0x03 - Fixed/Floating Point and Sequential Logic

ENGR 3410: Computer Architecture

Jon Tse

Fall 2020

Housekeeping

- HW2 due 9/21
- Lab 1, Part 1 & 2, due 9/21
  - You are allowed to work in groups for BOTH parts
  - I think Canvas allows for submission as a group

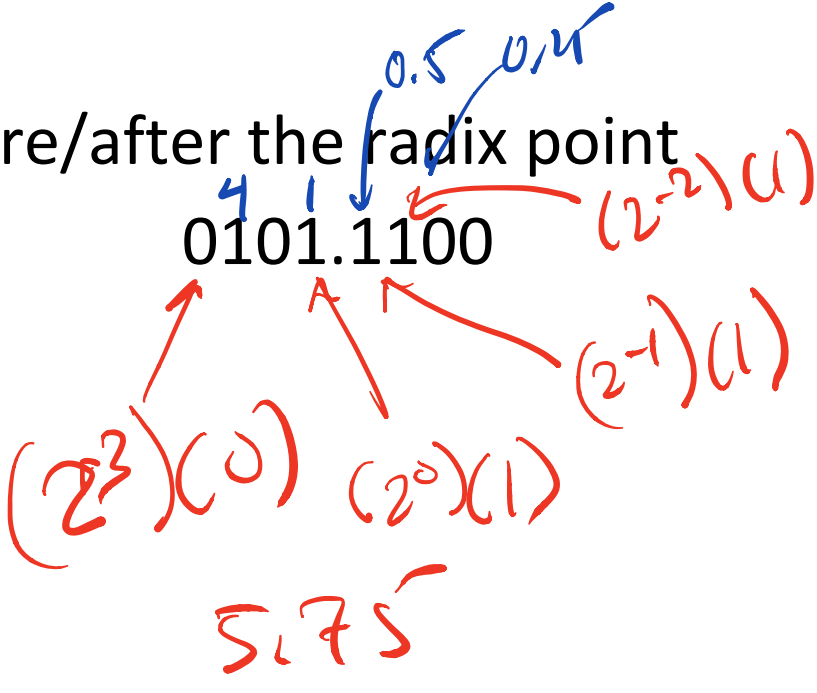
- Quick Aside about Makefiles

Wrap Up from ~~Last Week~~ Tuesday

Fixed Point: Integer scaled by a **fixed** factor.

Fixed # of digits before/after the radix point

$$x = \sum_i r^i d_i$$



Fixed Point Numbers

before. after.

IQ Notation

I4Q2 Signed, 4 bits before, 2 bits after

xxxx.xx

4 2

U2Q5 Unsigned, 2 before, 5 after

xx.xxxxx

2 5

I = 2's complement

U = unsigned

Fixed Point Addition

Align the radix point, then add!

$x_{10} = 4.5, y_{10} = 10.125$

$x_2 = 100.10000, y_2 = 01010.001$

U3Q5 U5Q3

```
  10.125
+  4.5
-------
  14.625
```

```
  01010.001
+   100.10000
-------------
  01110.10100
```

U3Q5

$8 + 4 + 2 + 0.5 + 0.125 = 14.625$

U5Q5 U5Q3

Fixed Point Multiplication

```
        1111.1000   -0.5 in I4Q4
      * 0011.0000    3.0 in I4Q4
      ------------
        .00000000
       0.0000000
      00.000000
     000.00000
    1111.1000
   11111.000
  000000.00
 0000000.0
------------------
00101110.10000000   46.5 In I8Q8
```

Fixed Point Multiplication

```
          1111.1000   -0.5 in I4Q4
        * 0011.0000    3.0 in I4Q4
        ------------
          .00000000
           0.0000000
          00.000000
         000.00000
        1111.1000
       11111.000
      000000.00
     0000000.0
     ------------------
0010 1110.10000000   46.5 In I8Q8
```

Incorrect?

Correct!

Fixed Point Multiplication

```
        1111.1000   -0.5 in I4Q4
      * 0011.0000    3.0 in I4Q4
  ---------------
 0000000.00000000
 0000000.00000000
 0000000.00000000
 0000000.00000000
 1111111.10000000   ☐ I8Q8!!
 1111111.00000000
 0000000.00000000
 0000000.00000000
  ---------------
11111110.10000000   -1.5 In I8Q8
```

Fixed Point Multiplication

```
        01.11     I2Q2    d1.75
      * 11.10     I2Q2   -d0.50
      ------
        .0000
       0.111
      01.11
     011.1
    0111.
   0111 .
  0111  .
 0111   .
0111    .
0110 1111.0010    I4Q4   -d0.875
```

Fixed Point Multiplication


```
   01.11     I2Q2   d1.75
*  11.10     I2Q2  -d0.50
-----------
    .0000
   0.111
  01.11
 011.1
0111.          ☐ From sign extension!
 0111 .
  0111  .
   0111 .
    0111.      ☐ No effect on output
-----------
0110 1111.0010  I4Q4  -d0.875
```

Observations

The product is wider than the inputs

InQx*ImQy=I(n+m)Q(x+y)

