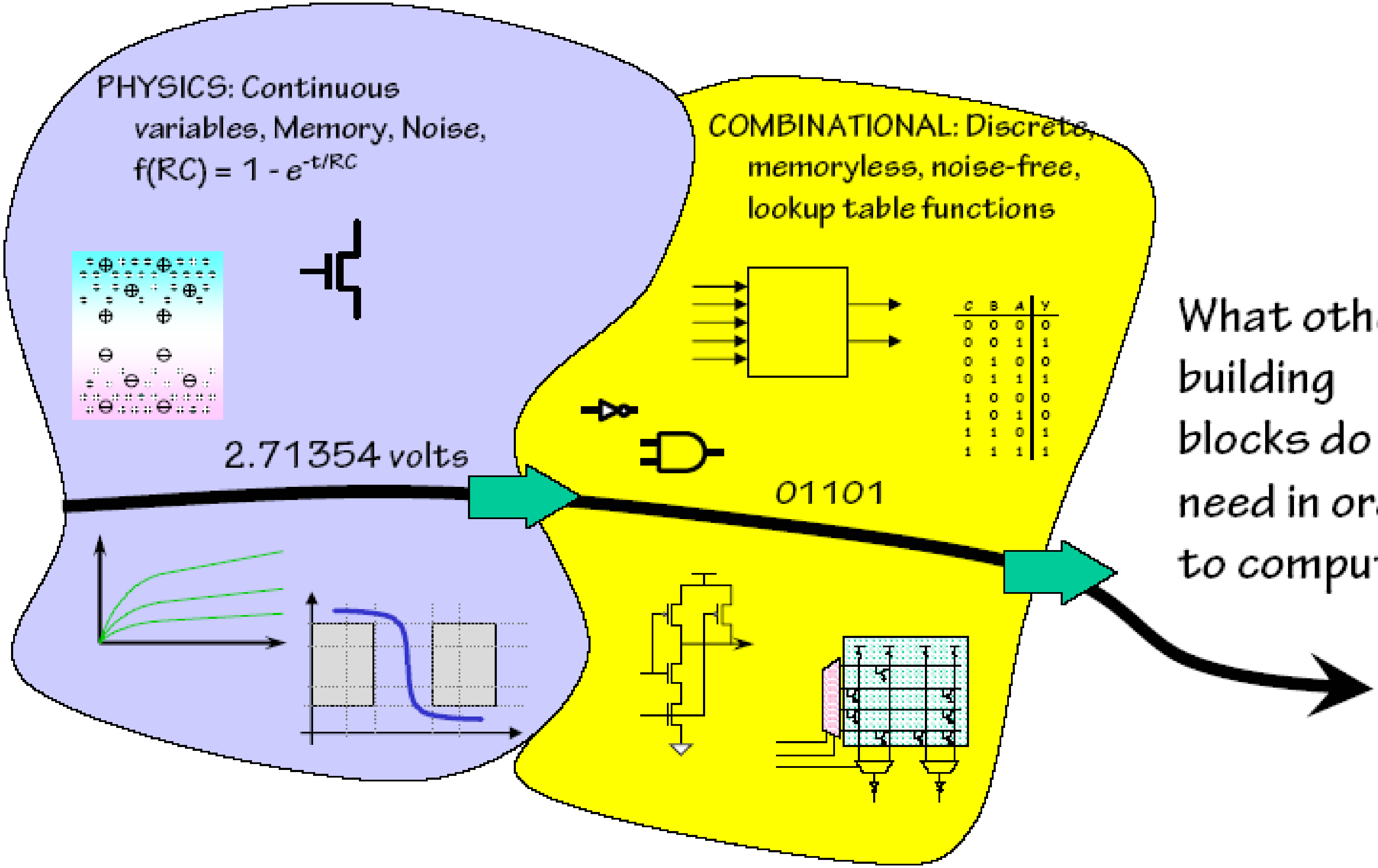
EECS 3201: Digital Logic Design Lecture 9

Ihab Amer, PhD, SMIEEE, P.Eng.

