EITF35: Introduction to Structured VLSI Design

Part 2.2.1: Sequential circuit

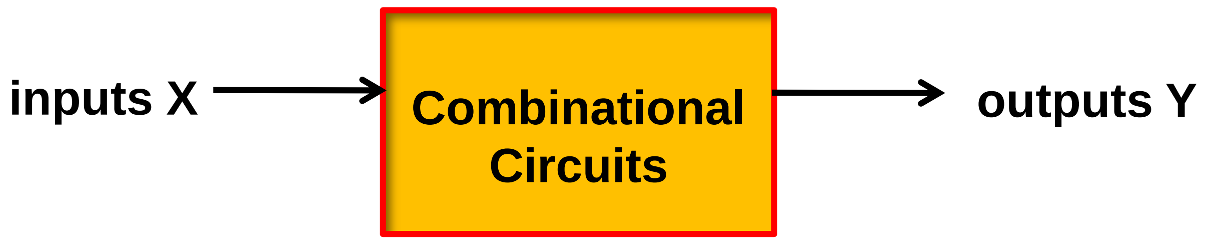
Liang Liu
liang.liu@eit.lth.se

Outline

- Sequential vs. Combinational
- Synchronous vs. Asynchronous
- Basic Storage Elements
- Timing
- Folding & Pipeline

Sequential vs. Combinational

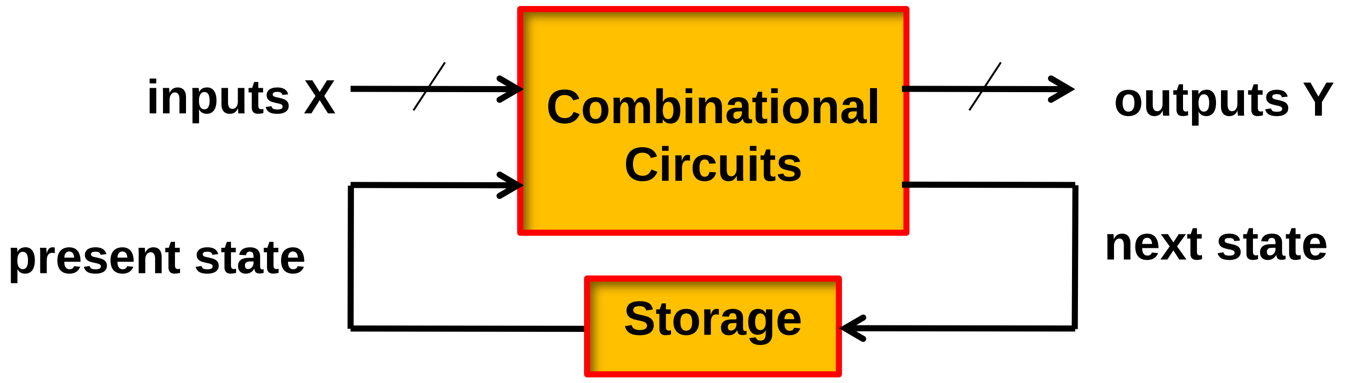
- A combinational circuit:



- At any time, outputs depend only on present inputs
  - Changing inputs changes outputs
- No regard for previous inputs
  - No memory (history)
- Time is "ignored" !
  - Time-independent circuit

Sequential vs. Combinational

- **A sequential circuit:**

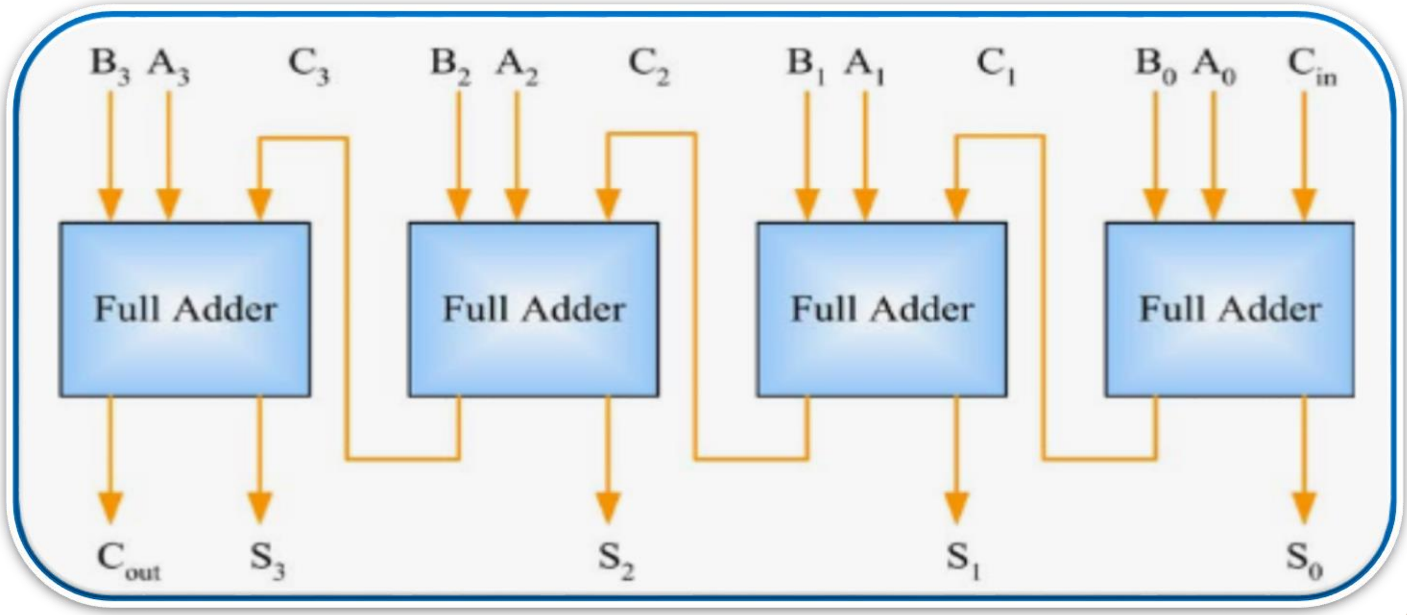


- **Outputs depends on inputs and past history of inputs**
  - Previous inputs can be stored into **storage elements**
  - **Input order matters**

Sequential vs. Combinational: adders

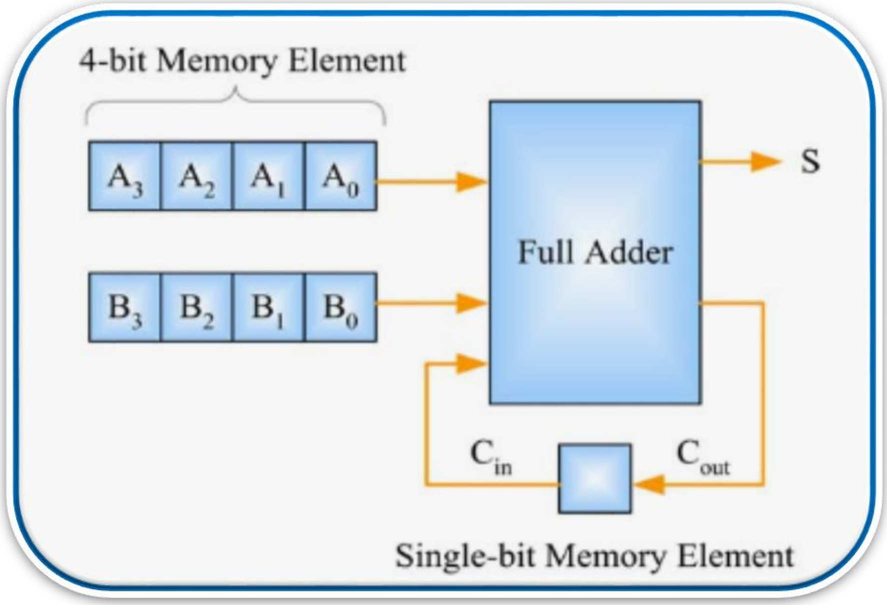
- **Calculate** $A_3A_2A_1A_0 + B_3B_2B_1B_0$

- **Combinational adder**
  - 4 full adders are required
  - One adder is active at a time slot



What we can do with storage elements?

- **Sequential Adder**



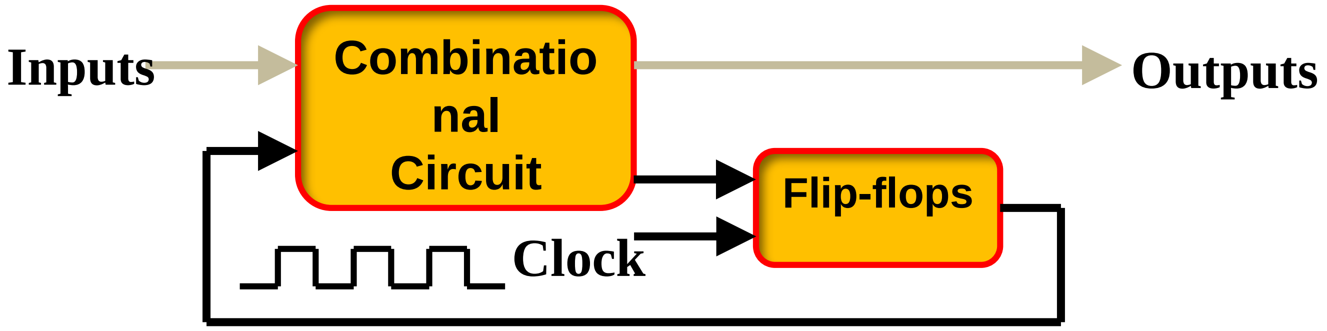
- **Folding!**
  - One full adder
  - 1-bit memory for carry
  - Two 4-bit memory for operators
- **4 clock cycles to get the output**