Sign extend everything

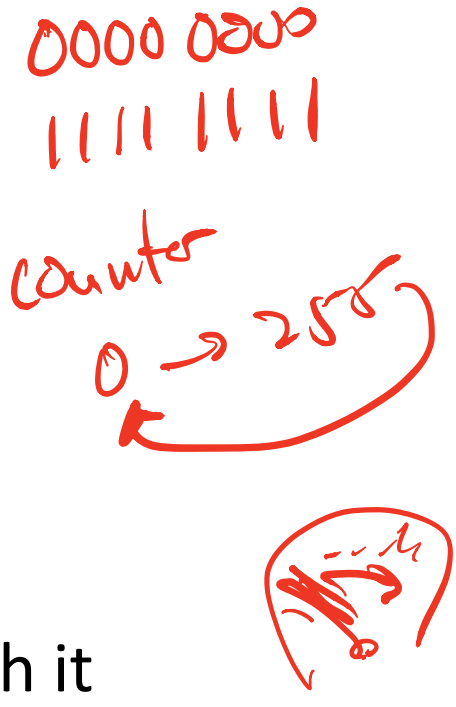
Finite Memory

- We can not expand every time.
- Usually, output format is input format.
- LSBs dropped are lost precision
  - Always a little bad
- MSBs dropped are occasional catastrophes.
  - Totally fine or absolutely terrible
  - Overflow Errors

1000 = -8

How to Handle?

- Overflow
  - Catastrophic value change
- Saturate
  - Wrong, but closer than Overflow
- Make the programmer deal with it
  - They know their needs better than we do



Precision vs Max Magnitude

- Humans handle this with scientific notation.
  - Lose **precision** to handle memory constraints
  - Like choosing IQ format after each calculation(ish)

$$1.234 \times 10^{2}$$

$$\text{Significand} \times R^{\text{Exponent}}$$

Lets Design Binary Scientific Notation

- Need to store:
  - Sign
  - Exponent
  - Significand
- Nice to be able to:
  - Fit in memory conveniently
  - Sort easily
  - Accommodate bad things happening

$$(-1)^{0} \times 1.234 \times 10^{2}$$

$$(-1)^{Sign} \times Significand \times R^{Exponent}$$

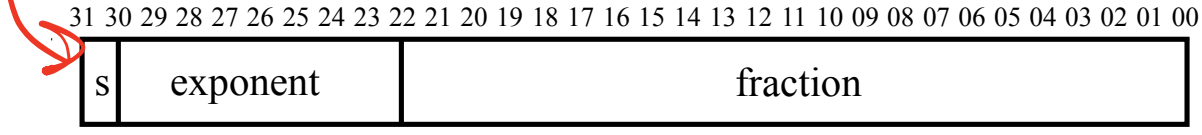
R = 2

Exponent Format

- Could use 2's complement.
  - But that doesn't sort well
- Use 'biased' notation instead.
  - Signed value 'biased' to be unsigned.
  - Biased means add a fixed value
  - Most negative number becomes 0.
- Makes sorting floats easy!

IEEE-754 Single Precision Float

$$\text{Float} = (-1)^{s} \times (1.\text{fraction}) \times 2^{(\text{exponent}-127)}$$



- Record Sign bit
- Convert Significand to U1Q23
  - Track changes to Exponent!
- Drop the MSB of Significand, record the rest
  - Significand = leading one + Fraction
- Bias Exponent by +127, record

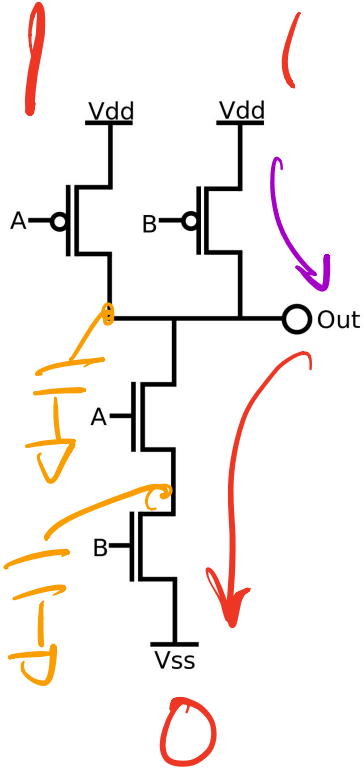
Today

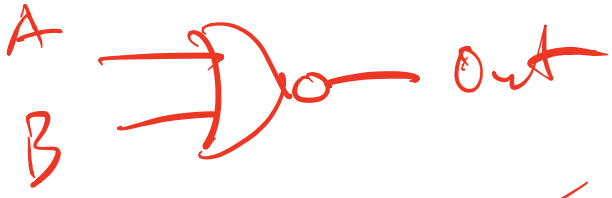
- Dealing with time
- State-holding gates: latches and flip flops