Progress so far…



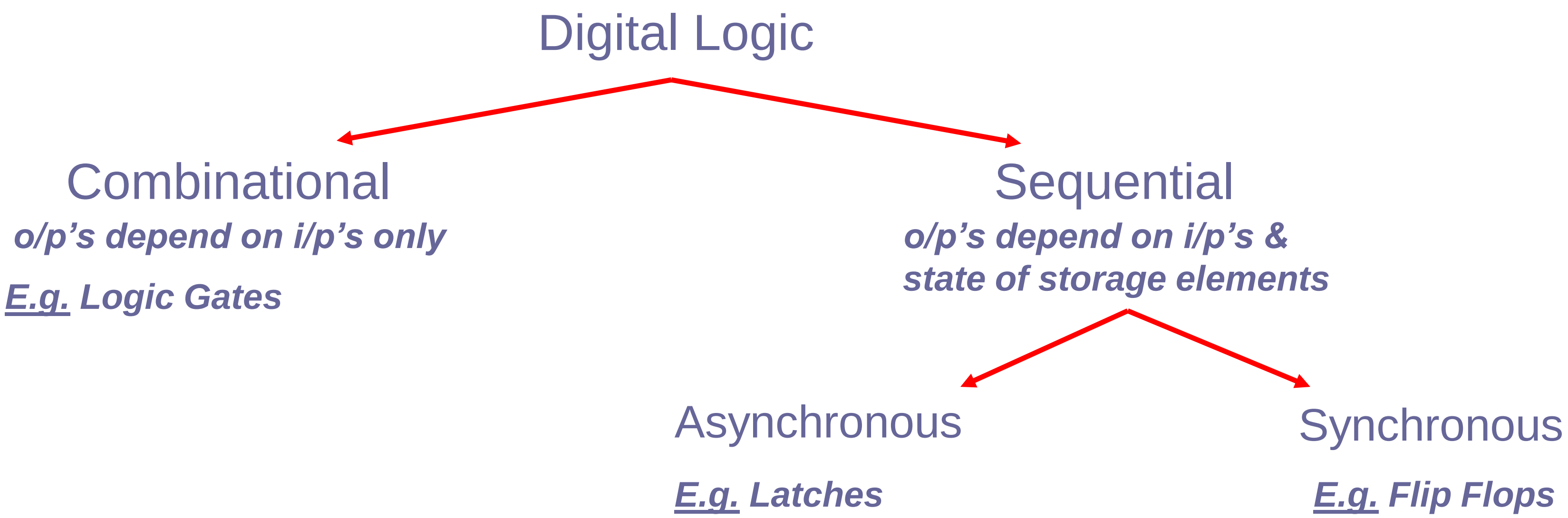
PHYSICS: Continuous variables, Memory, Noise, $f(RC) = 1 - e^{-t/RC}$

COMBINATIONAL: Discrete, memoryless, noise-free, lookup table functions

C	B	A	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

What other building blocks do we need in order to compute?

Digital Logic Classification



Digital Logic

Combinational

o/p's depend on i/p's only

E.g. Logic Gates

Sequential

o/p's depend on i/p's & state of storage elements

Asynchronous

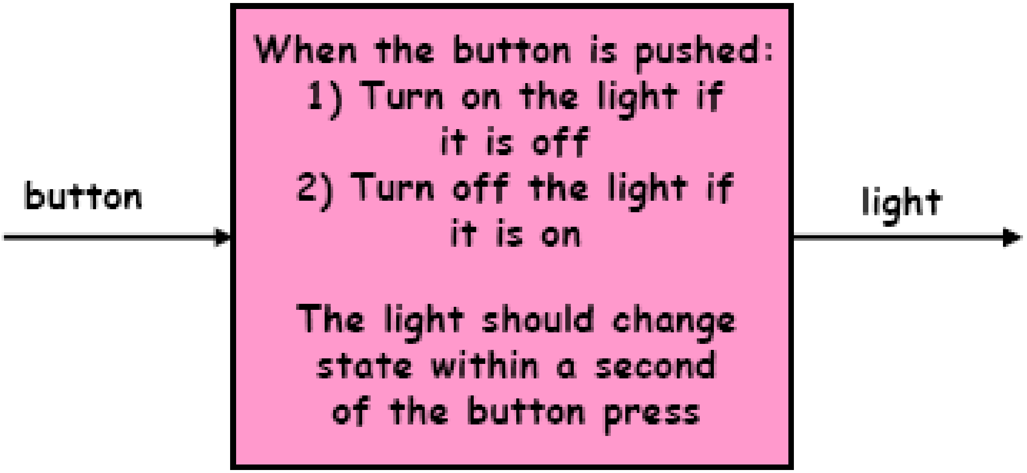
E.g. Latches

Synchronous

E.g. Flip Flops

Think about this…

With the info we encountered so far, can we build this?

