Chapter 3 :: Sequential Logic Design

Digital Design and Computer Architecture

David Money Harris and Sarah L. Harris

Chapter 3 :: Topics

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

Introduction

- Outputs of sequential logic depend on current *and* prior input values.
- Sequential logic thus 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

Sequential Circuits

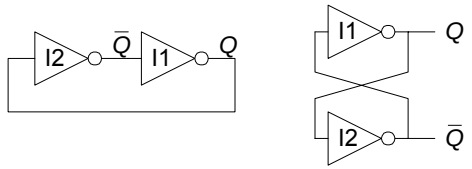
- give sequence to events
- have memory (short-term)
- use feedback from output to input to store information

State Elements

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

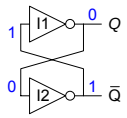
Bistable Circuit

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

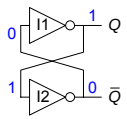


Bistable Circuit Analysis

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



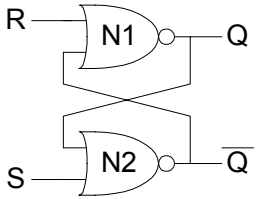
  - $Q = 1$: then $\overline{Q} = 0$ and $Q = 1$ (consistent)



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

SR Latch

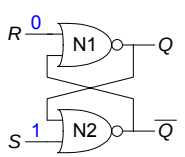
- SR Latch



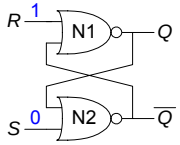
- Consider the four possible cases:
  - $S = 1, R = 0$
  - $S = 0, R = 1$
  - $S = 0, R = 0$
  - $S = 1, R = 1$

SR Latch Analysis

– $S = 1, R = 0$: then $Q = 1$ and $\overline{Q} = 0$



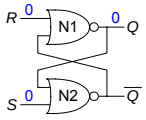
– $S = 0, R = 1$: then $Q = 0$ and $\overline{Q} = 1$



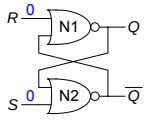
SR Latch Analysis

– $S = 0, R = 0$: then $Q = Q_{prev}$

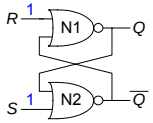
$Q_{prev} = 0$



$Q_{prev} = 1$



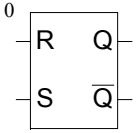
– $S = 1, R = 1$: then $Q = 0$ and $\overline{Q} = 0$



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
  - Reset: Make the output 0
  - When the set input, S, is 1 (and $R = 0$), Q is set to 1
  - When the reset input, R, is 1 (and $S = 0$), Q is reset to 0
  - Invalid state when $S = R = 1$

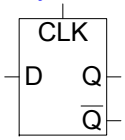
SR Latch Symbol



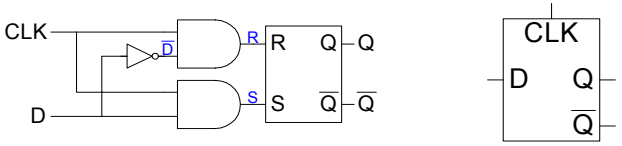
D Latch

- 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 $Q \neq$ NOT $\overline{Q}$

D Latch Symbol



D Latch Internal Circuit

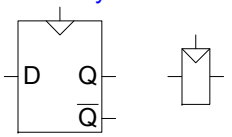


CLK	D	$\overline{D}$	S	R	Q	$\overline{Q}$
0	X					
1	0					
1	1					

D Flip-Flop

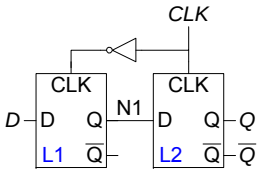
- Two inputs: *CLK*, *D*
- *Function*
  - The flip-flop "samples" *D* on the rising edge of *CLK*
    - When *CLK* rises from 0 to 1, *D* passes through to *Q*
    - Otherwise, *Q* holds its previous value
  - *Q* changes only on the rising edge of *CLK*
- A flip-flop is called an *edge-triggered* device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