Circuit Timing Behavior

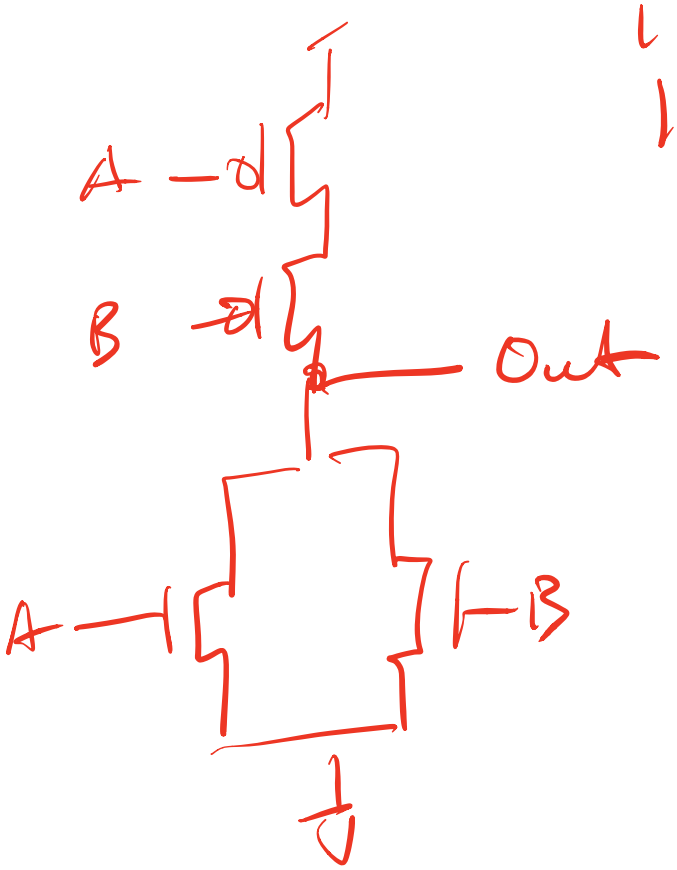
Simplified Model: Fixed Delay Gates

A	B	Out
0	0	1
0	1	1
1	0	1
1	1	0

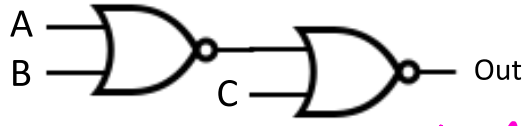




A	B	Out
0	0	1
0	1	0
1	0	0
1	1	0

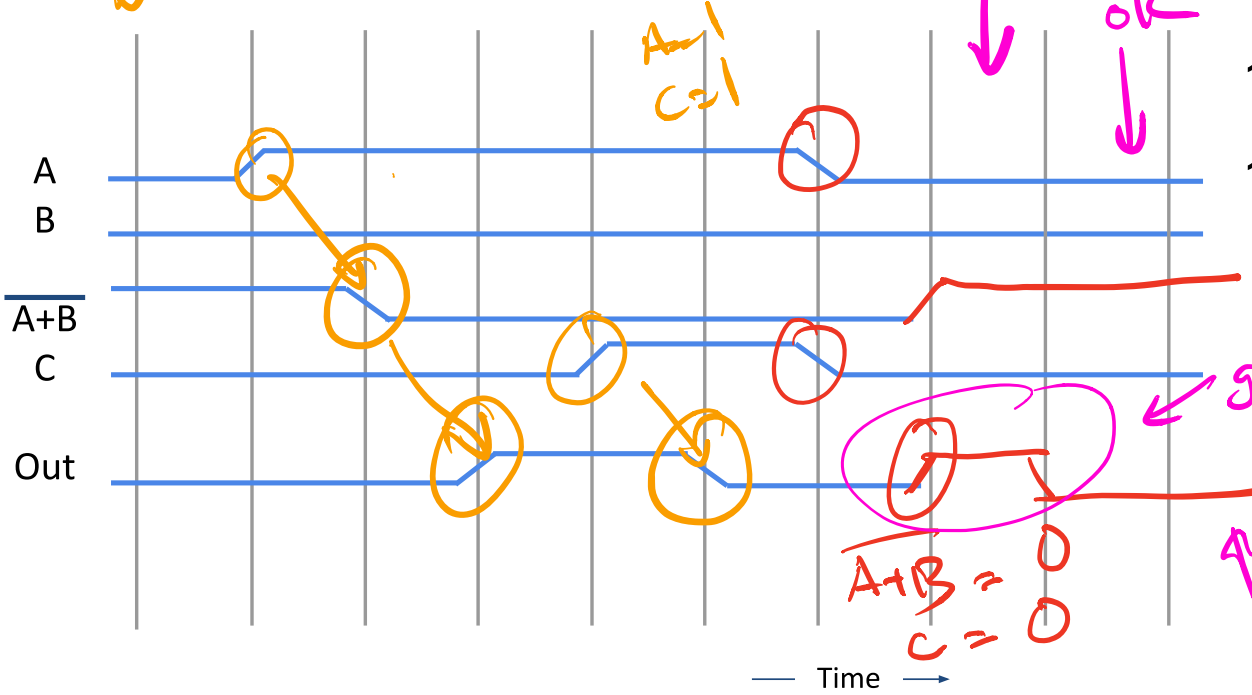


Gate Transitions/Timing Diagram



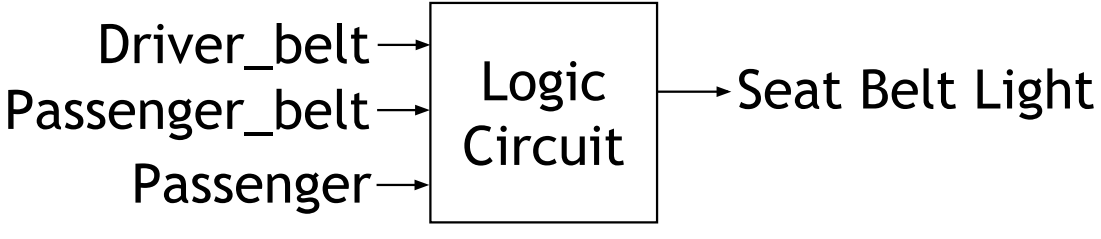
A	B	C	Out
0	0	0	0
0	0	1	0
1	0	0	1
1	0	1	0

$$\text{Out} = \overline{(A+B)} + C = \overline{(A+B)}\,\overline{C}$$



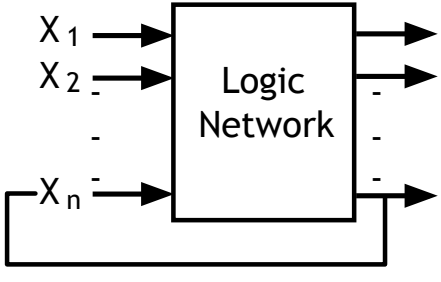
Combinational Logic

- Outputs are a pure function of the inputs
- No feedback between inputs and outputs

