Hardware Modeling [VU] (191.011)
– WS25 –

Synchronous Design Style

Guest Lecture by Prof. Steininger

WS 2025/26

Modified: 2025-12-16, 16:04 (f8a58e9)

Combinational Logic Gates

- Logic gates are the elementary blocks of a digital circuit (e.g. AND, OR, XOR)

Combinational Logic Gates

- Logic gates are the elementary blocks of a digital circuit (e.g. AND, OR, XOR)
- Gates without memory are called **combinational**

Combinational Logic Gates

- Logic gates are the elementary blocks of a digital circuit (e.g. AND, OR, XOR)
- Gates without memory are called **combinational**
    - Outputs only depend on inputs (c.f. mathematical function like $sin(x)$)

Combinational Logic Gates

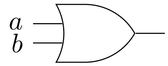
- Logic gates are the elementary blocks of a digital circuit (e.g. AND, OR, XOR)
- Gates without memory are called **combinational**
  - Outputs only depend on inputs (c.f. mathematical function like $sin(x)$)
- We can express their functionality using a *truth table*
  - Enumerate all inputs and write down output

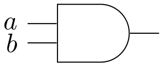
a	b	$a \wedge b$	$a \vee b$
F	F	F	F
F	T	F	T
T	F	F	T
T	T	T	T

Combinational Logic Gates

- Logic gates are the elementary blocks of a digital circuit (e.g. AND, OR, XOR)
- Gates without memory are called **combinational**
  - Outputs only depend on inputs (c.f. mathematical function like $sin(x)$)
- We can express their functionality using a *truth table*
  - Enumerate all inputs and write down output

a	b	$a \wedge b$	$a \vee b$
F	F	F	F
F	T	F	T
T	F	F	T
T	T	T	T





Sequential Logic Gates

- Gates with a memory are called **sequential**
  - Output depends on inputs and **previous state**

Sequential Logic Gates

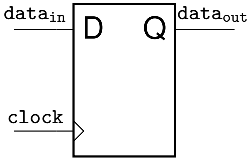
- Gates with a memory are called **sequential**
  - Output depends on inputs and **previous state**

  $\Rightarrow$ Expressed via truth table containing previous state or state diagram

Sequential Logic Gates

- Gates with a memory are called **sequential**
  - Output depends on inputs and **previous state**

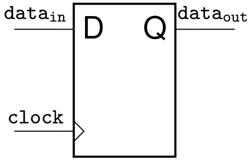
  ⇒ Expressed via truth table containing previous state or state diagram
- Prominent example: **flip-flop**



Sequential Logic Gates

- Gates with a memory are called **sequential**
  - Output depends on inputs and **previous state**

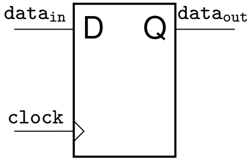
  ⇒ Expressed via truth table containing previous state or state diagram
- Prominent example: **flip-flop**
  - At each rising edge of the clock (CLK) the input data (D) is copied to the output (Q)



Sequential Logic Gates

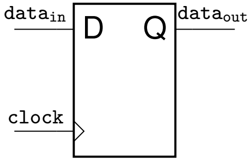
- Gates with a memory are called **sequential**
  - Output depends on inputs and **previous state**

  ⇒ Expressed via truth table containing previous state or state diagram
- Prominent example: **flip-flop**
  - At each rising edge of the clock (CLK) the input data (D) is copied to the output (Q)
  - Between rising clock edges the output is stable



Sequential Logic Gates

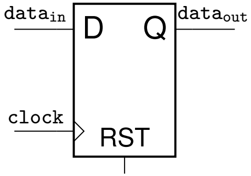
- Gates with a memory are called **sequential**
  - Output depends on inputs and **previous state**
  - ⇒ Expressed via truth table containing previous state or state diagram
- Prominent example: **flip-flop**
  - At each rising edge of the clock (CLK) the input data (D) is copied to the output (Q)
  - Between rising clock edges the output is stable



CLK	D	Q_{old}	Q
$\uparrow$	0	0	0
$\uparrow$	0	1	0
$\uparrow$	1	0	1
$\uparrow$	1	1	1
0	X	0	0 (Q_{old})
0	X	1	1 (Q_{old})
1	X	0	0 (Q_{old})
1	X	1	1 (Q_{old})

Sequential Logic Gates

- Gates with a memory are called **sequential**
  - Output depends on inputs and **previous state**
  - ⇒ Expressed via truth table containing previous state or state diagram
- Prominent example: **flip-flop**
  - At each rising edge of the clock (CLK) the input data (D) is copied to the output (Q)
  - Between rising clock edges the output is stable
- Optional: (synchronous or asynchronous) reset input (RST)



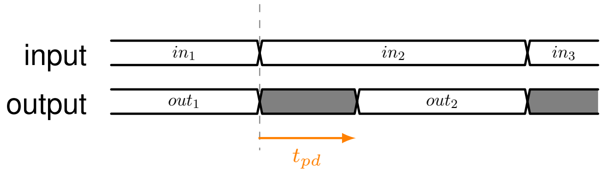
CLK	D	Q_{old}	Q
$\uparrow$	0	0	0
$\uparrow$	0	1	0
$\uparrow$	1	0	1
$\uparrow$	1	1	1
0	X	0	0 (Q_{old})
0	X	1	1 (Q_{old})
1	X	0	0 (Q_{old})
1	X	1	1 (Q_{old})