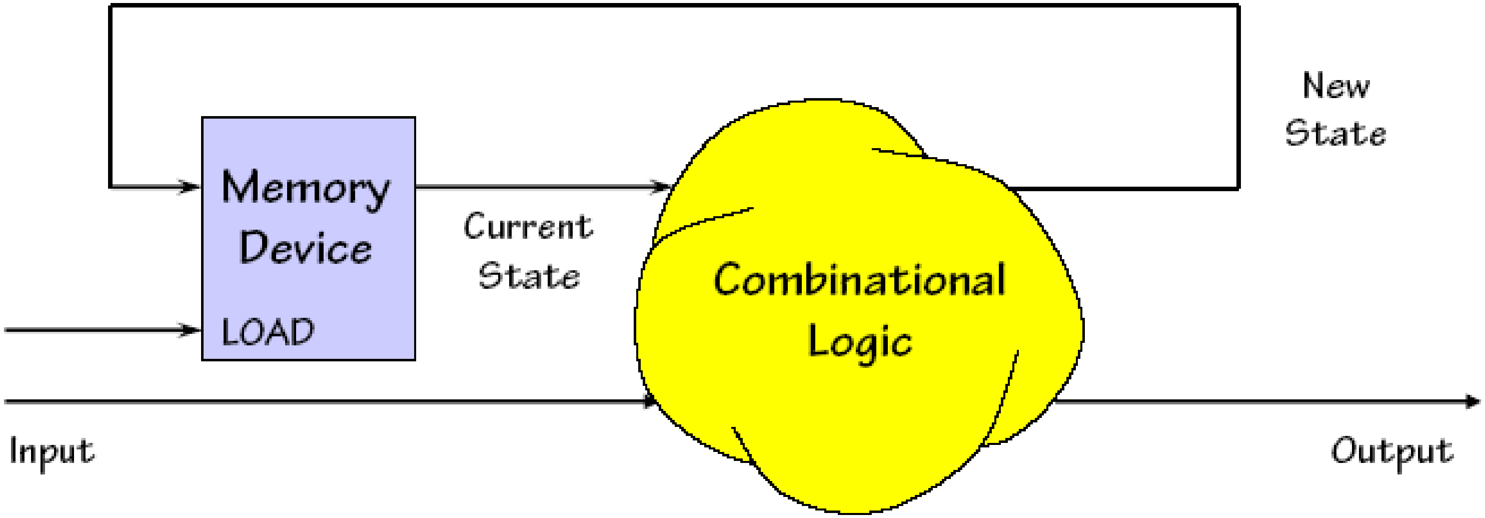

button
When the button is pushed:
1) Turn on the light if it is off
2) Turn off the light if it is on
The light should change state within a second of the button press
light

No!

1. "State" – i.e. the circuit should have memory
2. The o/p changes by an i/p "event" (pushing a button) rather than an input "value" (level)

What does it take?

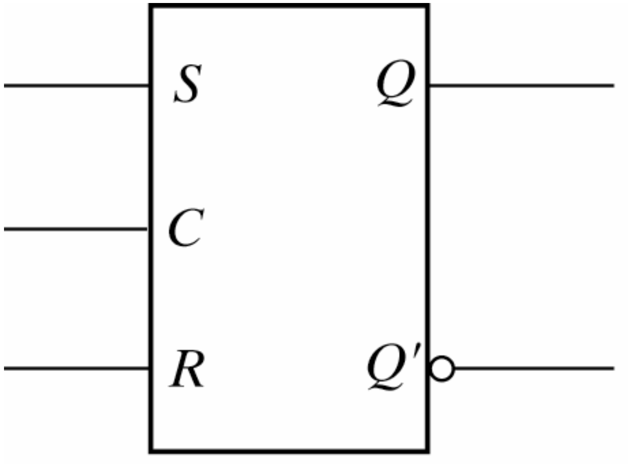
- Ability to store digital *state*

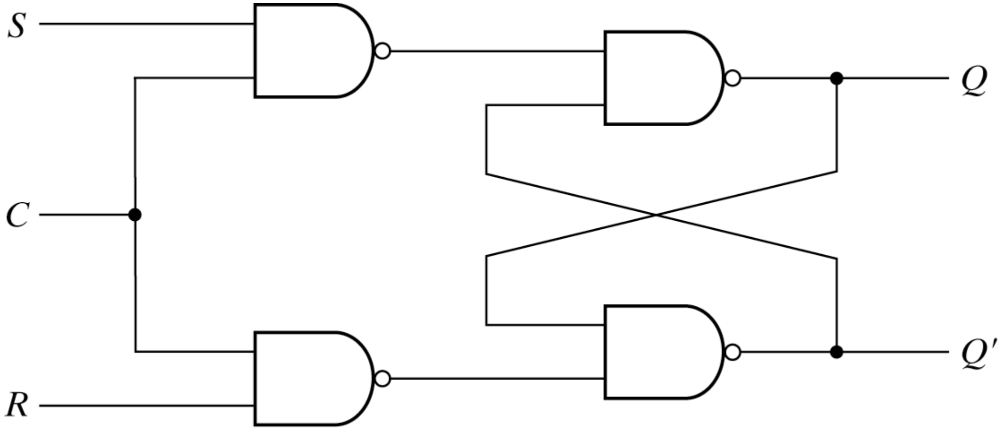


- Memory stores *current state*
- Combinational Logic computes
  - □ *Next State* (from input, current state)
  - □ *Output* (from input, current state)
- State changes on LOAD control input

What is a Latch?