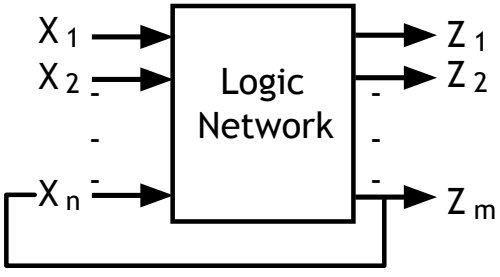


Sequential Logic

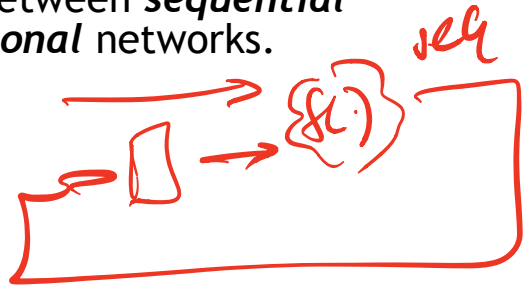
- Outputs are a function of current & previous inputs
- Implies some form of memory – “state holding”



Combinational vs. Sequential Logic

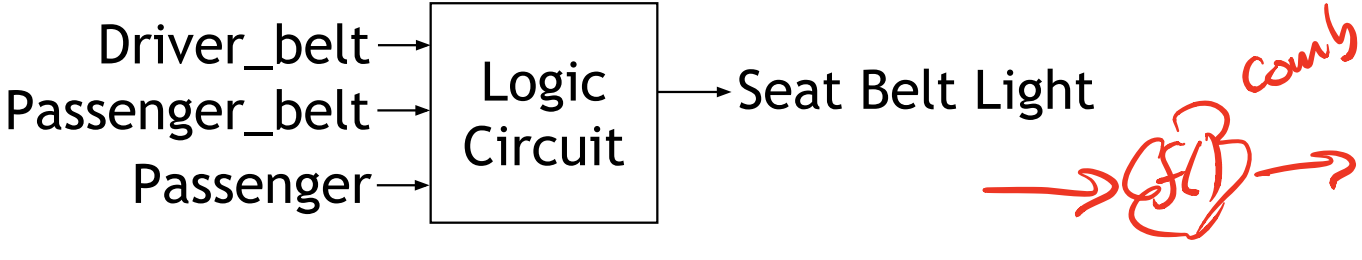


Network implemented from logic gates. The presence of feedback distinguishes between ***sequential*** and ***combinational*** networks.



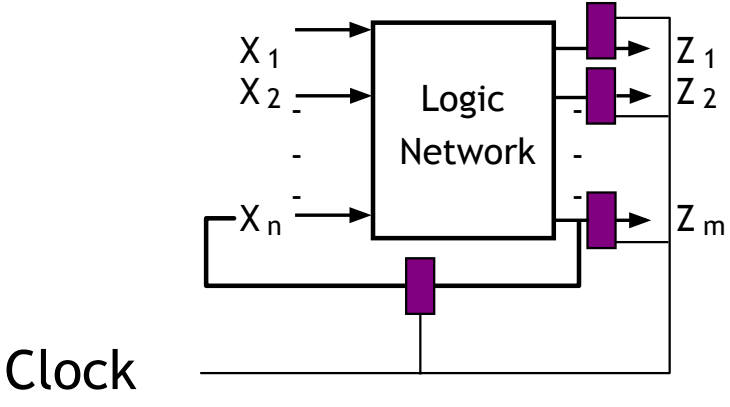
Combinational logic

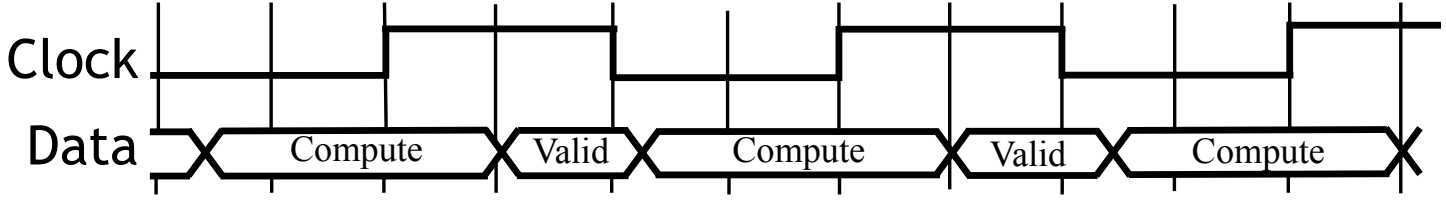
no feedback among inputs and outputs
outputs are a pure function of the inputs
e.g., seat belt light:
(Dbelt, Pbelt, Passenger) mapped into (Light)



