

Memory Elements

- Combinational logic cannot remember
 - Output logic values are function of inputs only
 - Feedback is needed to be able to remember a logic value
- Memory elements are needed in most digital logic circuits to hold (remember) logic values
- 2 basic types of memory elements
 - Latches
 - *Level-sensitive* to inputs
 - Flip-flops
 - *Edge-triggered* on active edge of clock

Reset-Set (RS) Latch (NOR)

- The simplest memory element

- Aka set-reset (SR) latch

- Cross-coupled NOR gates

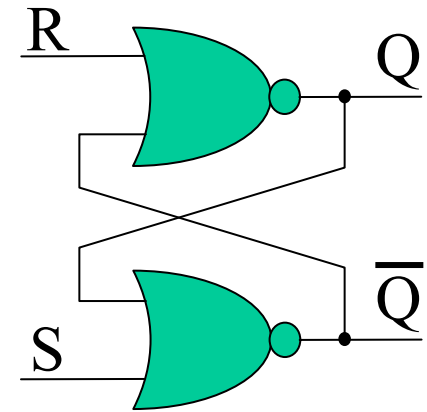
- Level sensitive

- Active high inputs

- R (reset)
 - S (set)
 - Only one input can be active
 - ✓ To avoid undefined state

- Outputs: Q and Q'

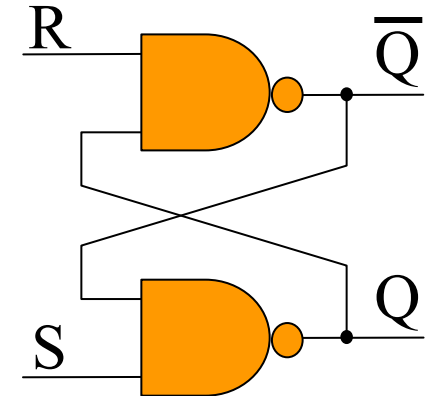
- Q = current state of latch



R	S	Q	Q'	Function
0	0	Q	Q'	Storage
0	1	1	0	Set
1	0	0	1	Reset
1	1	0-?	0-?	Undefined

Reset-Set (RS) Latch (NAND)

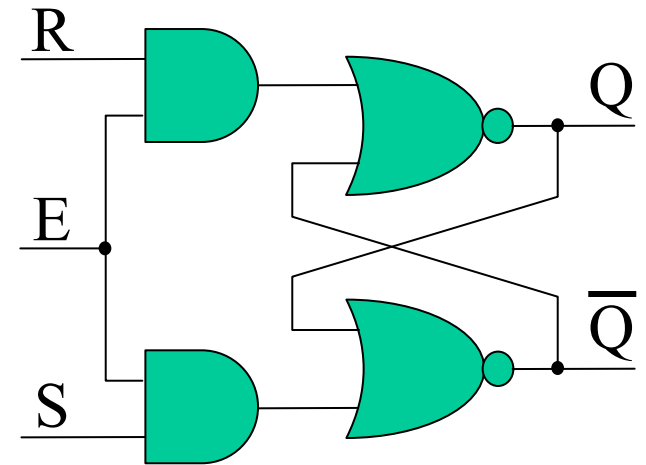
- Dual of NOR RS latch
- Cross-coupled NAND gates
 - Level sensitive
 - Active low inputs
 - R (reset)
 - S (set)
 - Only one input can be active
 - ✓ To avoid undefined state
 - Outputs: Q and Q'
 - Q = current state of latch



R	S	Q	Q'	Function
0	0	1-?	1-?	Undefined
0	1	0	1	Reset
1	0	1	0	Set
1	1	Q	Q'	Storage

Enabled Reset-Set (RS) Latch (NOR)

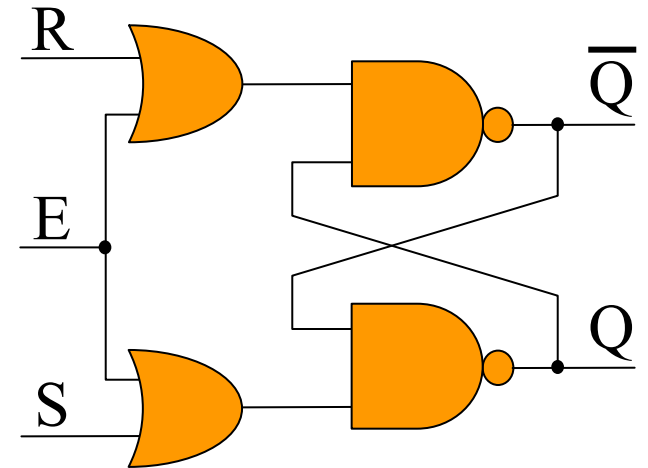
- Aka gated RS latch
 - When enable E is inactive, RS latch is forced into storage state
 - R and S can do nothing
- AND gates plus NOR RS latch
 - Level sensitive
 - Active high inputs
 - E (enable)
 - R (reset)
 - S (set)
 - R and S cannot both be active when E is active
 - ✓ To avoid undefined state
 - Outputs: Q and Q'
 - Q = current state of latch



E	R	S	Q	Q'	Function
0	X	X	Q	Q'	Storage
1	0	0	Q	Q'	Storage
1	0	1	1	0	Set
1	1	0	0	1	Reset
1	1	1	0-?	0-?	Undefined

Enabled Reset-Set (RS) Latch (NAND)

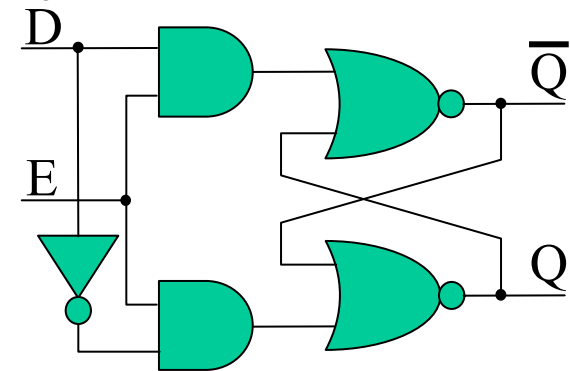
- Aka gated RS latch
 - When enable E is inactive, RS latch is forced into storage state
 - R and S can do nothing
- OR gates plus NAND RS latch
 - Level sensitive
 - Active low inputs
 - E (enable)
 - R (reset)
 - S (set)
 - R and S cannot both be active when E is active
 - ✓ To avoid undefined state
 - Outputs: Q and Q'
 - Q = current state of latch



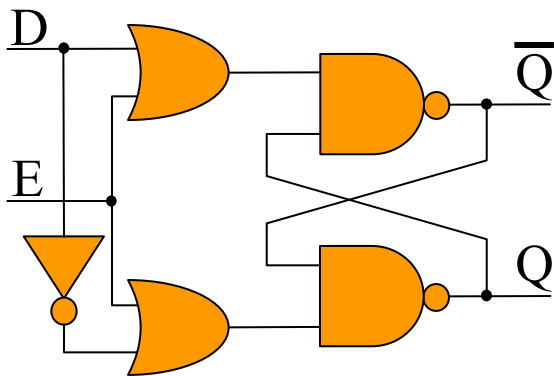
E	R	S	Q	Q'	Function
1	X	X	Q	Q'	Storage
0	0	0	1-?	1-?	Undefined
0	0	1	0	1	Reset
0	1	0	1	0	Set
0	1	1	Q	Q'	Storage