Timing Conditions for Proper Operation of Gates

- Real gates react to their inputs after their **propagation delay** (t_{pd})

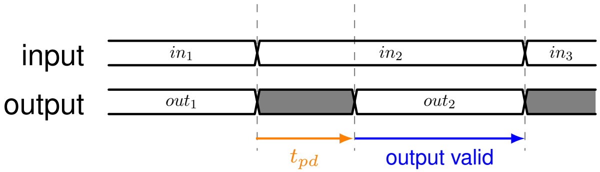
Timing Conditions for Proper Operation of Gates

- Real gates react to their inputs after their **propagation delay** (t_{pd})



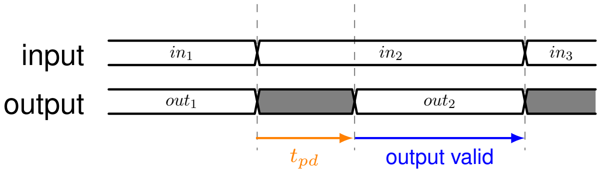
Timing Conditions for Proper Operation of Gates

- Real gates react to their inputs after their **propagation delay** (t_{pd})
- Before t_{pd} the output might **not** be valid



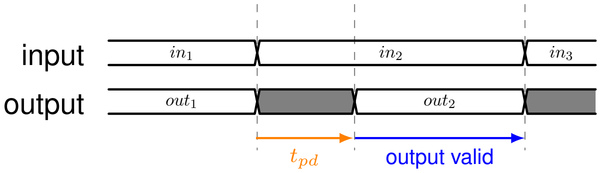
Timing Conditions for Proper Operation of Gates

- Real gates react to their inputs after their **propagation delay** (t_{pd})
- Before t_{pd} the output might **not** be valid
    - Output could be invalid voltage or make undesired transitions



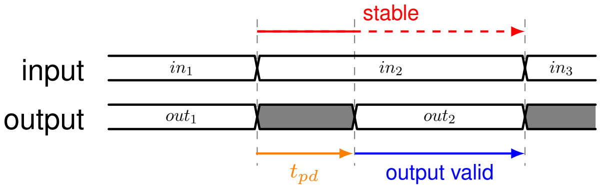
Timing Conditions for Proper Operation of Gates

- Real gates react to their inputs after their **propagation delay** (t_{pd})
- Before t_{pd} the output might **not** be valid
  - Output could be invalid voltage or make undesired transitions
- During the calculation of the output the inputs must be stable



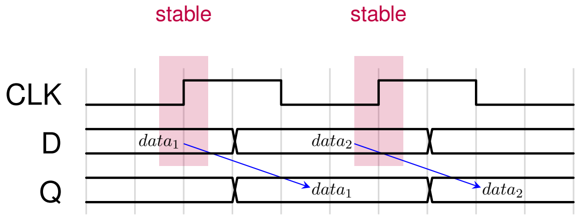
Timing Conditions for Proper Operation of Gates

- Real gates react to their inputs after their **propagation delay** (t_{pd})
- Before t_{pd} the output might **not** be valid
  - Output could be invalid voltage or make undesired transitions
- During the calculation of the output the inputs must be stable
  - After t_{pd} the output remains stable while the input does



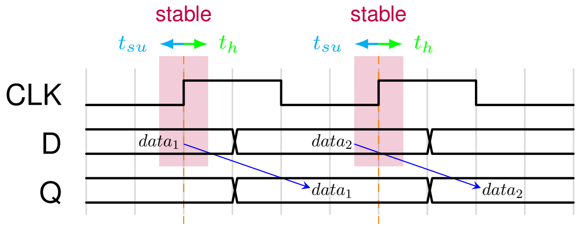
Timing Conditions for Proper Operation of FFs

- For the flip-flop the data input needs to be stable at the rising clock edge



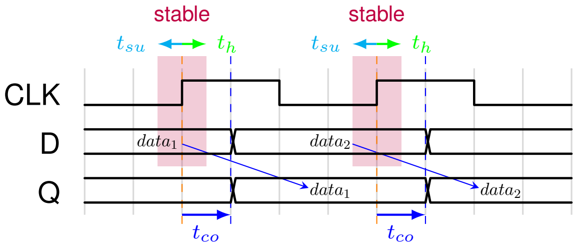
Timing Conditions for Proper Operation of FFs

- For the flip-flop the data input needs to be stable at the rising clock edge
  - Setup time t_{su} before / hold time t_h after the clock edge



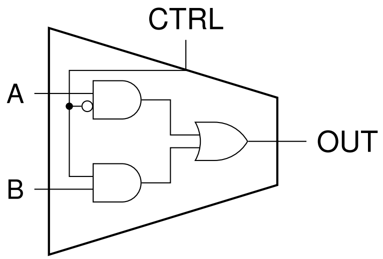
Timing Conditions for Proper Operation of FFs

- For the flip-flop the data input needs to be stable at the rising clock edge
  - Setup time t_{su} before / hold time t_h after the clock edge
  - Output changed after **clock-to-output** time (t_{co})



