Midterm 2 review

Thursday, March 29, 2018 7:50 PM

D Flip-Flop

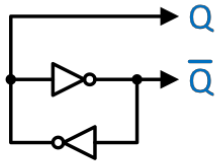
Combinational circuit:

Output only depends on current input values

Functions are limited because of the lack of storage elements.

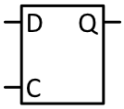
Storage structure:

- Once we can get a value into this structure, it stays there as long as the circuit has power



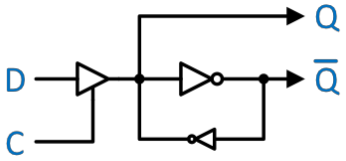
(We can store either a 0 or 1 in this structure.)

To control the storage structure, we can add a tristate buffer.



D latch symbol representation

D latch(Q=D when enabled):



C: control signal that enables the tristate buffer

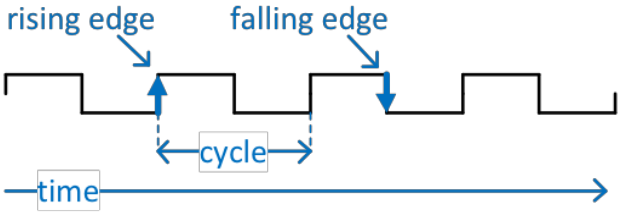
D: input

Q: output (Q bar: negated output)

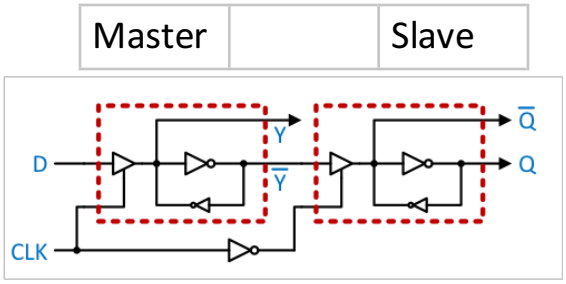
Level-sensitive: the value held in the latch can change whenever signal C enables the tristate buffer.

Transparent: when enabled, output = input

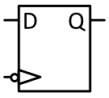
Level sensitivity is a problem because it makes the circuit unpredictable!
To keep control and have deterministic behavior:
We need clock: a special signal that oscillates btw 1 and 0 periodically at a specific frequency (Hz, # cycles/s)



After we use clock as control signal and separate the output logic (Q and not Q)
We can build the D flip-flop like this:



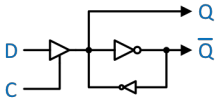
Leftside BLOCK updates on rising edge
Rightside BLOCK updates on falling edge



(symbol representation of negative edge triggering D flipflop or negative edge sensitive ff)

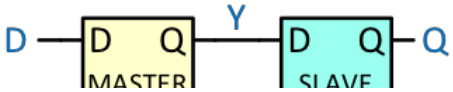
Difference btw D latch and D flip-flop:

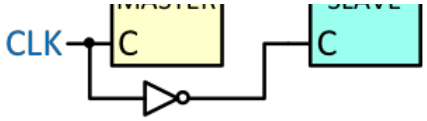
D-latch: <u>two update per clock cycle</u>



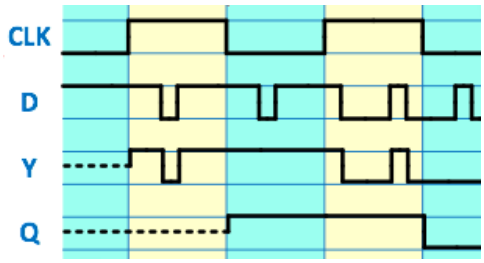
unified CLK signal

D flip-flop: <u>one update per CLK cycle</u>; <u>composed of two latches.</u>





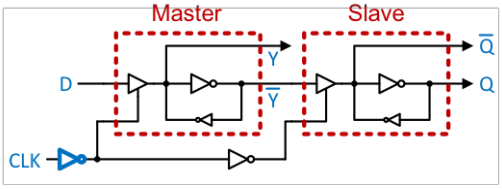
D: input of MASTER
Y: output of MASTER
input of SLAVE
Q: output of slave
output of the negative edge triggered D-FlipFlop
Only one latch will be changed given a input CLK signal.



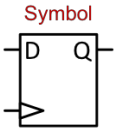
Y = D when CLK= 1
Q = Y when CLK = 0
Dashed line btw 0 & 1: unknown value

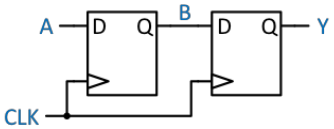
Q = D immediately when CLK is on falling edge
(it will take on value of D slightly before falling edge)

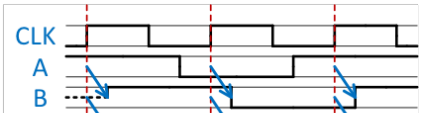
Positive-edge triggering D flip-flop



Negate the entire CLK signal from beginning.
representation:





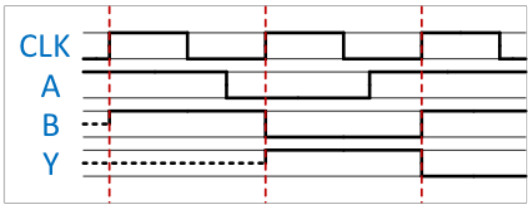


Chained flip-flop has delay:
The change to ff output is ALWAYS after the triggering CLK edge.

Functional waveform:
Ideal situation that delay = 0, but still expressing casuality of the circuit.

The value of a FF's output just after the active CLK edge is the value of FF's input right before the active CLK edge.

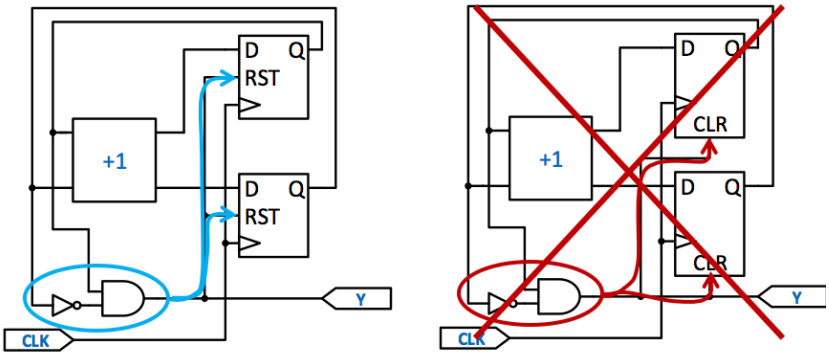
FF在triggering CLK edge之后的输出值 是由FF在triggering CLK edge之前的输入决定的



B input right before the first rising edge is unknown, resulting in the entire period of unknown in Y output before the second rising edge.
(B's changes do not show up at Y until next CLK edge)

Asynchronous input: affects ff immediately
Synchronous input: has no effect unless on active CLK edge

Flip-flop direct input(asynchronous, global!!):
To let FF to hold a specific value immediately before the next active CLK edge.
For any asynchronous inputs:
They should be never driven by signals other than global power-on reset.

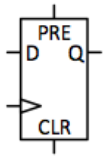


NEVER drive the flip-flop's asynchronous preset/clear inputs with your synchronous circuit.
They are only to be used for a power-on reset.

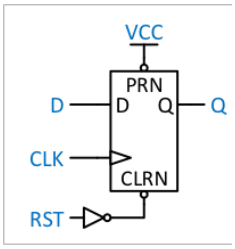
Commonly used to force a circuit's flip-flops into a known desired state on start-up

- Direct set (preset) forces flip-flop to **1**
- Direct reset (clear) forces flip-flop to **0**

Symbol



Preset or Clear should be disabled when not used:
Ex. Active low PRN & CLRN, disable PRN by connecting it to 1(VCC)



To have synchronous reset or preset, the logic you add should feed into flip-flop's D input.
Ex. To implement active low synchronous reset, need to add an AND gate to intercept D input and gate it off with reset signal to form new D input.
X and 0 = 0

Sequential Circuits

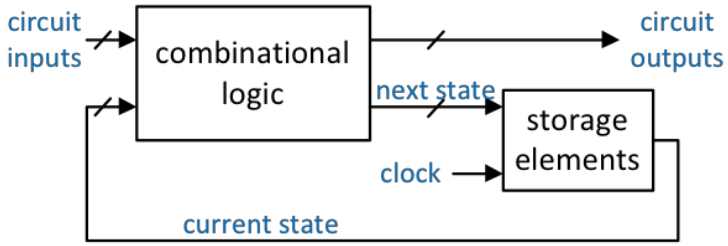
Combinational Circuit:
Output is a function of only current circuit inputs.
Sequential Circuit: (Comb+ffs)
Output are a function of current circuit inputs
What has happened in the past.
Ex. Vending machine

State: (of a circuit)
The set of values stored in the circuit's flip-flops(representing status).
Ex. The state of vending machine includes the amount of money entered.

Finite State Machine(FSM):
A sequential circuit that has a finite number of possible states(finite number of ffs in circuit).
All practical sequential circuits are FSMs. FSMs are machine that perform

- All practical sequential circuits are FSMs, FSMs are machine that perform tasks rather than being a memory devices.