Enabled Data or Delay (D) Latch

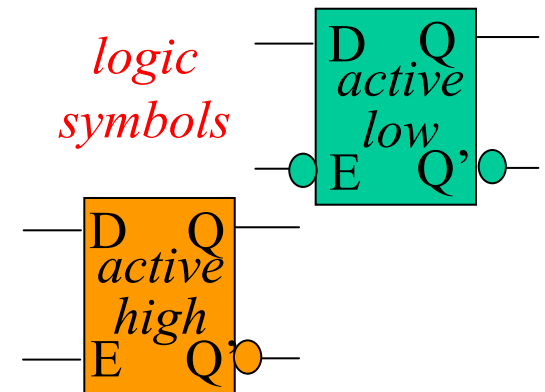
- Aka transparent D latch
 - Overcomes undefined state
 - R & S never active at same time
- Inverter plus enabled RS latch
 - Level sensitive
 - Active high enable for NOR latch
 - Active low enable for NAND latch



E	D	Q	Q'	Function
0	X	Q	Q'	Storage
1	0	0	1	Transparent
1	1	1	0	Transparent

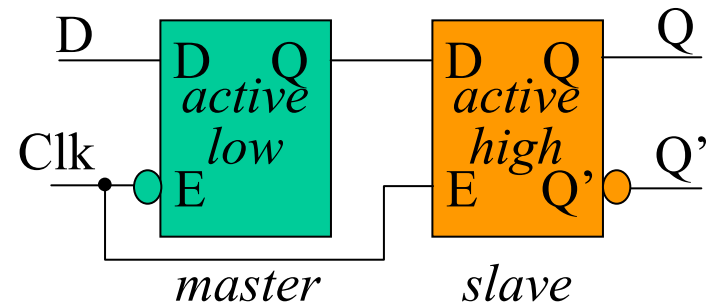


E	D	Q	Q'	Function
0	0	0	1	Transparent
0	1	1	0	Transparent
1	X	Q	Q'	Storage



D Flip-Flop

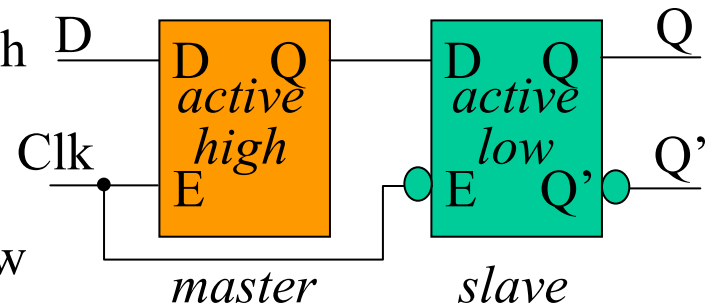
- Aka Master-Slave flip-flop
- Two transparent D latches
 - Sensitive to opposite levels of Clock
 - One is always in storage and the other transparent



master transparent master storage
slave storage slave transparent

rising edge

- Edge-triggered
 - Data moves through on Clock transition
 - Active-low latch followed by active-high
 - Rising edge-triggered
 - ✓ aka leading edge-triggered
 - Active-high latch followed by active-low
 - Falling edge-triggered
 - ✓ aka trailing edge-triggered



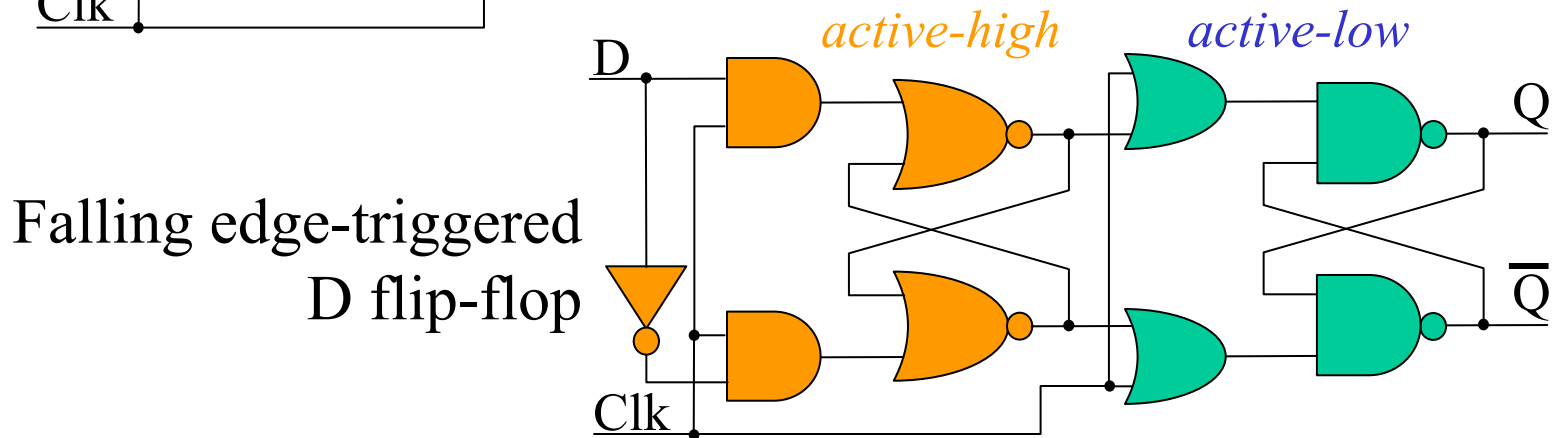
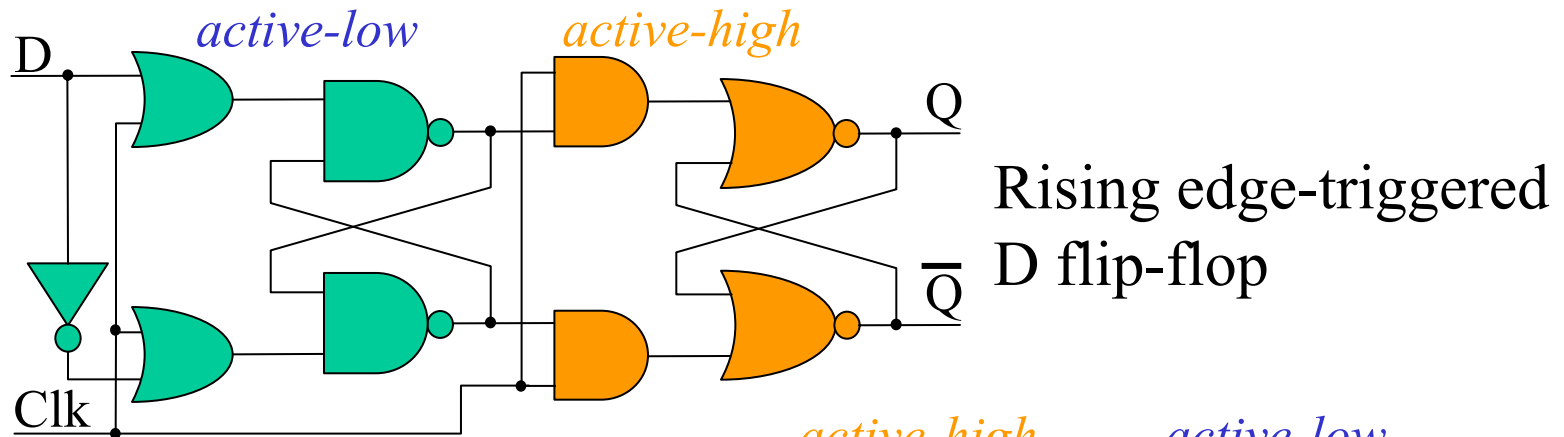
master transparent master storage
slave storage slave transparent