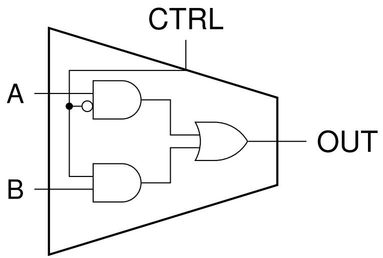
Building Functions from Gates

- Larger functions are composed of many simple gates



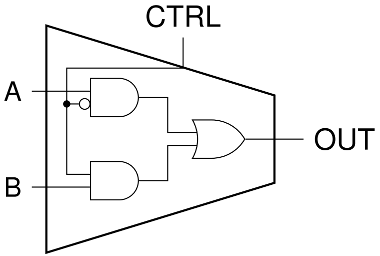
Building Functions from Gates

- Larger functions are composed of many simple gates
  - Gates operate concurrently
  - Some gates will provide inputs for others



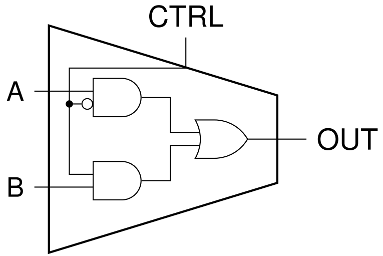
Building Functions from Gates

- Larger functions are composed of many simple gates
  - Gates operate concurrently
  - Some gates will provide inputs for others
- Each gate has an individual delay



Building Functions from Gates

- Larger functions are composed of many simple gates
  - Gates operate concurrently
  - Some gates will provide inputs for others
- Each gate has an individual delay
  - How to ensure proper operation?

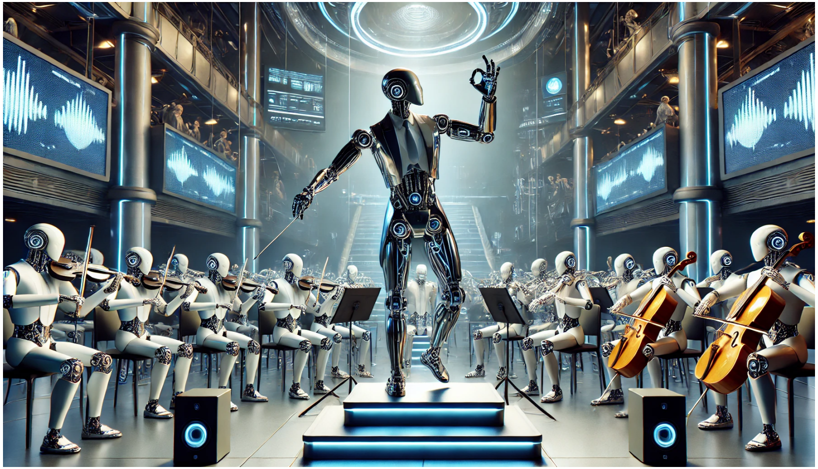


Building Functions from Gates

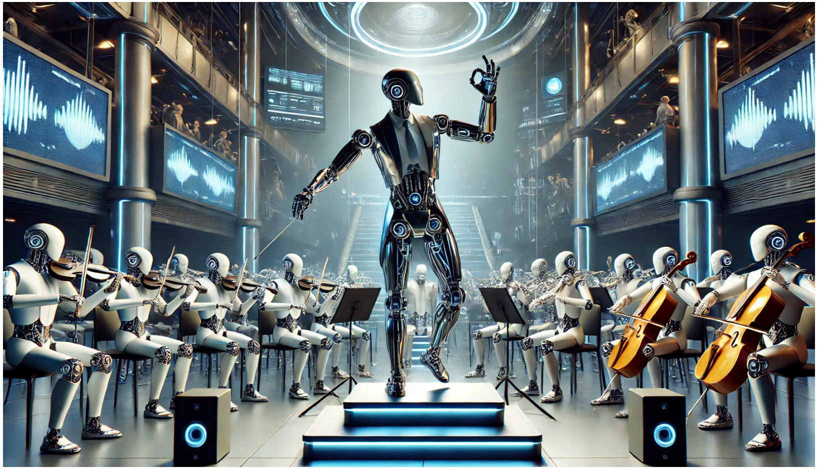
- Larger functions are composed of many simple gates
    - Gates operate concurrently
    - Some gates will provide inputs for others
- Each gate has an individual delay
    - How to ensure proper operation?

$\Rightarrow$ Requires coordination!

Coordination in Real Life



Coordination in Real Life



The Orchestra’s Coordination Principle

- There is no global notion of time

The Orchestra’s Coordination Principle

- There is no global notion of time $\Rightarrow$ the conductor introduces one

The Orchestra's Coordination Principle

- There is no global notion of time $\Rightarrow$ the conductor introduces one
- Each musician knows their specific schedule

The Orchestra’s Coordination Principle

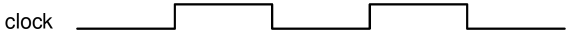
- There is no global notion of time $\Rightarrow$ the conductor introduces one
- Each musician knows their specific schedule
- A global plan ensures the desired result as a sum of all activities