Synchronous Sequential Circuits (ruled by clock)



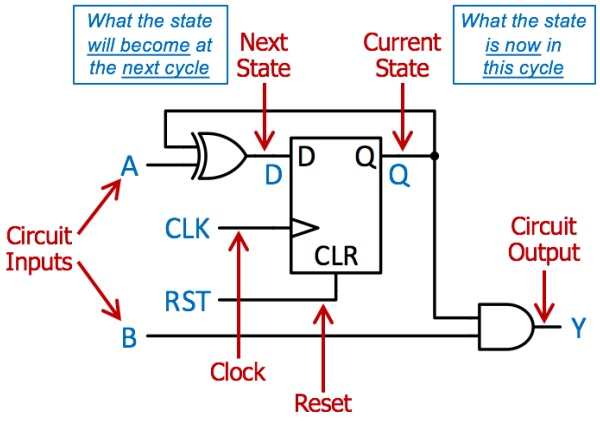
- **Current states:**

A code that determines how the machine will react to the inputs.
The set of values (currently) stored in flip-flops at a given time.
The Q output of 1 or a set of flip-flops.

- **Next state:**

Combinational logic of current state and inputs.
The set of values that will be stored into the flip-flops at next triggering clock edge.
The D inputs to the flip-flops that have not yet been stored into the flip-flops.
Circuit outputs: combinational logic of current state and possibly the inputs.
Once per clock cycle, **next state** becomes (update) the **current state.**

Anatomy of a Simple Seq. Circuit



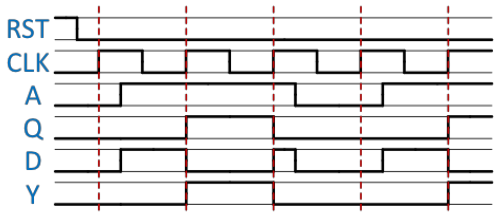
Ex. Two states (1ff) FSM
Current state: the value in the flip-flop.
Next state: the value at the input to the flipflop.
Output: a function of current state and inputs.

State table:

A truth table that shows all possible combinations of current state & circuit inputs, resulting next state & circuit outputs.
Input forming logic:
The combinational logic equations for the next state (FF inputs)

FSMs are either moore or mealy.
Moore machine: output based exclusively on current state(FF output)
Mealy machine: output based on current state(FF output) and circuit inputs.

State tables in circuit analysis:
Write forming logic equations for flip-flops input & outputs to get state tables, which can be written as kmap for simplification.
Input & output equations are all based on A(circuit inputs) and Q(flip-flops output/current state).
(A circuit that toggles output when input is 1)

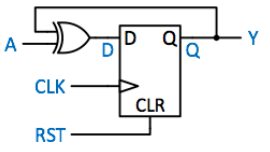


Can view D(input / next state) as asynchronous exclusive or result of A and Q.
At each rising clock edge, value of Q becomes D right before the edge.

State diagrams

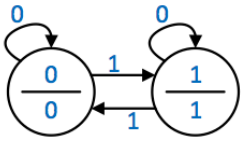
A graphical depiction of the state table.

Example **Moore** Diagram



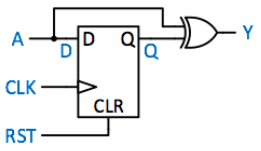
- Create state diagram based on state table
  - Two states (Q=0, Q=1)
  - Next-state transitions based on input A and current state
  - Output depends only on current state

Current State Q	Input A	Next State D	Output Y
0	0	0	0
0	1	1	0
1	0	1	1
1	1	0	1



Example **Mealy** Diagram

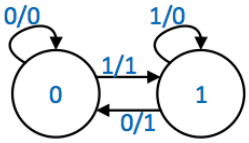
- Create state diagram



Create state diagram based on state table

- Two states (Q=0, Q=1)
- Next-state and output based on input A and current state
  - Input / Resulting Output

Current State Q	Input A	Next State D	Output Y
0	0	0	0
0	1	1	1
1	0	0	1
1	1	1	0



D Y as function of Q A.

- N flip-flops FSM = N bit binary encoding = 2^N possible states

Circles: states

Directed lines(possibly with labeled input): directed transition with input that cause the transition

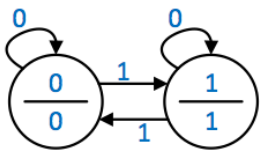
- Circuit outputs:

Mealy machine: shown on transition

Moore machine: inside state circle

Moore

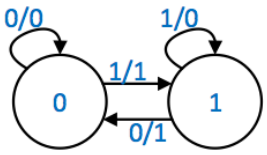
Output depends only on the current state



output shown in the state

Mealy

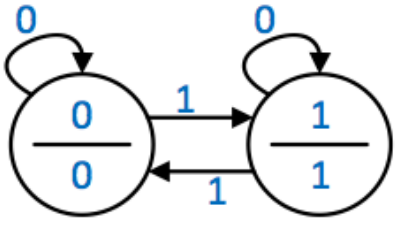
Output depends on current state and circuit inputs

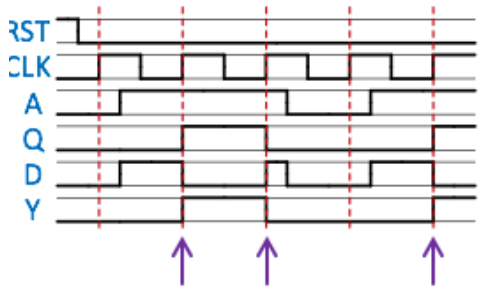


output shown on transitions

Moore

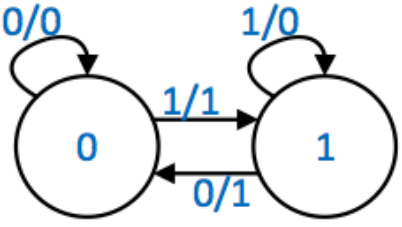
Output depends only on the **current state**

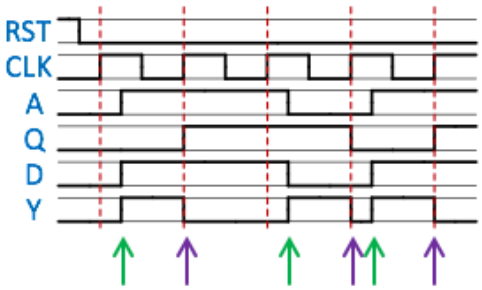




Mealy

Output depends on **current state** and **circuit inputs**



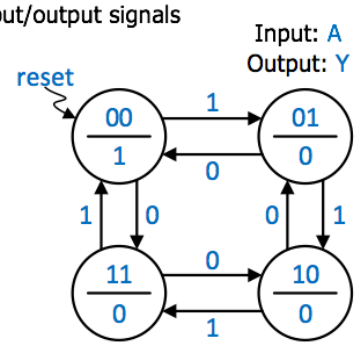


Moore output changes only when Q changes.
Mealy output can change when either Q or A changes.

Need to annotate state diagram:

- The state the circuit starts in when it is reset
- The names of the input/output signals

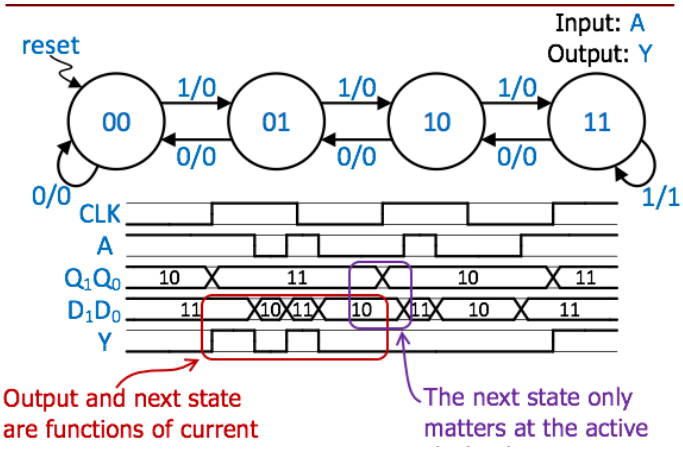
Current State Q_1	Q_0	Input A	Next State D_1	D_0	Output Y
0	0	0	1	1	1
0	0	1	0	1	1
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	1	0
1	0	1	1	1	0
1	1	0	1	0	0
1	1	1	0	0	0



This is an example of up1 down0 counter:
Count according to input A. Output 1 when the count is 00.

