + \overbrace{t_{setup}}^{\text{wasted}} \quad (1)$$

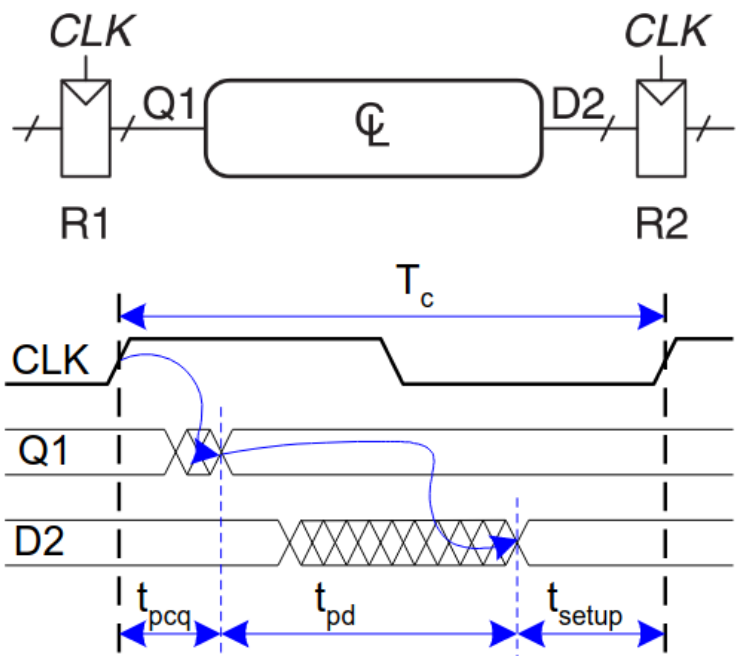


Figure: Sequential circuit flip-flop timing

Setup time and design performance

- Overall design performance is determined by the critical path t_{pd}
- Determines the minimum clock period (i.e., max operating frequency)
- If the critical path is too long, the design will run slowly
- if the critical path is too short, each cycle will do very little useful work, i.e., most of the cycle will be wasted in sequencing overhead

Hold Time Constraint

- Depends on the minimum delay from R1 to R2
- The input to R2 must be stable for at least t_hold after the clock edge

$$t_{hold} < t_{ccq} + t_{cd} \tag{2}$$

$$t_{cd} > t_{hold} - t_{ccq} \tag{3}$$

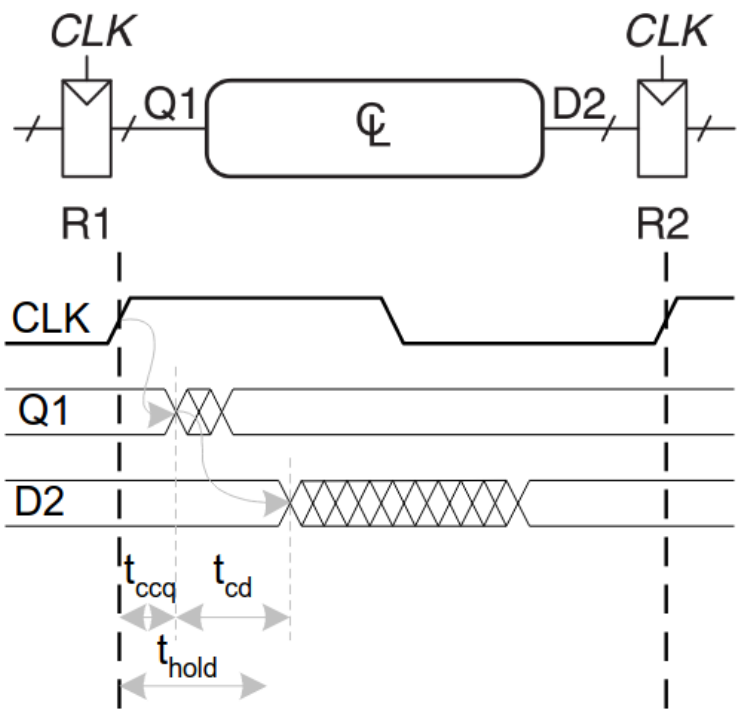
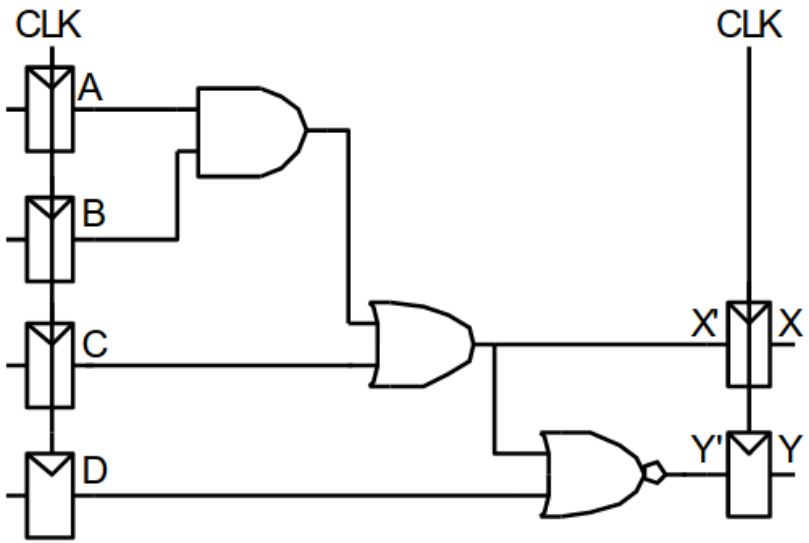