Coordination in Synchronous Logic

- We require a global notion of time

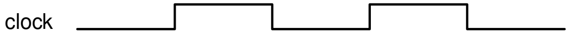
Coordination in Synchronous Logic

- We require a global notion of time
- Global clock distributed over circuit



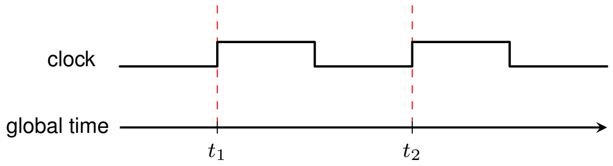
Coordination in Synchronous Logic

- We require a global notion of time
- Global clock distributed over circuit



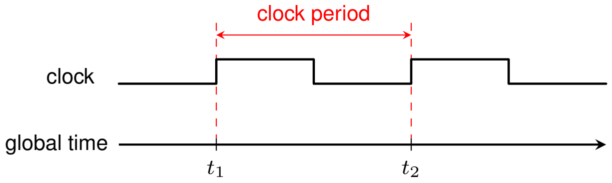
Coordination in Synchronous Logic

- We require a global notion of time
- Global clock distributed over circuit $\Rightarrow$ edges represent ticks of global time



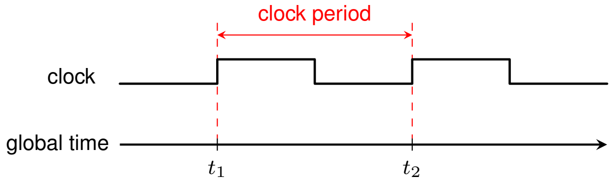
Coordination in Synchronous Logic

- We require a global notion of time
- Global clock distributed over circuit $\Rightarrow$ edges represent ticks of global time



Coordination in Synchronous Logic

- We require a global notion of time
- Global clock distributed over circuit $\Rightarrow$ edges represent ticks of global time
- Combinational gates cannot be controlled by a clock

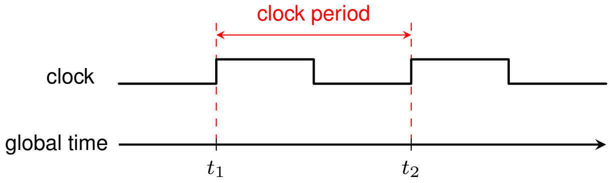


Coordination in Synchronous Logic

- We require a global notion of time
- Global clock distributed over circuit $\Rightarrow$ edges represent ticks of global time
- Combinational gates cannot be controlled by a clock

$\Rightarrow$ We put flip-flops between them to

  - capture gates' outputs at the right moment, and
  - keep gates' inputs stable sufficiently long enough

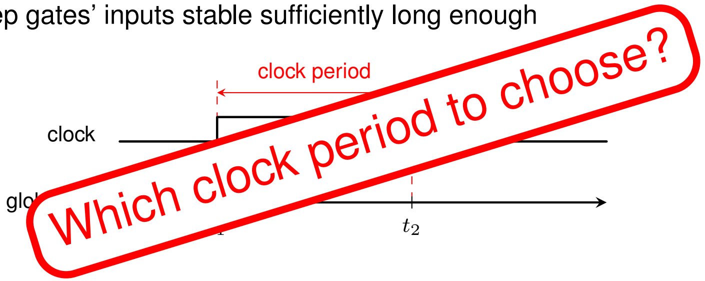


Coordination in Synchronous Logic

- We require a global notion of time
- Global clock distributed over circuit $\Rightarrow$ edges represent ticks of global time
- Combinational gates cannot be controlled by a clock

$\Rightarrow$ We put flip-flops between them to

- capture gates' outputs at the right moment, and
- keep gates' inputs stable sufficiently long enough