Safe Sequential Circuits

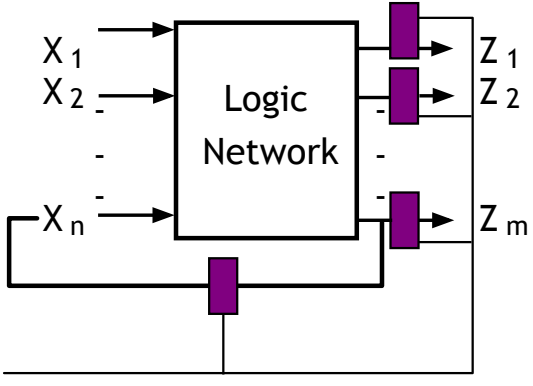
- Clocked elements on feedback, perhaps in/outputs
  - Clock signal synchronizes operation
  - Clocked elements hide glitches/hazards

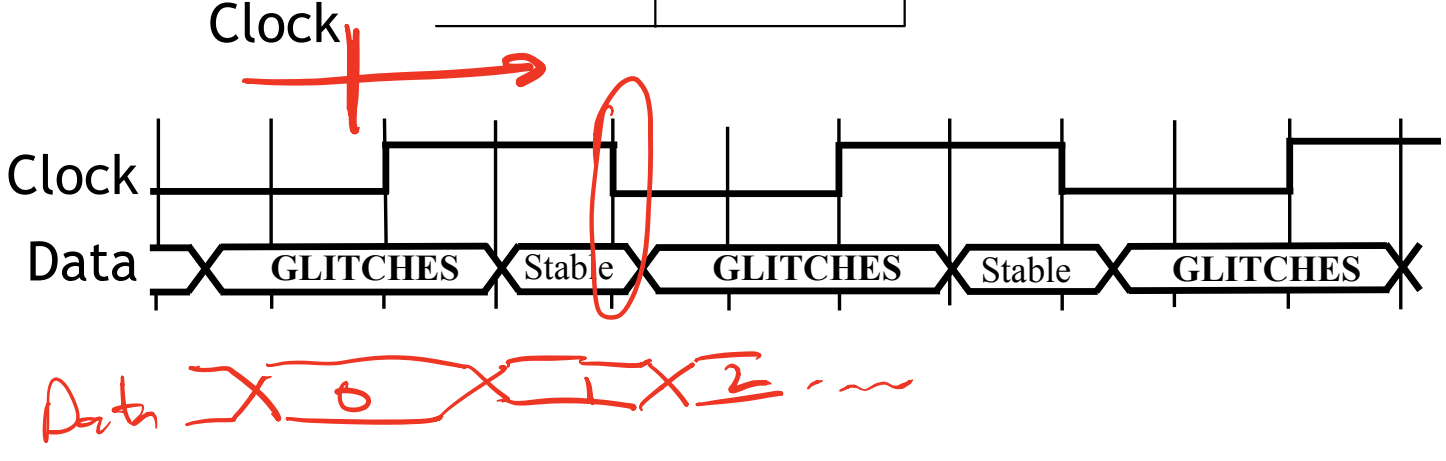




Safe Sequential Circuits

- Clocked elements on feedback, perhaps in/outputs
  - Clock signal synchronizes operation
  - Clocked elements hide glitches/hazards





State-Holding Elements

- Two main categories:
  - Flip Flops and Latches
- “Hold on” to state until triggered to update.
- *Many* varieties

The SR Latch

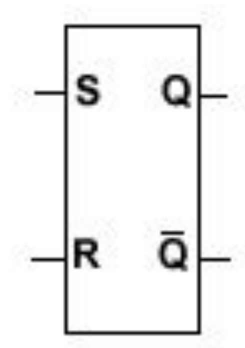
- "SR" for "Set Reset"