Outline

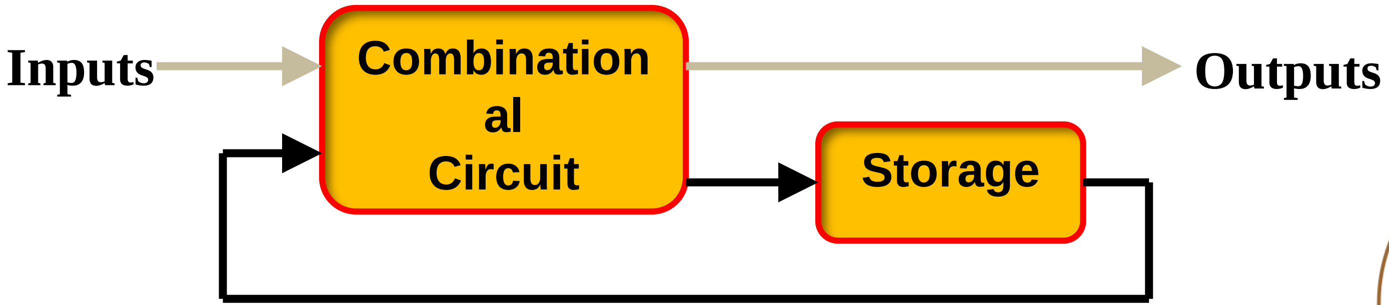
- Sequential vs. Combinational
- Synchronous vs. Asynchronous
- Basic Storage Elements
- Timing
- Folding & Pipeline

Synchronous vs. Asynchronous

- **Two types of sequential circuits:**
  - Synchronous: The behavior of the circuit depends on the input signal at discrete instances of time (also called **clocked**)

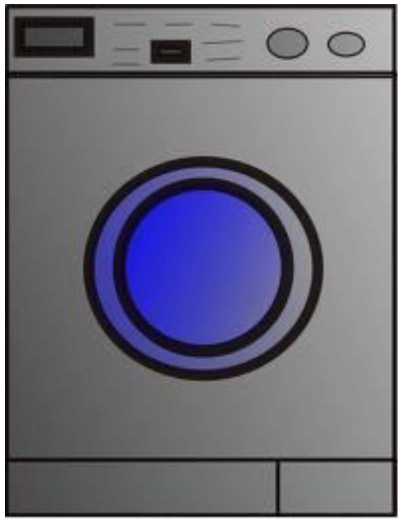


  - Asynchronous: The behavior of the circuit depends on the input signals at ***any instance of time***



Synchronous vs. Asynchronous

- **When you have a clock**
- **You know that washer takes 1 hour**
- **You put the laundry in the washer and leave**
- **Dry 1hour later**



Synchronous vs. Asynchronous

- What if you don't have a clock …

Synchronous or Asynchronous?

- **Sync. Advantages: Simplicity to design, debug, and test**
  - Timing is controlled by one simple clock
  - No hand-shake circuits
  - Well supported by EDA tools
  - Recommended for **VLSI**
- **Sync. Disadvantages:**
  - Performance constrained by worst-case: critical path
  - Overhead for clock network
  - Less power efficient

We will focus on synchronous circuits in this course

Power Example

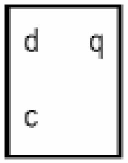
```
                   Internal  Switching  Leakage    Total
Power Group          Power     Power     Power     Power  (   %)  Attrs
--------------------------------------------------------------------
io_pad             0.0000    0.0000    0.0000    0.0000 ( 0.00%)
memory             0.0000    0.0000    0.0000    0.0000 ( 0.00%)
black_box          0.0000    0.0000    0.0000    0.0000 ( 0.00%)
clock_network      0.0137 4.982e-03 3.116e-05    0.0187 (13.76%)
register        3.029e-03 1.298e-03 8.082e-04 5.136e-03 ( 3.79%)
combinational      0.0518    0.0557 4.337e-03    0.1118 (82.45%)
```

Outline

- Sequential vs. Combinational
- Synchronous vs. Asynchronous
- **Basic Storage Elements**
- Timing
- Folding & Pipeline

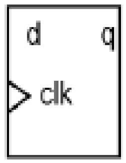
Basic storage element

- D latch: level sensitive
- D flip-flop (D-FF): edge sensitive



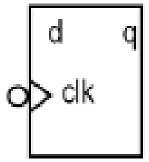
c	q*
0	q
1	d

D latch



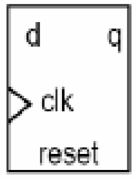
clk	q*
0	q
1	q
↑	d

pos-edge triggered D-FF



clk	q*
0	q
1	q
↓	d

neg-edge triggered D-FF



reset	clk	q*
1	-	0
0	0	q
0	1	q
0	↑	d

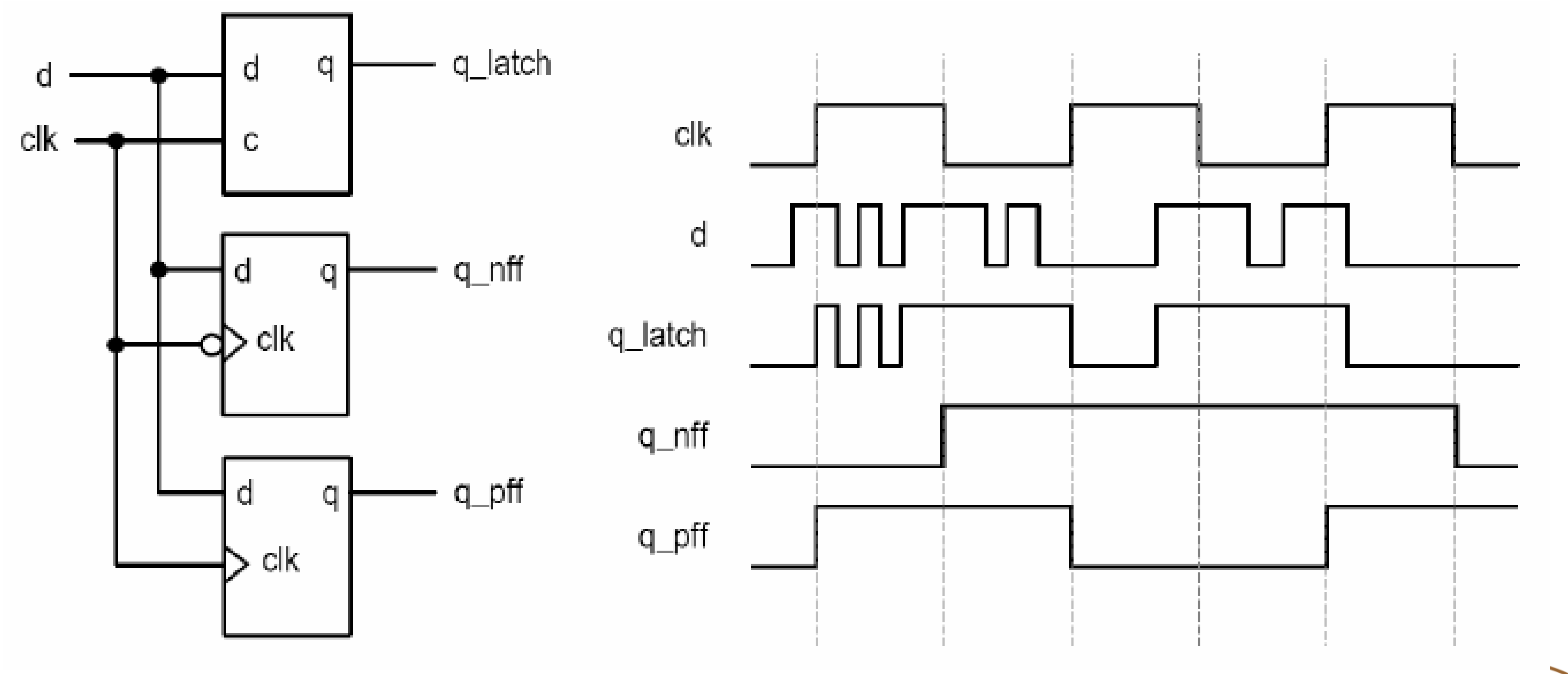
D-FF with reset

Why Reset?

- **Initial State**
- **Some Hints**
  - Efficient sync. design for complicated system
  - The importance of sync initial state
  - A good clock is crucial
  - No timing violation

Basic storage element

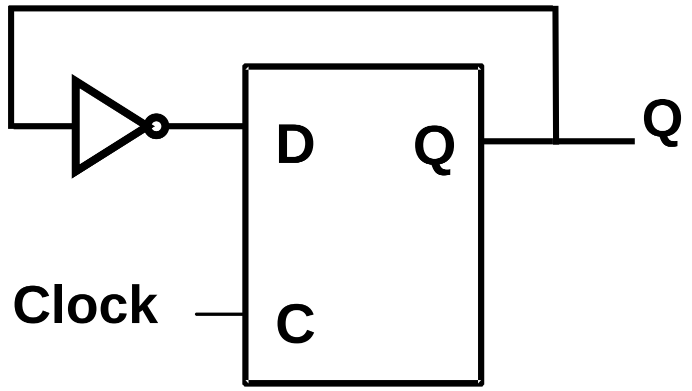
- D latch: level sensitive
- D flip-flop (D-FF): edge sensitive



Problem with Latches

- Problem: A latch is transparent; state keep changing as long as the clock remains active
- Due to this uncertainty, latches can not be reliably used as storage elements.
- What happens if Clock=1? What will be the value of Q when Clock goes to 0?

Latch Example



Most EDA software tools have difficulty with latches.