SR Latch



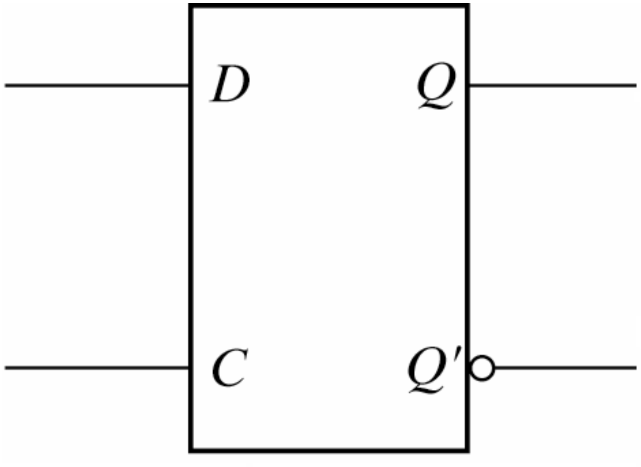


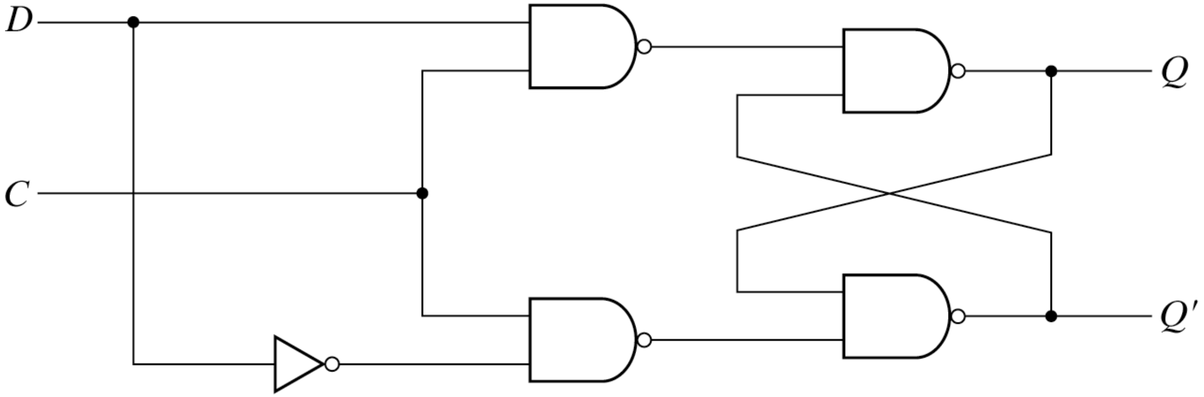
(a) Logic diagram

C	S	R	Next state of Q
0	X	X	No change
1	0	0	No change
1	0	1	Q = 0; Reset state
1	1	0	Q = 1; set state
1	1	1	Indeterminate

(b) Function table

D Latch





(a) Logic diagram

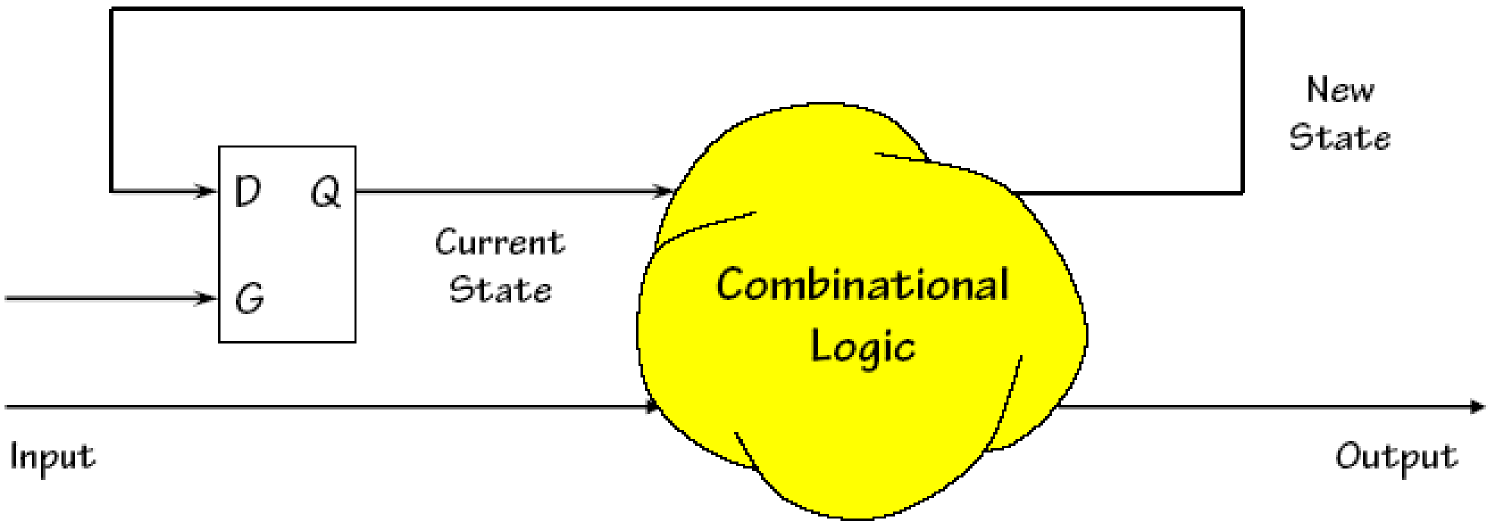
C	D	Next state of Q
0	X	No change
1	0	$Q = 0$; Reset state
1	1	$Q = 1$; Set state

(b) Function table

HDL for D Latch

```
module D_latch (Q,D,control);
   output Q;
   input D,control;
   reg Q;
   always @ (control or D)
   if (control) Q = D;     //Same as: if (control == 1)
endmodule
```