Enables in counter:
Enabled: behaves normally(update at next triggering edge)
Disabled: keep current value
Can also count in different sequence: arithmetic/grey/bcd

Note:
On mealy machine, output labeled on the transition will only happen if input is on active clock edge.
(it doesn't matter how input change btw two active clock edges, but they do change)
The circuit output changes in response to the circuit input changes OR triggering clock edge
(circuit input changes: different transition, triggering clock edge: output is genrated)



state and input clock edge

Q1Q0 becomes D1D0 whenever on rising clock edge.
Y becomes 1 if current state Q1Q0 is 11 and input A is 1

A sequential circuit with n inputs and m states:
state transitions coming IN to a given state >= 0
state transitions leaving a given state = 2^n

Sequential circuit design process

Specification:
- Purpose, input output signals&meaning interface, constraints in reality

State diagram and state table:
- States required, next state required with all possible input combinations
- Validate state diagram by testing under different input scenarios

Optimization:
- Determine unused state & merge

State assignment:
- Required number of ffs, binary value assignment

Combinational logic design:
- Find D ff input and Y circuit output equation

Implementation: create schematic
Verification: use test vector to correct the design

State minimization

#FFs >= ceil(log2(#states)) more storage elements added than expected
To eliminate redundancy:
- Reuse states when possible (loop back)
- Merge states when appropriate(same outgoing transitions(targets) & output)

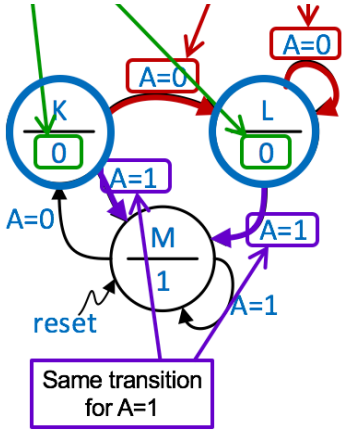
In practice:
Moore machine: merge states with same outputs & next state

Identifying Merge-able States

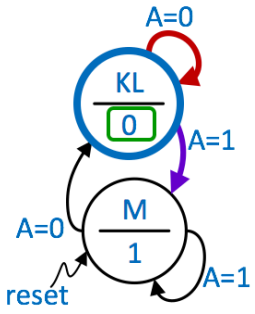
Same outputs

Same transition for A=0

Moore Machine:
- Merge states w/same



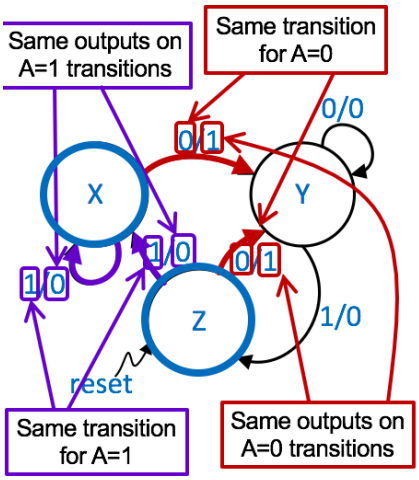
outputs & next state



FSM with merged KL state

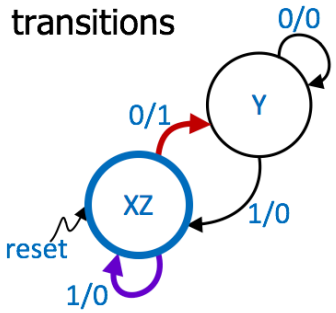
- Mealy machine: merge states with same next state & output transition (same edges labeling & targets)

Identifying Merge-able States



Mealy Machine:

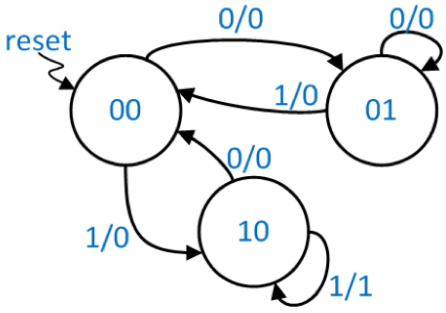
- Merge states w/same next state / output transitions



FSM with merged XZ state

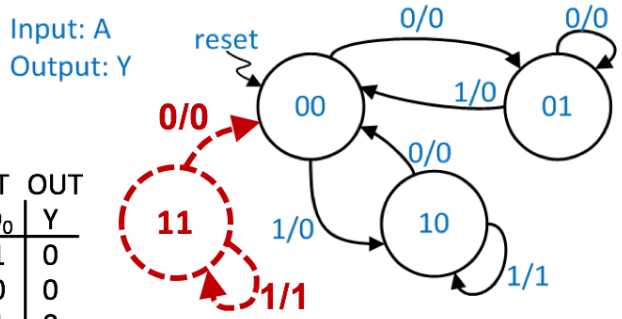
- Unused states(states unused in remaining of 2^n states for n ffs FSM) Regard next state & output of them as don't care:

Input: A
Output: Y



CURR		IN	NEXT		OUT
Q_1	Q_0	A	D_1	D_0	Y
0	0	0	0	1	0
0	0	1	1	0	0
0	1	0	0	1	0
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	1	1	1
1	1	0	X	X	X
1	1	1	X	X	X

Find optimal solution using k-map:



CURR Q_1	CURR Q_0	IN A	NEXT D_1	NEXT D_0	OUT Y
0	0	0	0	1	0
0	0	1	1	0	0
0	1	0	0	1	0
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	1	1	1
1	1	0	**0**	**0**	**0**
1	1	1	**1**	**1**	**1**

$D_0 = Q_1 A + \overline{Q_1}\,\overline{A}$

$D_1 = Q_0 A$

$Y = Q_1 A$

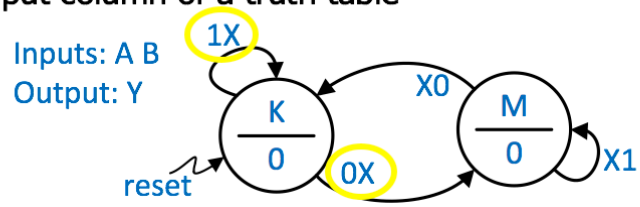
Determine the behavior, draw the state back.

Note: state 11 should never be entered under normal conditions.

Therefore, it can be used as a flag for debugging.

Don't care can also be used in input on state diagram.

- Don't-cares can express that a particular variable does not affect a transition
  - Similar to shorthand notation where we use X in an input column of a truth table



  - Example: In state K, only care about A, and in state M, only care about B...
    - In this case, each arrow represents **two** transitions

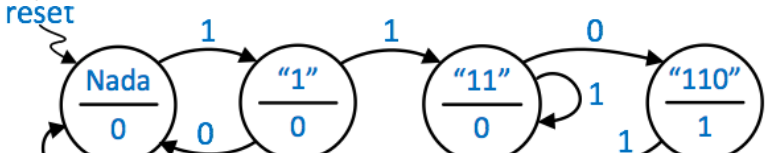
FSM design and verification

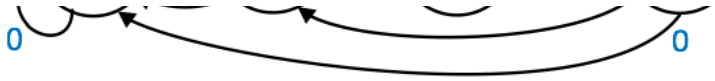
Step 1. draw state diagram

Ex.

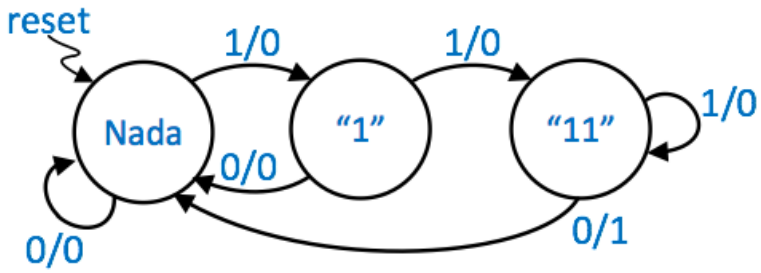
Output a 1 whenever the circuit recognizes a sequence of 110 within a serial bit pattern.

Try Moore machine:





Try Mealy machine:



Note:

- The mealy machine needs fewer states and detects the pattern earlier.

- Difference in functionality:
  Moore machine reacts to the past input value, then output.
- This means Moore output cannot be affected by input change unless on active clock edge.
- Mealy machine reacts to the current input value change immediately.

Step 2. state assignment

Assign state numbers:

- N filp-flops can represents at most 2^N states. If we require m states for the diagram, then m <= 2^N for some N.
- The number of unused states is 2^N - m.

Minimum FF design: as few FFs as possible.

- Choice of state numbers assignment can affect complexity of the next-state and output logic.

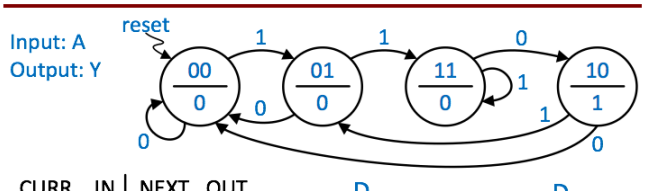
Ex. Use gray code, binary arithmetic sequence. The name should be meaningful.

Determine the reset state!

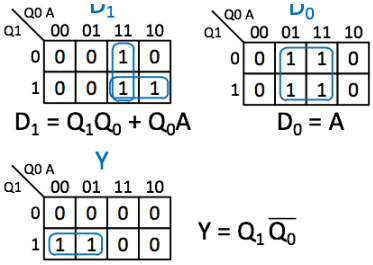
Step 3. transcript the state diagram to state table.

Optimize and derive combinational logic equations for each flip-flop input and circuit output.