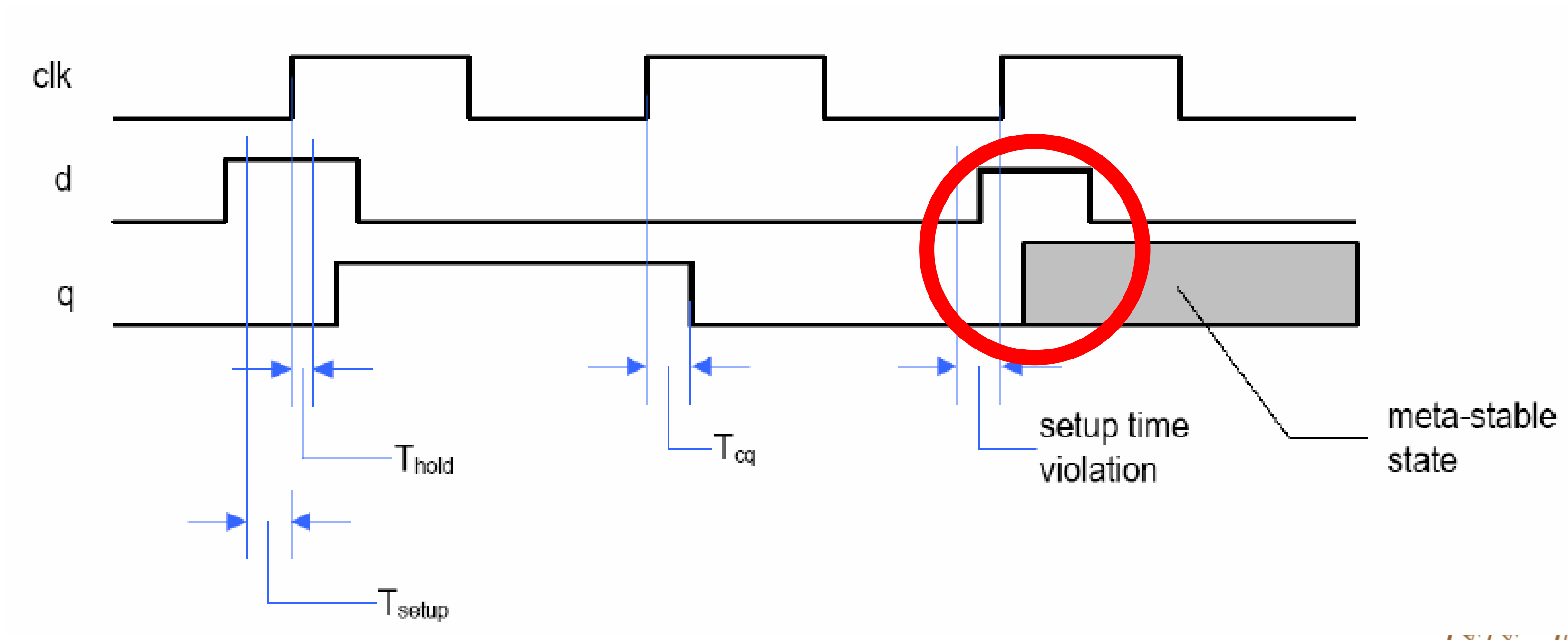
Outline

- Sequential vs. Combinational
- Synchronous vs. Asynchronous
- Basic Storage Elements
- Timing
- Folding & Pipeline

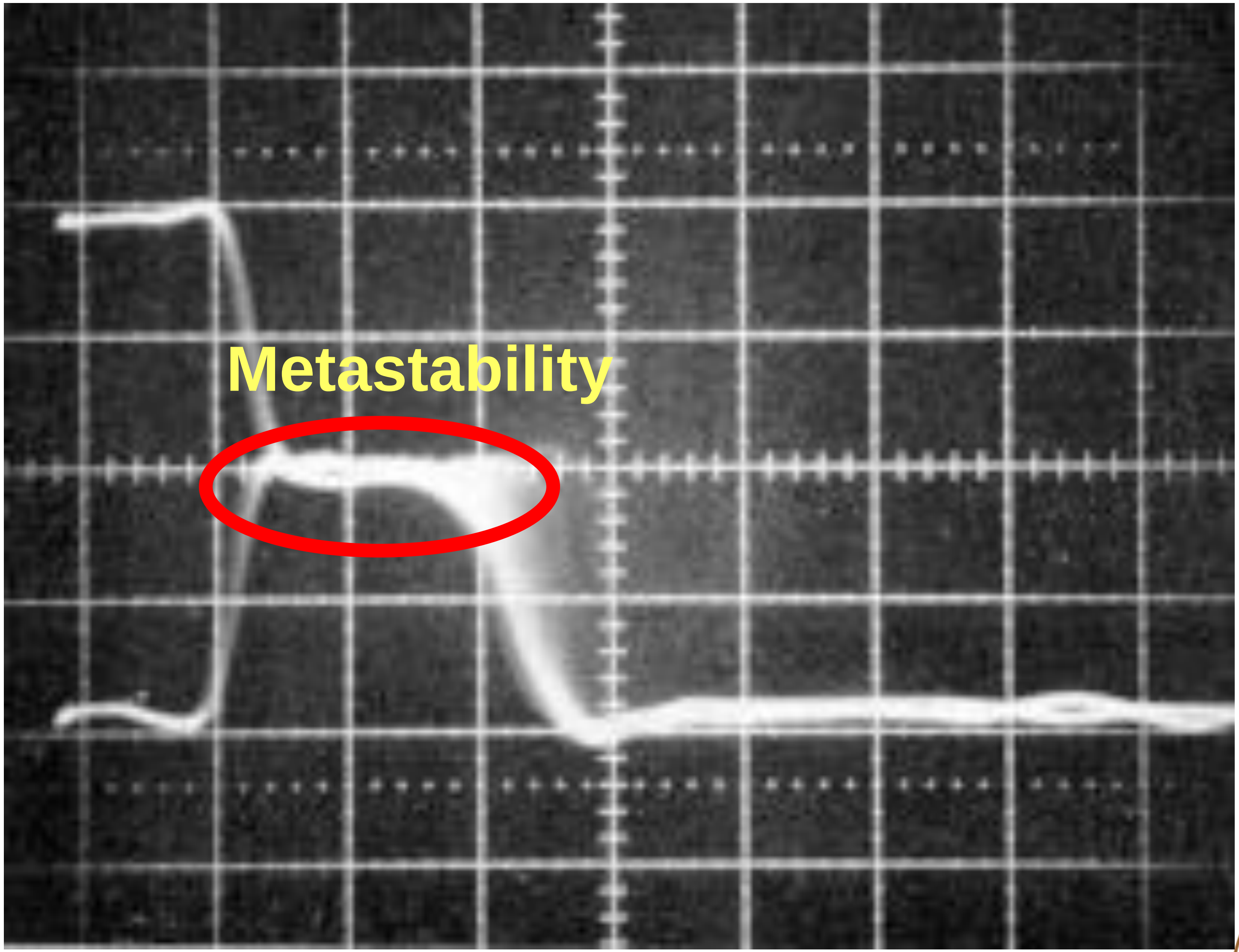
Flip Flops Timing

Very Important Timing Considerations!

- ***Setup Time*** **(Ts): The minimum time during which D input must be maintained before the clock transition occurs.**
- ***Hold Time*** **(Th): The minimum time during which D input must not be changed after the clock transition occurs.**

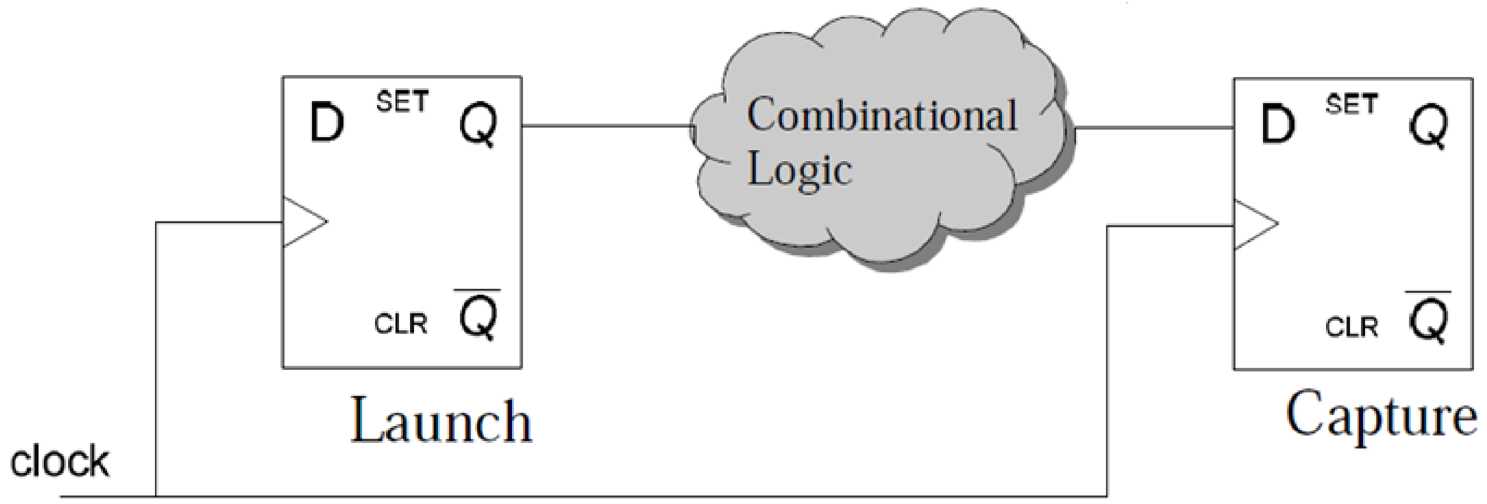


Metastability in Digital Logic



How fast can a synchronous circuit run?