Lets try this out…



G = 1 → Latch transparent

G = 0 → Latch stores state

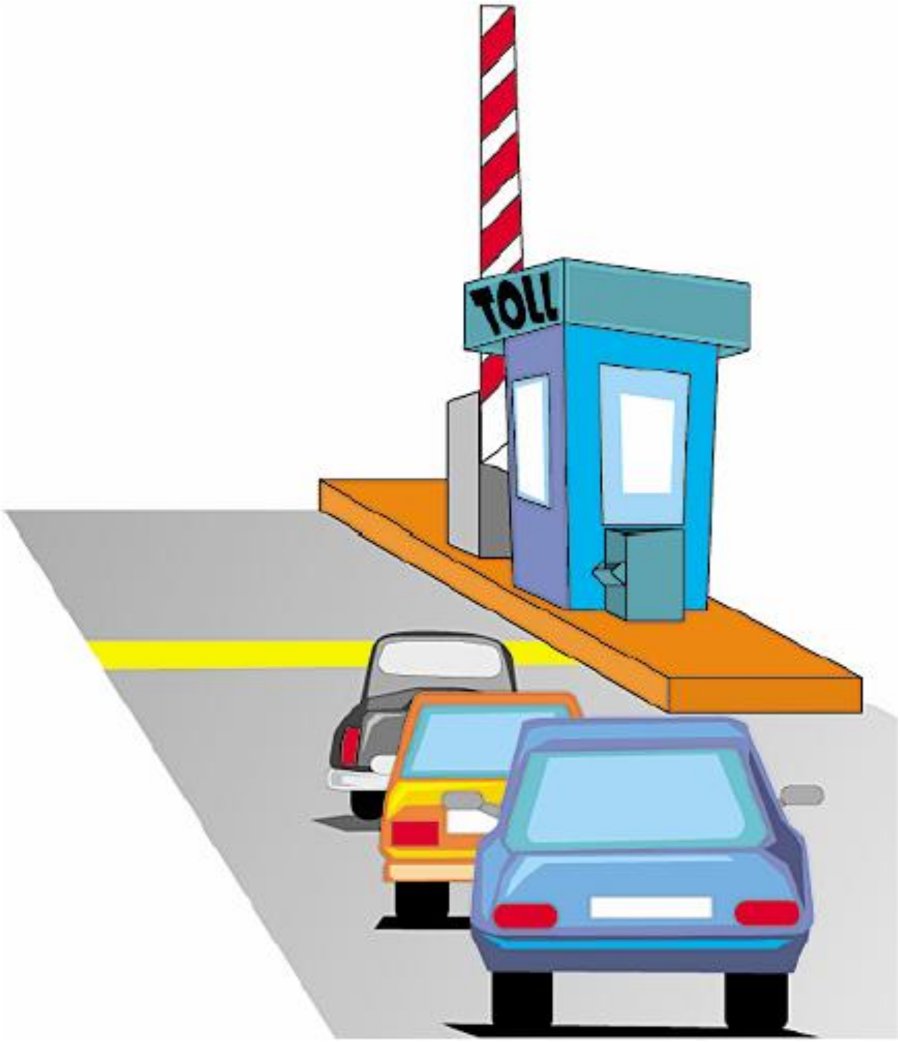
Special timing considerations should be taken!

Flakey Control System

Here is a strategy to save a couple of dollars in the coming holidays!

Flakey Control System

Here is a strategy to save a couple of dollars in the coming holidays!



Flakey Control System

Here is a strategy to save a couple of dollars in the coming holidays!

Escapement Strategy

The Solution:
Add two gates and only open one at a time

Escapement Strategy

The Solution:
Add two gates and only open one at a time

Escapement Strategy

The Solution:
Add two gates and only open one at a time

Escapement Strategy

The Solution:
Add two gates and only open one at a time

Escapement Strategy

The Solution:
Add two gates and only open one at a time

Escapement Strategy

The Solution:
Add two gates and only open one at a time

Escapement Strategy

The Solution:
Add two gates and only open one at a time

Escapement Strategy

The Solution:
Add two gates and only open one at a time

Escapement Strategy

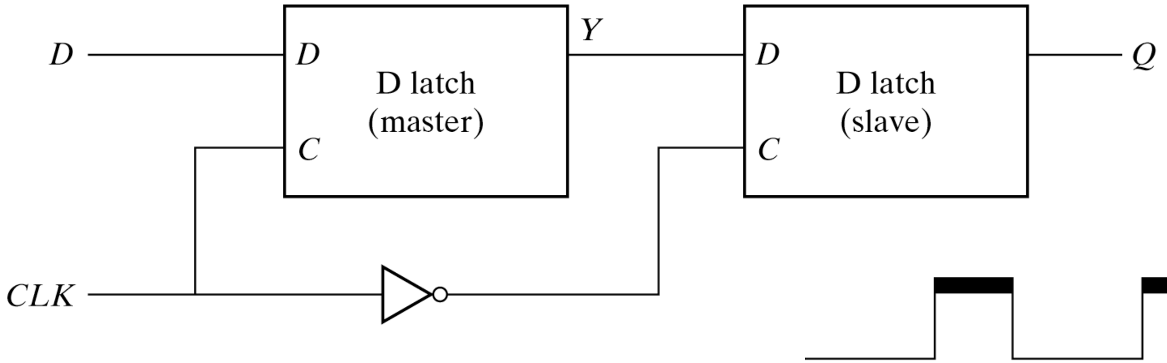
The Solution:
Add two gates and only open one at a time