Figure: Hold time constraint

Timing constraint



Timing Characteristics

t_{ccq} = 30 ps

t_{pcq} = 50 ps

t_{setup} = 60 ps

t_{hold} = 70 ps

per gate:

t_{pd} = 35 ps

t_{cd} = 25 ps

t_{pd} = 3 x 35 ps = 105 ps

t_{cd} = 25 ps

Check setup time constraints:

$T_c \geq (50 + 105 + 60)$ ps = 215 ps

$f_{max} = 1/T_c$ = 4.65 GHz

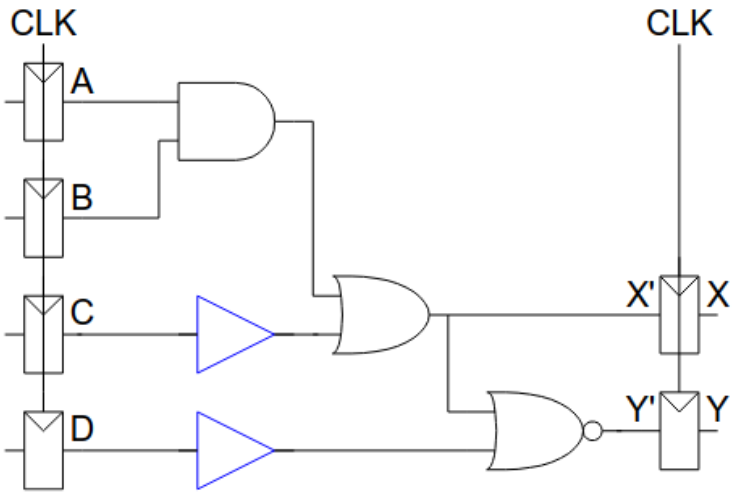
Check hold time constraint:

$t_{ccq} + t_{cd} > t_{hold}$?

(30 + 25) ps > 70 ps ?

Timing constraint

Add buffers to the short paths:



t_{pd} = 3 x 35 ps = 105 ps

t_{cd} = 2 x 25 ps = 50 ps

Check setup time constraints:

$T_c \geq (50 + 105 + 60)$ ps = 215 ps

$f_c = 1/T_c$ = 4.65 GHz

Timing Characteristics

t_{ccq}	= 30 ps
t_{pcq}	= 50 ps
t_{setup}	= 60 ps
t_{hold}	= 70 ps

per gate

t_{pd}	= 35 ps
t_{cd}	= 25 ps

Check hold time constraint:

$t_{ccq} + t_{cd} > t_{hold}$?

(30 + 50) ps > 70 ps ?

Clock skew

- To make matters worse, clocks have delays too!
- The clock does not reach all parts of the chip simultaneously!
- TL;DR: Reality sucks.