D Flip-Flop Symbols

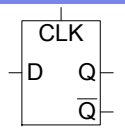


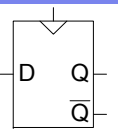
D Flip-Flop Internal Circuit

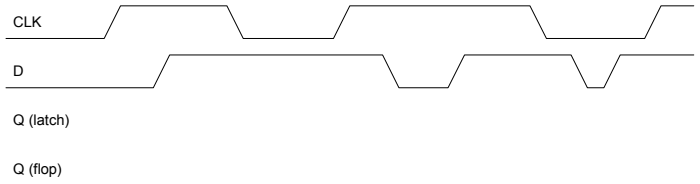
- Two back-to-back latches (L1 and L2) controlled by complementary clocks
- When *CLK* = 0
  - L1 is transparent
  - L2 is opaque
  - *D* passes through to N1
- When *CLK* = 1
  - L2 is transparent
  - L1 is opaque
  - N1 passes through to *Q*
- Thus, on the edge of the clock (when *CLK* rises from 0→1)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



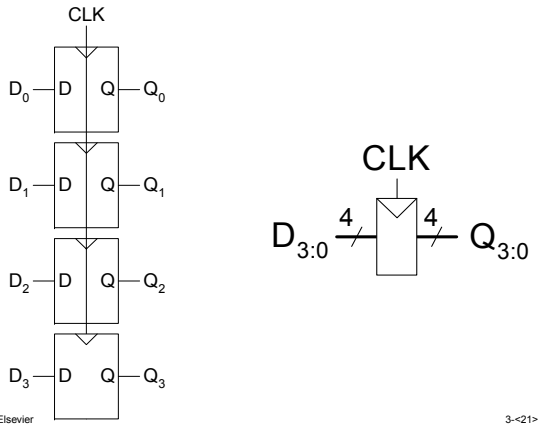
D Flip-Flop vs. D Latch







Registers

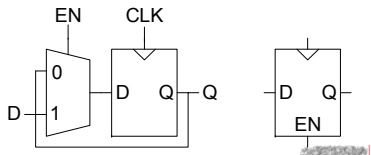


Enabled Flip-Flops

- 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

Internal Circuit

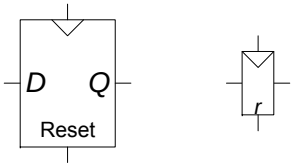
Symbol



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

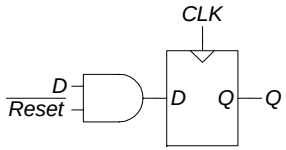
Symbols



Resettable Flip-Flops

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

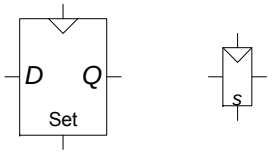
Internal Circuit



Settable Flip-Flops

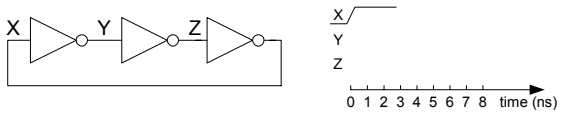
- Inputs: *CLK*, *D*, *Set*
- Funtion:
  - *Set* = 1
    - *Q* is set to 1
  - *Set* = 0
    - the flip-flop behaves like an ordinary D flip-flop

Symbols



Sequential Logic

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



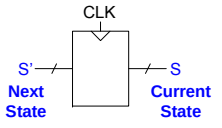
- This circuit has no inputs and 1-3 outputs

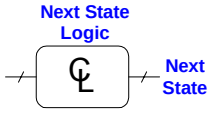
Synchronous Sequential Logic Design

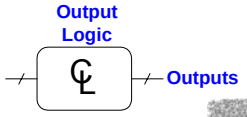
- Breaks cyclic paths by inserting registers
- These registers contain the state of the system
- The state changes at the clock edge, so we say the system is synchronized to the clock
- Rules of synchronous sequential circuit composition:
  - Every circuit element is either a register or a combinational 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