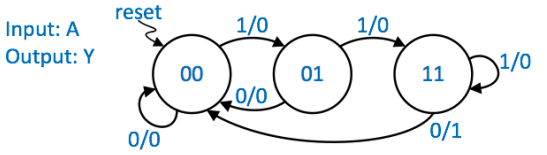
Example: Moore Machine



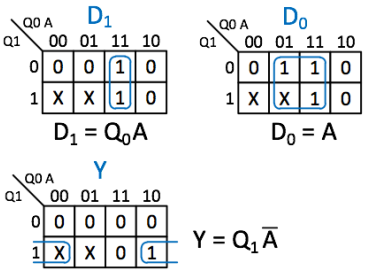
Q_1	Q_0	A	D_1	D_0	Y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	0	1	1
1	1	0	1	0	0
1	1	1	1	1	0



Example: Mealy Machine



CURR Q_1	CURR Q_0	IN A	NEXT D_1	NEXT D_0	OUT Y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	1	0
1	0	0	X	X	X
1	0	1	X	X	X
1	1	0	0	0	1
1	1	1	1	1	0



State 10 is unused, so we mark it with x for next state and output.

Example Circuits

- Moore

$D_1 = Q_1Q_0 + Q_0A$ (can factor out Q_0)

$D_0 = A$

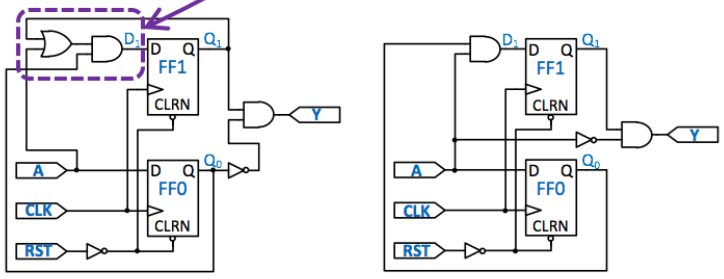
$Y = Q_1\overline{Q_0}$

- Mealy

$D_1 = Q_0A$

$D_0 = A$

$Y = Q_1\overline{A}$

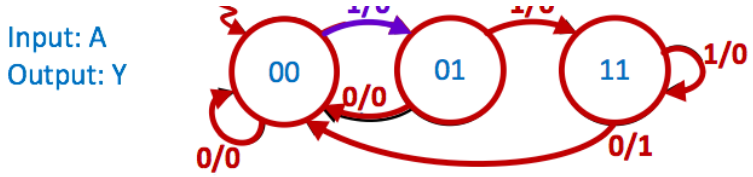


Notice that mealy machine is simpler because we have don't cares for our benefits.

Step 4. verification

We should test all state transition by come up with a minimized input sequence that tests:

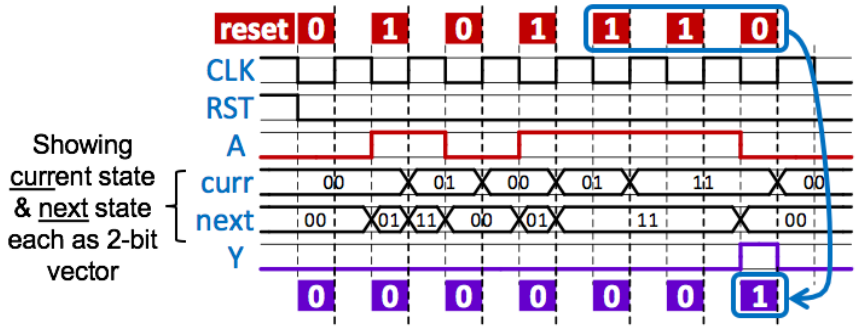
All possible input value in every possible state.



test vector: reset 0 1 0 1 1 1 0

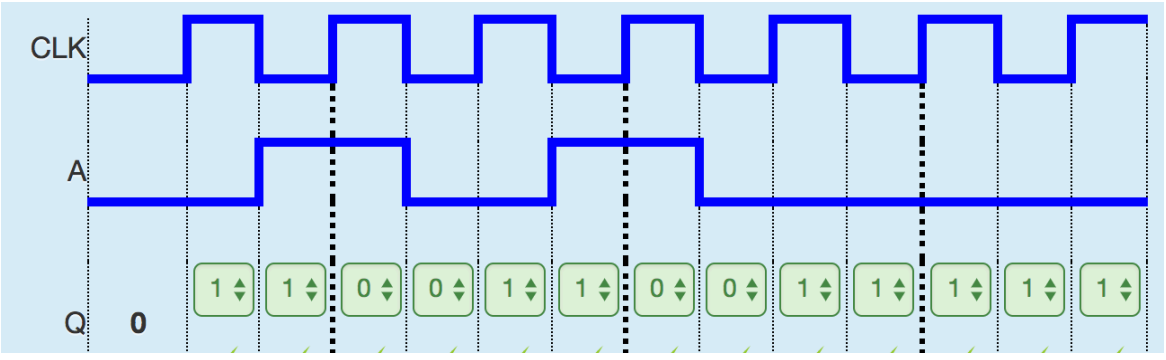
Remember: do not change input values at the same time as the active clock edge!

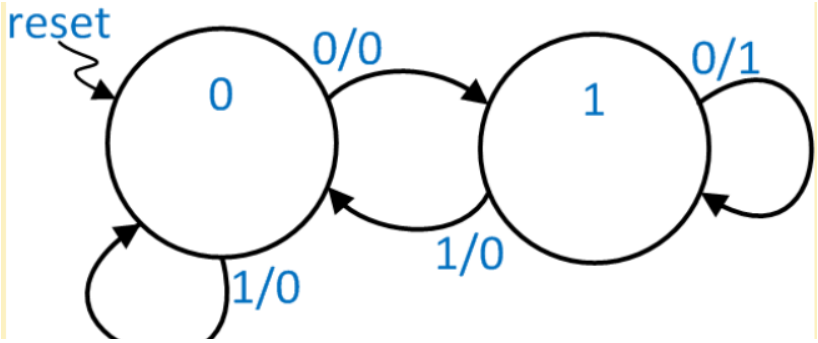
(Only change inputs at the inactive edge of the clock)



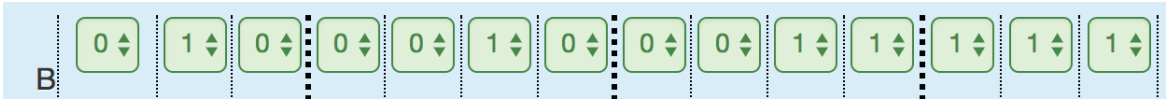
Ex. This is a positive edge triggered waveform of above sequential circuit.

As input change, next state change accordingly w.r.t current state. At each positive edge, current state become next state prior to the positive edge.

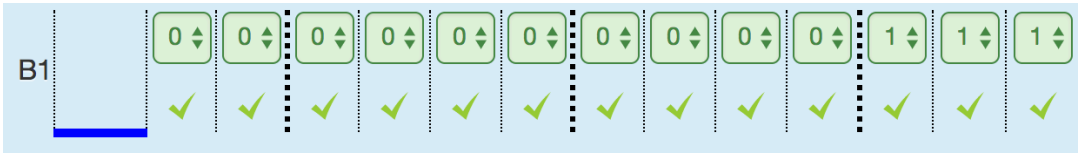


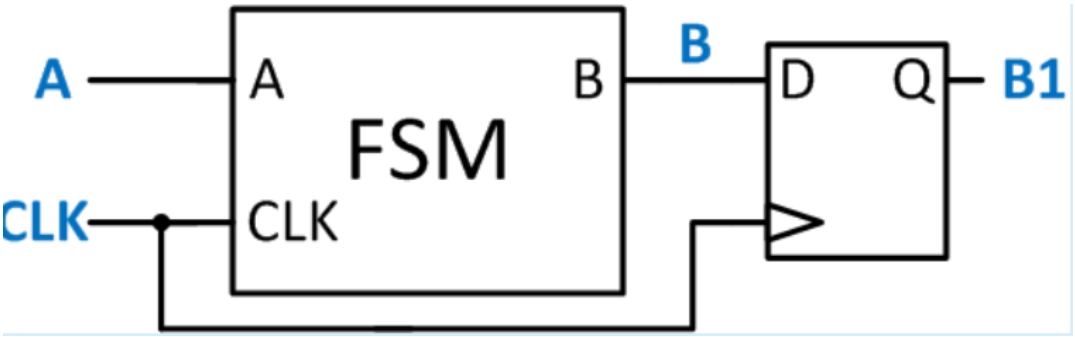


To find output, check each vertical group of Q and A and corresponding transition arrow.



The mealy output is only guaranteed to be meaningful right before the clock edge because the input value at that time period affect state change.





B1 is dependent on value of B before clock edge.

Conversion relationship:
Moore machine can be converted to mealy machine.
But mealy cannot be converted to moore.

Flip-Flop Timing Parameters

Synchronous inputs change at clock edge: indeterminate input value.

- Therefore, we want to make sure input does not change too close to the clock edge.

- Secondly, the output of a flip-flop will not update instantaneously at clock edge. There is a delay before new value to propagate through flip-flop to Q output.

