

Chapter 3 :: Sequential Logic Design

Digital Design and Computer Architecture

David Money Harris and Sarah L. Harris

Copyright © 2007 Elsevier

3-<1>



Chapter 3 :: Topics

- Introduction
- Latches and Flip-Flops
- Synchronous Logic Design
- Finite State Machines
- Timing of Sequential Logic
- Parallelism

Copyright © 2007 Elsevier

3-<2>



Introduction

- Outputs of sequential logic depend on current *and* prior input values – it has **memory**.
- Some definitions:
 - **State**: all the information about a circuit necessary to explain its future behavior
 - **Latches and flip-flops**: state elements that store one bit of state
 - **Synchronous sequential circuits**: combinational logic followed by a bank of flip-flops

Copyright © 2007 Elsevier

3-<3>



Sequential Circuits

- Give sequence to events
- Have memory (short-term)
- Use feedback from output to input to store information

Copyright © 2007 Elsevier

3-<4>



State Elements

- The state of a circuit influences its future behavior
- State elements store state
 - Bistable circuit
 - SR Latch
 - D Latch
 - D Flip-flop

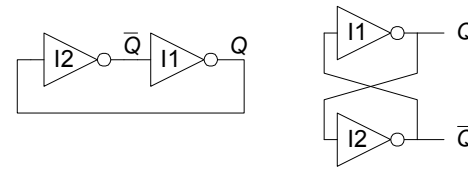
Copyright © 2007 Elsevier

3-<6>



Bistable Circuit

- Fundamental building block of other state elements
- Two outputs: Q, \bar{Q}
- No inputs



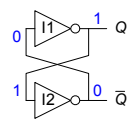
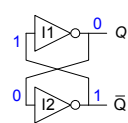
Copyright © 2007 Elsevier

3-<6>



Bistable Circuit Analysis

- Consider the two possible cases:
 - $Q = 0$: then $\bar{Q} = 1$ and $Q = 0$ (consistent)
 - $Q = 1$: then $\bar{Q} = 0$ and $Q = 1$ (consistent)



- Bistable circuit stores 1 bit of state in the *state variable*, Q
 - (or \bar{Q})
- But there are **no inputs to control the state**

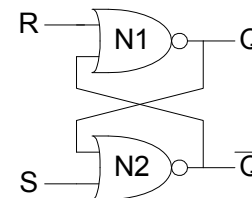
Copyright © 2007 Elsevier

3-<7>



SR (Set/Reset) Latch

- SR Latch



- Consider the four possible cases:
 - $S = 0, R = 1$
 - $S = 1, R = 0$
 - $S = 0, R = 0$
 - $S = 1, R = 1$

Copyright © 2007 Elsevier

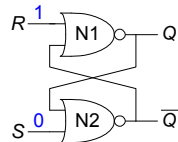
3-<8>



SR Latch Analysis

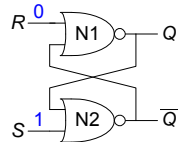
- $R = 1, S = 0$:

$Q =$
 $\bar{Q} =$



- $R = 0, S = 1$:

$Q =$
 $\bar{Q} =$



Copyright © 2007 Elsevier

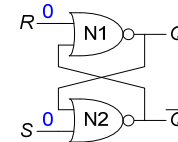
3-<9>



SR Latch Analysis

- $R = 0, S = 0$:

$Q =$
 $\bar{Q} =$



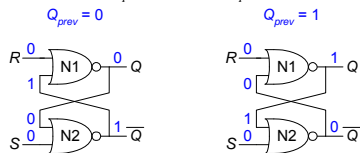
Copyright © 2007 Elsevier

3-<10>



SR Latch Analysis

- $S = 0, R = 0$: then $Q = Q_{prev}$ and $\bar{Q} = \bar{Q}_{prev}$



- Q remembers its previous value (state)

- This is a way to build memory

Copyright © 2007 Elsevier

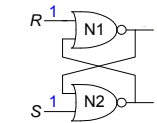
3-<11>



SR Latch Analysis

- $S = 1, R = 1$:

$Q =$
 $\bar{Q} =$



Copyright © 2007 Elsevier

3-<12>

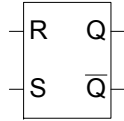


SR Latch Symbol

- SR stands for Set/Reset Latch
 - Stores one bit of state (Q)
- Control what value is being stored with S, R inputs
 - **Set:** Make the output 1 ($S = 1, R = 0, Q = 1$)
 - **Reset:** Make the output 0 ($S = 0, R = 1, Q = 0$)

- **Must do something to avoid invalid state (when $S = R = 1$)**

SR Latch Symbol



Copyright © 2007 Elsevier

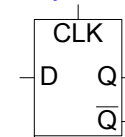
3-<13>



D Latch

- Two inputs: CLK, D
 - CLK : controls *when* the output changes
 - D (the data input): controls *what* the output changes to
- Function
 - When $CLK = 1$, D passes through to Q (the latch is *transparent*)
 - When $CLK = 0$, Q holds its previous value (the latch is *opaque*)
- Avoids invalid case when $Q \neq \text{NOT } \bar{Q}$