NOR based

S	R	Q Next
0	0	Q
0	1	0
1	0	1
1	1	X

→ 0 bad } illegal state

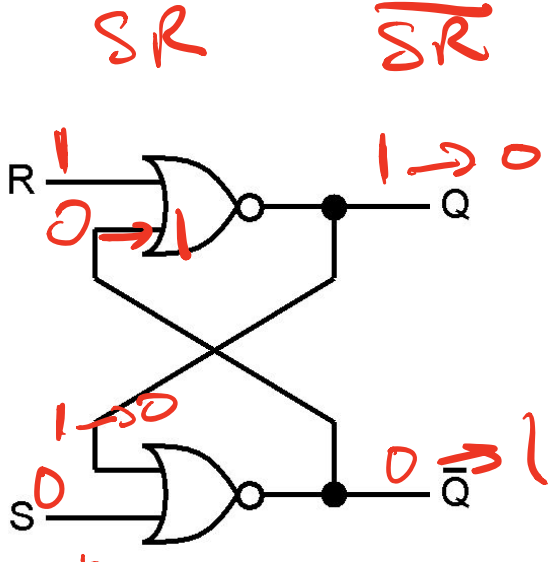
avoid by design



Reinventing the SR Latch

- Implementation: Cross-coupled NOR/NAND

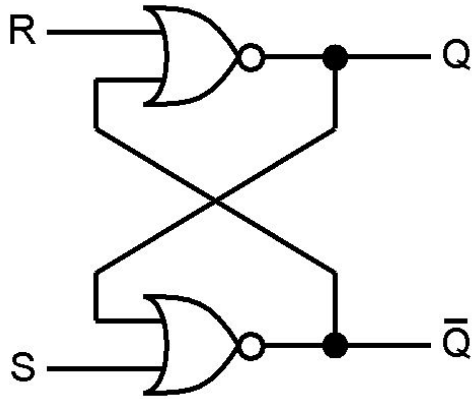
S	R	Q Next
0	0	Q
0	1	0
1	0	1
1	1	X

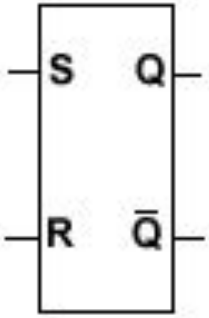


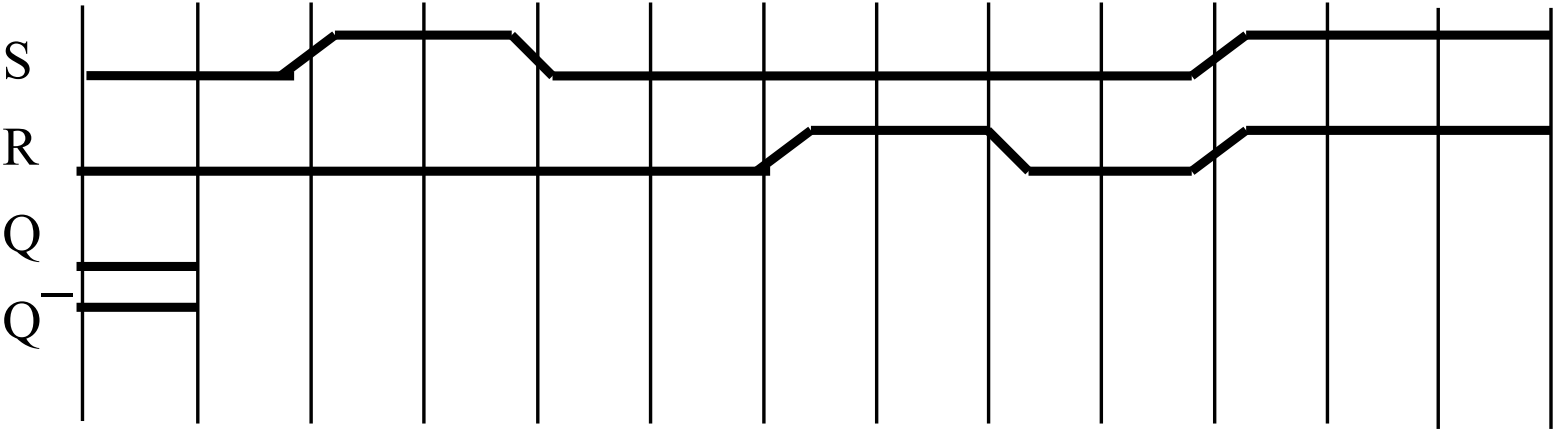
NOR

A	B	Out
0	0	1
0	1	0
1	0	0
1	1	0

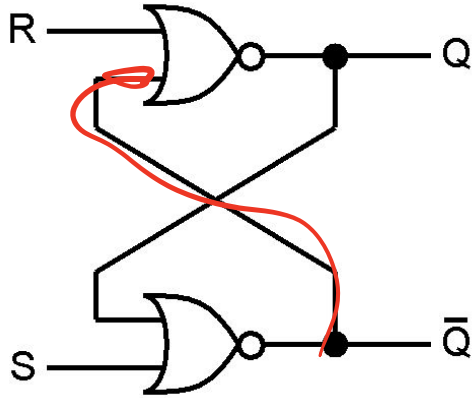
SR Latch Timing

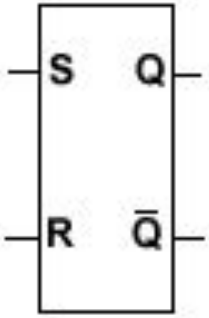


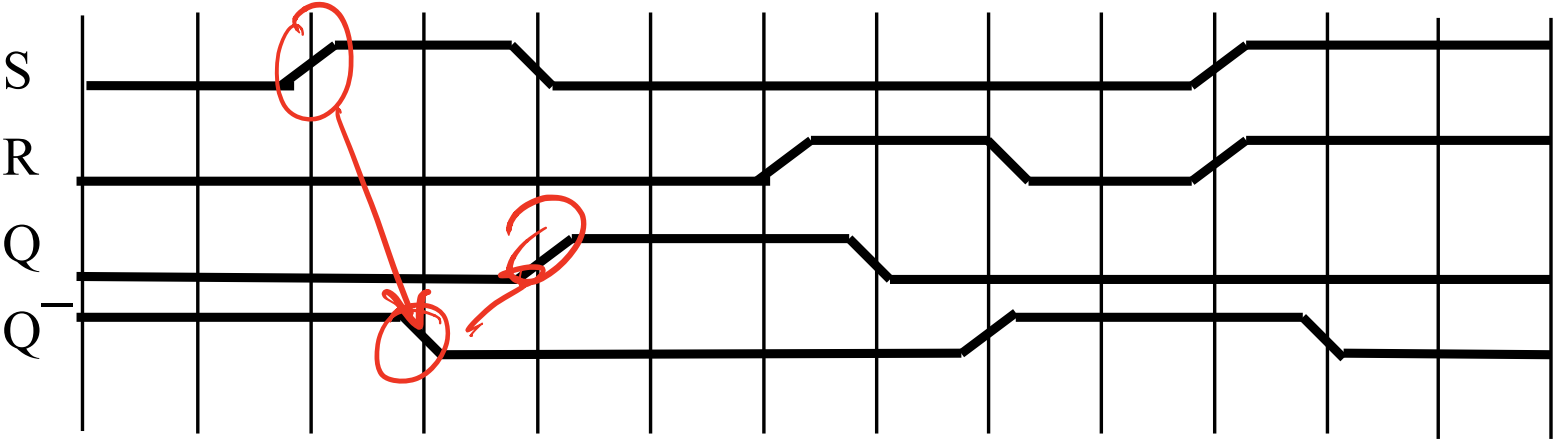




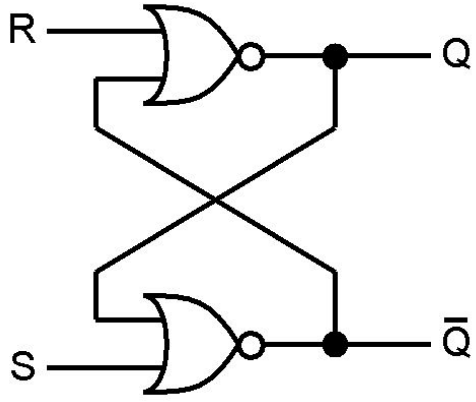
SR Latch Timing

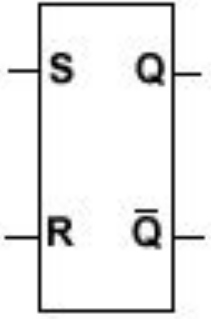




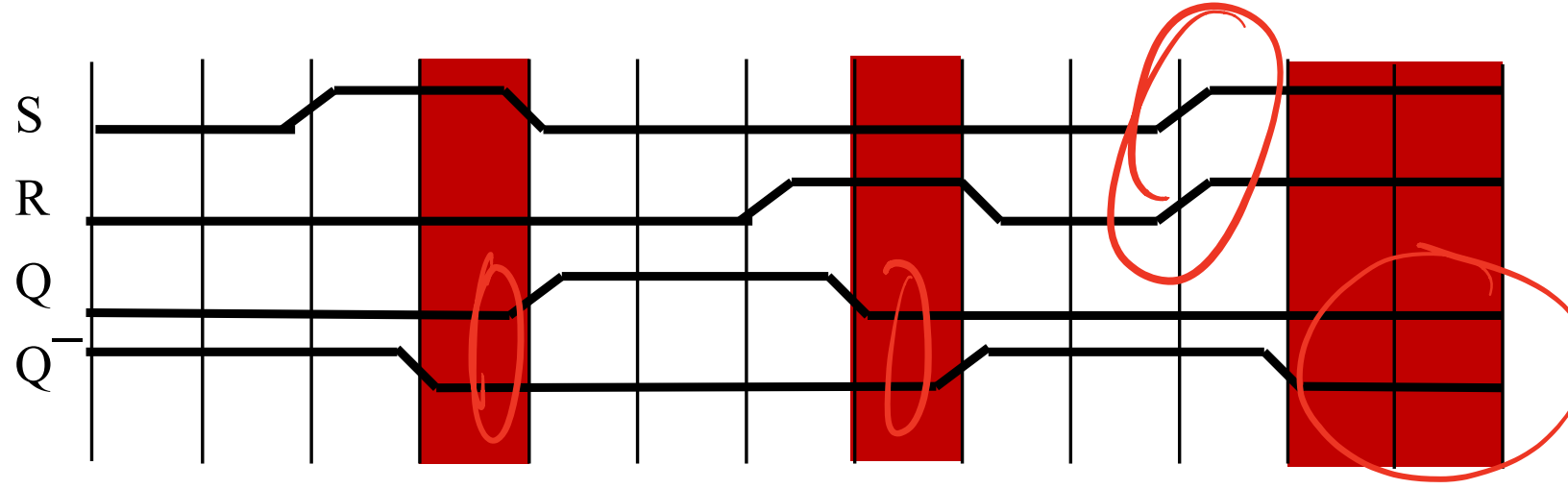


SR Latch Timing

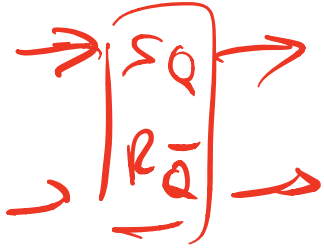




illegal