falling edge

D Flip-Flop

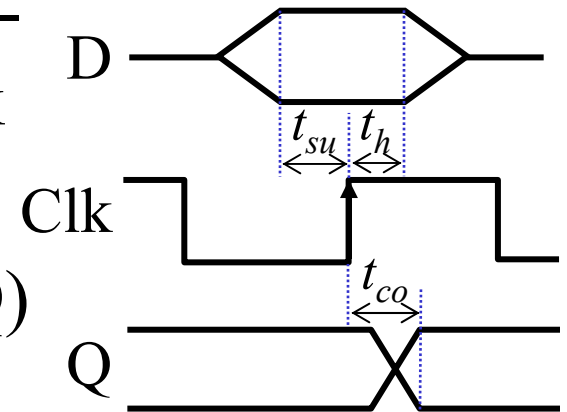
- Gate-level implementation

➤ No need for inverter in slave latch since master has Q & Q'



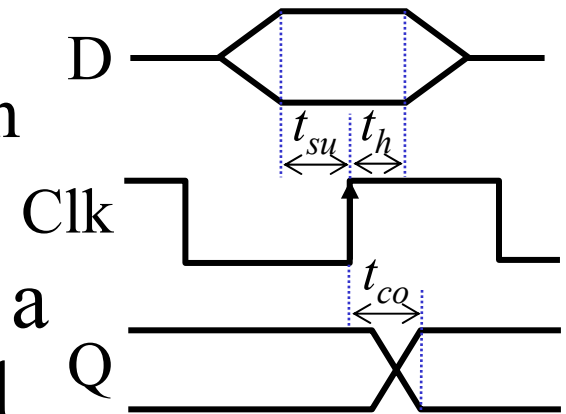
Timing Considerations

- Set-up time (t_{su}) = minimum time data (D) must be valid at input to flip-flop prior to the active edge of the clock
- Hold time (t_h) = minimum time data (D) must remain valid at input to flip-flop after the active edge of the clock
- Clock-to-output delay (t_{co}) = maximum time before output data (Q) is valid after the active edge of the clock



Timing Considerations

- Set-up & hold time violations in a real circuit result in metastability
 - Flip-flop goes to intermediate logic levels ($Q = Q'$)
 - Eventually resolves to an unknown state
- Set-up & hold time violations in a vector set for simulation referred to as clock-data-races
 - Leads to invalid simulation results & manufacturing testing problems



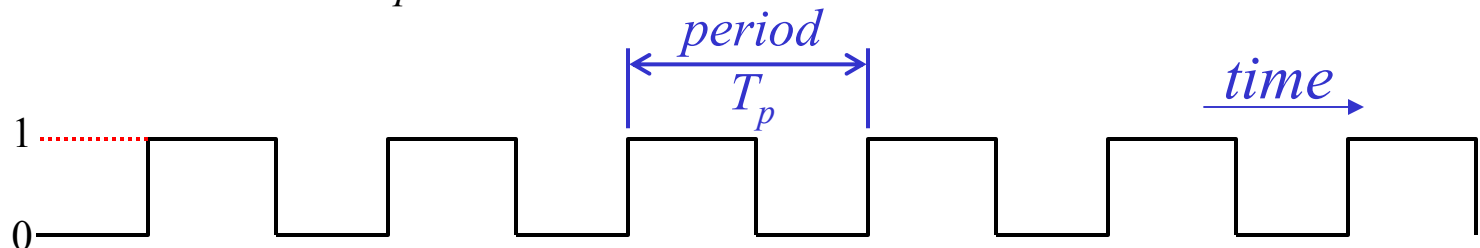
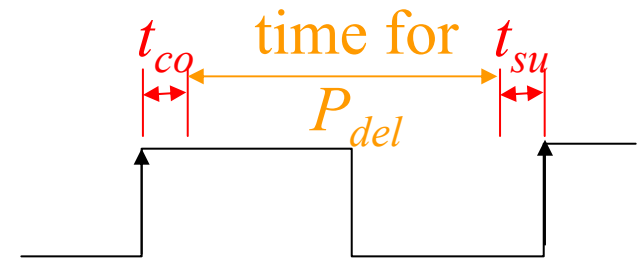
What is the Clock?

- Typically a periodic signal (a sequence of pulses) used to:
 - sample data, *and*
 - store the sampled data in memory elements
- *Clock frequency = 1/period*