D Latch Symbol

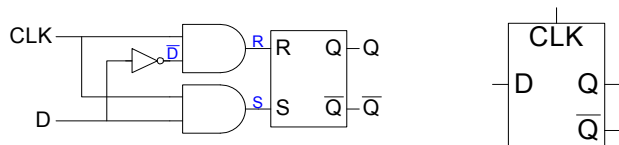


Copyright © 2007 Elsevier

3-<14>



D Latch Internal Circuit



CLK	D	\bar{D}	S	R	Q	\bar{Q}
0	X					
1	0					
1	1					

Copyright © 2007 Elsevier

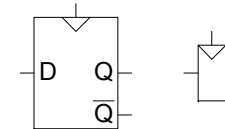
3-<15>



D Flip-Flop

- Two inputs: CLK, D
- Function
 - The flip-flop “samples” D on the rising edge of CLK
 - When CLK rises from 0 to 1, D passes through to Q
 - Otherwise, Q holds its previous value
 - Q changes only on the rising edge of CLK
- A flip-flop is called an *edge-triggered* device because it is activated on the clock edge

D Flip-Flop Symbols



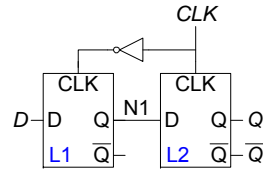
Copyright © 2007 Elsevier

3-<16>



D Flip-Flop Internal Circuit

- Two back-to-back latches (L1 and L2) controlled by complementary clocks
- When $CLK = 0$
 - L1 is
 - L2 is
 - D
- When $CLK = 1$
 - L2 is
 - L1 is
 - $N1$
- Thus, on the edge of the clock (when CLK rises from 0→1)
 - D

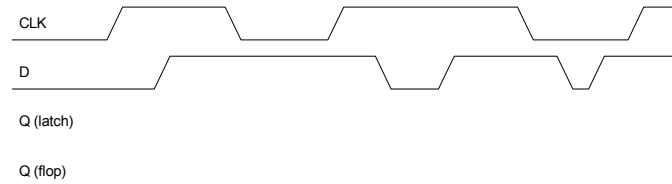


Copyright © 2007 Elsevier

3-<17>



D Flip-Flop vs. D Latch

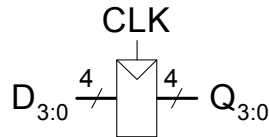
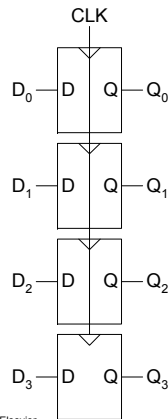


Copyright © 2007 Elsevier

3-<18>



Registers



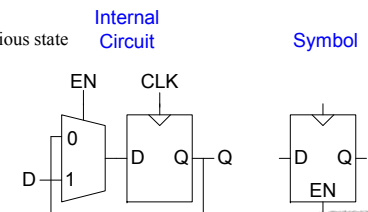
Copyright © 2007 Elsevier

3-<19>



Enabled Flip-Flops

- Inputs: CLK, D, EN
 - The enable input (EN) controls when new data (D) is stored
- Function
 - $EN = 1$
 - D passes through to Q on the clock edge
 - $EN = 0$
 - the flip-flop retains its previous state



Copyright © 2007 Elsevier

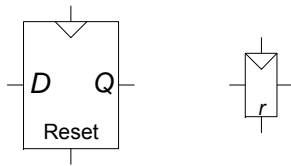
3-<20>



Resettable Flip-Flops

- Inputs: $CLK, D, Reset$
- Function:
 - $Reset = 1$
 - Q is forced to 0
 - $Reset = 0$
 - the flip-flop behaves like an ordinary D flip-flop

Symbols



Copyright © 2007 Elsevier

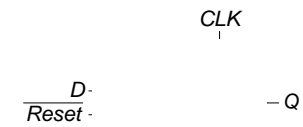
3-<21>



Resettable Flip-Flops

- Two types:
 - Synchronous: resets at the clock edge only
 - Asynchronous: resets immediately when $Reset = 1$
- Asynchronously resettable flip-flop requires changing the internal circuitry of the flip-flop (see Exercise 3.10)
- Synchronously resettable flip-flop?

Internal Circuit



Copyright © 2007 Elsevier

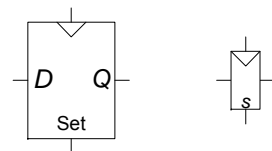
3-<22>



Settable Flip-Flops

- Inputs: CLK, D, Set
- Function:
 - $Set = 1$
 - Q is set to 1
 - $Set = 0$
 - the flip-flop behaves like an ordinary D flip-flop

Symbols



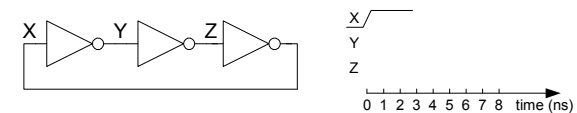
Copyright © 2007 Elsevier

3-<23>



Sequential Logic

- Sequential circuits: all circuits that aren't combinational
- A problematic circuit:



- This circuit has no inputs and 1-3 outputs

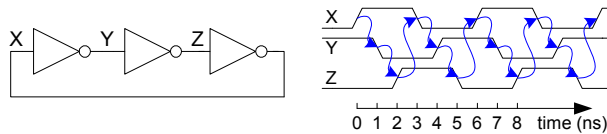
Copyright © 2007 Elsevier

3-<24>



Sequential Logic

- Sequential circuits: all circuits that aren't combinational
- A problematic circuit:



- This circuit has no inputs and 1-3 outputs
- It is an astable circuit that oscillates
- Its period depends on the delay of the inverters – which depends on the manufacturing process, temperature, etc
- The circuit has a *cyclic path*: output fed back to input

Copyright © 2007 Elsevier

3-<25>



Synchronous Sequential Logic Design

- Breaks cyclic paths by inserting registers
- These registers contain the state of the system
- The state changes at the clock edge, so we say the system is *synchronized* to the clock
- Rules of synchronous sequential circuit composition:
 - Every circuit element is either a register or a combinational circuit
 - At least one circuit element is a register
 - All registers receive the same clock signal
 - Every cyclic path contains at least one register
- Two common synchronous sequential circuits
 - Finite State Machines (FSMs)
 - Pipelines

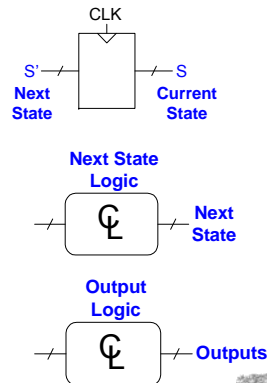
Copyright © 2007 Elsevier

3-<26>



Finite State Machine (FSM)

- Consists of:
 - State register that
 - Stores the current state and
 - Loads the next state at the clock edge
 - Combinational logic that
 - Computes the next state
 - Computes the outputs



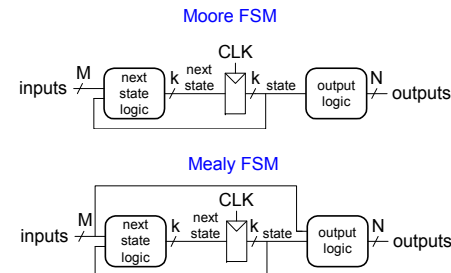
Copyright © 2007 Elsevier

3-<27>



Finite State Machines (FSMs)

- Next state is determined by the current state and the inputs
- Two types of finite state machines differ in the output logic:
 - Moore FSM: outputs depend only on the current state
 - Mealy FSM: outputs depend on the current state *and* the inputs



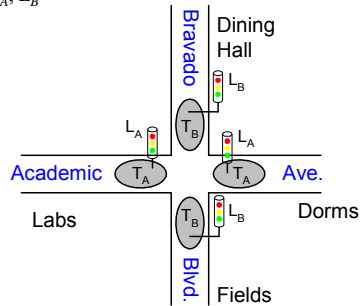
Copyright © 2007 Elsevier

3-<28>



Finite State Machine Example

- Traffic light controller
 - Traffic sensors: T_A, T_B (TRUE when there's traffic)
 - Lights: L_A, L_B



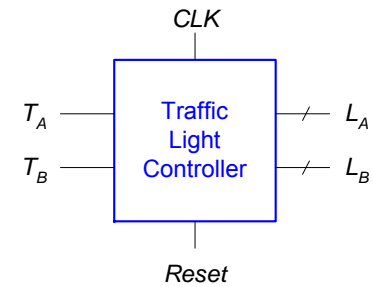
Copyright © 2007 Elsevier

3-<29>



FSM Black Box

- Inputs: $CLK, Reset, T_A, T_B$
- Outputs: L_A, L_B



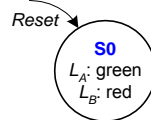
Copyright © 2007 Elsevier

3-<30>



FSM State Transition Diagram

- Moore FSM: outputs labeled in each state
- States: Circles
- Transitions: Arcs



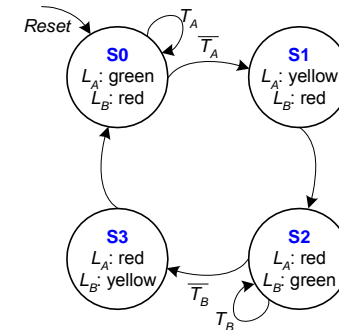
Copyright © 2007 Elsevier

3-<31>



FSM State Transition Diagram

- Moore FSM: outputs labeled in each state
- States: Circles
- Transitions: Arcs



Copyright © 2007 Elsevier

3-<32>



FSM State Transition Table

Current State S	Inputs		Next State S'
	T_A	T_B	
S0	0	X	
S0	1	X	
S1	X	X	
S2	X	0	
S2	X	1	
S3	X	X	

Copyright © 2007 Elsevier

3-<33>



FSM Encoded State Transition Table

Current State		Inputs		Next State	
S_1	S_0	T_A	T_B	S'_1	S'_0
0	0	0	X		
0	0	1	X		
0	1	X	X		
1	0	X	0		
1	0	X	1		
1	1	X	X		