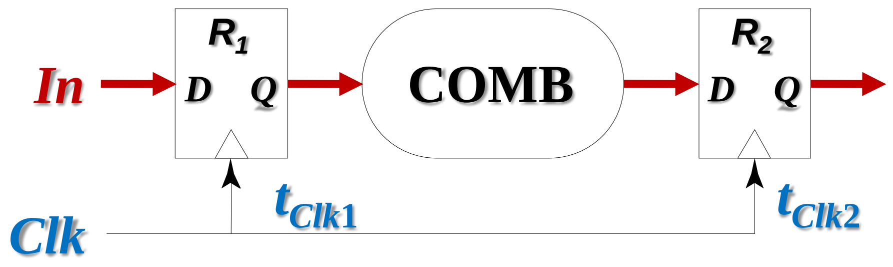
- **RTL (Register Transfer Level)**



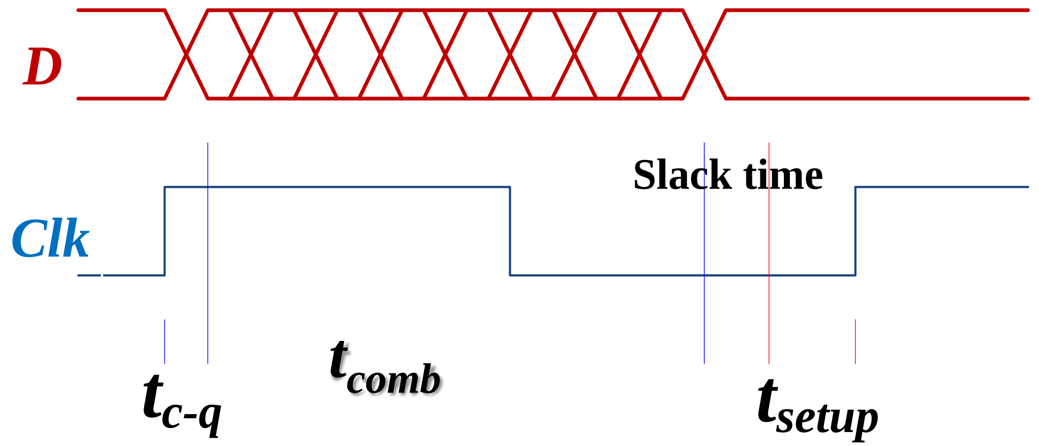
- **Timing analysis:**
  - Starting with the clock rising edge at the launch FF, end with the clock rising edge (next period or same period) of the capture FF

Setup Time

- **Setup Timing analysis:**
  - Starting with the clock rising edge at the launch FF, end with the clock rising edge (next period) of the capture FF

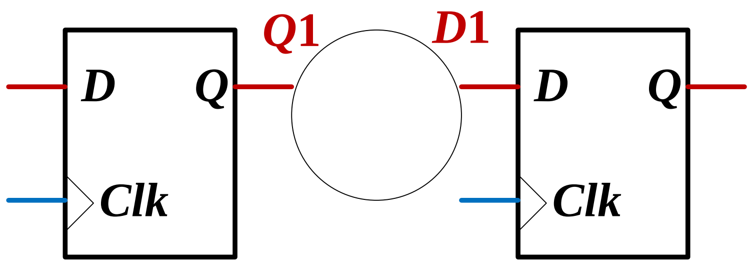


- **Data-Path (arrive time):** $T_{\text{Combinational logic}} + FF_{\text{launch}}(\text{clk} \to Q)$
- **Clock-Path (required time):** Clock Period - FF_{tSetup}
- **Timing constraint :** $T_{\text{Combinational logic}} + FF_{\text{launch}}(\text{clk} \to Q) < \text{Clock Period} - FF_{\text{tSetup}}$

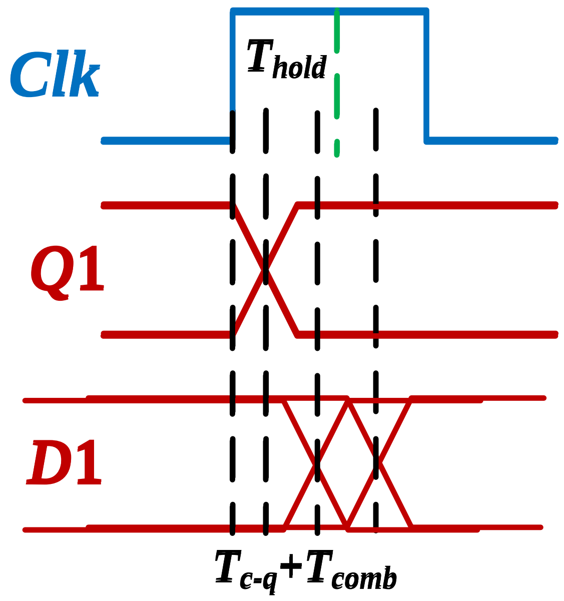


Hold Time

- **Hold Timing analysis:**
  - Starting with the clock rising edge at the launch FF, end with the clock rising edge (same period) of the capture FF

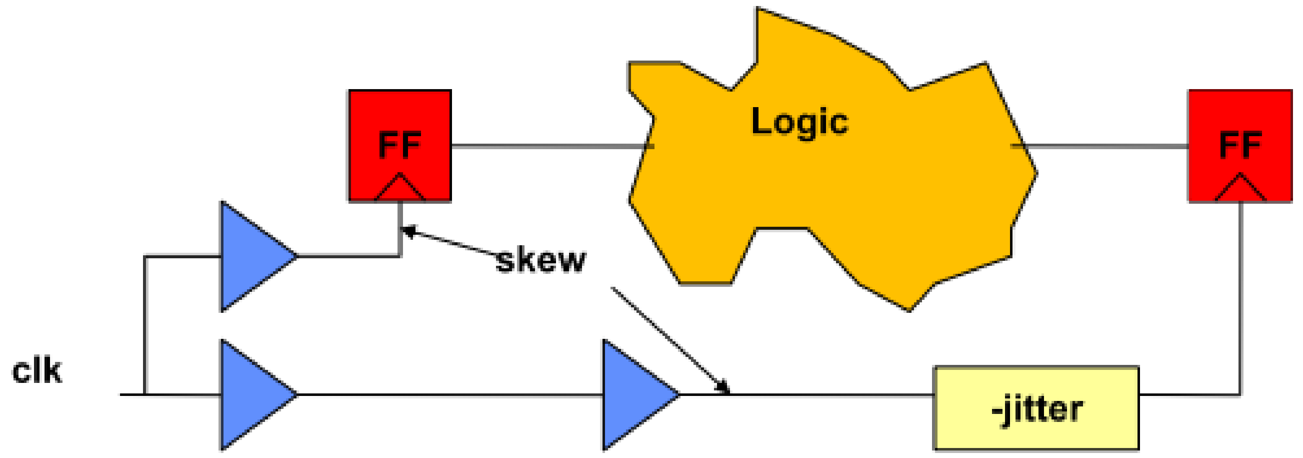


- **Data-Path (arrive time):** $T_{\text{Combinational logic}}$ + FF_{launch}(clk -> Q)
- **Clock-Path (required time):** FF_{tHold}
- **Timing constraint :** $T_{\text{Combinational logic}}$ + FF_{launch}(clk -> Q)> FF_{tHold}



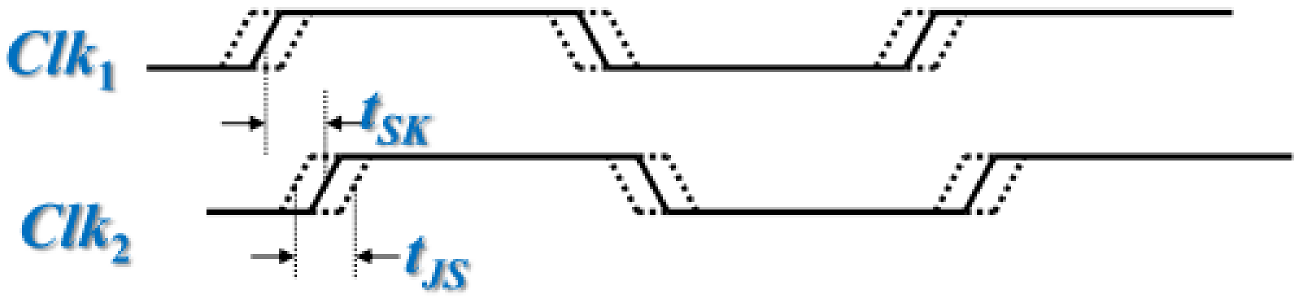
Clock uncertainty

- Clock uncertainty = skew ±jitter
- Clock skew
  - The (knowable) difference in clock arrival times at each flip-flop
  - Caused mainly by imperfect balancing of clock tree/mesh
- Clock jitter
  - The random (unknowable) difference in clock arrival times at each flip-flop
  - Caused by on-die process, V_{dd}, temperature variation, PLL jitter, crosstalk.
- Clock tree to minimize clock uncertainty