- $f_{clk} = 1/T_p$

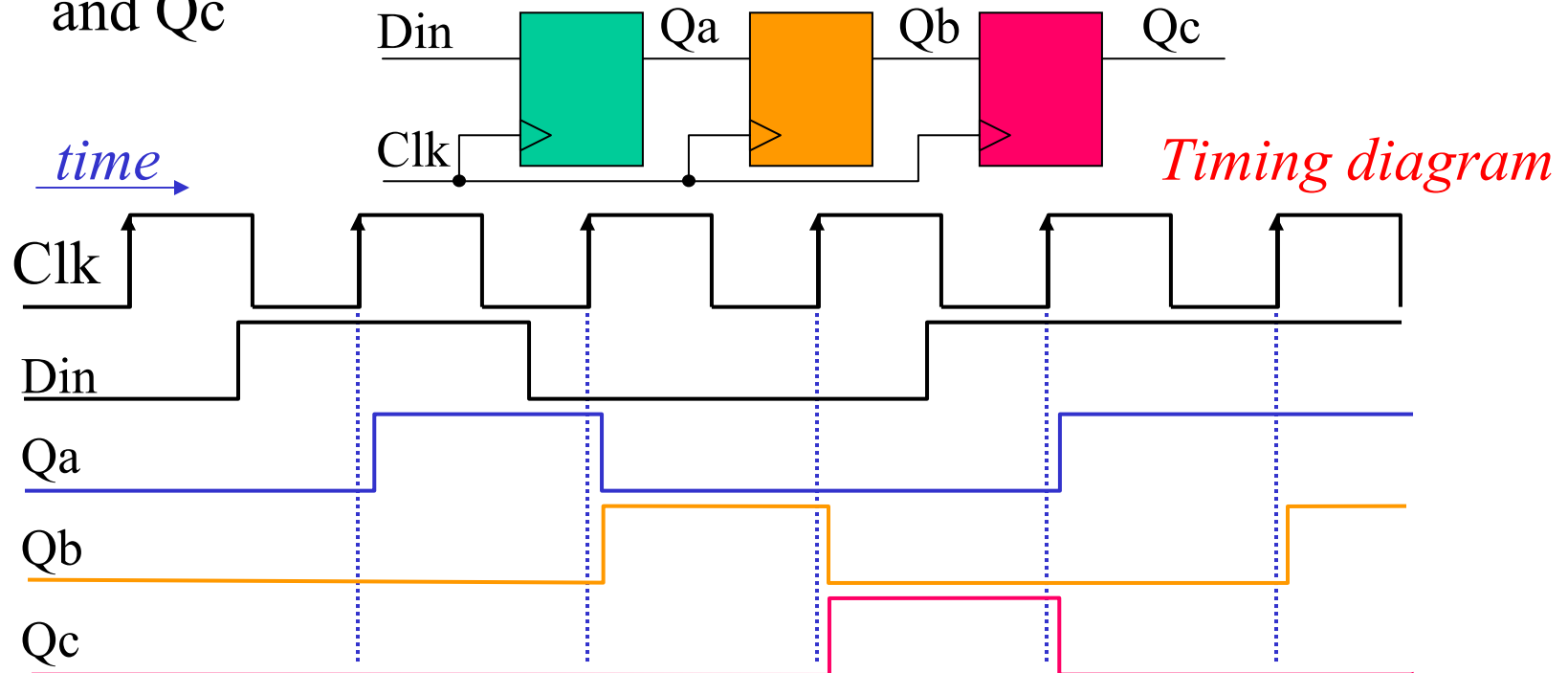
- $T_p \geq t_{co} + P_{del} + t_{su}$

- $P_{del} \leq T_p - t_{co} - t_{su}$



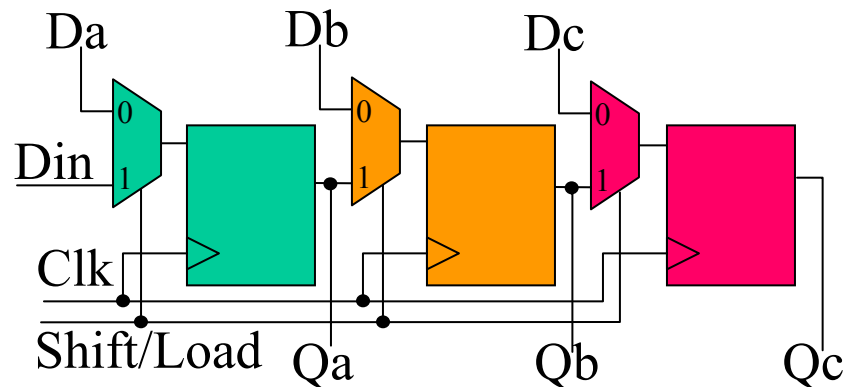
Serial Shift Register Example

- A series of D flip-flops whose outputs are connected to the input of the next flip-flop
 - *serial-in, serial-out* = data in on Din, data out on Qc
 - *serial-in, parallel-out* = data in on Din, data out on Qa, Qb, and Qc

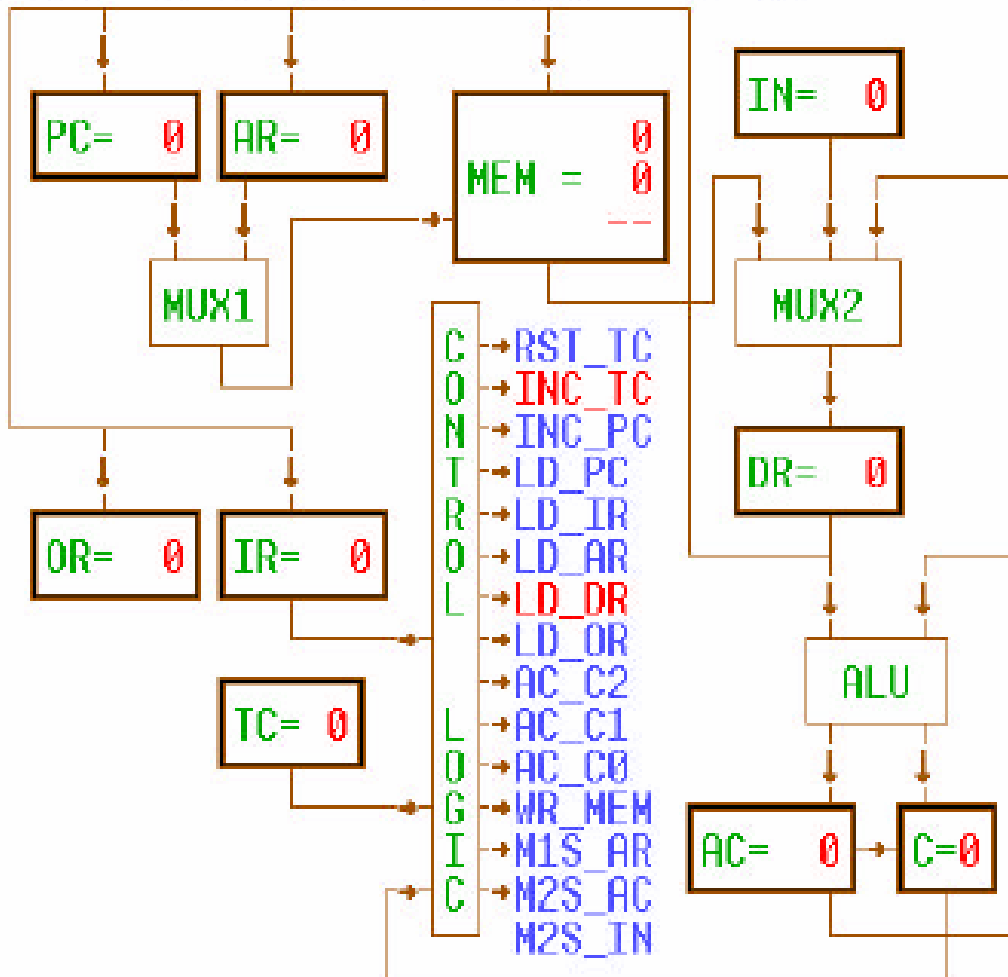


Another Shift Register Example

- A series of multiplexers and D flip-flops whose outputs are connected to the input of the next flip-flop
 - *parallel-in, parallel-out* = data in on Da, Db, and Dc; data out on Qa, Qb, and Qc (Shift/Load = 0)
 - *parallel-in, serial-out* = data in on Da, Db, and Dc; data out on Qc (Shift/Load = 0, then Shift/Load = 1)
 - *Serial-in, serial-out* = data in on Din, data out on Qc (Shift/Load = 1)
 - *Serial-in, parallel-out* = data in on Din, data out on Qa, Qb, and Qc (Shift/Load = 1)



PSIM Architecture



Sequential Logic:

Program Memory (MEM)

Program Counter (PC)

Address Register (AR)

Data Register (DR)

Input Register (IN)

Output Register (OR)

Accumulator (AC)

ALU Carry Register (C)

Instruction Register (IR)

Timing Counter (TC)

Combinational Logic:

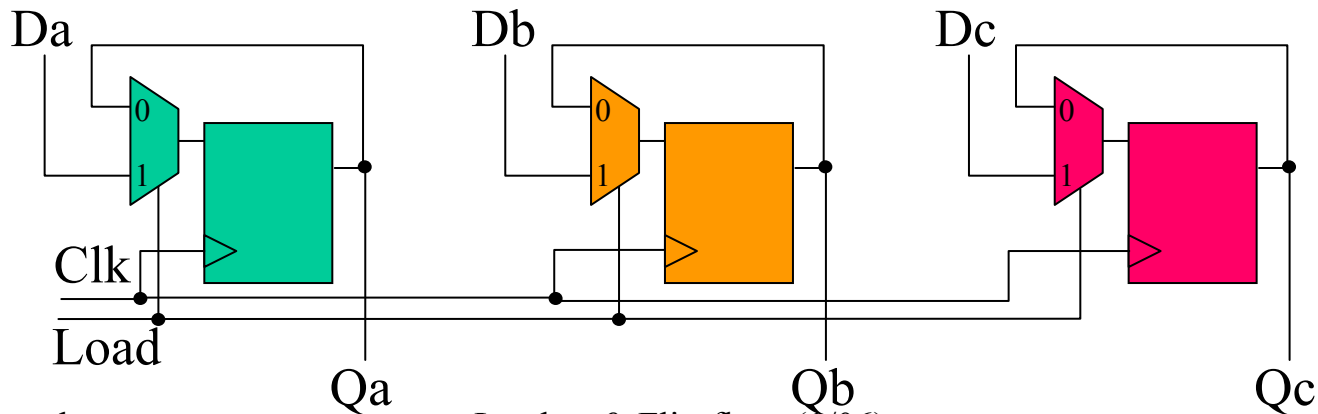
Control Logic

Arithmetic/Logic Unit (ALU)

Multiplexers 1&2 (MUX)

Another Register Example

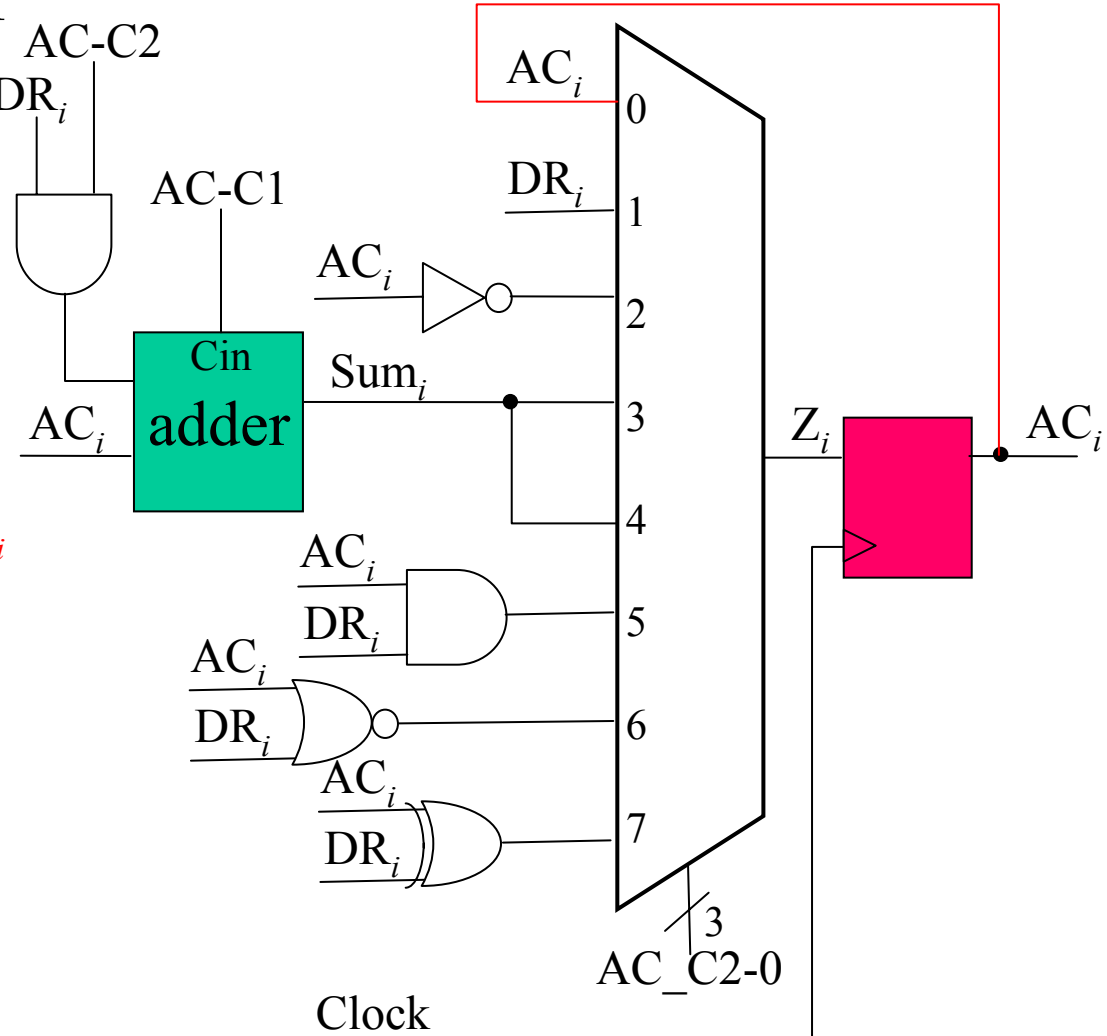
- A series of multiplexers and D flip-flops whose outputs are connected to the input of the MUX
 - Register with active high Load
 - **Load = 1 & rising edge of clock:** *parallel-in, parallel-out* = data in on Da, Db, and Dc; data out on Qa, Qb, and Qc
 - **Otherwise:** Holds data; data out remains on Qa, Qb, and Qc
 - Basic register design used in PSIM for:
 - AR, DR, OR, IN (all 8-bits) and IR (4-bits)



Accumulator Register Example

- Accumulator in PSIM

- Functions controlled by combinational logic design
 - Including holding data when no operations are specified
 - ✓ Via feedback of AC_i
- Only need a flip-flop at output of MUX
 - AC register (8-bits)
 - C register (1-bit)
 - ✓ Similar to AC_i design shown here



Random Access Memory (RAM)

- Assuming MEM from PSIM

- 8-bit address => 256 words

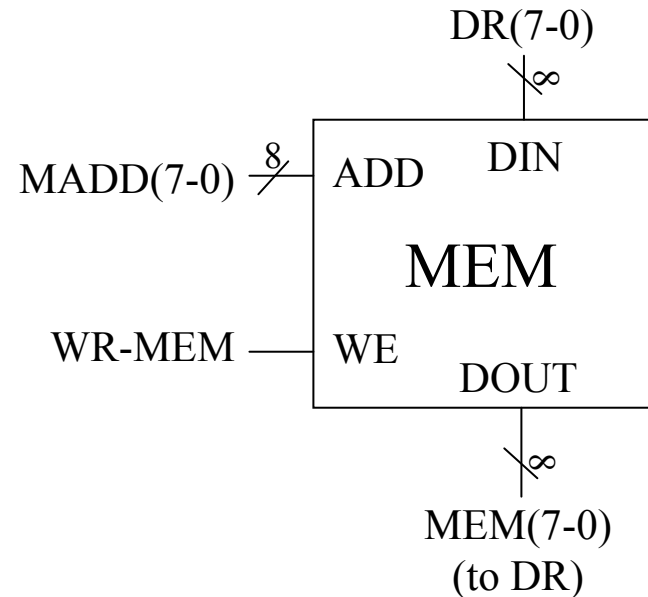
- MADD

- 8-bit words

- Input data = 8-bits
 - ✓ From DR
- Output data = 8-bits
 - ✓ From MEM

- Active high write enable

- WR-MEM
 - ✓ When WR-MEM = 1, data from DR is written into address location specified by MADD