Assembly Line Optimization

Assembly Line Optimization

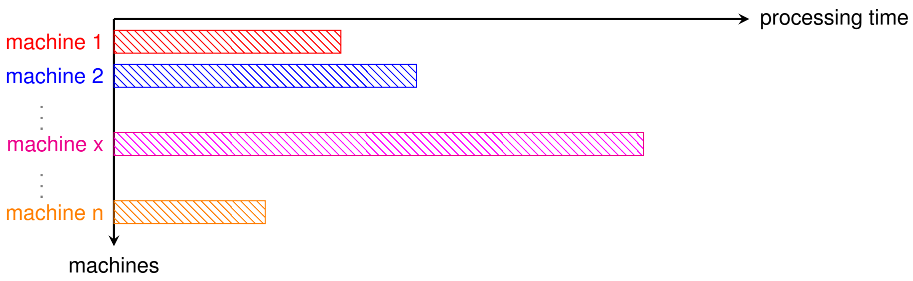
Assembly Line Optimization

Timing the Assembly Line

- Conveyor belt can only move once all machines are done

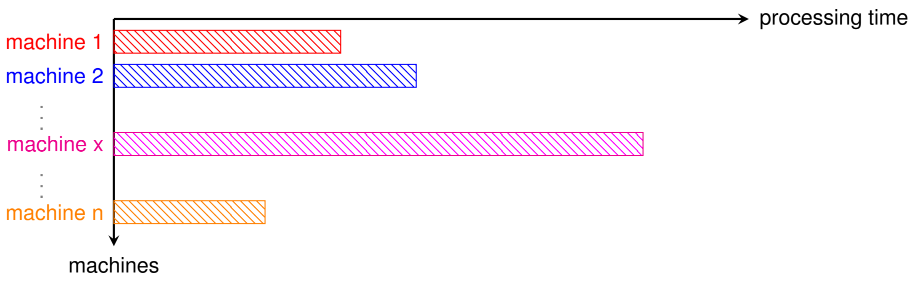
Timing the Assembly Line

- Conveyor belt can only move once all machines are done



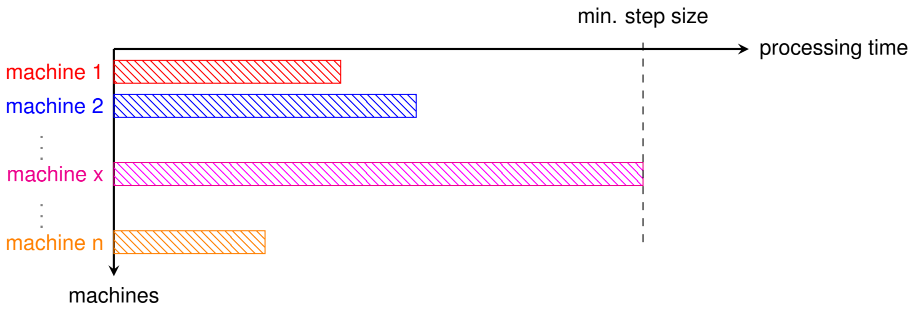
Timing the Assembly Line

- Conveyor belt can only move once all machines are done



Timing the Assembly Line

- Conveyor belt can only move once all machines are done
- Max. machine processing time $\Rightarrow$ Min. time step between movements



Static Timing Analysis (STA)

- In a synchronous design the circuit is partitioned into blocks through the insertion of flip-flops

Static Timing Analysis (STA)

- In a synchronous design the circuit is partitioned into blocks through the insertion of flip-flops

Static Timing Analysis (STA)

- In a synchronous design the circuit is partitioned into blocks through the insertion of flip-flops

Static Timing Analysis (STA)

- In a synchronous design the circuit is partitioned into blocks through the insertion of flip-flops
- To identify the minimum clock period we use **static timing analysis** (STA)

Static Timing Analysis (STA)

- In a synchronous design the circuit is partitioned into blocks through the insertion of flip-flops
- To identify the minimum clock period we use **static timing analysis** (STA)
    - Determine the signal delays through each block (take the slowest)

Static Timing Analysis (STA)

- In a synchronous design the circuit is partitioned into blocks through the insertion of flip-flops
- To identify the minimum clock period we use **static timing analysis** (STA)
  - Determine the signal delays through each block (take the slowest)
  - The longest of all such block delays is the **critical path**

Static Timing Analysis (STA)

- In a synchronous design the circuit is partitioned into blocks through the insertion of flip-flops
- To identify the minimum clock period we use **static timing analysis** (STA)
    - Determine the signal delays through each block (take the slowest)
    - The longest of all such block delays is the **critical path**
    - After this delay even the slowest output is stable

Static Timing Analysis (STA)

- In a synchronous design the circuit is partitioned into blocks through the insertion of flip-flops
- To identify the minimum clock period we use **static timing analysis** (STA)
  - Determine the signal delays through each block (take the slowest)
  - The longest of all such block delays is the **critical path**
  - After this delay even the slowest output is stable
  - We must also ensure stable flip-flop inputs around clock edges

Static Timing Analysis (STA)

- In a synchronous design the circuit is partitioned into blocks through the insertion of flip-flops
- To identify the minimum clock period we use **static timing analysis** (STA)
  - Determine the signal delays through each block (take the slowest)
  - The longest of all such block delays is the **critical path**
  - After this delay even the slowest output is stable
  - We must also ensure stable flip-flop inputs around clock edges
- This critical path delay determines the minimum clock period

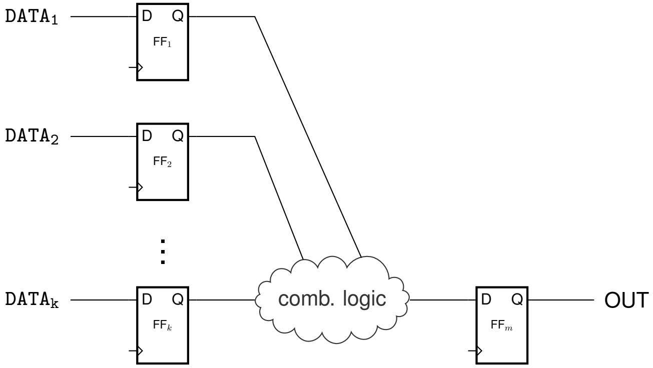
Static Timing Analysis (STA)

- In a synchronous design the circuit is partitioned into blocks through the insertion of flip-flops
- To identify the minimum clock period we use **static timing analysis** (STA)
  - Determine the signal delays through each block (take the slowest)
  - The longest of all such block delays is the **critical path**
  - After this delay even the slowest output is stable
  - We must also ensure stable flip-flop inputs around clock edges
- This critical path delay determines the minimum clock period
  - The maximum clock frequency is the inverse of this period

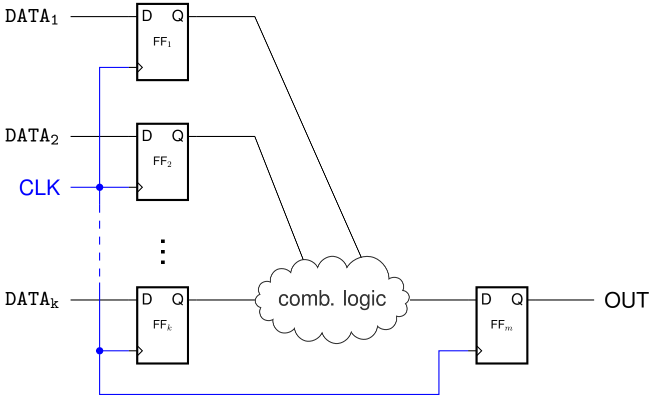
Static Timing Analysis (STA)