State	Encoding
S0	00
S1	01
S2	10
S3	11

$$S'_1 =$$

$$S'_0 =$$

Copyright © 2007 Elsevier

3-<34>



FSM Output Table

Current State		Outputs			
S_1	S_0	L_{A1}	L_{A0}	L_{B1}	L_{B0}
0	0				
0	1				
1	0				
1	1				

Output	Encoding
green	00
yellow	01
red	10

$$L_{A1} =$$

$$L_{A0} =$$

$$L_{B1} =$$

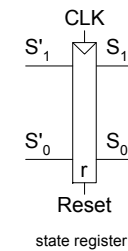
$$L_{B0} =$$

Copyright © 2007 Elsevier

3-<35>



FSM Schematic: State Register

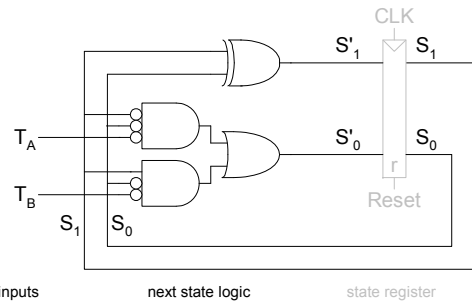


Copyright © 2007 Elsevier

3-<36>



FSM Schematic: Next State Logic

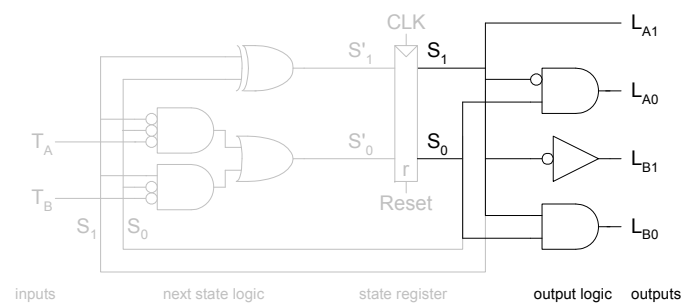


Copyright © 2007 Elsevier

3-<37>



FSM Schematic: Output Logic

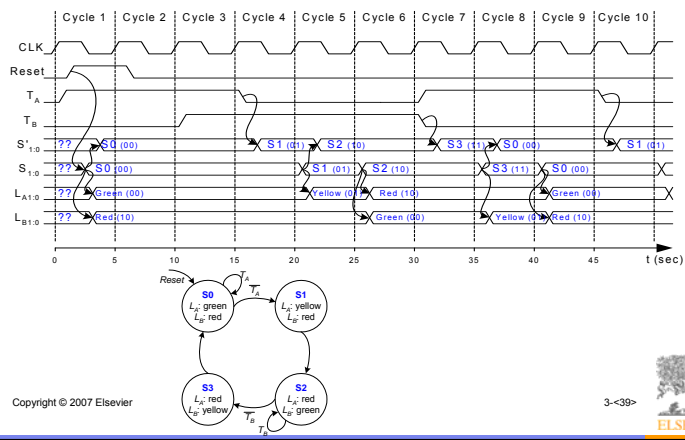


Copyright © 2007 Elsevier

3-<38>



FSM Timing Diagram



Copyright © 2007 Elsevier

3-<39>



FSM State Encoding

- Binary encoding: i.e., for four states, 00, 01, 10, 11
- One-hot encoding
 - One state bit per state
 - Only one state bit is HIGH at once
 - I.e., for four states, 0001, 0010, 0100, 1000
 - Requires more flip-flops
 - Often next state and output logic is simpler

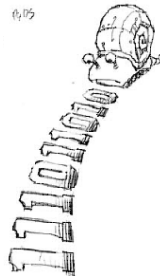
Copyright © 2007 Elsevier

3-<40>



Moore vs. Mealy FSM

- Alyssa P. Hacker has a snail that crawls down a paper tape with 1's and 0's on it. The snail smiles whenever the last four digits it has crawled over are 1101. Design Moore and Mealy FSMs of the snail's brain.



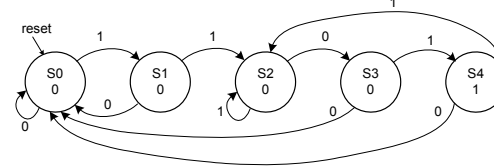
Copyright © 2007 Elsevier

3-<41>



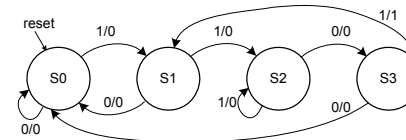
State Transition Diagrams

Moore FSM



Mealy FSM: arcs indicate input/output

Mealy FSM



Copyright © 2007 Elsevier

3-<42>



Moore FSM State Transition Table

Current State			Inputs	Next State		
S_2	S_1	S_0	A	S'_2	S'_1	S'_0
0	0	0	0			
0	0	0	1			
0	0	1	0			
0	0	1	1			
0	1	0	0			
0	1	0	1			
0	1	1	0			
0	1	1	1			
1	0	0	0			
1	0	0	1			

State	Encoding
S0	000
S1	001
S2	010
S3	011
S4	100

Copyright © 2007 Elsevier

3-<43>



Moore FSM Output Table

Current State			Output
S_2	S_1	S_0	Y
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	