RAM continued

- RAM consists of:

- Address decoder with enable

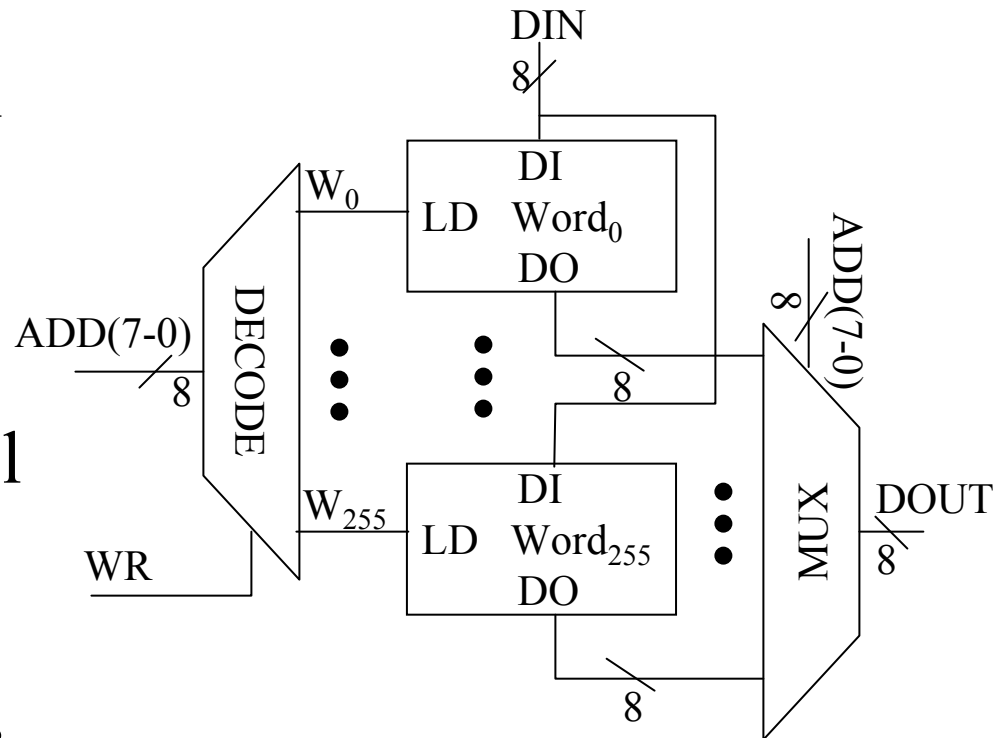
- Produces active high enables to registers

- Registers with parallel load

- Stores data associated with specified address

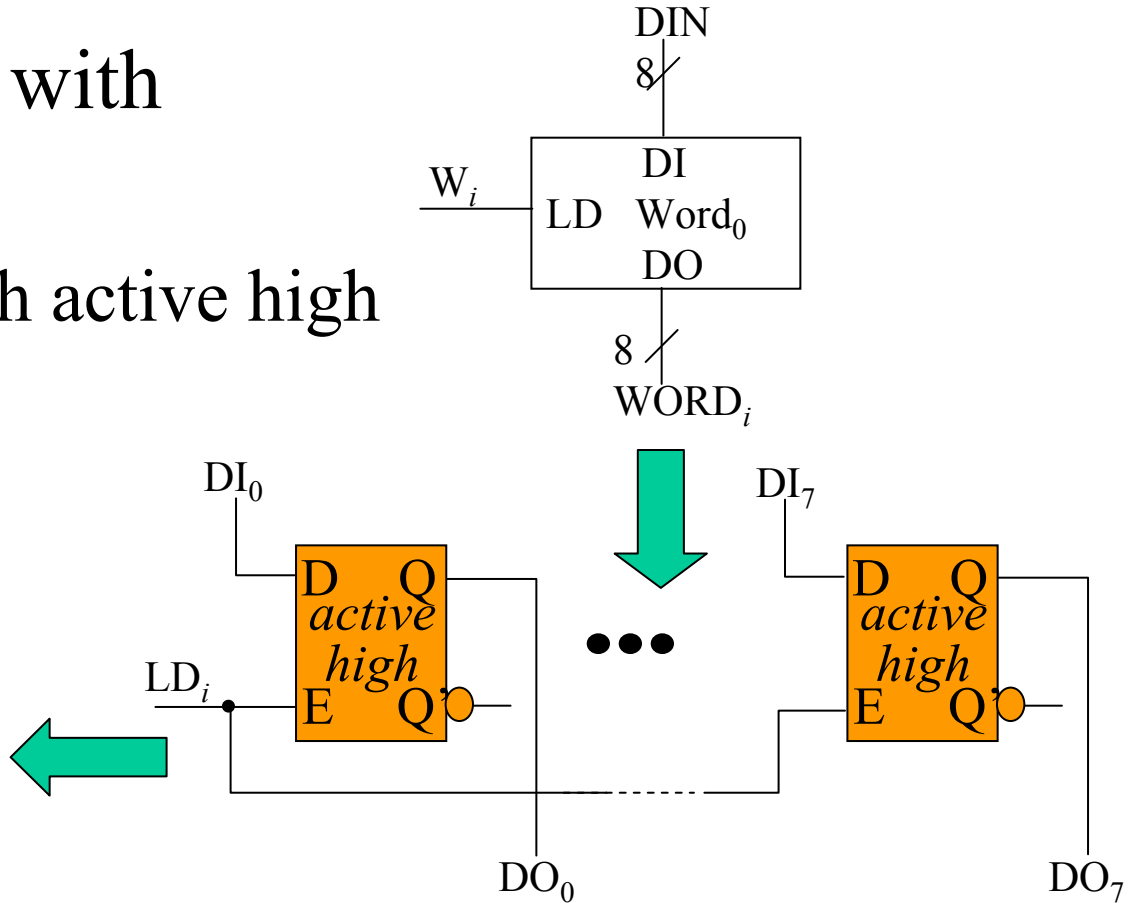
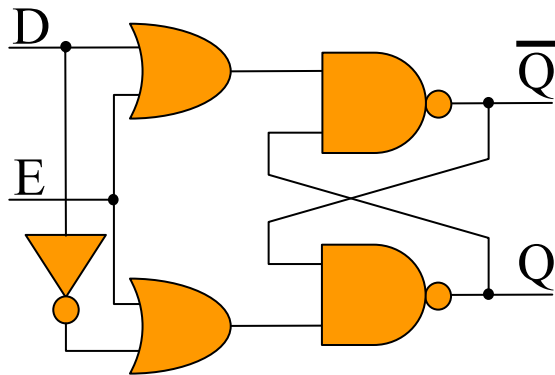
- Read MUX