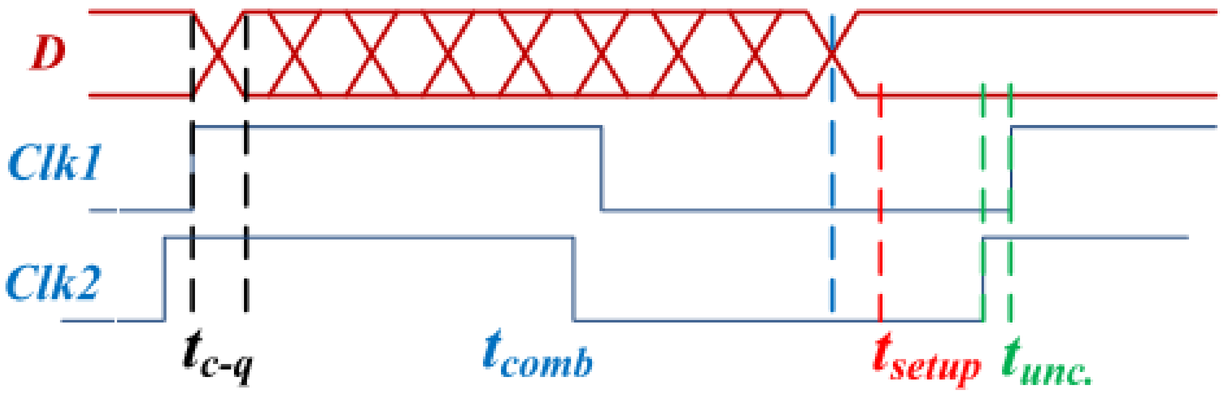


Clock uncertainty

- Clock uncertainty = skew ±jitter

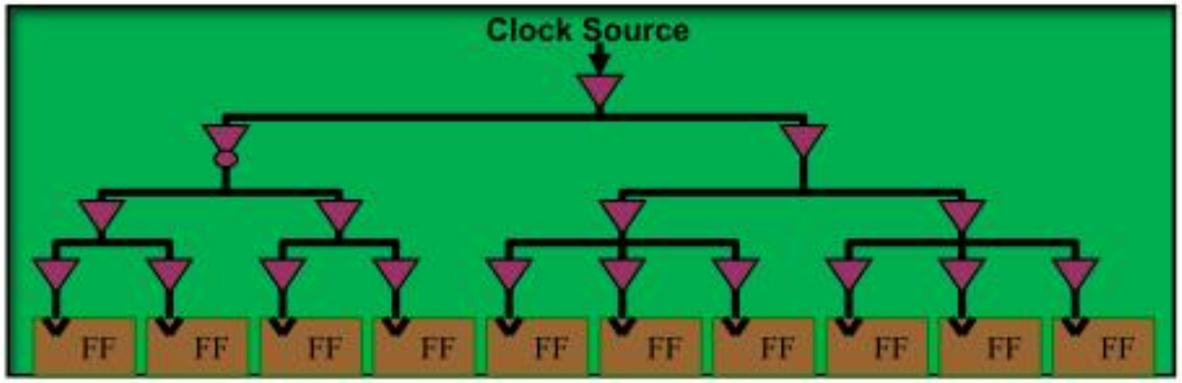


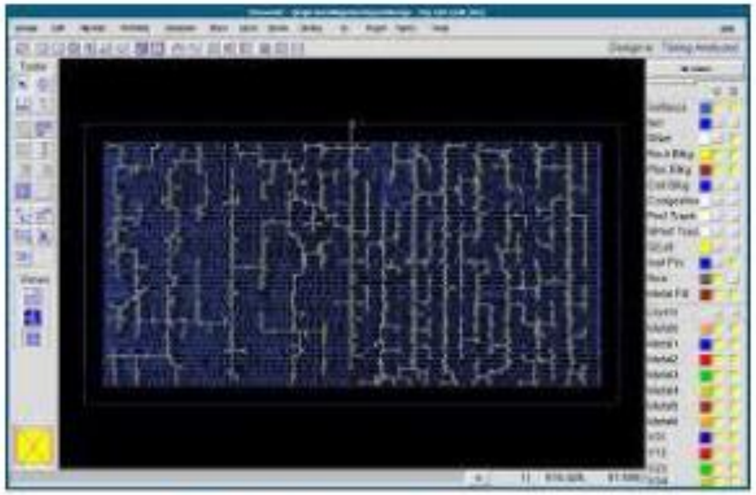
- Timing analysis with clock uncertainty



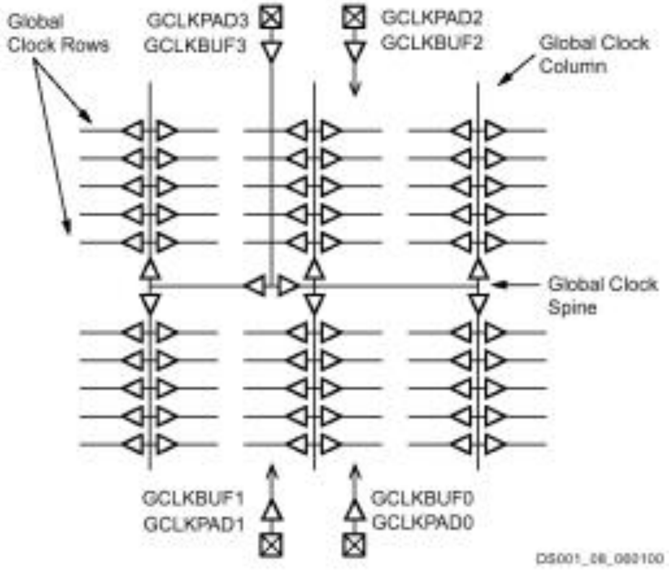
$T_{\text{Combinational logic}}$ + FF_{lauch}(clk -> Q) < Clock Period - FF $_{\text{tSetup}}$ - Clock Uncertainty