- In a synchronous design the circuit is partitioned into blocks through the insertion of flip-flops
- To identify the minimum clock period we use **static timing analysis** (STA)
  - Determine the signal delays through each block (take the slowest)
  - The longest of all such block delays is the **critical path**
  - After this delay even the slowest output is stable
  - We must also ensure stable flip-flop inputs around clock edges
- This critical path delay determines the minimum clock period
  - The maximum clock frequency is the inverse of this period
- For the best performance we choose our clock frequency close to the maximum from the STA

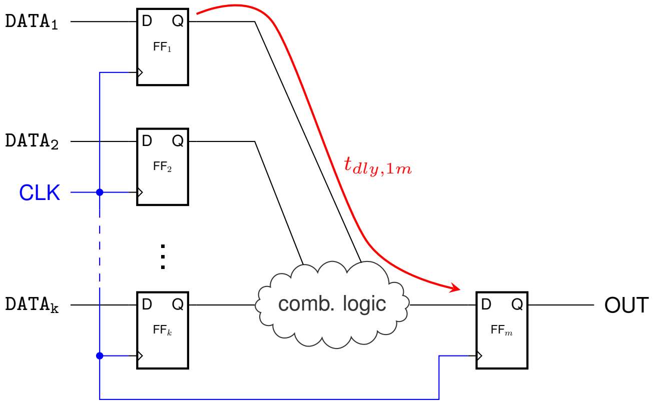
Static Timing Analysis Illustration



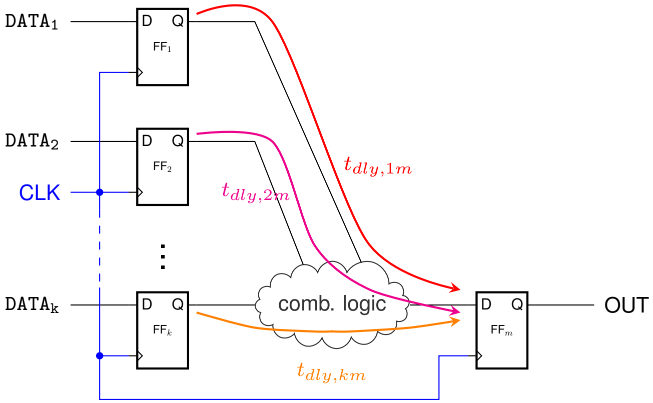
Static Timing Analysis Illustration



Static Timing Analysis Illustration

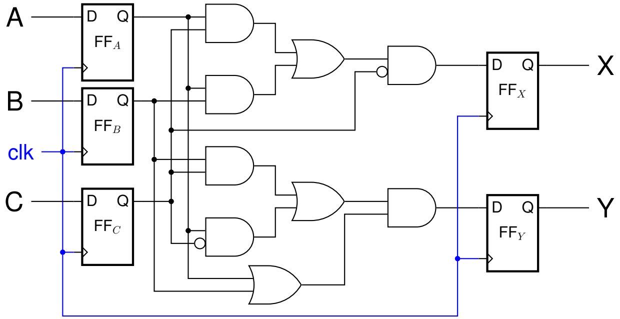


Static Timing Analysis Illustration



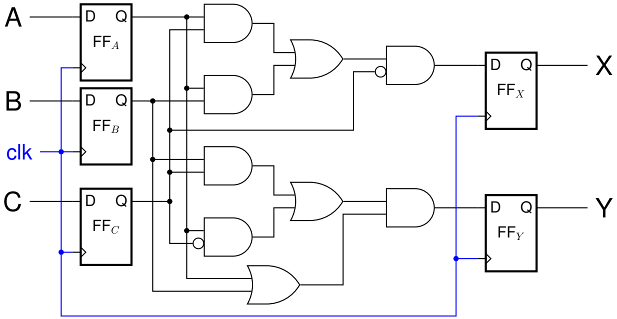
Calculation Example

What is the highest possible clock frequency f_{clk}?



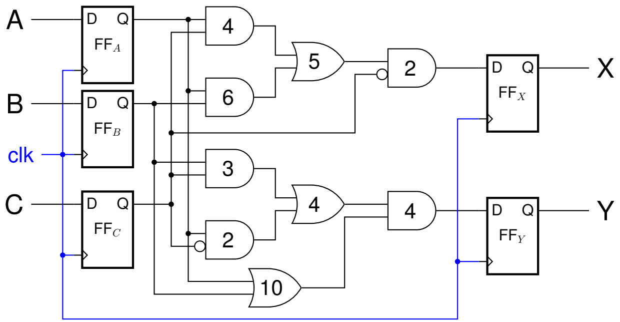
Calculation Example

What is the highest possible clock frequency f_{clk}?
Flip-flop parameters: $t_{co} = t_{su} = 1ns$