- Reads specified address



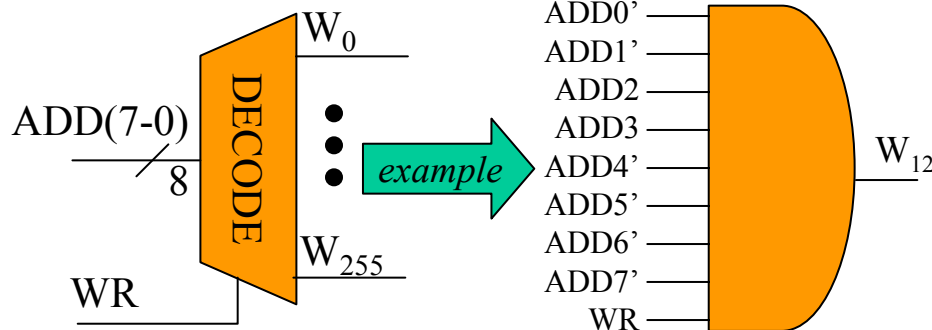
RAM continued

- Word Registers with parallel load
 - 8 D-latches with active high enable

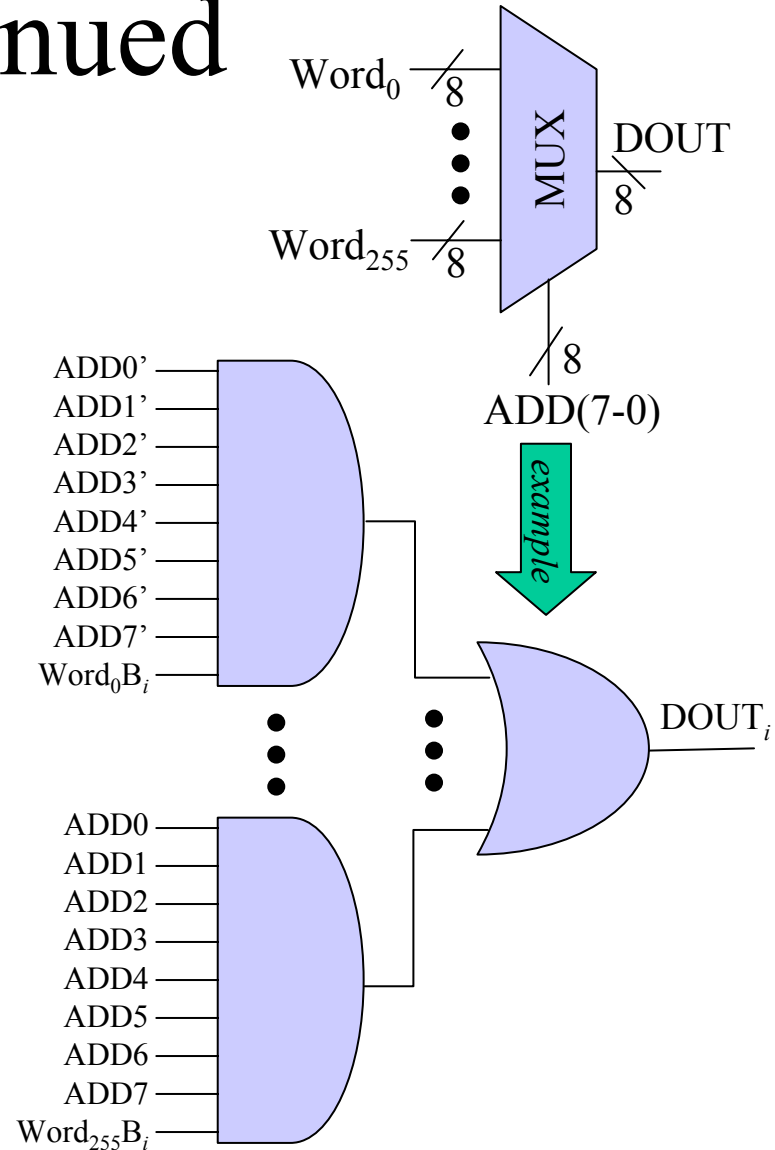


RAM continued

- Read MUX
 - 8 256-to-1 MUXs
 - Functional equivalent
- Address decoder
 - 256 9-input AND gates
 - 8 inverters



C. E. Stroud

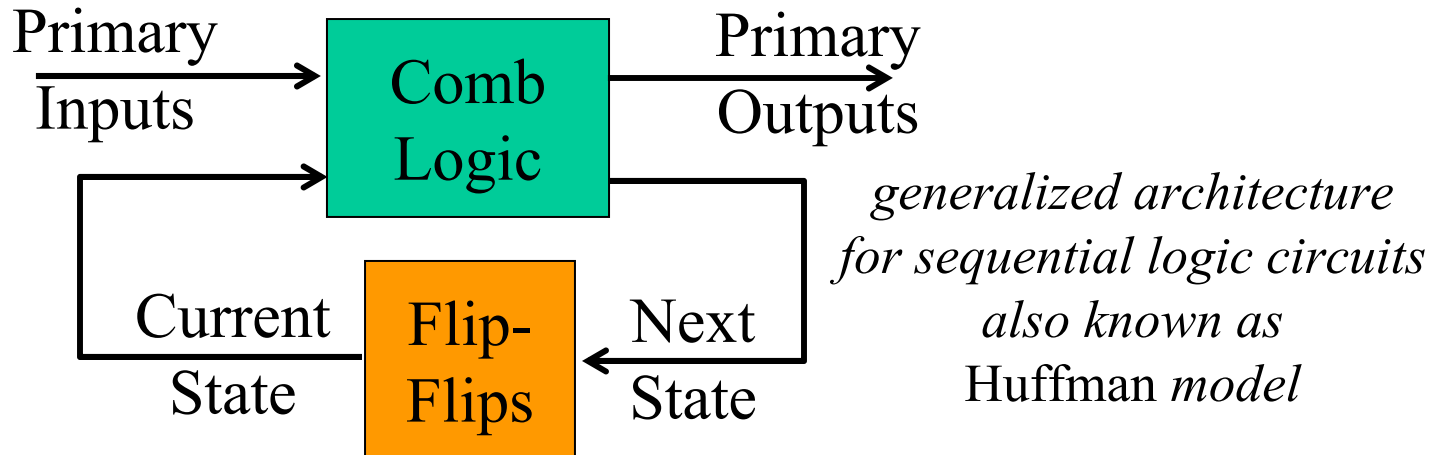


Latches & Flip-flops (1/06)

20

What is Sequential Logic?

