

## Spiral 1 / Unit 6

### Flip-flops and Registers

© Mark Redekopp

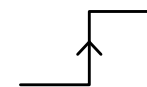
## Outcomes

- I know the difference between combinational and sequential logic and can name examples of each.
- I understand latency, throughput, and at least 1 technique to improve throughput
- I can identify when I need state vs. a purely combinational function
  - I can convert a simple word problem to a logic function (TT or canonical form) or state diagram
- I can use Karnaugh maps to synthesize combinational functions with several outputs
- I understand how a register with an enable functions & is built
- I can design a working state machine given a state diagram
- I can implement small logic functions with complex CMOS gates

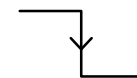
## FLIP FLOPS AND REGISTERS

## Flip-Flops

- Outputs only change once per clock period
  - Outputs change on either the *positive edges* of the clock or the *negative edges*



Positive-Edge of the Clock

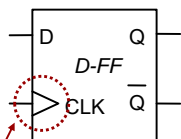


Negative-Edge of the Clock

# Flip-Flops

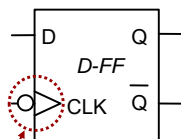
- To indicate negative-edge triggered use a bubble in front of the clock input

Positive-Edge Triggered D-FF



No bubble indicates positive-edge triggered

Negative-Edge Triggered D-FF

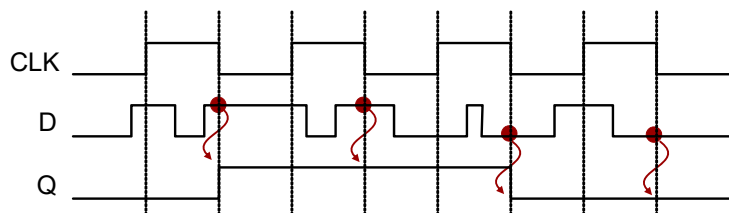


Bubble indicates negative-edge triggered

# Negative-Edge Triggered D-FF

- Q looks at D only at the negative-edge

CLK	D	Q*	Q**
0	x	Q	Q'
1	x	Q	Q'
↓	0	0	1
↓	1	1	0

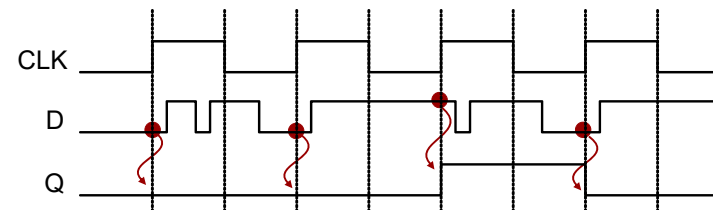


Q only samples D at the negative edges and then holds that value until the next edge

# Positive-Edge Triggered D-FF

- Q looks at D only at the positive-edge

CLK	D	Q*	Q**
0	x	Q	Q'
1	x	Q	Q'
↑	0	0	1
↑	1	1	0

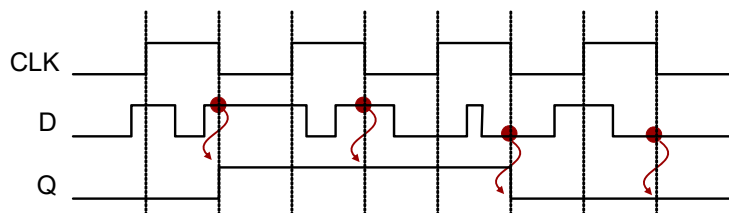


Q only samples D at the positive edges and then holds that value until the next edge

# Negative-Edge Triggered D-FF

- Q looks at D only at the negative-edge

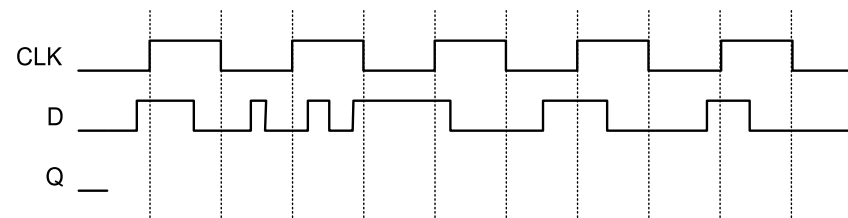
CLK	D	Q*	Q**
0	x	Q	Q'
1	x	Q	Q'
↓	0	0	1
↓	1	1	0