Finite State Machine (FSM)

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



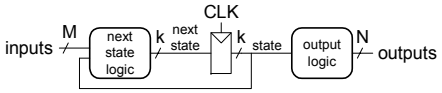




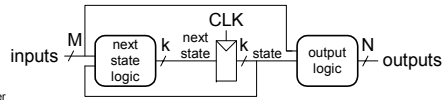
Finite State Machines (FSMs)

- 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

Moore FSM

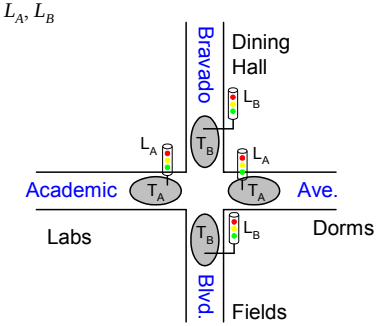


Mealy FSM



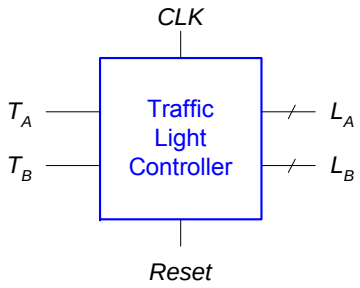
Finite State Machine Example

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



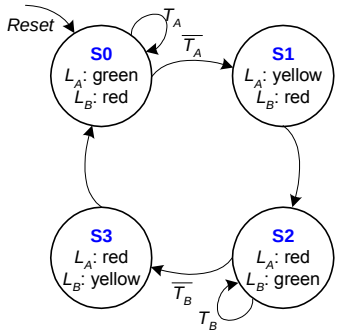
FSM Black Box

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



FSM State Transition Diagram

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



FSM State Transition Table

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

FSM Encoded State Transition Table

Current State S_1	S_0	Inputs T_A	T_B	Next State 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		

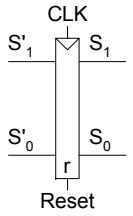
State	Encoding
S0	00
S1	01
S2	10
S3	11

FSM Output Table

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

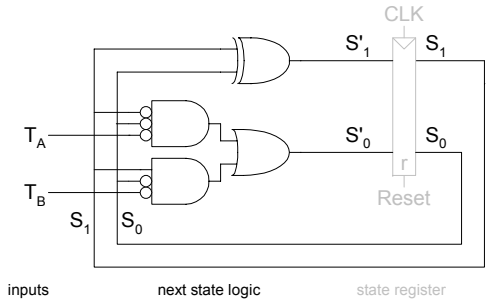
Output	Encoding
green	00
yellow	01
red	10