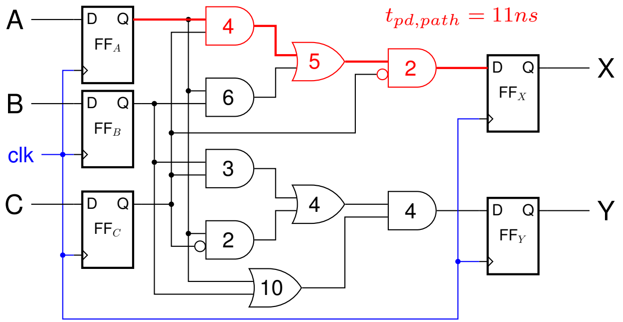
Calculation Example

What is the highest possible clock frequency f_{clk}?
Flip-flop parameters: $t_{co} = t_{su} = 1ns$



Calculation Example

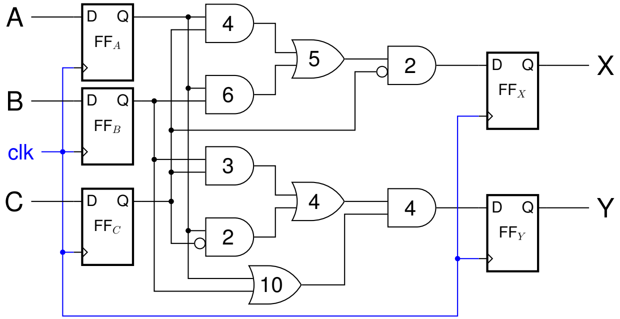
What is the highest possible clock frequency f_{clk}?
Flip-flop parameters: $t_{co} = t_{su} = 1ns$



path	delay [ns]
$FF_A \rightarrow FF_X$	11

Calculation Example

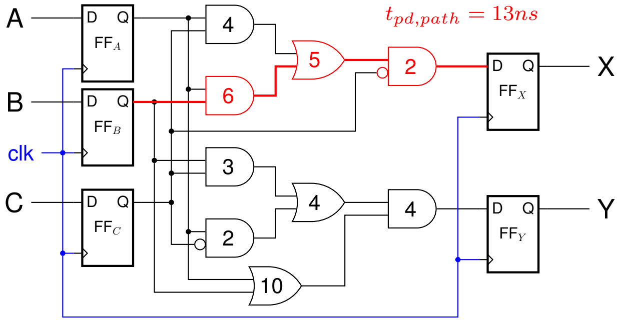
What is the highest possible clock frequency f_{clk}?
Flip-flop parameters: $t_{co} = t_{su} = 1ns$



path	delay [ns]
$FF_A \rightarrow FF_X$	11
$FF_A \rightarrow FF_X$	13
$FF_A \rightarrow FF_Y$	10
$FF_A \rightarrow FF_Y$	14

Calculation Example

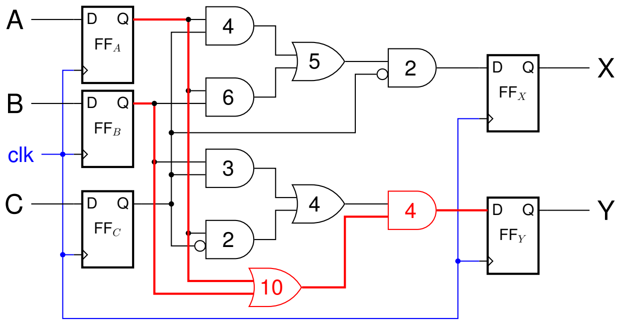
What is the highest possible clock frequency f_{clk}?
Flip-flop parameters: $t_{co} = t_{su} = 1ns$



path	delay [ns]
$FF_A \rightarrow FF_X$	11
$FF_A \rightarrow FF_X$	13
$FF_A \rightarrow FF_Y$	10
$FF_A \rightarrow FF_Y$	14
$FF_B \rightarrow FF_X$	13
$FF_B \rightarrow FF_Y$	11
$FF_B \rightarrow FF_Y$	14
$FF_C \rightarrow FF_X$	11
$FF_C \rightarrow FF_X$	2
$FF_C \rightarrow FF_Y$	11
$FF_C \rightarrow FF_Y$	10

Calculation Example

What is the highest possible clock frequency f_{clk}?
Flip-flop parameters: $t_{co} = t_{su} = 1ns$

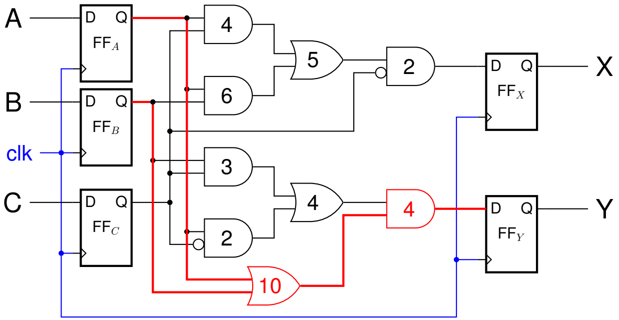


path	delay [ns]
$FF_A \rightarrow FF_X$	11
$FF_A \rightarrow FF_X$	13
$FF_A \rightarrow FF_Y$	10
$FF_A \rightarrow FF_Y$	14
$FF_B \rightarrow FF_X$	13
$FF_B \rightarrow FF_Y$	11
$FF_B \rightarrow FF_Y$	14
$FF_C \rightarrow FF_X$	11
$FF_C \rightarrow FF_X$	2
$FF_C \rightarrow FF_Y$	11
$FF_C \rightarrow FF_Y$	10