Y =

Copyright © 2007 Elsevier

3-<44>



Mealy FSM State Transition and Output Table

Current State		Input	Next State		Output
S_1	S_0	A	S'_1	S'_0	Y
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

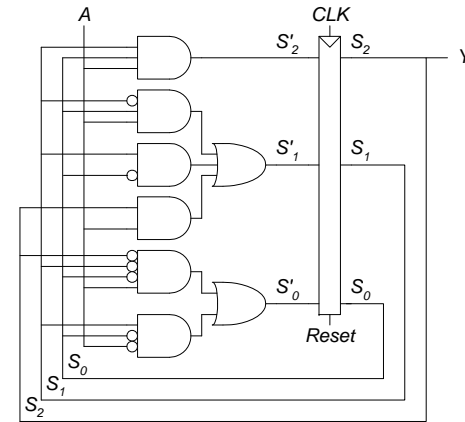
State	Encoding
S0	00
S1	01
S2	10
S3	11

Copyright © 2007 Elsevier

3-<45>



Moore FSM Schematic

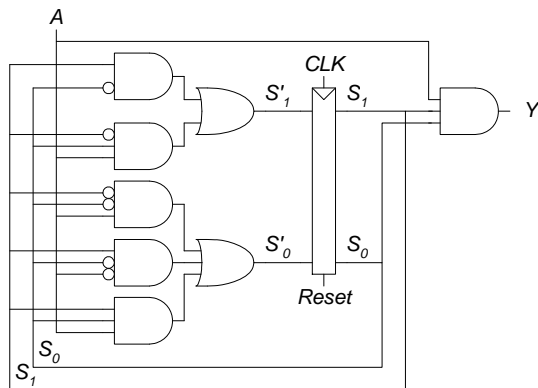


Copyright © 2007 Elsevier

3-<46>



Mealy FSM Schematic

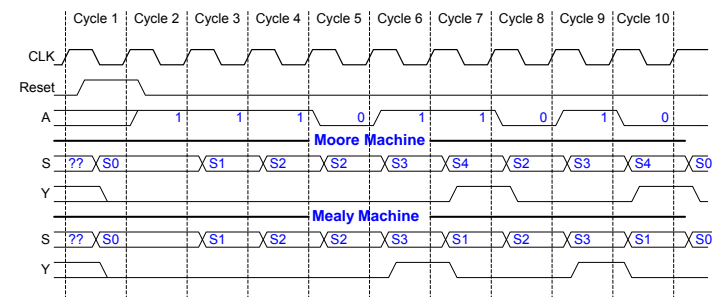


Copyright © 2007 Elsevier

3-<47>



Moore and Mealy Timing Diagram



Copyright © 2007 Elsevier

3-<48>



Factoring State Machines

- Break complex FSMs into smaller interacting FSMs
- Example: Modify the traffic light controller to have a Parade Mode.
 - The FSM receives two more inputs: P, R
 - When $P = 1$, it enters Parade Mode and the Bravado Blvd. light stays green.
 - When $R = 1$, it leaves Parade Mode

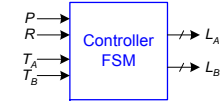
Copyright © 2007 Elsevier

3-<49>

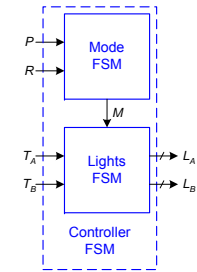


Parade FSM

Unfactored FSM



Factored FSM

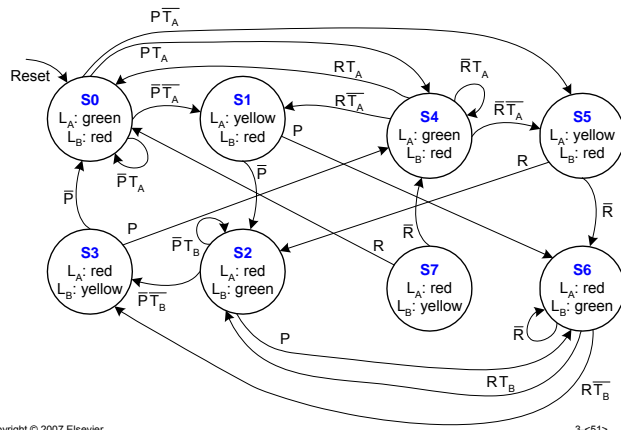


Copyright © 2007 Elsevier

3-<50>



Unfactored FSM State Transition Diagram

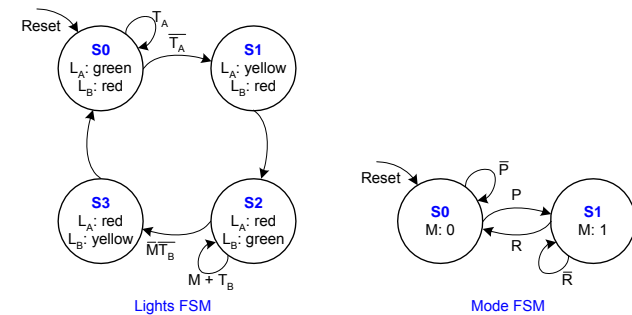


Copyright © 2007 Elsevier

3-<51>



Factored FSM State Transition Diagram



Copyright © 2007 Elsevier

3-<52>



FSM Design Procedure

- Identify the inputs and outputs
- Sketch a state transition diagram
- Write a state transition table
- Select state encodings
- For a Moore machine:
 - Rewrite the state transition table with the selected state encodings
 - Write the output table
- For a Mealy machine:
 - Rewrite the combined state transition and output table with the selected state encodings
- Write Boolean equations for the next state and output logic
- Sketch the circuit schematic