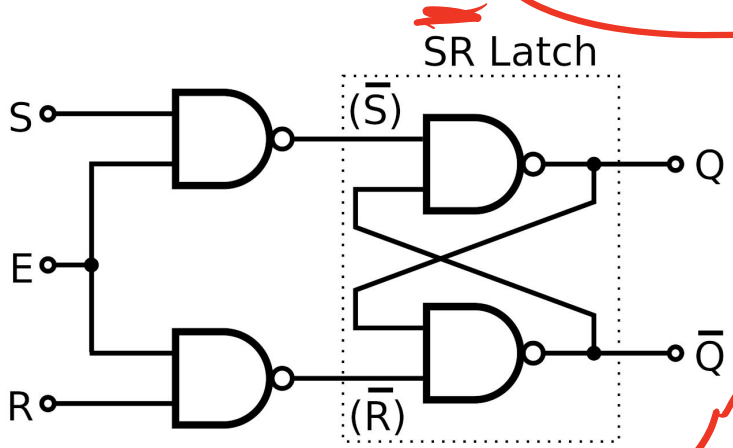


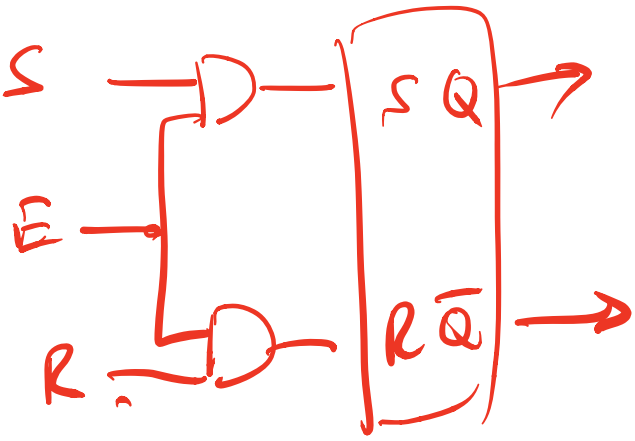
Latches vs Flip Flops

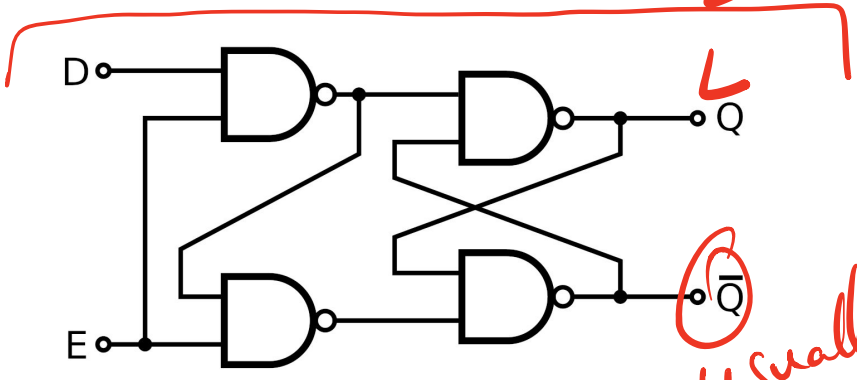


- Confusing vocab – they are similar
- A latch is typically **Enabled**
  - Updates continuously or holds value
- A flip flop is typically **Clocked**
  - Updates only at a specific moment
- All are forms of ‘Bistable Multivibrators’

D-Latch



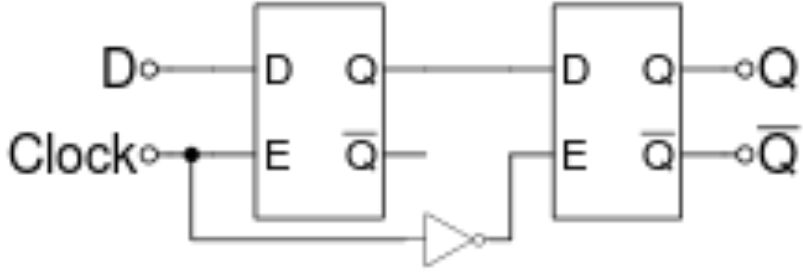




Usually ignore

D-Flip Flop (first draft)

- Two D-Latches in series with opposite polarity Enable lines



- Pulse Triggered
  - Capture on one edge
  - Display on the other

http://en.wikipedia.org/wiki/File:Negative-edge_triggered_master_slave_D_flip-flop.svg

Edge Triggered D-Flip Flop

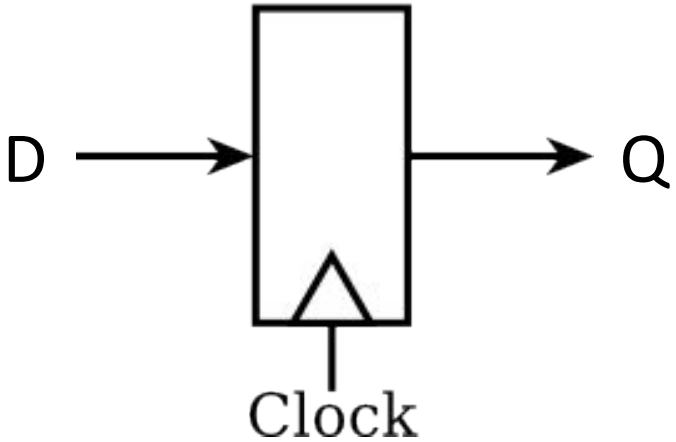