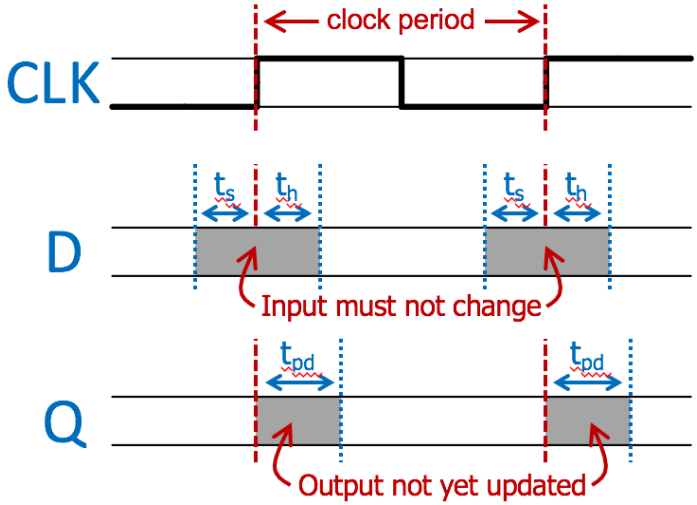
We need input value to be stable before and after clock edge:

Flip-flop setup Time t_s

Flip-flop hold Time t_h

The delay between the actual appearance of flip-flop output after clock edge.

Flip-flop propagation delay t_{pd}



Hold up time violation occur when $t_h > t_{pd} + t_{comb.min}$(delay of fastest ff to ff path)

This means a change on a flip-flop output propagate through the circuit back to a flip-flop input before the end of the hold up time.

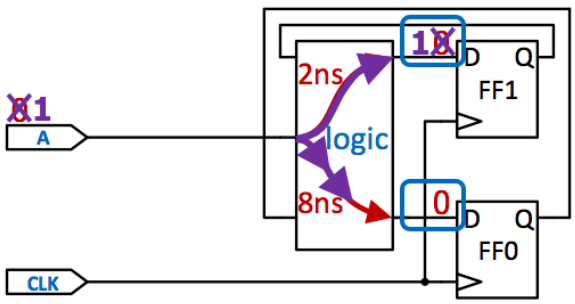
To avoid hold up time violation, we usually set $t_h < t_{pd}$, hold up time cannot be violated even when $t_{comb.min} = 0$

Inputs to a synchronous circuit change asynchronously can also cause problem:

Setup/hold time violations if input changes too close to clock edge.

Unequal delay paths sometimes mean that input change does not propagate to all of a circuit's flip-flops in the same clock cycle.

Assume that in the FSM below the next state is 00_2 if A=0 and 11_2 if A=1



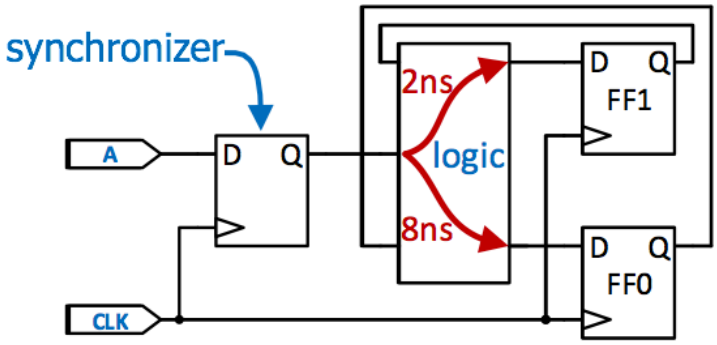
Suppose signal A changes from 0→1 at 5ns
before the positive clock edge – what happens?

- Assume t_S and t_H are not violated

5ns before the next clock edge, signal A is passed in. The uneven delay results in faulty behavior of the circuit.
(Signal A can arrive FF1 using 2 s, but cannot arrive FF0, which need 8 ns)
Then, at the clock edge, FF0 is not updated.

Solution:

To solve the problem, add a flip-flop to **synchronize** the changes in A to the clock.



Now, input changes can only propagate to FF1/FF0 immediately after a clock edge, so we can guarantee correct operation **...except...**

Synchronizer input can change at any time, which may violates synchronizer FF's setup hold time itself, which may results in a metastable synchronizer output.
A metastable state is nondeterministic state that oscillates and settles to 1 or 0.
To reduce the chances and the bad effect of being metastable, a chain of synchronizer FFs can be used to reduce the probability. (can approach 0 but never equal 0)

Sequential Circuit Timing

How fast can we clock a sequential circuit?
Critical path: longest (largest delay) path that begins and ends at a flip-flop(can be the same FF).

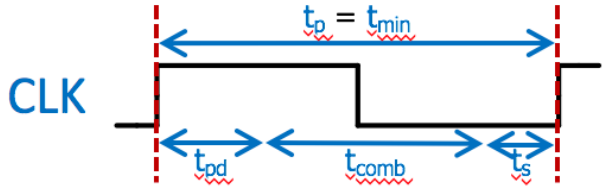
Minimum Clock period t_{min} : smallest feasible clk period

$$t_{min} = t_{pd} + t_{comb} + t_s$$

t_{comb} : the delay of the critical path between a FF output and a FF input.

NOT CIRCUIT INPUTS!!!!!

Need time for a change to get out of a flip-flop, through the circuit, and be present at another flip-flop input before the start of the set-up time



Slack t_{slack} :

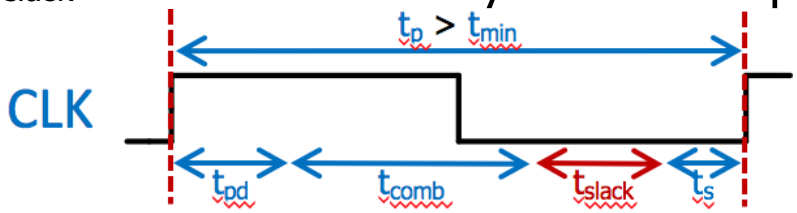
difference btw actual clock period t_p and minimum clock period t_{min}.

$t_{slack} = t_p - t_{min}$

$t_{slack} > 0$ clock can be faster

$t_{slack} < 0$ clock is too fast

$t_{slack} = 0$ clock is exactly the fastest possible



Clock frequency:

The clock speed of the sequential circuit.

Faster clock = higher clock frequency

Maximum circuit frequency: $\mathbf{f_{max} = 1/t_{min}}$

Actual circuit frequency: $\mathbf{f = 1/t_p}$

Remember:

- Clock period is ps, ns, µs, ms, s...
- Frequency is Hz, kHz, MHz, GHz...

If t_{min} = 10ns...

$$f_{max} = \frac{1}{t_{min}} = \frac{1}{10\text{ ns}} = \frac{1}{10^{-8}\text{ s}} = 10^8\text{ Hz} = 100\text{ MHz}$$

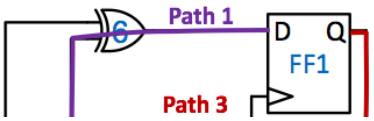
Metric Prefix Table

Prefix	Symbol	Exponential
yotta	Y	10^{24}
zetta	Z	10^{21}
exa	E	10^{18}
peta	P	10^{15}
tera	T	10^{12}
giga	G	10^{9}
mega	M	10^{6}
kilo	k	10^{3}
hecto	h	10^{2}
deca	da	10^{1}
		10^{0}
deci	d	10^{-1}
centi	c	10^{-2}
milli	m	10^{-3}
micro	μ	10^{-6}
nano	n	10^{-9}
pico	p	10^{-12}
femto	f	10^{-15}
atto	a	10^{-18}
zepto	z	10^{-21}
yocto	y	10^{-24}

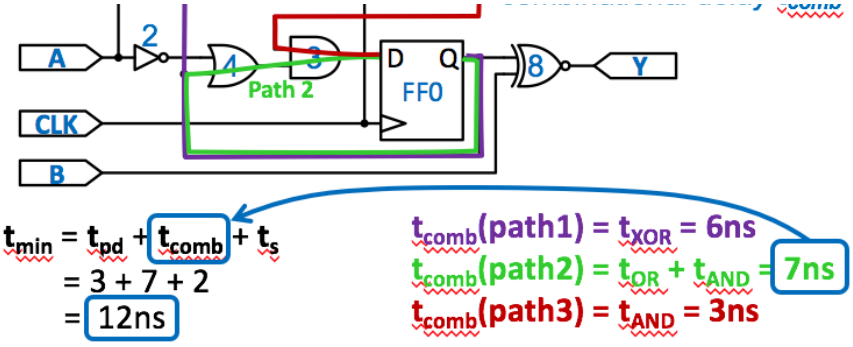
EXAMPLE

Critical Path Example 1

- Find t_{min}, given the following delays (in ns):
  $t_{pd} = 3$, $t_s = 2$, $t_{NOT} = 2$, $t_{AND} = 3$, $t_{OR} = 4$, $t_{XOR} = 6$, $t_{XNOR} = 8$