Copyright © 2007 Elsevier

3-<53>



Timing

- Flip-flop samples D at clock edge
- D must be stable when it is sampled
- Similar to a photograph, D must be stable around the clock edge
- If D is changing when it is sampled, metastability can occur

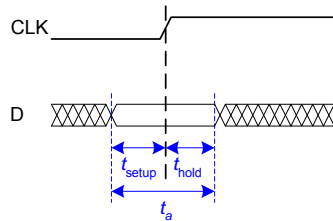
Copyright © 2007 Elsevier

3-<54>



Input Timing Constraints

- **Setup time:** t_{setup} = time *before* the clock edge that data must be stable (i.e. not changing)
- **Hold time:** t_{hold} = time *after* the clock edge that data must be stable
- **Aperture time:** t_a = time around clock edge that data must be stable ($t_a = t_{\text{setup}} + t_{\text{hold}}$)



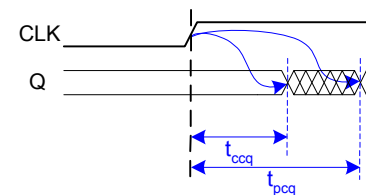
Copyright © 2007 Elsevier

3-<55>



Output Timing Constraints

- **Propagation delay:** t_{pcq} = time after clock edge that the output Q is guaranteed to be stable (i.e., to stop changing)
- **Contamination delay:** t_{ccq} = time after clock edge that Q might be unstable (i.e., start changing)



Copyright © 2007 Elsevier

3-<56>



Dynamic Discipline

- The input to a synchronous sequential circuit must be stable during the aperture (setup and hold) time around the clock edge.
- Specifically, the input must be stable
 - at least t_{setup} before the clock edge
 - at least until t_{hold} after the clock edge

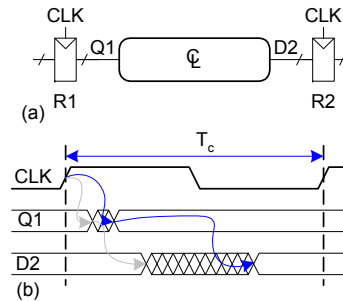
Copyright © 2007 Elsevier

3-<57>



Dynamic Discipline

- The delay between registers has a **minimum** and **maximum** delay, dependent on the delays of the circuit elements



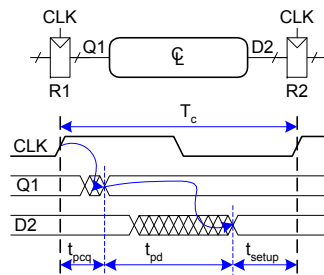
Copyright © 2007 Elsevier

3-<58>



Setup Time Constraint

- Depends on the **maximum** delay from register R1 through the combinational logic to R2
- The input to register R2 must be stable at least t_{setup} before the clock edge



$$T_c \geq t_{pd}$$

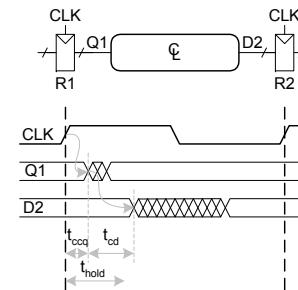
Copyright © 2007 Elsevier

3-<59>



Hold Time Constraint

- Depends on the **minimum** delay from register R1 through the combinational logic to R2
- The input to register R2 must be stable for at least t_{hold} after the clock edge



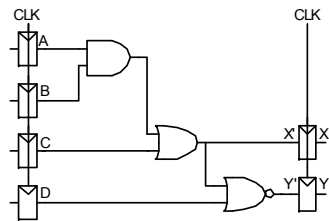
$$t_{\text{hold}} < t_{cd}$$

Copyright © 2007 Elsevier

3-<60>



Timing Analysis



Timing Characteristics

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

per gate
 $t_{pd} = 35 \text{ ps}$
 $t_{cd} = 25 \text{ ps}$

$t_{pd} =$

$t_{cd} =$

Setup time constraint:

$$T_c \geq$$

$$f_c = 1/T_c =$$

Hold time constraint:

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

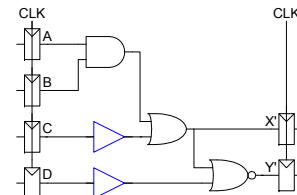
Copyright © 2007 Elsevier

3-<61>



Fixing Hold Time Violation

Add buffers to the short paths:



Timing Characteristics

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

per gate
 $t_{pd} = 35 \text{ ps}$
 $t_{cd} = 25 \text{ ps}$

$t_{pd} =$

$t_{cd} =$

Setup time constraint:

$$T_c \geq$$

$$f_c = 1/T_c =$$

Hold time constraint:

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

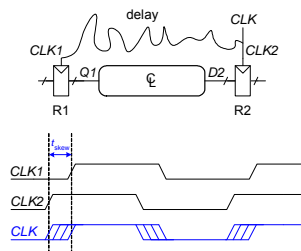
Copyright © 2007 Elsevier

3-<62>



Clock Skew

- The clock doesn't arrive at all registers at the same time
- Skew is the difference between two clock edges
- Examine the worst case to guarantee that the dynamic discipline is not violated for any register – many registers in a system!