Q only samples D at the negative edges and then holds that value until the next edge

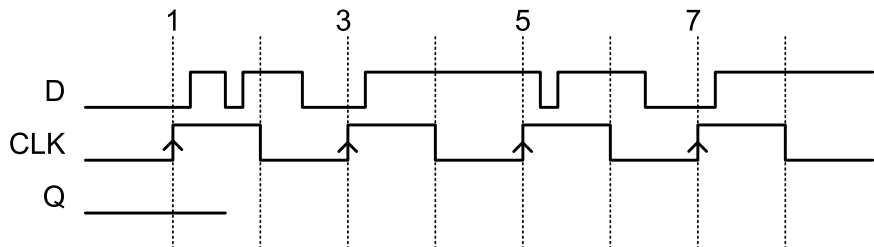
# D FF Example

- Assume positive edge-triggered FF



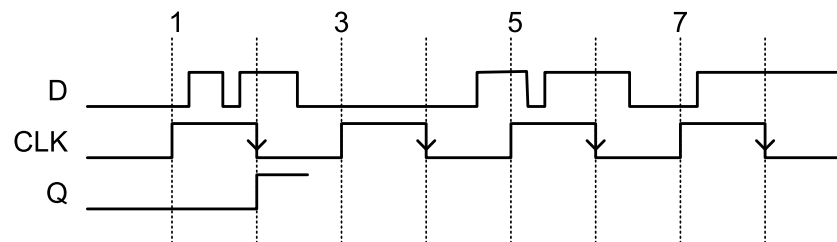
## D FF Example

- Assume positive edge-triggered FF



## D FF Example

- Assume negative edge-triggered FF

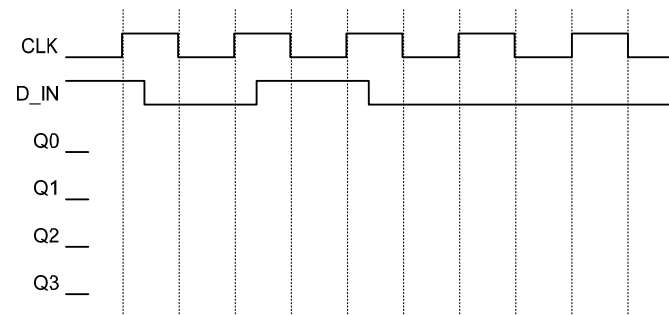
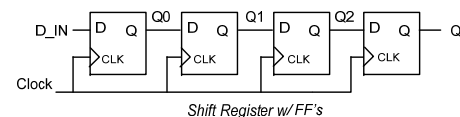


## Shift Register

- A shift register is a device that acts as a 'queue' or 'FIFO' (First-in, First-Out).
- It can store n bits and each bit moves one step forward each clock cycle
  - One bit comes in the overall input per clock
  - One bit 'falls out' the output per clock



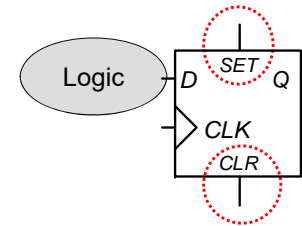
## Shift Register



# INITIALIZING OUTPUTS

# Initializing Outputs

- Need to be able to initialize Q to a known value (0 or 1)
- FF inputs are often connected to logic that will produce values after initialization
- Two extra inputs are often included: PRESET and CLEAR

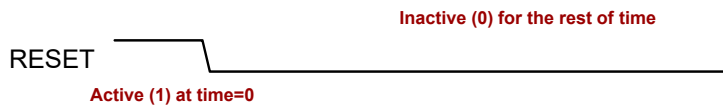


When CLEAR = active  
Q\*=0  
When SET = active  
Q\*=1  
When NEITHER = active  
Normal FF operation