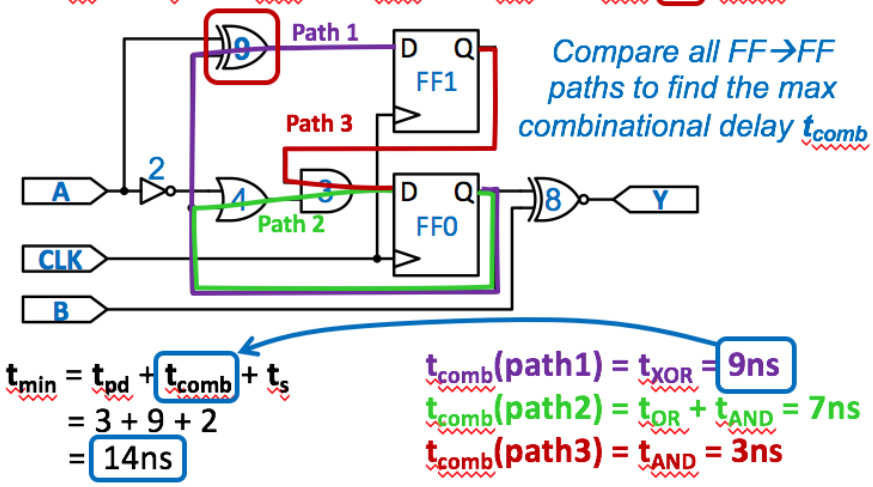


Compare all FF→FF paths to find the max combinational delay t_{comb}



Critical Path Example 2

- Find t_{min}, given the following delays (in ns):
  $t_{pd} = 3$, $t_s = 2$, $t_{NOT} = 2$, $t_{AND} = 3$, $t_{OR} = 4$, $t_{XOR} = 9$, $t_{XNOR} = 8$



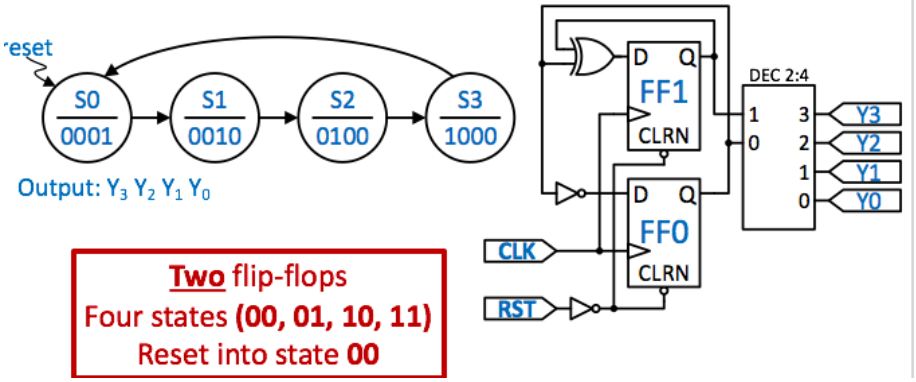
DO NOT CONSIDER INPUT PATH!!!!!!!!!
WE ONLY CARE ABOUT PATH STARTING FROM A FF OUTPUT TO A FF INPUT!!!!!

BE cautious about gating clock signals, usually resulting in faulty behaviors.
our whole design methodology is based on the clock signal arriving at all flip-flops SIMULTANEOUSLY, and clock gating changes that!!!!!!!!!!

One-Hot State Machine Design

Ex. A ring counter

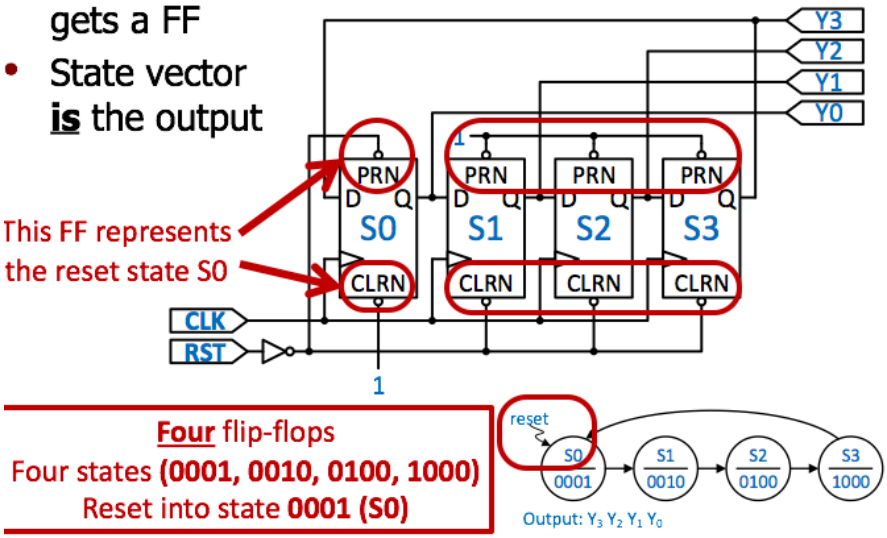
- Could use a minimum flip-flop technique...
  - Use an arithmetic order for state assignment
  - A decoder takes in current state and produces the circuit outputs



Two flip-flops
Four states **(00, 01, 10, 11)**
Reset into state **00**

Reset will clear all flip-flops to state S0.

- Each state gets a FF
- State vector **is** the output



Four flip-flops
Four states **(0001, 0010, 0100, 1000)**
Reset into state **0001 (S0)**

An alternate minimum flip-flop design:

One-hot state assignment/One flip-flop per state

PRO: SIMPLICITY

CON: MORE Flip-flops are used, but this has undetermined effects on the circuit depends on implementation

Flip-flop = state

m states = m flip-flops

The circuit is only in one state at a time(a lot of unused state).

Current state :

FF that has 1 in output, Q = 1.

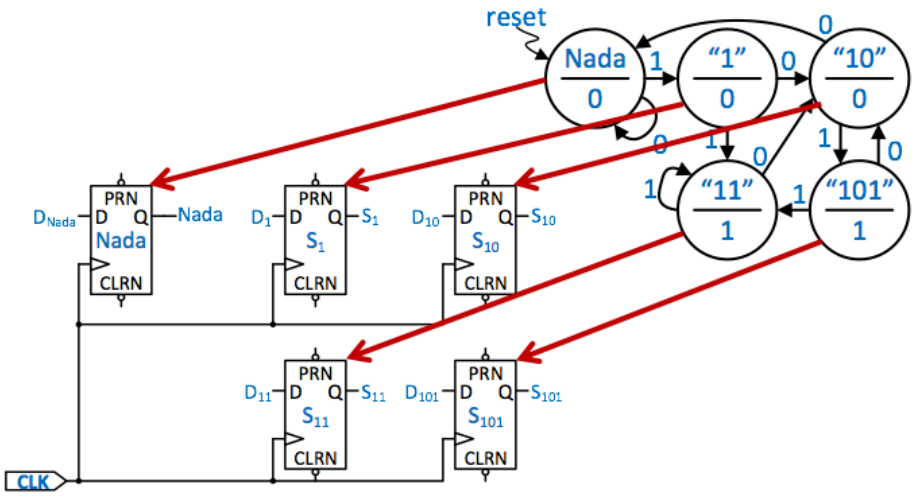
The complete state output vector such as S0 = 0001.
Next state : the flip-flop that has input D = 1.
Circuit reset : load 1 into FF reset flip-flop and clear remaining
Circuit clear :
State Transition:
Flip-flop for current state passing a token "1" to the flip-flop that corresponds to the next state.

We can create hardware directly from one hot diagram:
One FF per state.
For each state (FF), create an AND gate for each incoming transition, or these together as FF input D.
(All the ways this can be the next state)

Moore outputs: OR together all the outputs of FFs that represent states where the output should be 1.
Mealy outputs: create AND gate for each transition where the output is 1, and OR those together.
(All the ways this output can be 1)

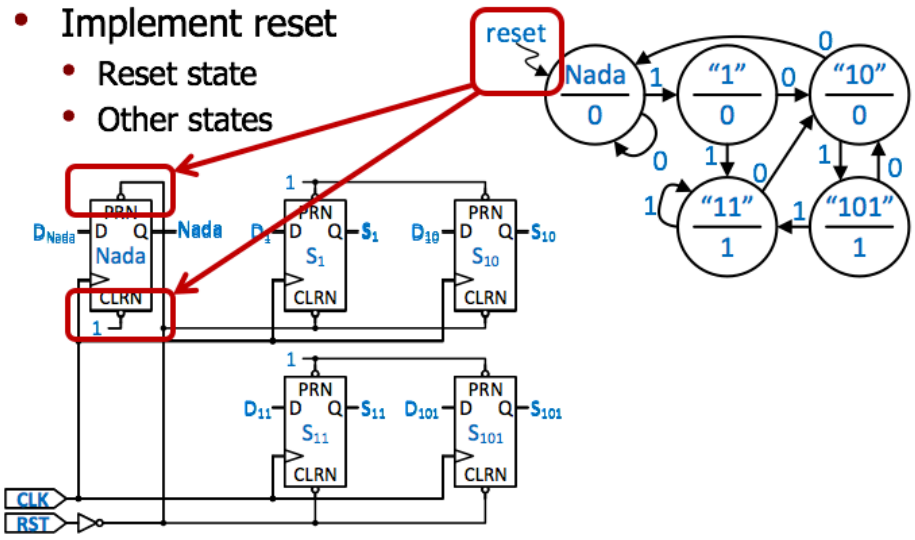
Example One-Hot State Machine

- Create a flip-flop for each state



Example One-Hot State Machine

- Create a flip-flop for each state
- Implement reset
  - Reset state
  - Other states