FSM Schematic: State Register

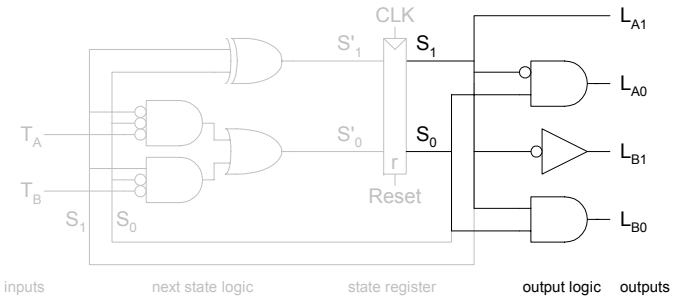


state register

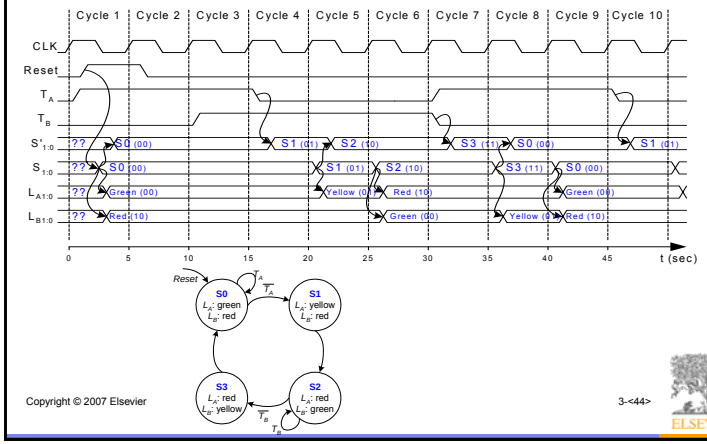
FSM Schematic: Next State Logic



FSM Schematic: Output Logic



FSM Timing Diagram



FSM State Encoding

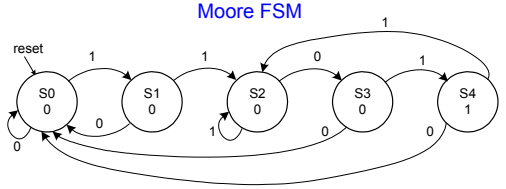
- Binary encoding: i.e., for four states, 00, 01, 10, 11
- One-hot encoding
  - One state bit per state
  - Only one state bit is HIGH at once
  - I.e., for four states, 0001, 0010, 0100, 1000
  - Requires more flip-flops
  - Often next state and output logic is simpler

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.

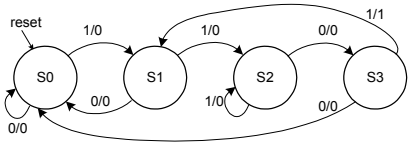
State Transition Diagrams

Moore FSM



Mealy FSM: arcs indicate input/output

Mealy FSM



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

Moore FSM Output Table

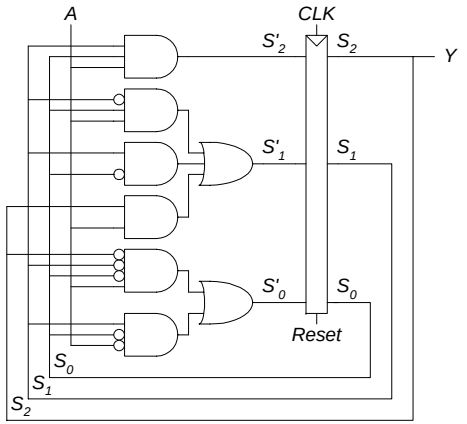
Current State			Output
S_2	S_1	S_0	Y
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	

Mealy FSM State Transition and Output Table

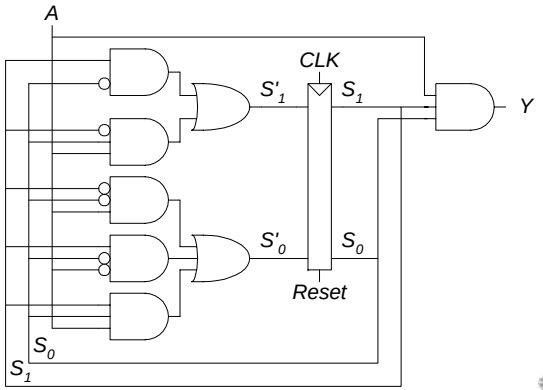
Current State		Input	Next State		Output
S_1	S_0	A	S'_1	S'_0	Y
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