Note: CLR and SET have priority over normal FF inputs

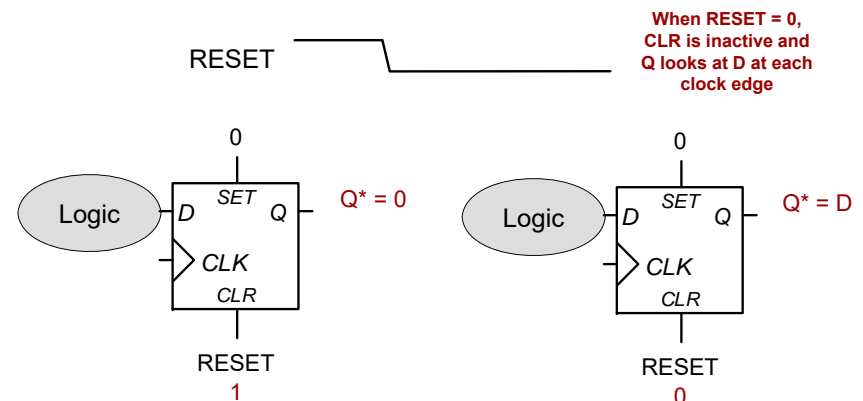
# Initializing Outputs

- To help us initialize our FF's use a RESET signal
  - Generally produced for us and given along with CLK
- It starts at *Active (1)* when power turns on and then goes to *Inactive (0)* for the rest of time
- When it's active use it to initialize the FF's and then it will go inactive for the rest of time and the FF's will work based on their inputs



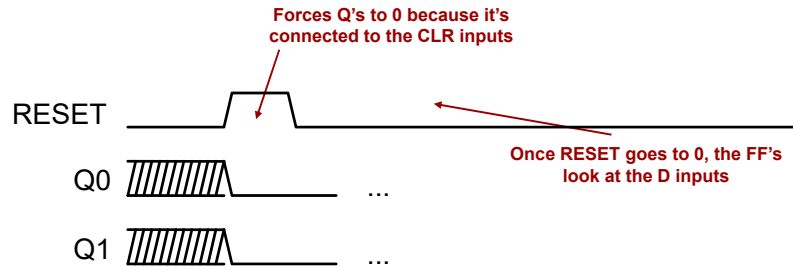
# Initializing Outputs

- Need to be able to initialize Q to a known value (0 or 1)



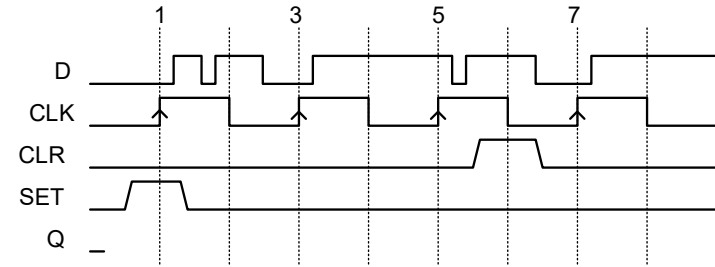
## Implementing an Initial State

- When RESET is activated Q's initialize to 0 and then when it goes back to 1 the Q's look at the D inputs



## Preset / Clear Example

- Assume an **synchronous** Preset



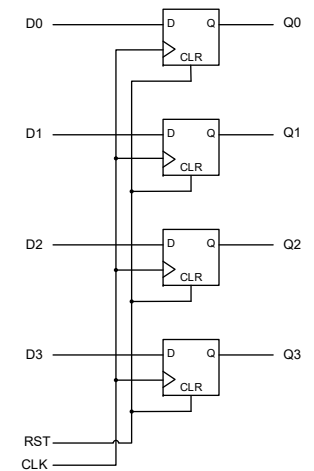
Using muxes to control when register save data