- A collection of logic gates and flip-flops
 - The logic values stored in the flip-flops establish the *current state* of the sequential logic circuit
 - The logic values at the inputs in conjunction with the current state determines the next state of the sequential logic circuit after the active edge of the clock



Flip-Flop Information for Sequential Logic Design

- Types of flip-flops
 - D (data)
 - T (toggle)
 - SR (set-reset)
 - Also known as RS (reset-set)
 - JK (Jack Kilby)
- Each type has associated:
 - Characteristic equation
 - Characteristic table
 - sometimes called state table
 - State diagram
 - Excitation table

We will consider only edge-triggered flip-flops

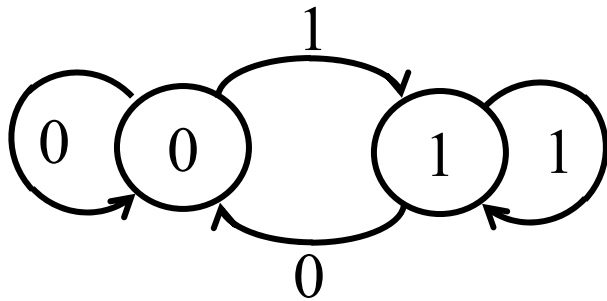
All provide same basic information but in slightly different forms

State Diagrams & State Tables

- Describe complete operation of sequential logic circuit
 - Vertices (nodes) represent states
 - Edges represent state transitions on active edge of clock based on primary input logic values
- State diagram & state tables provide exact same information
 - Diagram is graphical representation of same info as in state table
- Given current state and primary input values we can determine the next state after active edge of clock

D Flip-Flop Specification

state diagram

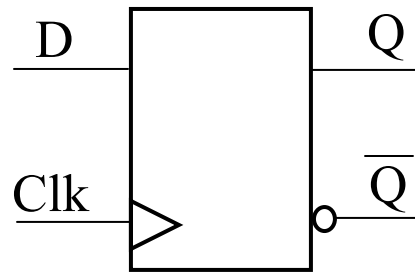


characteristic equation

$$Q^+ = D$$

characteristic table

D	Q ⁺
0	0
1	1



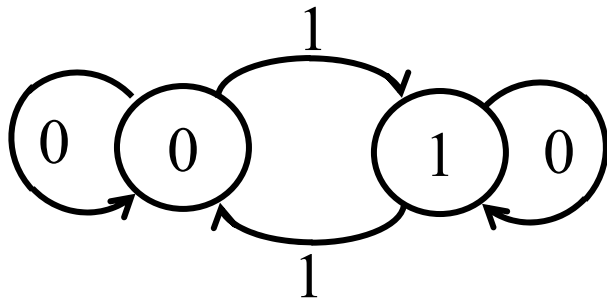
logic diagram

excitation table

Q	Q ⁺	D
0	0	0
0	1	1
1	0	0
1	1	1

T Flip-Flop Specification

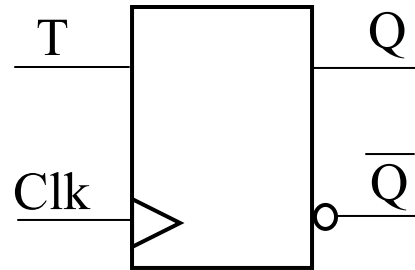
state diagram



characteristic equation

$$Q^+ = TQ' + T'Q$$

$$= T \oplus Q$$



logic diagram

excitation table

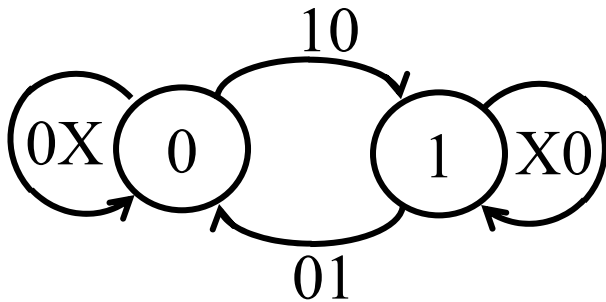
Q	Q ⁺	T
0	0	0
0	1	1
1	0	1
1	1	0

characteristic table

T	Q ⁺	Mode
0	Q	Storage
1	Q'	Toggle

RS Flip-Flop Specification

state diagram



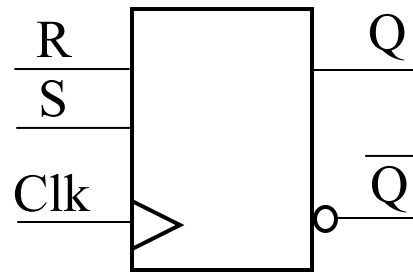
input ordering = SR

characteristic table

S	R	Q+	Mode
0	0	Q	Storage
0	1	0	Reset
1	0	1	Set
1	1	?	Indeterminant

characteristic equation

$$Q^+ = S + R'Q$$



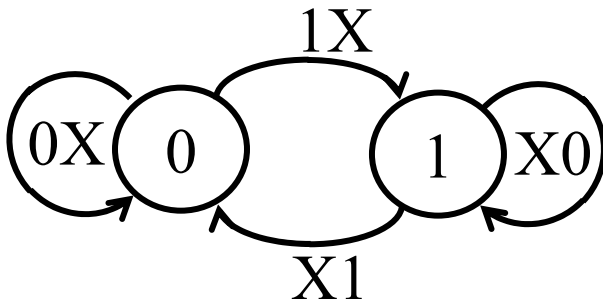
logic diagram

excitation table

Q	Q+	SR
0	0	0X
0	1	10
1	0	01
1	1	X0

JK Flip-Flop Specification

state diagram



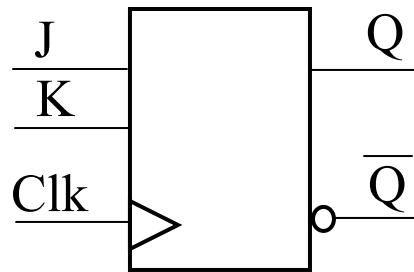
input ordering = JK

characteristic table

J	K	Q+	Mode
0	0	Q	Storage
0	1	0	Reset
1	0	1	Set
1	1	Q'	Toggle

characteristic equation

$$Q^+ = JQ' + K'Q$$



logic diagram

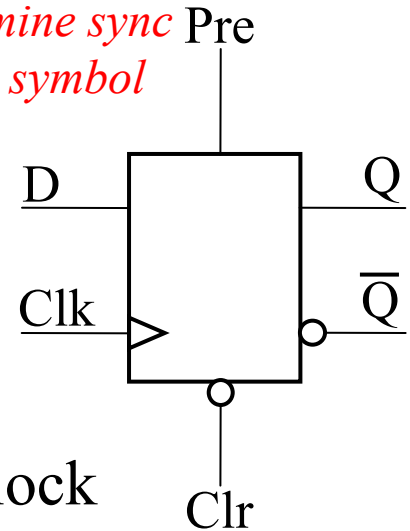
excitation table

Q	Q+	JK
0	0	0X
0	1	1X
1	0	X1
1	1	X0

Flip-Flop Initialization

- Preset (aka set) $\Rightarrow Q^+ = 1$
- Clear (aka reset) $\Rightarrow Q^+ = 0$
- Some flip-flops have:
 - Both preset and clear (set and reset)
 - A preset or a clear
 - Neither (JK & SR flops have set/reset functions)
- Preset and/or clear can be
 - Active high or active low
 - Synchronous \Rightarrow with respect to active edge of clock
 - Asynchronous \Rightarrow independent of clock edges
- Initialization important for:
 - logic simulation to remove undefined logic values (2, 3, U, etc.)
 - system operation to put system in a known state

*Typical logic symbol
with active high preset
and active low clear
Cannot determine sync
or async from symbol*



Synchronous vs. Asynchronous

- Synchronous => states of memory elements change only with respect to active edge of clock
- Asynchronous => states of memory elements can change without an active edge of clock
 - Asynchronous designs often have timing problems

*Example:
assume sync preset
and async clear*