Definition

Clock skew: the time difference between two clock edges

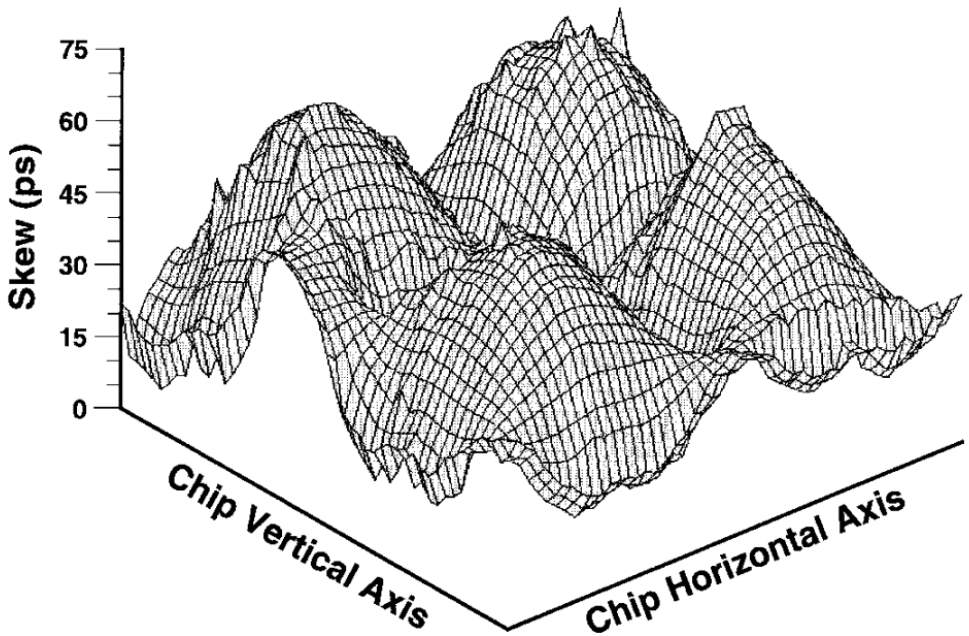


Figure: Skew in a processor - Alpha 21264

Clock Jitter

- The definition of clock skew we used considers both the spatial skew and the jitter, i.e., the timing deviation from true periodicity.
- We chose this definition to simplify the description. And consider one variable as the worst of the worst-case scenario.

Revised setup timing with Clock Skew

- Safe timing requires considering the worst-case skew
  - Clock arrives at R2 before R1
  - Leaves as little time as possible for the combinational logic

$$T_c \geq t_{pcq} + t_{pd} + t_{setup} + t_{skew} \tag{4}$$

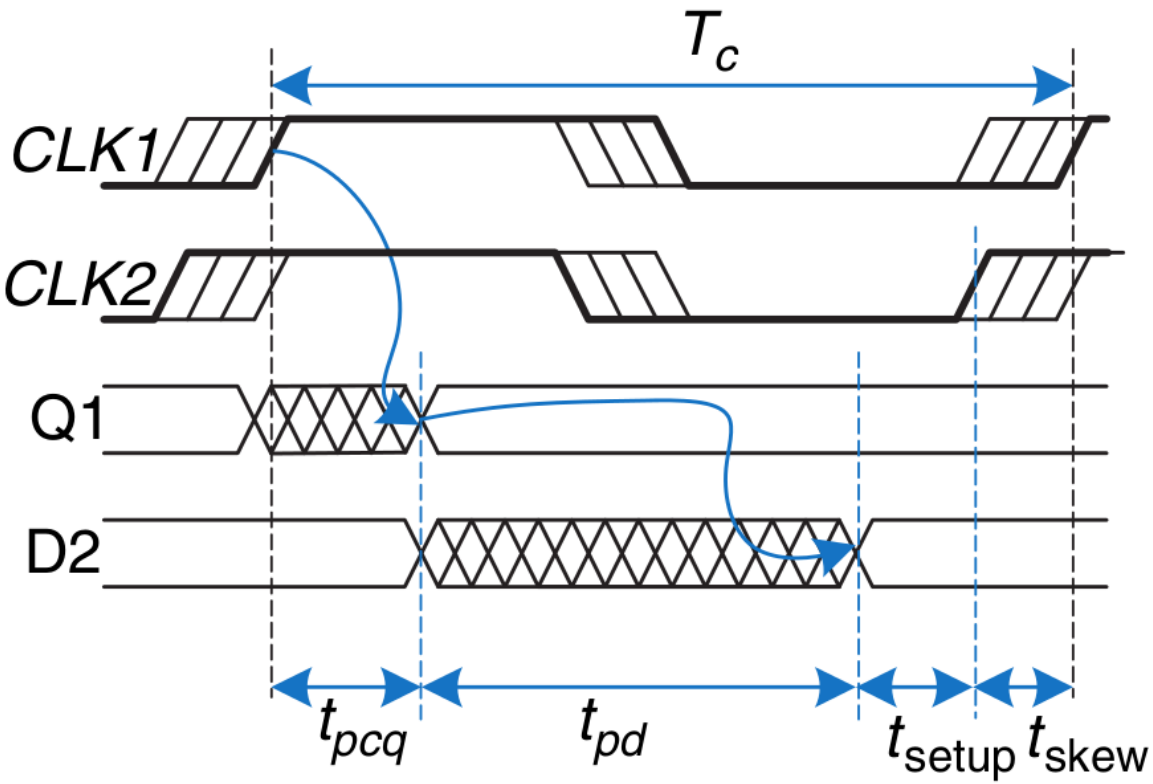


Figure: Caption

Revised hold timing with Clock Skew

- Safe timing requires considering the worst-case skew
  - Clock arrives at R2 after R1
  - Increases the minimum required delay for the combinational logic