State	Encoding
S0	00
S1	01
S2	10
S3	11

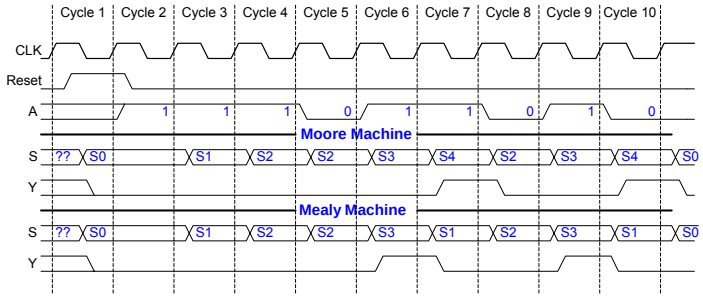
Moore FSM Schematic



Mealy FSM Schematic



Moore and Mealy Timing Diagram

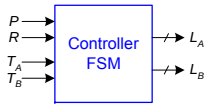


Factoring State Machines

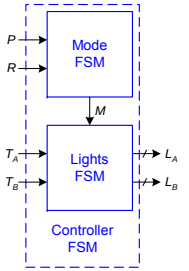
- Break complex FSMs into smaller interacting FSMs
- Example: Modify the traffic light controller to have a Parade Mode.
  - The FSM receives two more inputs: *P*, *R*
  - When *P* = 1, it enters Parade Mode and the Bravado Blvd. light stays green.
  - When *R* = 1, it leaves Parade Mode

Parade FSM

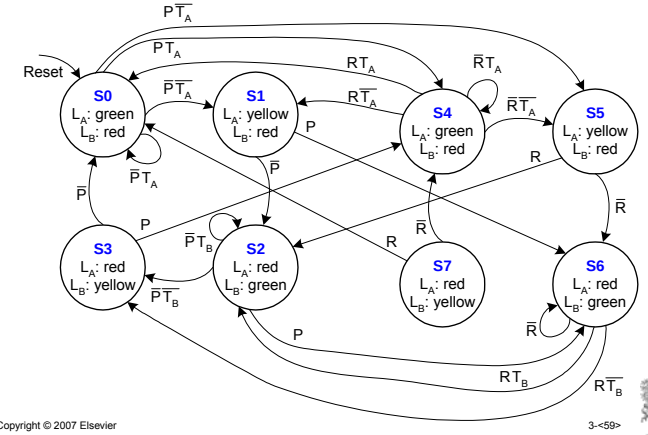
Unfactored FSM



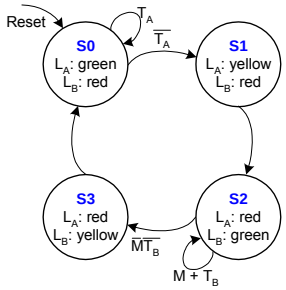
Factored FSM



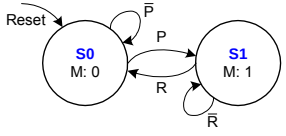
Unfactored FSM State Transition Diagram



Factored FSM State Transition Diagram



Lights FSM



Mode FSM

FSM Design Procedure

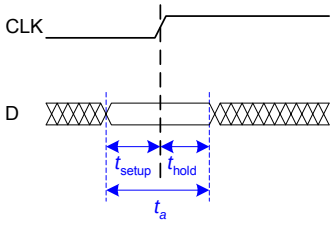
- Identify the inputs and outputs
- Sketch a state transition diagram
- Write a state transition table
- Select state encodings
- For a Moore machine:
  - Rewrite the state transition table with the selected state encodings
  - Write the output table
- For a Mealy machine:
  - Rewrite the combined state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

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

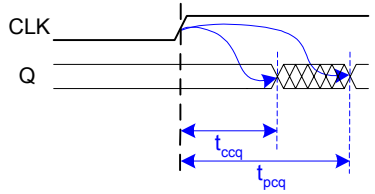
Input Timing Constraints

- 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 ($t_a = t_{setup} + t_{hold}$)



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)