Clock tree



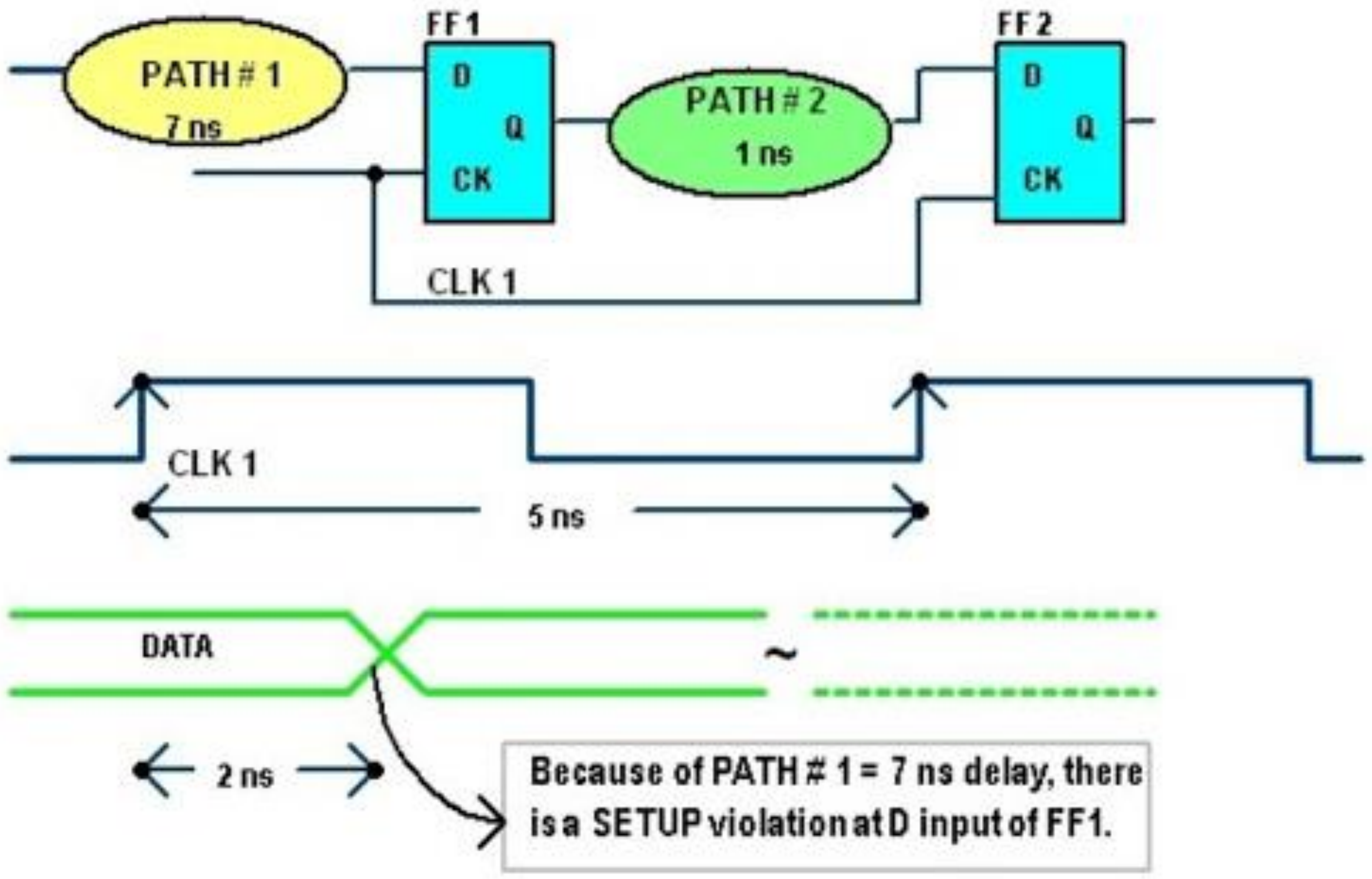


Clock-tree Asic



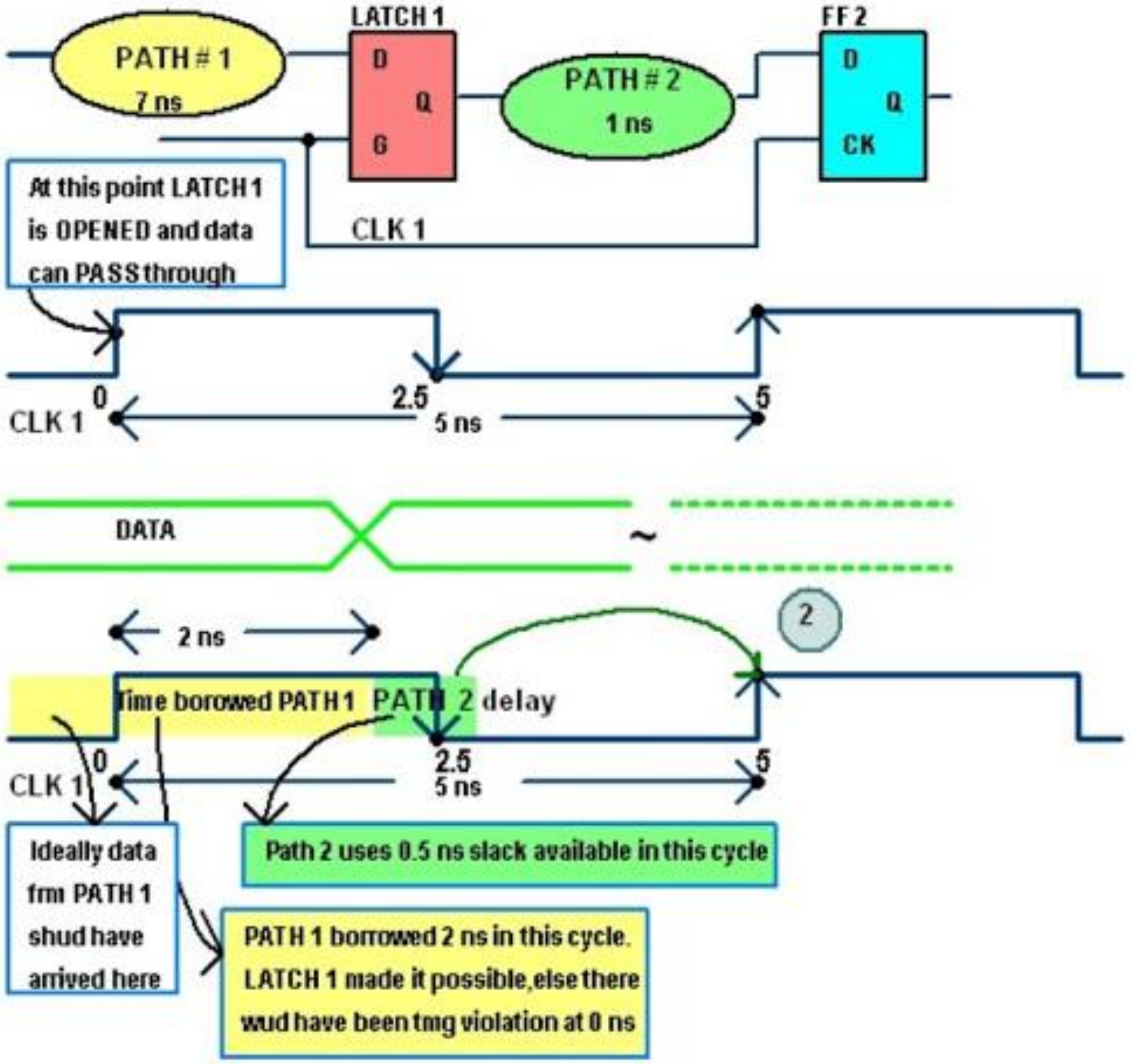
Global clock network in Xilinx FPGA

Time borrow



Because of PATH # 1 = 7 ns delay, there is a SETUP violation at D input of FF1.

Time borrow


LATCH 1
FF 2
PATH # 1
7 ns
D
Q
G
PATH # 2
1 ns
D
Q
CK
CLK 1
At this point LATCH 1 is OPENED and data can PASS through
CLK 1
0
2.5
5 ns
5
DATA
~
2 ns
2
Time borowed PATH 1
PATH 2 delay
CLK 1
0
2.5
5 ns
5
Ideally data frm PATH 1 shud have arrived here
Path 2 uses 0.5 ns slack available in this cycle
PATH 1 borrowed 2 ns in this cycle. LATCH 1 made it possible, else there wud have been tmg violation at 0 ns

Outline

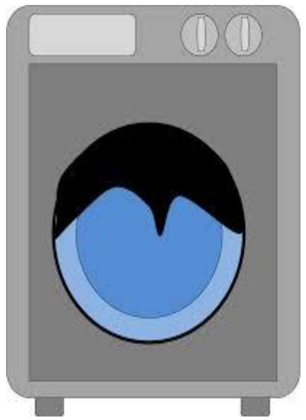
- ◻ Sequential vs. Combinational
- ◻ Synchronous vs. Asynchronous
- ◻ Basic Storage Elements
- ◻ Timing
- ◻ **Folding & Pipeline**

Pipeline

- **Acknowledgement:**
  - The following slides have been provided by Prof. Ward in September 2004.
  - Reformatting of PowerPoint and addition of two more slide done September 2007 by Jens Sparsø.
  - Slides are used in DTU course 02154 Digital Systems Engineering (fall 2008).
  - Due to Joachim Rodrigues' position at DTU, I used some of the slides in EITF35.

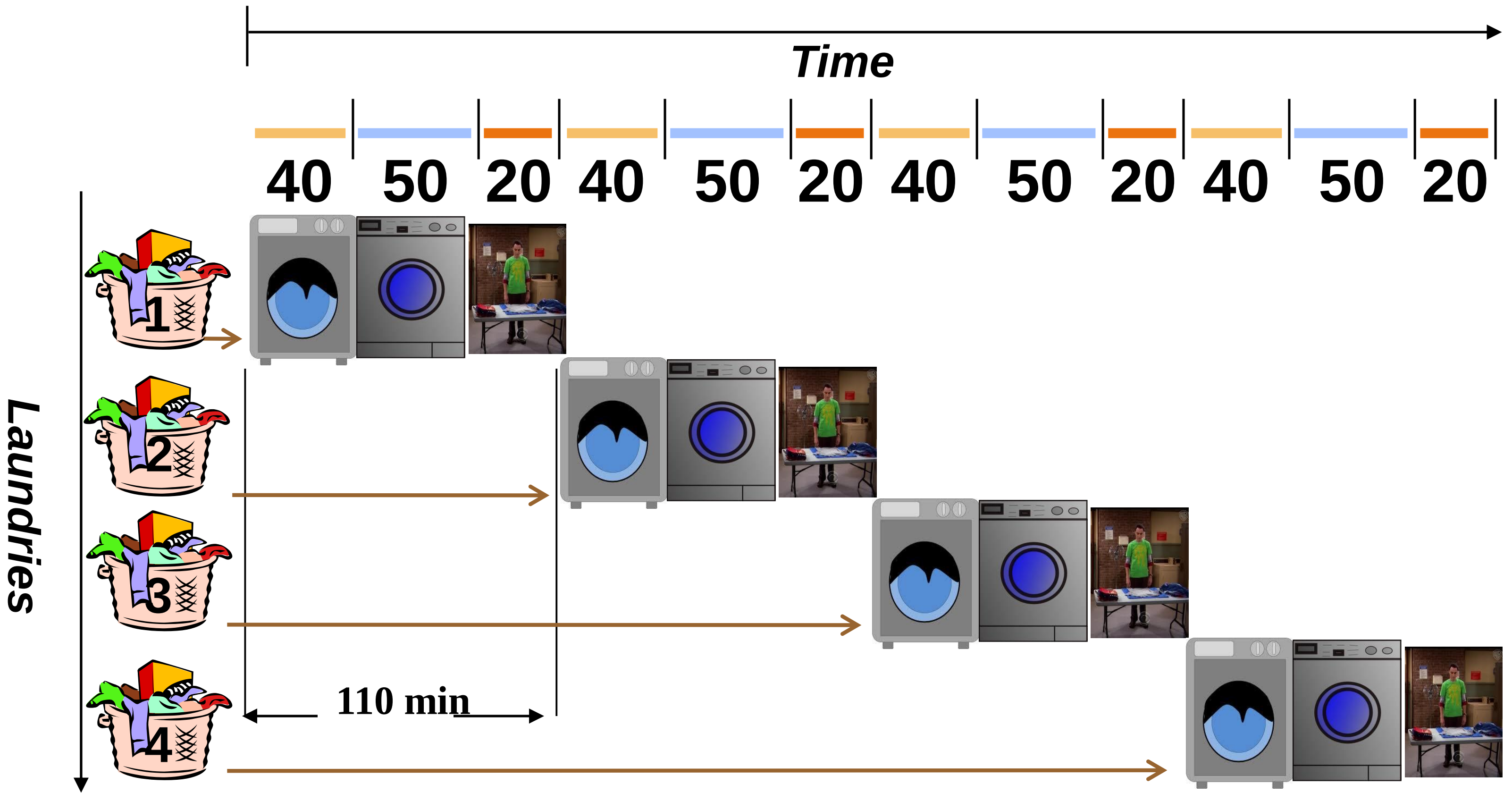
Pipelining

- Start again from laundry room



- Small laundry has one washer, one dryer and one folder, it takes 110 minutes to finish one load:
  - Washer takes 40 minutes
  - Dryer takes 50 minutes
  - "Folding" takes 20 minutes
- Need to do 4 laundries