Escapement Strategy

The Solution:
Add two gates and only open one at a time

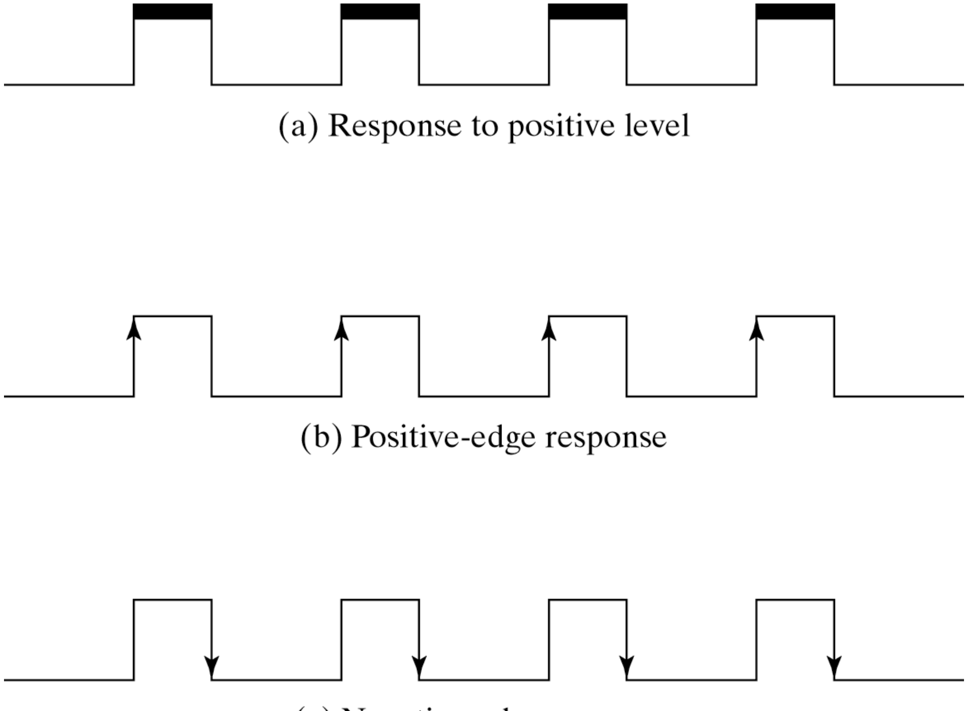
Key: At no time is there an open path through both gates

Back to Digital Systems…



Master-Slave Flip-Flop

Same idea as double-gate toll station



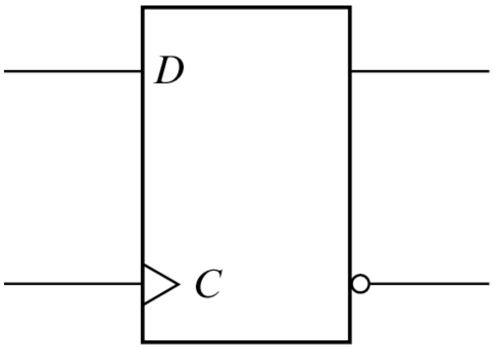
(a) Response to positive level

(b) Positive-edge response

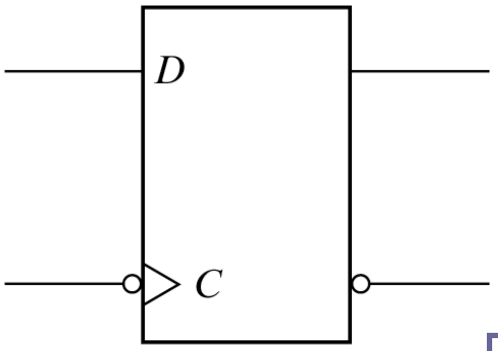
(c) Negative-edge response

What is a FF?