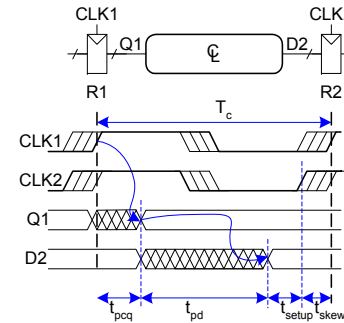
Copyright © 2007 Elsevier

3-<63>



Setup Time Constraint with Clock Skew

- In the worst case, CLK2 is earlier than CLK1



$$T_c \geq$$

$$t_{pd} \leq$$

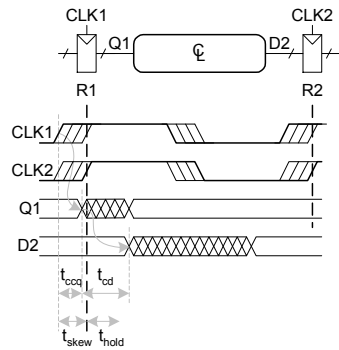
Copyright © 2007 Elsevier

3-<64>



Hold Time Constraint with Clock Skew

- In the worst case, CLK2 is later than CLK1



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

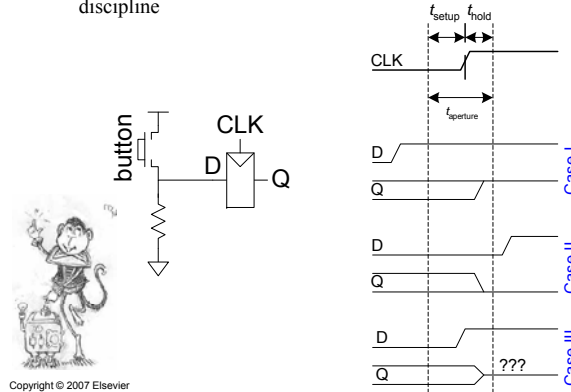
Copyright © 2007 Elsevier

3-<65>



Violating the Dynamic Discipline

- Asynchronous (for example, user) inputs might violate the dynamic discipline



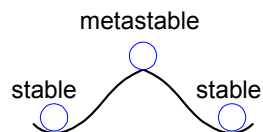
Copyright © 2007 Elsevier

3-<66>



Metastability

- Any bistable device has two stable states and a metastable state between them
- A flip-flop has two stable states (1 and 0) and one metastable state
- If a flip-flop lands in the metastable state, it could stay there for an undetermined amount of time



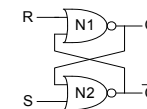
Copyright © 2007 Elsevier

3-<67>



Flip-flop Internals

- Because the flip-flop has feedback, if Q is somewhere between 1 and 0, the cross-coupled gates will eventually drive the output to either rail (1 or 0, depending on which one it is closer to).



- A signal is considered metastable if it hasn't resolved to 1 or 0
- If a flip-flop input changes at a random time, the probability that the output Q is metastable after waiting some time, t , is:

$$P(t_{res} > t) = (T_0/T_c) e^{-t/\tau}$$

t_{res} : time to resolve to 1 or 0

T_0, τ : properties of the circuit

Copyright © 2007 Elsevier

3-<68>



Metastability

- Intuitively:
 - T_0/T_c describes the probability that the input changes at a bad time, i.e., during the aperture time

$$P(t_{res} > t) = (T_0/T_c) e^{-t/\tau}$$
 - τ is a time constant indicating how fast the flip-flop moves away from the metastable state; it is related to the delay through the cross-coupled gates in the flip-flop

$$P(t_{res} > t) = (T_0/T_c) e^{-t/\tau}$$
- In short, if a flip-flop samples a metastable input, if you wait long enough (t), the output will have resolved to 1 or 0 with high probability.

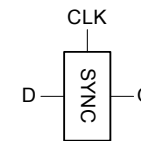
Copyright © 2007 Elsevier

3-<69>



Synchronizers

- Asynchronous inputs (D) are inevitable (user interfaces, systems with different clocks interacting, etc.).
- The goal of a synchronizer is to make the probability of failure (the output Q still being metastable) low.
- A synchronizer cannot make the probability of failure 0.