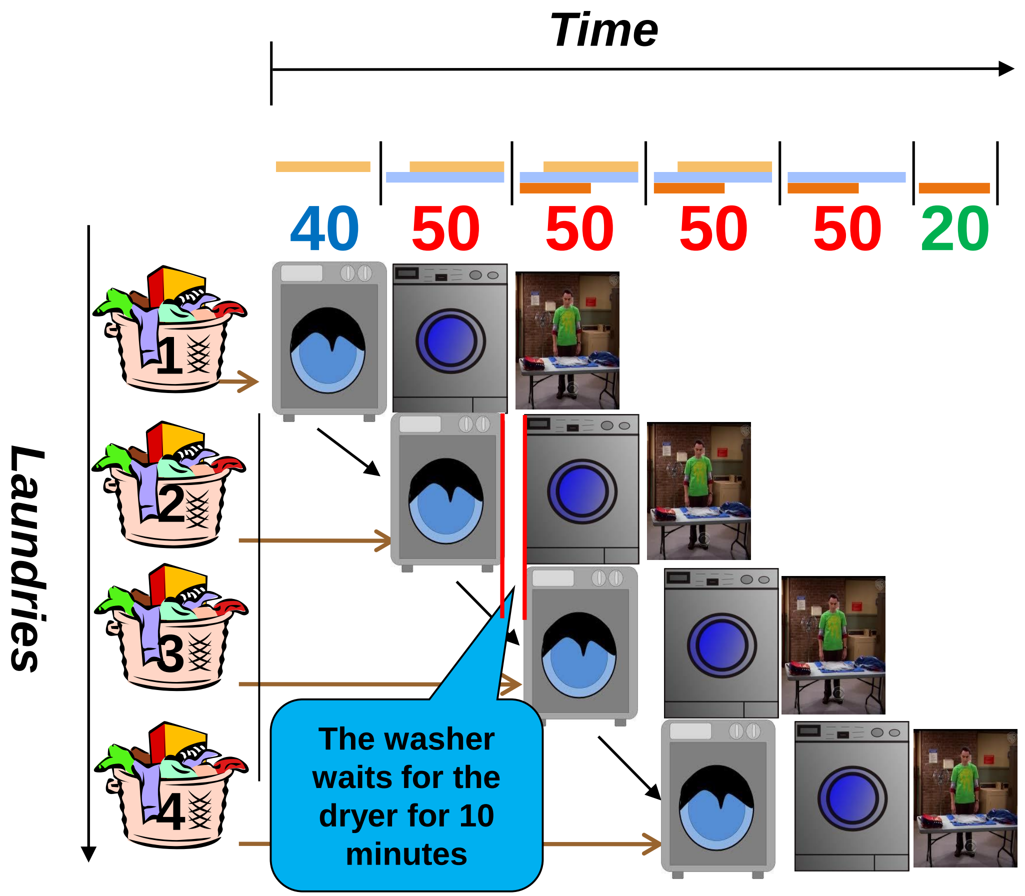
Not very smart way...



Total = N*(Washer+ Dryer+Folder)

= 440 mins

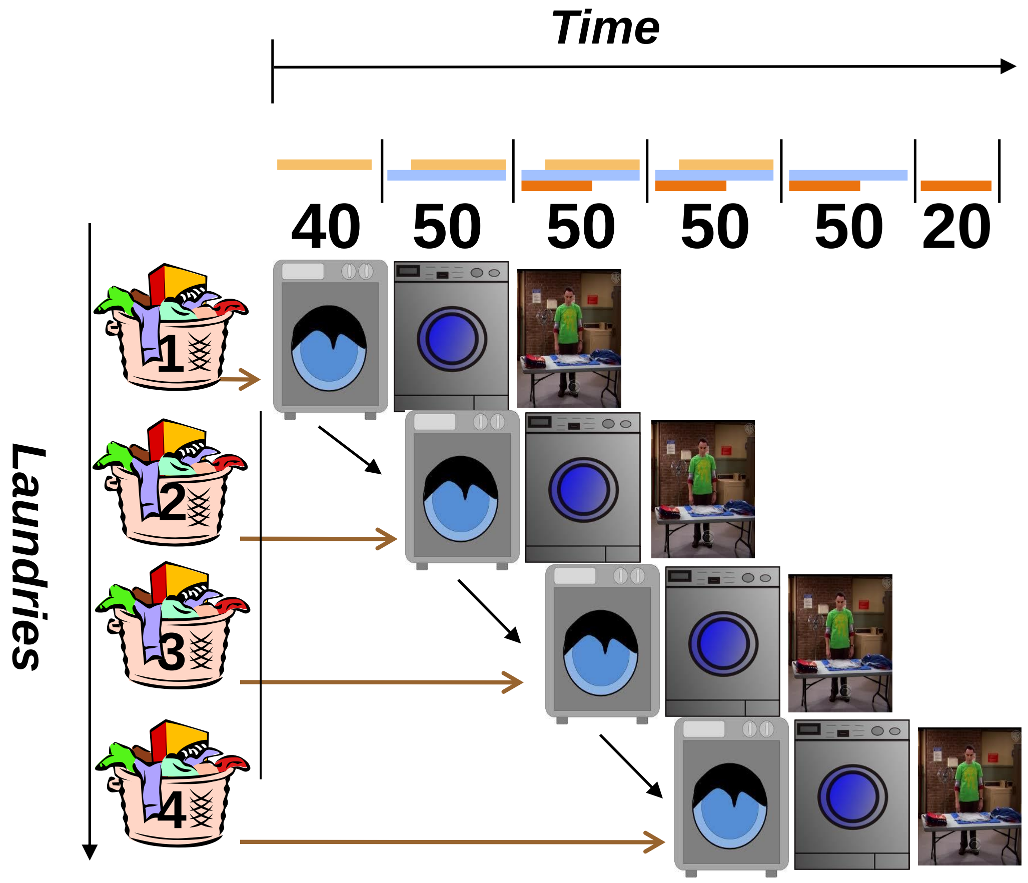
If we pipelining



Total = Washer+N*Max(Washer,Dryer,Folder)+Folder

= 260 mins

Pipeline Facts



- Multiple tasks operating simultaneously
- Pipelining doesn't help latency of single task, it helps throughput of entire workload
- Pipeline rate limited by slowest pipeline stage
- Unbalanced lengths of pipe stages reduces speedup
- Potential speedup ∝ Number of pipe stages

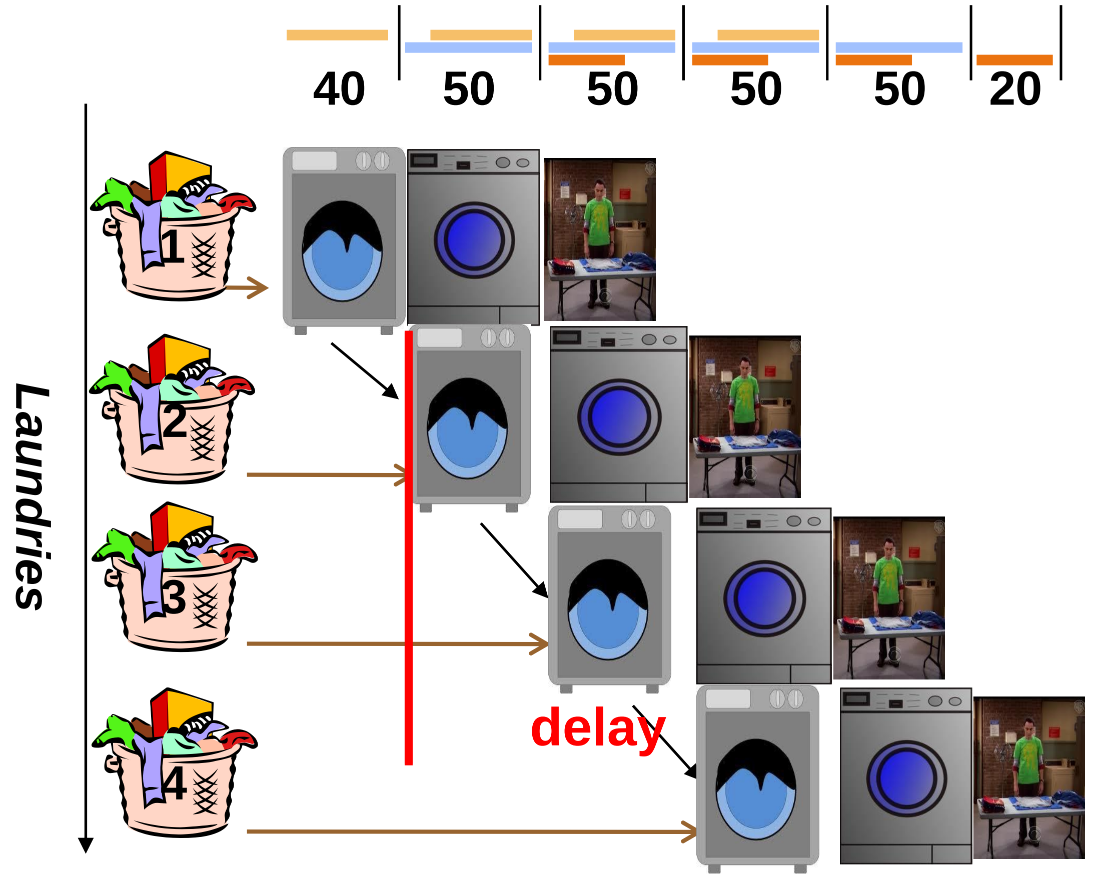
Some definitions

Very Important!