Simply, it is a *clocked* latch



(a) Positive-edge



(a) Negative-edge

Characteristics Table

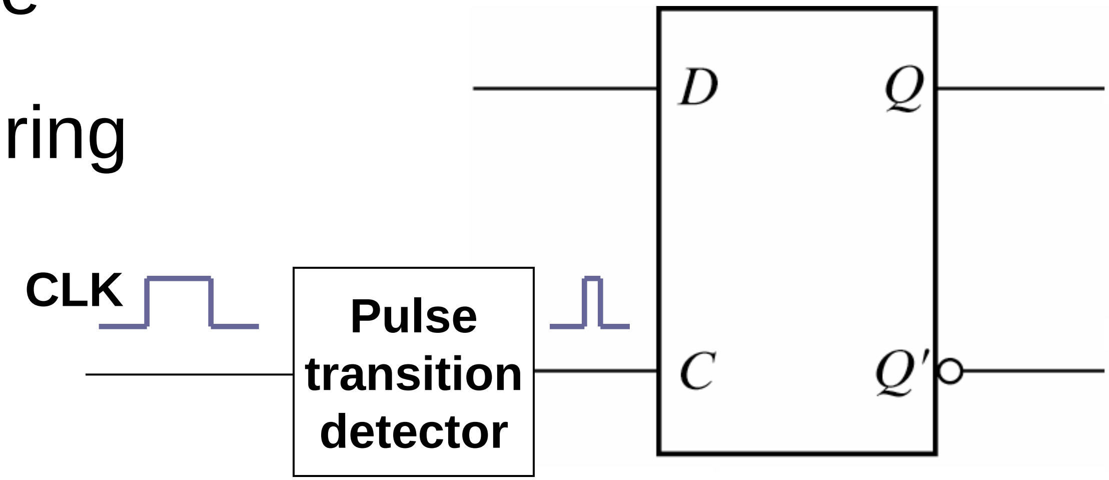
D	**Q(t+1)**	
0	0	**Reset**
1	1	**Set**

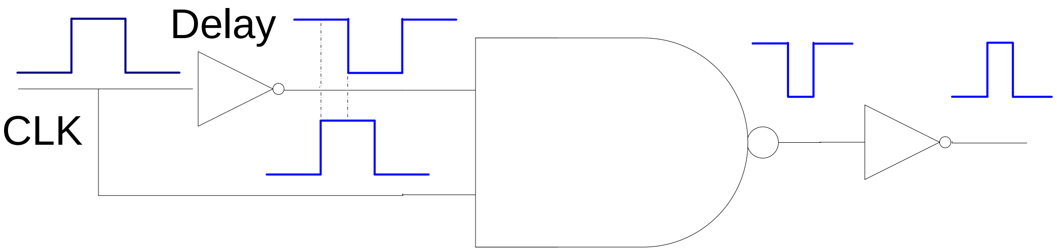
HDL *(Behavioral)*

```
module D_FF (Q,D,CLK);
  output Q;
  input D,CLK;
  reg Q;
  always @ (posedge CLK)
    Q = D;
endmodule
```

How to construct it?

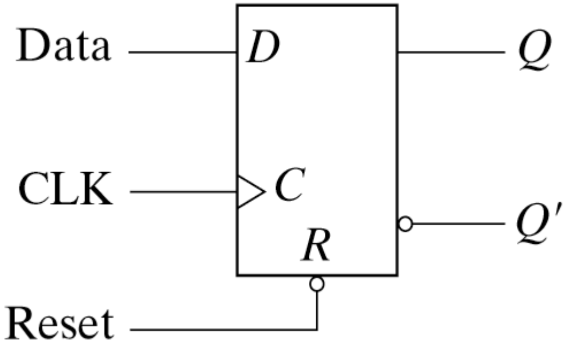
- Master-Slave
- Edge Triggering





A type of pulse transition detector

D Flip-Flop with Asynch RST



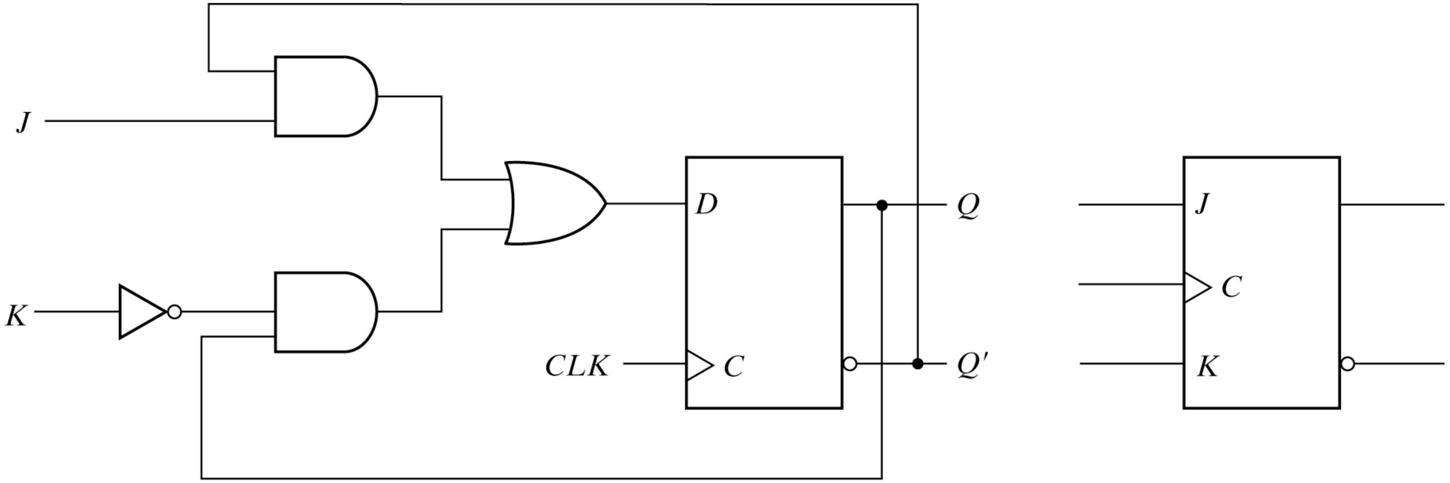
R	C	D	Q	Q'
0	X	X	0	1
1	↑	0	0	1
1	↑	1	1	0

HDL *(Behavioral)*

```
module DFF (Q,D,CLK,RST);
  output Q;
  input D,CLK,RST;
  reg Q;
  always @ ( posedge CLK or negedge RST)
    if (~RST) Q = 1'b0;
    else Q = D;
endmodule
```

How would the D-FF with Synch RST look like?