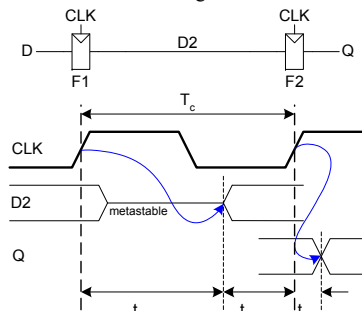
Copyright © 2007 Elsevier

3-<70>



Synchronizer Internals

- A synchronizer can be built with two back-to-back flip-flops.
- Suppose the input D is transitioning when it is sampled by flip-flop 1, $F1$.
- The amount of time the internal signal $D2$ can resolve to a 1 or 0 is $(T_c - t_{setup})$.



Copyright © 2007 Elsevier

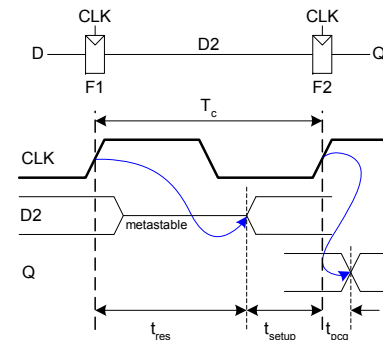
3-<71>



Synchronizer Probability of Failure

For each sample, the probability of failure of this synchronizer is:

$$P(\text{failure}) = (T_0/T_c) e^{-(T_c - t_{setup})/\tau}$$



Copyright © 2007 Elsevier

3-<72>



Synchronizer Mean Time Before Failure

- If the asynchronous input changes once per second, the probability of failure per second of the synchronizer is simply $P(\text{failure})$.
- In general, if the input changes N times per second, the probability of failure per second of the synchronizer is:

$$P(\text{failure})/\text{second} = (NT_0/T_c) e^{-(T_c - t_{\text{setup}})/\tau}$$

- Thus, the synchronizer fails, on average, $1/[P(\text{failure})/\text{second}]$
- This is called the *mean time between failures*, MTBF:

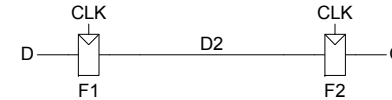
$$\text{MTBF} = 1/[P(\text{failure})/\text{second}] = (T_c/NT_0) e^{(T_c - t_{\text{setup}})/\tau}$$

Copyright © 2007 Elsevier

3-<73>



Example Synchronizer



- Suppose: $T_c = 1/500 \text{ MHz} = 2 \text{ ns}$ $\tau = 200 \text{ ps}$
 $T_0 = 150 \text{ ps}$ $t_{\text{setup}} = 100 \text{ ps}$
 $N = 10 \text{ events per second}$

- What is the probability of failure? MTBF?

$P(\text{failure}) =$

$P(\text{failure})/\text{second} =$

MTBF =

Copyright © 2007 Elsevier

3-<74>



Parallelism

- **Two types of parallelism:**
 - **Spatial parallelism**
 - duplicate hardware performs multiple tasks at once
 - **Temporal parallelism**
 - task is broken into multiple stages
 - also called pipelining
 - for example, an assembly line

Copyright © 2007 Elsevier

3-<75>



Parallelism Definitions

- Some definitions:
 - **Token:** A group of inputs processed to produce a group of outputs
 - **Latency:** Time for one token to pass from start to end
 - **Throughput:** The number of tokens that can be produced per unit time
- Parallelism increases throughput.

Copyright © 2007 Elsevier

3-<76>



Parallelism Example

- Ben Bitdiddle is baking cookies to celebrate the installation of his traffic light controller. It takes 5 minutes to roll the cookies and 15 minutes to bake them. After finishing one batch he immediately starts the next batch. What is the latency and throughput if Ben doesn't use parallelism?

Latency =

Throughput =

Copyright © 2007 Elsevier

3-<77>



Parallelism Example

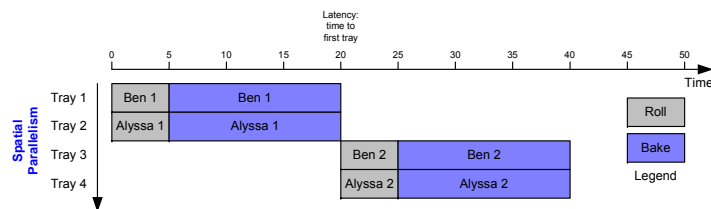
- What is the latency and throughput if Ben uses parallelism?
 - Spatial parallelism:** Ben asks Alyssa P. Hacker to help, using her own oven
 - Temporal parallelism:** Ben breaks the task into two stages: roll and baking. He uses two trays. While the first batch is baking he rolls the second batch, and so on.

Copyright © 2007 Elsevier

3-<78>



Spatial Parallelism



Latency =

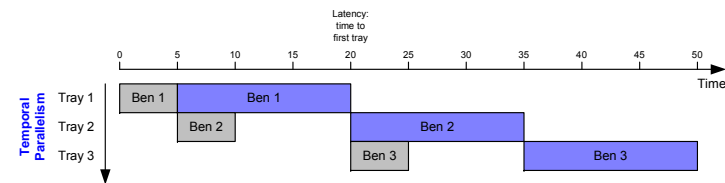
Throughput =

Copyright © 2007 Elsevier

3-<79>



Temporal Parallelism



Latency =

Throughput =

Using both techniques, the throughput would be 8 trays/hour

Copyright © 2007 Elsevier

3-<80>