Calculation Example

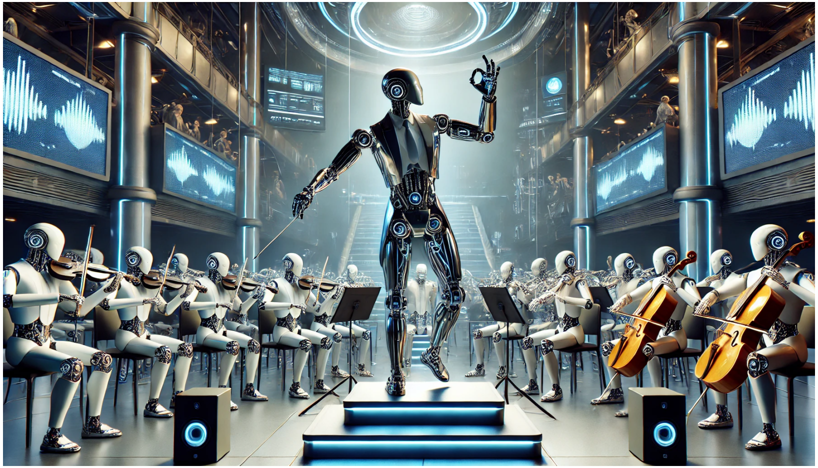
What is the highest possible clock frequency f_{clk}?
Flip-flop parameters: $t_{co} = t_{su} = 1ns$

$$T_{clk} = (14 + 1 + 1)ns \Rightarrow f_{clk} = T_{clk}^{-1} = (16ns)^{-1} = 62.5MHz$$



path	delay [ns]
$FF_A \rightarrow FF_X$	11
$FF_A \rightarrow FF_X$	13
$FF_A \rightarrow FF_Y$	10
$FF_A \rightarrow FF_Y$	14
$FF_B \rightarrow FF_X$	13
$FF_B \rightarrow FF_Y$	11
$FF_B \rightarrow FF_Y$	14
$FF_C \rightarrow FF_X$	11
$FF_C \rightarrow FF_X$	2
$FF_C \rightarrow FF_Y$	11
$FF_C \rightarrow FF_Y$	10

Benefits of Synchronous Design



Benefits of Synchronous Design

- Discretization of Time
    - Concentrate on points in time where all inputs and outputs are stable

  $\Rightarrow$ Designing synchronous circuits is relatively easy and efficient

Benefits of Synchronous Design

- Discretization of Time
  - Concentrate on points in time where all inputs and outputs are stable
  - $\Rightarrow$ Designing synchronous circuits is relatively easy and efficient
- High efficiency
  - Just one single signal required to coordinate all activities in the circuit
  - This periodic clock signal is easy to generate

Benefits of Synchronous Design

- Discretization of Time
  - Concentrate on points in time where all inputs and outputs are stable

  $\Rightarrow$ Designing synchronous circuits is relatively easy and efficient
- High efficiency
  - Just one single signal required to coordinate all activities in the circuit
  - This periodic clock signal is easy to generate
- Proven in practice
  - Billion working designs

Issues with Synchronous Design

Issues with Synchronous Design

- Clock distribution

Issues with Synchronous Design

- Clock distribution
  - Clock edges must arrive at all flip flops at (nearly) the same time

Issues with Synchronous Design

- Clock distribution
  - Clock edges must arrive at all flip flops at (nearly) the same time

  $\Rightarrow$ The clock network is power hungry and challenging to design

Issues with Synchronous Design

- Clock distribution
  - Clock edges must arrive at all flip flops at (nearly) the same time

  $\Rightarrow$ The clock network is power hungry and challenging to design
- Delay uncertainties

Issues with Synchronous Design

- Clock distribution
  - Clock edges must arrive at all flip flops at (nearly) the same time

  $\Rightarrow$ The clock network is power hungry and challenging to design
- Delay uncertainties
  - Propagation delays vary with temperature, supply voltage and are subject to fabrication tolerances

Issues with Synchronous Design

- Clock distribution
  - Clock edges must arrive at all flip flops at (nearly) the same time
  - $\Rightarrow$ The clock network is power hungry and challenging to design
- Delay uncertainties
  - Propagation delays vary with temperature, supply voltage and are subject to fabrication tolerances
  - $\Rightarrow$ Require worst-case assumptions, wasting performance

Issues with Synchronous Design

- Clock distribution
  - Clock edges must arrive at all flip flops at (nearly) the same time

  $\Rightarrow$ The clock network is power hungry and challenging to design
- Delay uncertainties
  - Propagation delays vary with temperature, supply voltage and are subject to fabrication tolerances

  $\Rightarrow$ Require worst-case assumptions, wasting performance
- Rigid timing, no graceful degradation
  - Propagation delay exceeds clock period $\Rightarrow$ completely wrong results

Issues with Synchronous Design

- Clock distribution
  - Clock edges must arrive at all flip flops at (nearly) the same time

  $\Rightarrow$ The clock network is power hungry and challenging to design
- Delay uncertainties
  - Propagation delays vary with temperature, supply voltage and are subject to fabrication tolerances

  $\Rightarrow$ Require worst-case assumptions, wasting performance
- Rigid timing, no graceful degradation
  - Propagation delay exceeds clock period $\Rightarrow$ completely wrong results
- However: synchronous design is the most widely used design style
  - Alternatives exist (advanced courses)

Lecture Complete!