Example One-Hot State Machine

- Create FF input and circuit output equations

$D_{Nada} = Nada \cdot \overline{A} + S_{10} \cdot \overline{A}$
$= (Nada + S_{10}) \cdot \overline{A}$

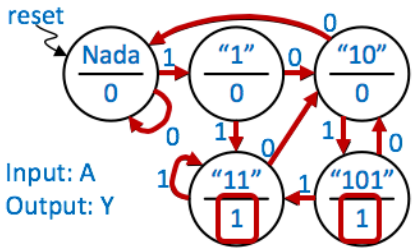
$D_1 = Nada \cdot A$

$D_{10} = S_1 \cdot \overline{A} + S_{11} \cdot \overline{A} + S_{101} \cdot \overline{A}$
$= (S_1 + S_{11} + S_{101}) \cdot \overline{A}$

$D_{101} = S_{10} \cdot A$

$D_{11} = S_1 \cdot A + S_{11} \cdot A + S_{101} \cdot A$
$= (S_1 + S_{11} + S_{101}) \cdot A$

$Y = S_{11} + S_{101}$



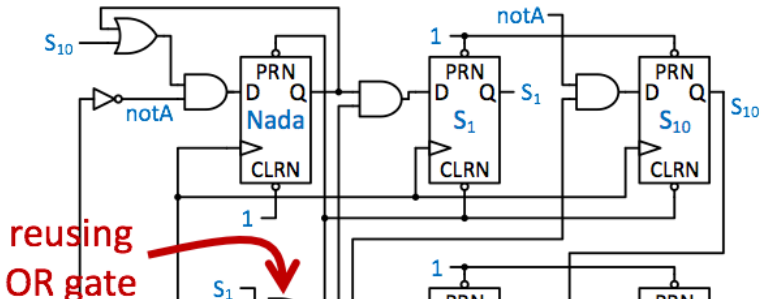
*Notice that the asynchronous reset is **NOT** an input to the D_{Nada} equation!*

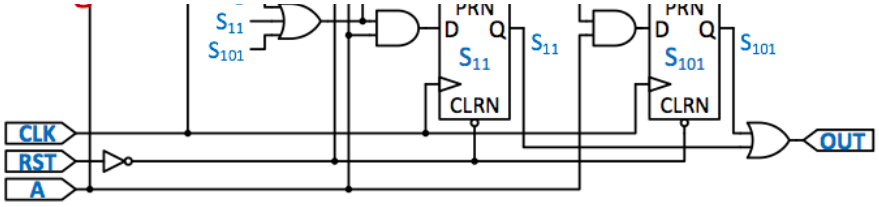
*The reset behavior is handled by the **asynchronous clear/preset inputs***

Can use Boolean algebra to simplify…

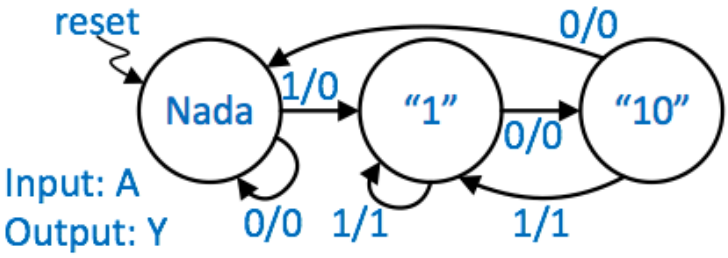
Example One-Hot State Machine

- The completed circuit….





Mealy One-Hot



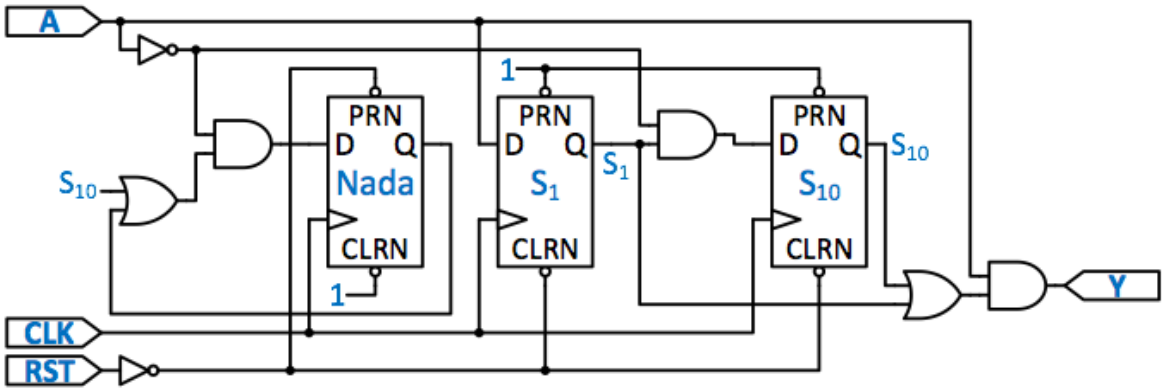
one of these has to be a 1 because it always has to be in some state...

$$D_{Nada} = Nada\cdot\overline{A} + S_{10}\cdot\overline{A} = (Nada + S_{10})\cdot\overline{A}$$

$$D_1 = Nada\cdot A + S_1\cdot A + S_{10}\cdot A = (Nada + S_1 + S_{10})\cdot A = A$$

$$D_{10} = S_1\cdot\overline{A}$$

$$Y = S_1\cdot A + S_{10}\cdot A = (S_1 + S_{10})\cdot A$$



As shown above, the logic expressions obtained directly from the state diagram could often be simplified algebraically. However, the main motivation for using the one-hot design method is simplicity of design along with ease of debugging.

As a general rule, you should only do algebraic simplification when it results in a circuit that is easier for you to implement.

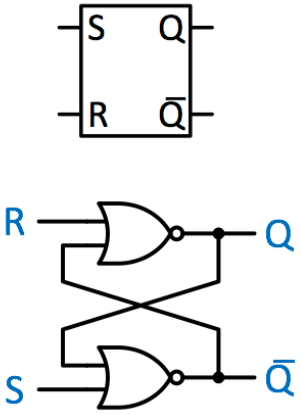
Other Flip-flop type

characteristic table: describe behavior of a flip-flop.

SR Latch (NOR)

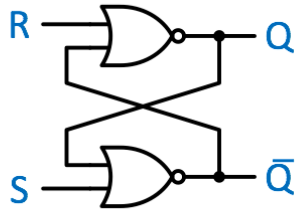
- S refers to "set", R refers to "reset"

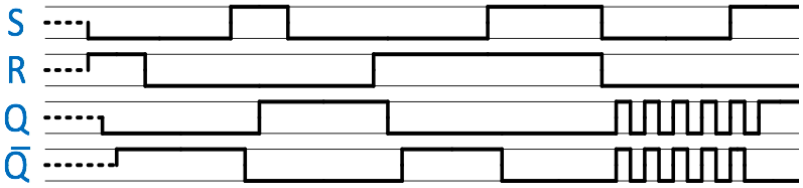
NOR SR Latch Characteristic Table			
Inputs		Next State	Operation
S	R	Q(t+1)	
0	0	Q(t)	Hold (no change)
0	1	0	Reset
1	0	1	Set
1	1	???	Undefined



SR Latch Timing Waveform

- The Q and $\bar{Q}$ outputs are both unknown until S and R are known
- When S = R = 1, Q = $\bar{Q}$ = 0!
  - If we try to "hold" afterwards with S = R = 0, the value in the latch oscillates…

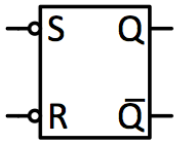




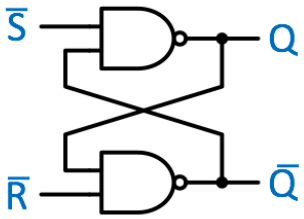
$\bar{S}\bar{R}$ Latch (NAND)

- Can make an $\bar{S}\bar{R}$ latch with NANDs
  - Inputs become active-low

NAND $\bar{S}\bar{R}$ Latch Characteristic Table		
Inputs	Next State	



$\bar{S}$	$\bar{R}$	Q(t+1)	Operation
0	0	???	Undefined
0	1	1	Set
1	0	0	Reset
1	1	Q(t)	Hold (no change)

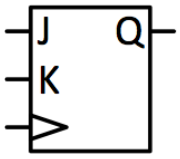


The active low version of SR latch.

JK Flip-Flop

- Based on SR latches
  - J → S, K → R
- Avoids the undefined case!