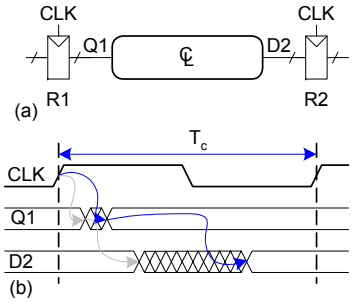


Dynamic Discipline

- 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

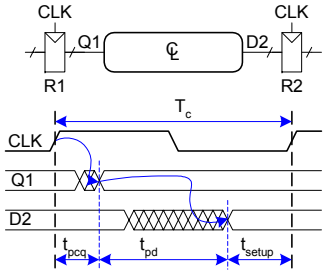
Dynamic Discipline

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



Setup Time Constraint

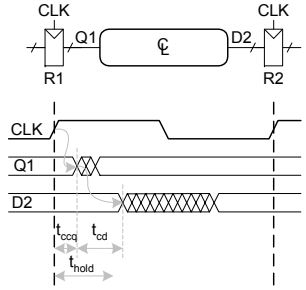
- The setup time constraint depends on the **maximum** delay from register R1 through the combinational logic.
- The input to register R2 must be stable at least t_{setup} before the clock edge.



$T_c \geq$

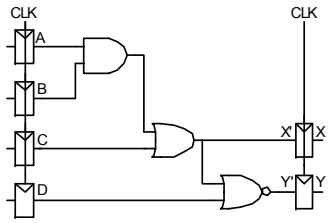
Hold Time Constraint

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



$t_{hold} <$

Timing Analysis



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} =
t_{cd} =

Setup time constraint:

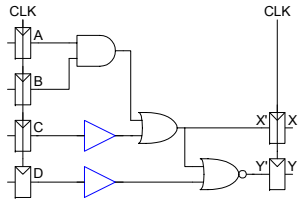
$T_c \geq$
$f_c = 1/T_c =$

Hold time constraint:

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

Fixing Hold Time Violation

Add buffers to the short paths:



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} =
t_{cd} =

Setup time constraint:

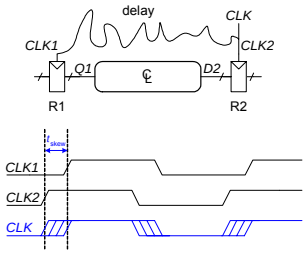
$T_c \geq$
$f_c =$

Hold time constraint:

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

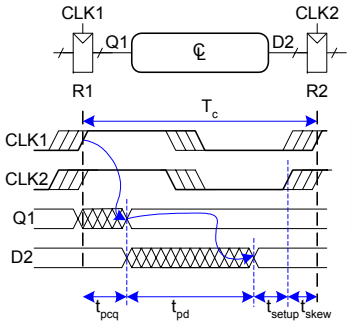
Clock Skew

- The clock doesn't arrive at all registers at the same time
- This may be caused by delay or other timing noise
- Skew is the difference between two clock edges
- Because there may be many registers in a system, we examine the worst case for each case to guarantee that the dynamic discipline is not violated for any register



Setup Time Constraint with Clock Skew

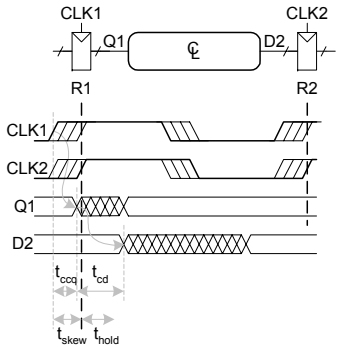
- In the worst case, the CLK2 is earlier than CLK1



$T_c \geq$

Hold Time Constraint with Clock Skew

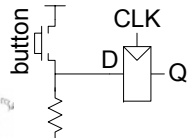
- In the worst case, CLK2 is later than CLK1

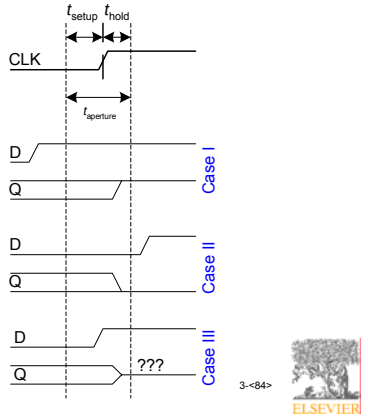


$t_{ccq} + t_{cd} > t_{cd} >$

Violating the Dynamic Discipline

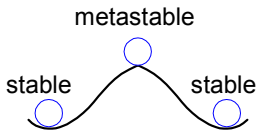
- Asynchronous (for example, user) inputs might violate the dynamic discipline





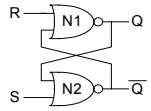
Metastability

- 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



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

Metastability

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

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

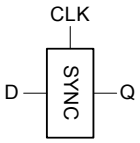
  - τ is a time constant indicating how fast the flip-flop moves away from the metastable state; it is related to the delay through the cross-coupled gates in the flip-flop

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

- 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.