## REGISTER WITH ENABLES

## Register Resets/Clears

- When the power turns on the bit stored in a flip-flop will initialize to a random value
- Better to initialize it to a known value (usually 0's)
- Use a special signal called "reset" to force the flip-flops to 0's

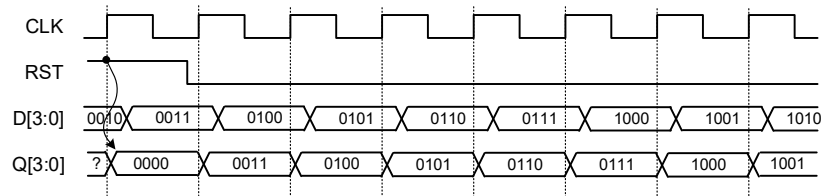
CLK	RST	D <sub>i</sub>	Q <sub>i</sub> *
1,0	X	X	Q <sub>i</sub>
↑	1	X	0
↑	0	0	0
↑	0	1	1



4-bit Register

## Register Problem

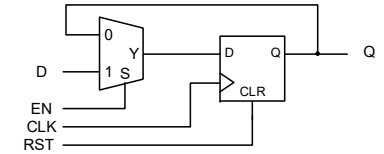
- Whatever the D value is at the clock edge is sampled and passed to the Q output until the next clock edge
- Problem: Register will save data on EVERY edge
  - Often we want the ability to save on one edge and then keep that value for many more cycles



4-bit Register – On clock edge, D is passed to Q

## Solution

- Registers (D-FF's) will sample the D bit every clock edge and pass it to Q
- Sometimes we may want to hold the value of Q and ignore D even at a clock edge
- We can add an enable input and some logic in front of the D-FF to accomplish this

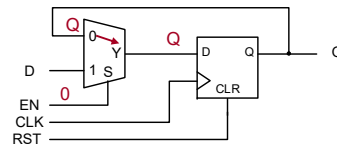


FF with Data Enable  
(Always clocks, but selectively chooses old value, Q, or new value D)

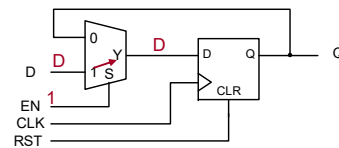
CLK	RST	EN	D <sub>i</sub>	Q <sub>i</sub> *
0,1	X	X	X	Q <sub>i</sub>
↑	1	X	X	0
↑	0	0	X	Q <sub>i</sub>
↑	0	1	0	0
↑	0	1	1	1

## Registers w/ Enables

- When EN=0, Q value is passed back to the input and thus Q will maintain its value at the next clock edge
- When EN=1, D value is passed to the input and thus Q will change at the edge based on D



When EN=0, Q is recycled back to the input

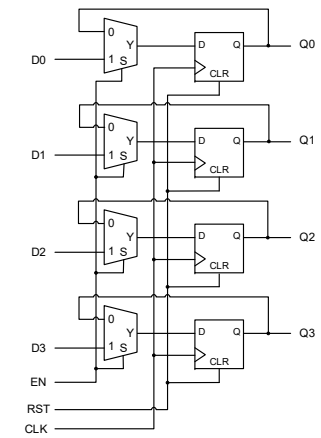


When EN=1, D input is passed to FF input

## 4-bit Register w/ Data (Load) Enable

- Registers (D-FF's) will sample the D bit every clock edge and pass it to Q
- Sometimes we may want to hold the value of Q and ignore D even at a clock edge
- We can add an enable input and some logic in front of the D-FF to accomplish this

CLK	RST	EN	D <sub>i</sub>	Q <sub>i</sub> *
0,1	X	X	X	Q <sub>i</sub>
↑	1	X	X	0
↑	0	0	X	Q <sub>i</sub>
↑	0	1	0	0
↑	0	1	1	1



4-bit register with 4-bit wide 2-to-1 mux in front of the D inputs

## Registers w/ Enables

- The D value is sampled at the clock edge only if the enable is active
- Otherwise the current Q value is maintained