JK Flip-Flop



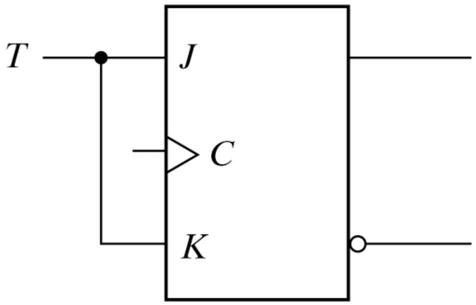
(a) Circuit diagram

(b) Graphic symbol

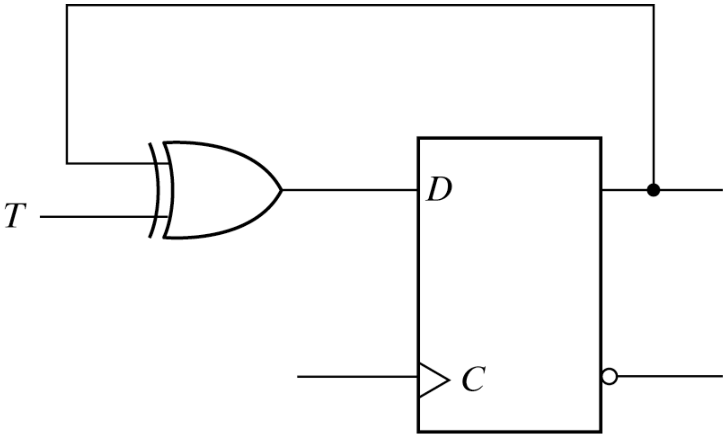
Characteristics Table

J	K	Q(t+1)	
0	0	Q(t)	**No Change**
0	1	0	**Reset**
1	0	1	**Set**
1	1	Q'(t)	**Complement**

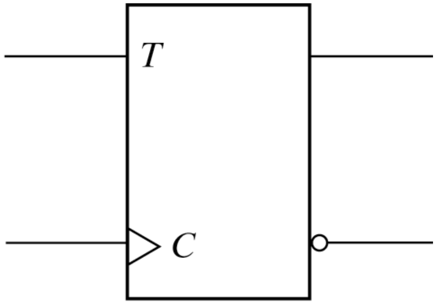
T Flip-Flop



(a) From *JK* flip-flop



(b) From *D* flip-flop



(c) Graphic symbol

Characteristics Table

T	**Q(t+1)**	
0	Q(t)	**No Change**
1	Q'(t)	**Complement**

HDL for JK & T Flip-Flops

```
//T flip-flop from D flip-flop and gates
module TFF (Q,T,CLK,RST);
   output Q;
   input T,CLK,RST;
   wire DT;
   assign DT = Q ^ T ;
//Instantiate the D flip-flop
   DFF TF1 (Q,DT,CLK,RST);
endmodule
/******************************************/
//JK flip-flop from D flip-flop and gates
module JKFF (Q,J,K,CLK,RST);
   output Q;
   input J,K,CLK,RST;
   wire JK;
   assign JK = (J & ~Q) | (~K & Q);
//Instantiate D flipflop
   DFF JK1 (Q,JK,CLK,RST);
endmodule
```

FF's with asynchronous RST

```
//D flip-flop
module DFF (Q,D,CLK,RST);
   output Q;
   input D,CLK,RST;
   reg Q;
   always @ (posedge CLK or negedge RST)
     if (~RST) Q = 1'b0;
     else Q = D;
endmodule
```

HDL for JK FF (Functional)

```
module JK_FF (J,K,CLK,Q,Qnot);
   output Q,Qnot;
   input  J,K,CLK;
   reg  Q;
   assign Qnot = ~ Q ;
   always @ (posedge CLK)
          case ({J,K})
            2'b00: Q = Q;
            2'b01: Q = 1'b0;
            2'b10: Q = 1'b1;
            2'b11: Q = ~ Q;
          endcase
endmodule
```

JK FF without asynchronous RST

Important Book Chapters

- Related sections of chapter 5 in the textbook

References

- “Digital Design (3rd and 4th Editions)”, Morris Mano , Prentice Hall, (2002/2007)
- “Digital Fundamentals (10th Edition)”, Thomas L. Floyd, Prentice Hall, 2010
- http://ece.gmu.edu/coursewebpages/ECE/ECE448/S10/
- cpk.auc.dk/education/SSU-2007/mm10/ssu_mm10.pdf
- www.ece.cmu.edu/~thomas/VSLIDES.pdf
- http://ece.gmu.edu/courses/ECE448/index_S06.htm
- MIT Lecture Notes on: http://www.ece.concordia.ca/~asim