- **Latency: The delay from when an input is established until the output associated with that input becomes valid.**

  (non-pipeline Laundry = ___**110**___ *mins)*

  (pipeline Laundry = ___**120**___ *mins)*



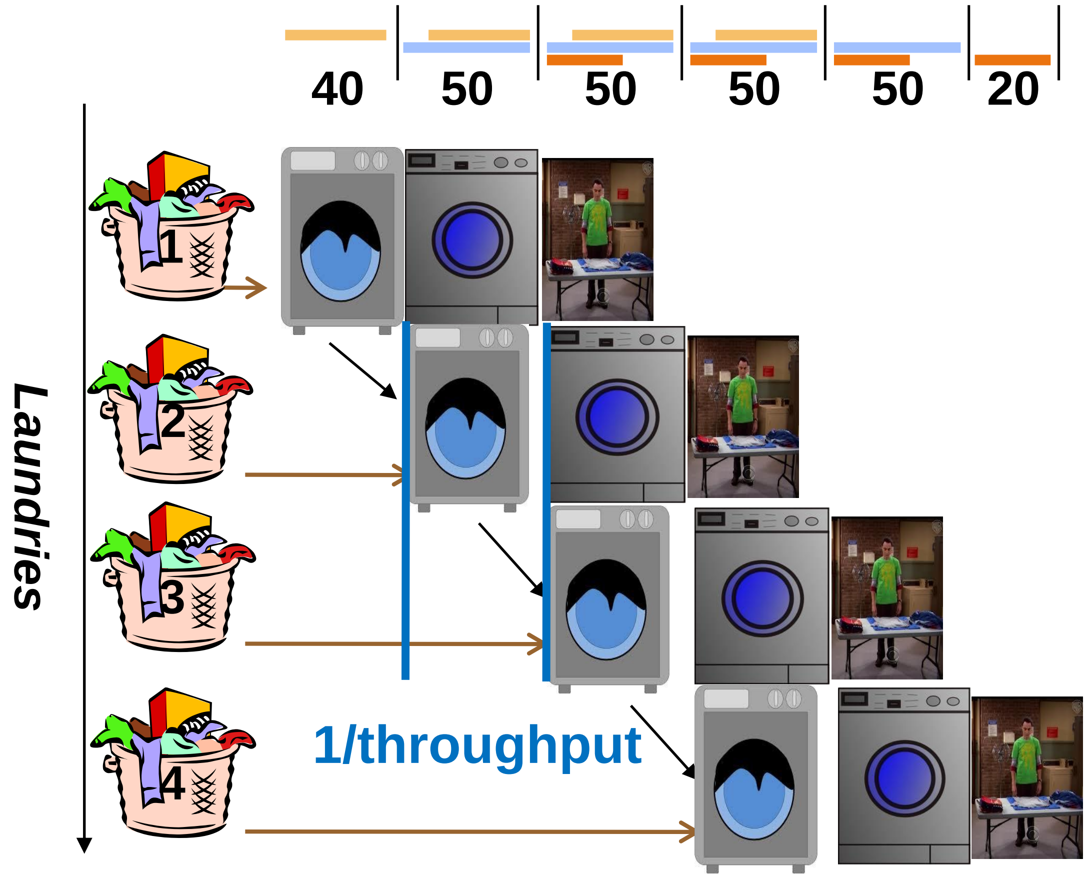
Some definitions

Very Important!

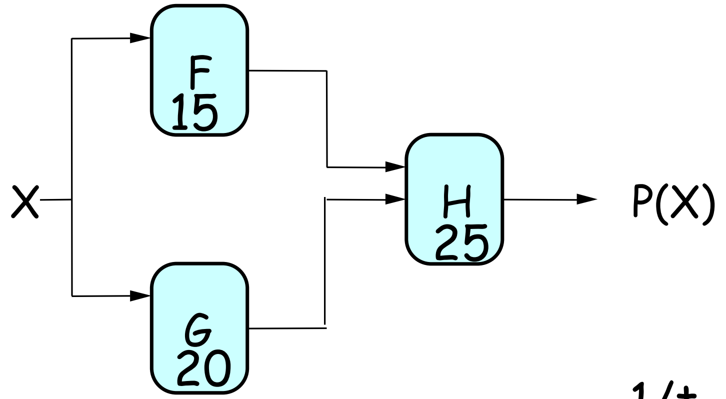
- **Throughput: The rate of which inputs or outputs are processed or how frequently a laundry can be loaded**

  (non-pipeline Laundry = **1/110** *outputs/min)*

  (pipeline Laundry = **1/50** *outputs/min)*

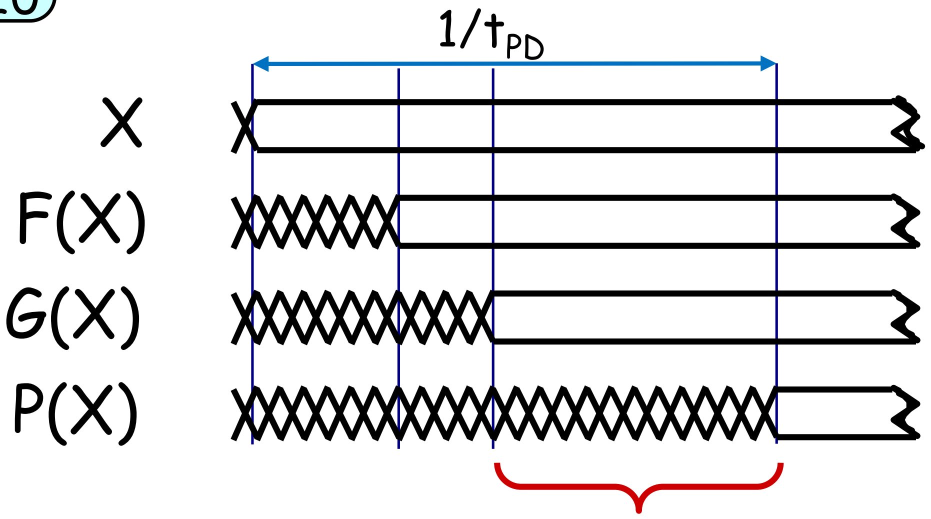


Okay, back to circuits…



- **Combinational logic:**
  latency = t_{PD},
  throughput = $1/t_{PD}$.

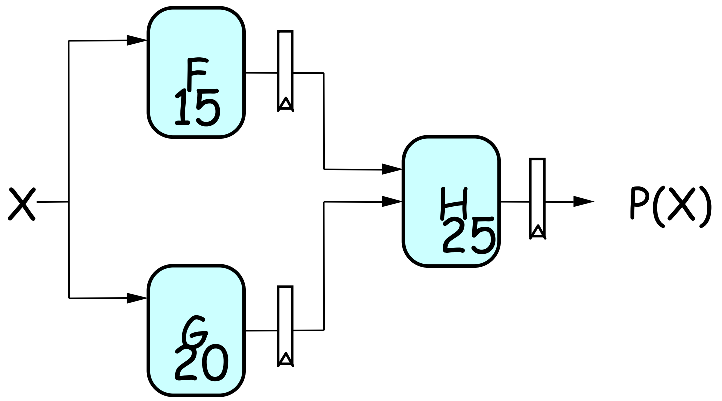
- **Can we use the hardware more efficiently?**



F & G are "idle", just holding their outputs stable while H performs its computation

Pipelined Circuits

use registers to hold
H' s input stable!

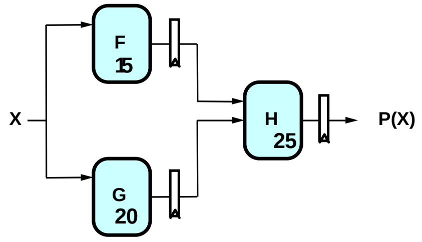


- **Pipelined circuit:**
  - 2-stage *pipeline*: if we have a valid input X during clock cycle j, P(X) is valid during clock j+2.
  - Now F & G can be working on input X_{i+1} while H is performing its computation on X_i.

Suppose F, G, H have propagation delays of 15, 20, 25 ns and we are using ideal zero-delay registers:

	latency	throughput
un-pipelined	45	1/45
2-stage pipelined	50	1/25
	worse	better

Pipeline timing diagrams



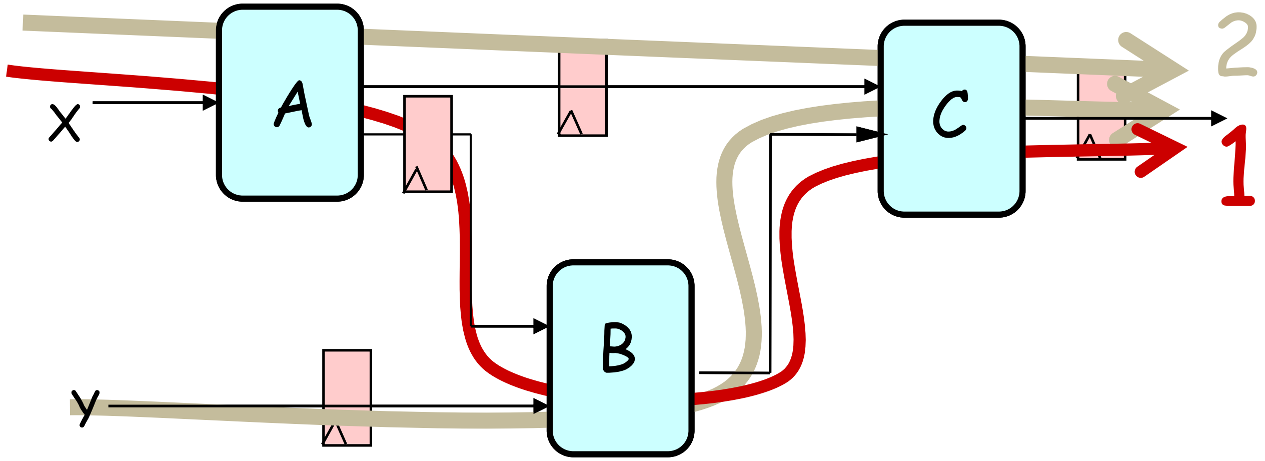
Clock cycle →

Pipeline stages ↓

	i	i+1	i+2	i+3	
Input	X_i	X_{i+1}	X_{i+2}	X_{i+3}	…
F Reg		$F(X_i)$	$F(X_{i+1})$	$F(X_{i+2})$	…
G Reg		$G(X_i)$	$G(X_{i+1})$	$G(X_{i+2})$	
H Reg			$H(X_i)$	$H(X_{i+1})$	$H(X_{i+2})$

Ill-formed pipelines

Consider a BAD job of pipelining:



Problem:

Some paths from inputs to outputs had 2 registers, and some had only 1!

Make sure every paths have been pipelined with same stages

Combinational, Folding and Pipelined

- **Combinational Circuits**
  - Advantage: low latency
  - Disadvantage: low throughput, more hardware, low utilization

- **Folding**
  - Advantage: less hardware, high utilization
  - Disadvantage: high latency, limited application

- **Pipeline**
  - Advantage: very high throughput
  - Disadvantages: pipeline latency, more hardware

Thanks!