- Functionally similar to prior dual latch flip flop
- Edge Triggered, not Pulse
- Implementation process dependent
  - Can be roughly three SR Latches
  - Can be dynamic logic – learn in VLSI

Edge Triggered D-Flip Flop

- Always has:
  - Clk, D, Q

Clk	D	Q
↑	0	0
↑	1	1
Other	X	Q

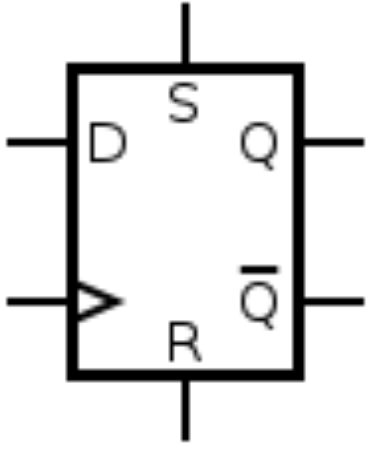
Rising edge



Edge Triggered D-Flip Flop

- Always has:
  - Clk, D, Q

Clk	D	Q
↑	0	0
↑	1	1
Other	X	Q



- May also have:
  - S, R, ~Q
  - **S**et and **R**eset are asynchronous (ignore clock)
    - Can be used to define initial conditions (e.g. at boot)

http://en.wikipedia.org/wiki/File:D-Type_Flip-flop.svg

Common Uses

- In between processing stages
- “Debounce” inputs
  - Hide external noise / uncertainty from the inputs
- Synchronization