JK Flip-Flop Characteristic Table			
Inputs		Next State	Operation
J	K	Q(t+1)	
0	0	Q(t)	Hold (no change)
0	1	0	Reset
1	0	1	Set
1	1	$\overline{Q(t)}$	Complement

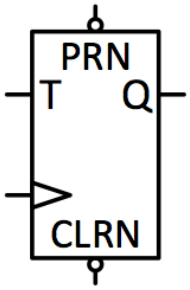


Note there is no undefined like SR latch: instead we toggle current state.

T Flip-Flop

- "Toggle" FF is equivalent to JK FF with J=K=T
- MUST have a set or reset input to start at a known initial value

T Flip-Flop Characteristic Table		
Input	Next State	Operation
T	Q(t+1)	
0	Q(t)	Hold (no change)
1	$\overline{Q(t)}$	Complement



How to use other flip-flops:

Create **state table** with column: current state/next state/circuit input/FF input

For each current state -> next state transition:

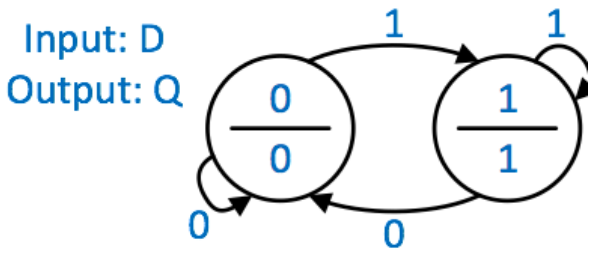
Find the required FF input values in characteristic table.

Find the required FF input values in characteristic table.
Create combinational logic for FF inputs and circuit output.

How to make D flip-flop using JK flip-flop?
Draw D flip-flop state diagram and characteristic table:

Express DFF behavior using a state diagram

Create a state table from the state diagram



Input D	Curr State Q	Next State N
0	0	0
0	1	0
1	0	1
1	1	1

D is not the next state in this case – it is the **input**!

The output is equal to the current state, so we are not showing it separately

Add column FF inputs. Find the required input that causes transition.

Input D	Curr State Q	Next State N	FF Inputs J K
0	0 →	0	01 or 00
0	1 →	0	01 or 11
1	0 →	1	10 or 11
1	1 →	1	10 or 00

JK Characteristic Table		
Inputs J	K	Next State Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	$\overline{Q(t)}$

FF Inputs J K	
01 or 00	→ JK = 0X
01 or 11	→ JK = X1

Take advantage of the multiple options by using don't-cares!

10 or 11	→ JK = 1X
10 or 00	→ JK = X0

J

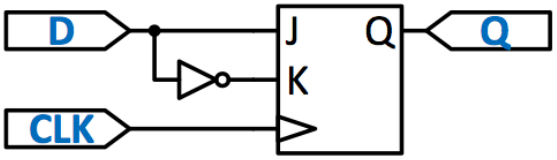
D \ Q	0	1
0	0	X
1	1	X

J = D

K

D \ Q	0	1
0	X	1
1	X	0

K = $\bar{D}$

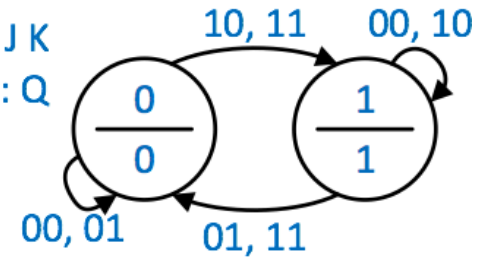


D = 0 → J = 0, K = 1 (reset)
D = 1 → J = 1, K = 0 (set)

Example: Make JK From D

- Express JK behavior using a state diagram
- Create a state table

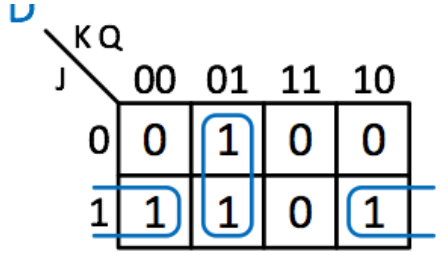
Inputs: J K
Output: Q



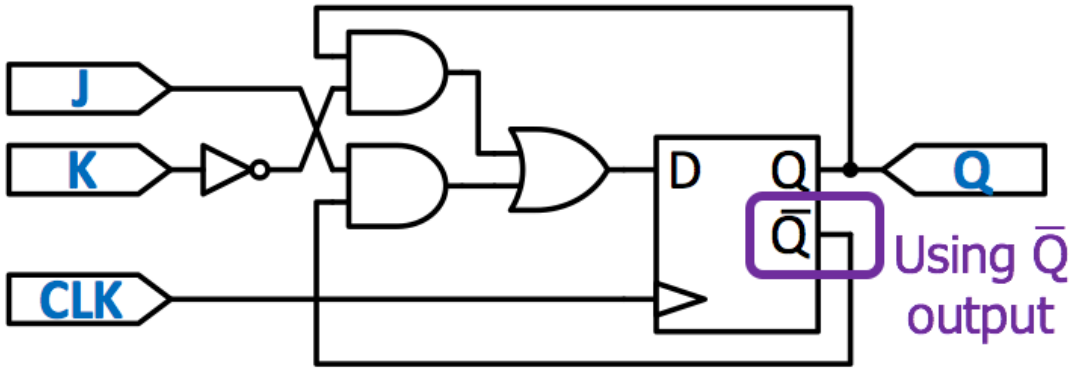
Input J K	Curr State Q	Next State D
0 0 (hold)	0 1	0 1
0 1 (reset)	0 1	0 0
1 0 (set)	0 1	1 1
1 1 (toggle)	0 1	1 0

JK Characteristic Table		
Inputs J	K	Next State Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	$\overline{Q(t)}$

Create the required logic based on the state table

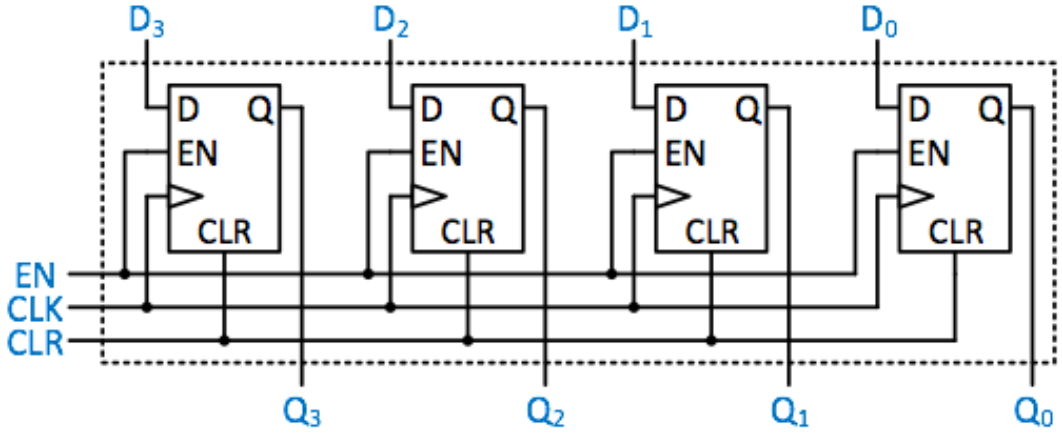


$$D = J\bar{Q} + \bar{K}Q$$



Register

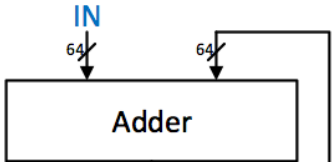
Definition: a **set** of contextually-related flip-flops as a unit.
Usage: store a multi-bit binary word.
Operations: read write simultaneously on all FFs as a group, affected by the same clock and asynchronous signals.



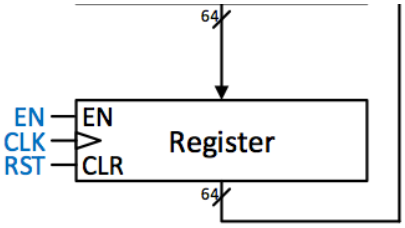
Benefits: simplicity in divide&concquer modular design.
E.g. 64 bit up-down counter 2^64 states impossible to be implemented as FSM
E.g. 64 bit accumulator design :

How can we easily build an accumulator?

- Build a 64-bit register



- Build a 64-bit adder
- Connect them!

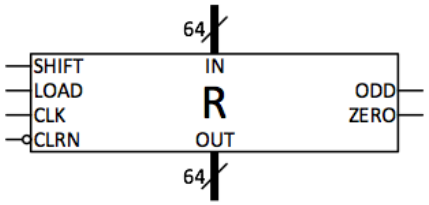


Register type:

- Storage register: store data from other circuits, often with only "load" ability.
  Often used by RISC(Reduced instruction set computers) machine with a shared ALU to get operands in need.
- Operational registers:
  Can perform operations on stored data and inputs.
  Often used by CISC(complex instruction set computer) machine to execute complex instructions.

Register organization:
Datapath(output/input): data process and storage
Control unit: control datapath how and when
Status signal(output): register output that provide information to the controller
Control signals(input): register inputs to feed into control unit. Clock and asynchronous inputs (global).

64-bit register **R** with shift and load capabilities



8-bit register **Y** that can perform various operations on itself and/or its data inputs **A** and **B**