$$t_{cd} \geq -t_{ccq} + t_{hold} + t_{skew} \tag{5}$$

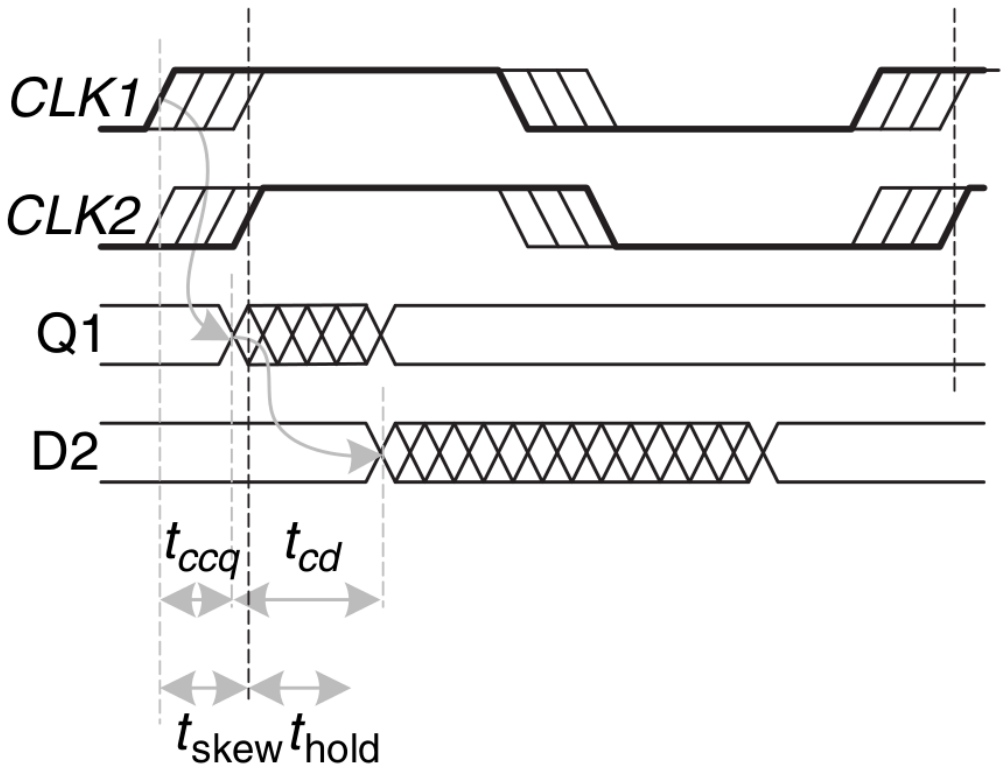


Figure: Caption

Outline

Does it work?

There are many levels of "it works":

- You have designed a circuit! Is it functionally correct? Does the hardware meet all the timing constraints, even if it is logically correct?
- How can you test for functionality and timing?
- Answer: simulation tools! For everything, more or less.

Note: aim for something higher than "It just works."

Testing

Testing can be the most time-consuming design stage

- Functional correctness of all logic paths
- Timing, power, etc. of all circuit elements

Unfortunately, low-level (e.g., circuit) simulation is much slower than high-level (e.g., HDL, C) simulation.
Solution: splitting responsibilities.

1. Check only functionality at a high level (e.g., C, HDL)
   - (Relatively) fast simulation time allows high code coverage (remember, this is very TDD-like).
   - Easy to write and run tests
2. Check only timing, power, etc., at low level (e.g., circuit)
   - No functional testing of low-level model
   - Instead, test functional equivalence to high-level model
   - Hard, but easier than testing logical functionality at this level

Functional Verification - Test-bench

- Testbench: a module created specifically to test a design
- Tested design is called the “Unit under test (UUT)” or "Device Under Test."
- A Testbench provides inputs (called test patterns) to the UUT
  - Hand-crafted values
  - Automatically generated (e.g., sequential or random values)
- A testbench checks outputs of the UUT against
  - Hand-crafted values
  - A “golden design” that is known to be bug-free

Tiiming Verification

High-level simulation (e.g., C, VHDL)

- Useful for hierarchical modeling
- Insert delays in FFs, basic gates, memories, etc.
- High-level design will have some notion of timing
- Usually not as accurate as real circuit timing

Circuit-level timing verification

- Needed to synthesize your design to actual circuits
- No one general approach: it is very design flow specific. FPGA/ASIC/etc. has special tools: Vivado/Quartus for FPGA, Cadence for ASIC/VLSI.

Tools

- Tools exist, and they work (most of the time)
- When they fail, place-and-route and manual optimization (many fun, very lol).

Conclusion

Enough for today.
And it doesn't make noises.