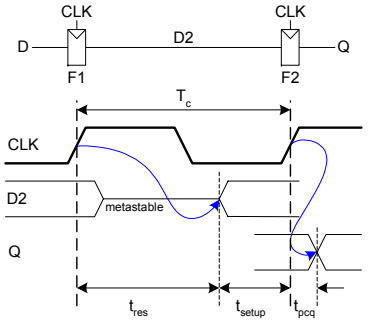
Synchronizers

- 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.



Synchronizer Internals

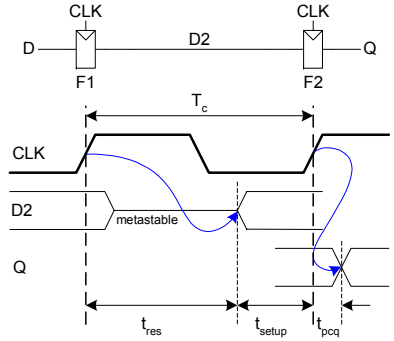
- A synchronizer can be built with two back-to-back flip-flops.
- Suppose the input D is transitioning when it is sampled by flip-flop 1, F1.
- The amount of time the internal signal D2 can resolve to a 1 or 0 is (T_c - t_{setup}).



Synchronizer Probability of Failure

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

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



Synchronizer Mean Time Before Failure

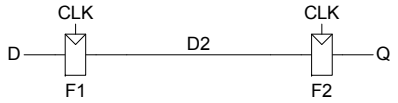
- If the asynchronous input changes once per second, the probability of failure per second of the synchronizer is simply P(failure):
- In general, if the input changes N times per second, the probability of failure per second of the synchronizer is:

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

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

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

Example Synchronizer



- Suppose: T_c = 1/500 MHz, τ = 200 ps, T_0 = 150 ps, t_{setup} = 100 ps, N = 10 events per second
- What is the probability of failure? MTBF?

P(failure) =

P(failure)/second =

MTBF =

Parallelism

- 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.
- 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

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?

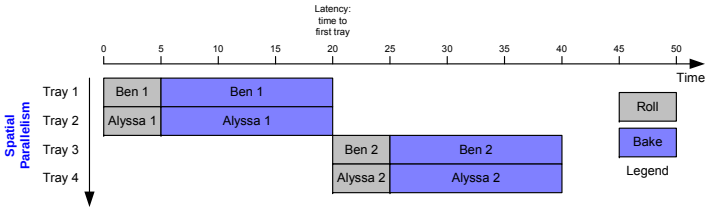
Latency = 5 + 15 = 20 minutes = 1/3 hour

Throughput = 1 tray/ 1/3 hour = 3 trays/hour

Parallelism Example

- What is the latency and throughput if Ben uses parallelism?
  - **Spatial parallelism:** Ben asks Allysa 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.

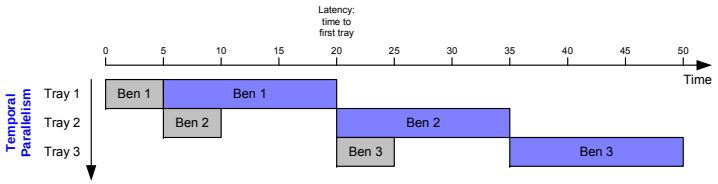
Spatial Parallelism



Latency =

Throughput =

Temporal Parallelism



Latency =

Throughput =