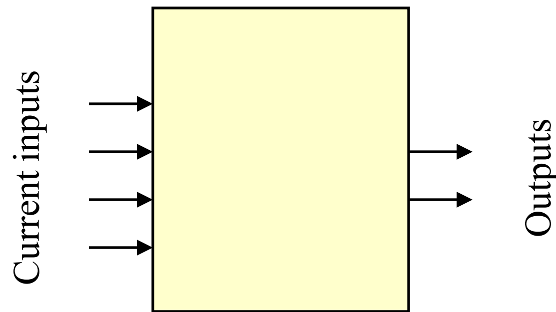
Introduction to Digital Logic

Lecture 16:
Combinational vs. Sequential Logic
Bistables

© Mark Redekopp, All rights reserved

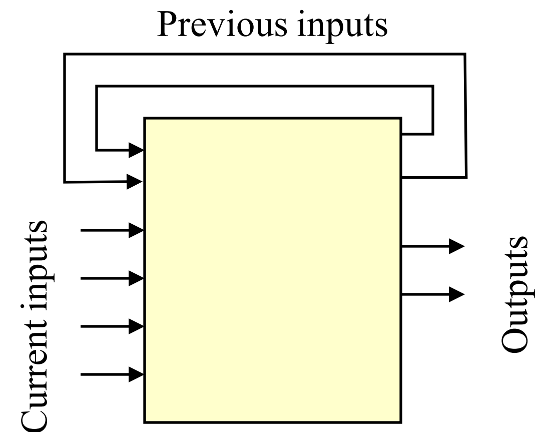
Combinational vs. Sequential

Combinational Logic



Outputs depend only on current outputs

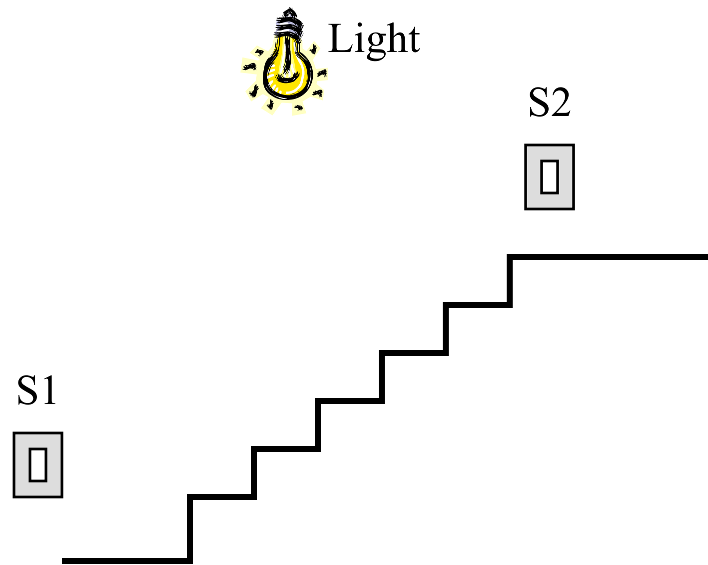
Sequential Logic



Outputs depend on current inputs and previous inputs

© Mark Redekopp, All rights reserved

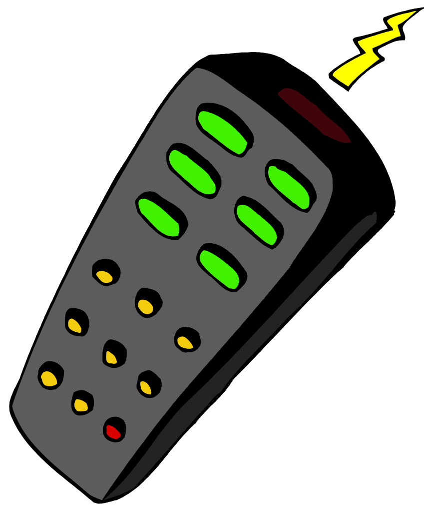
Combinational Example: Staircase Light Switch



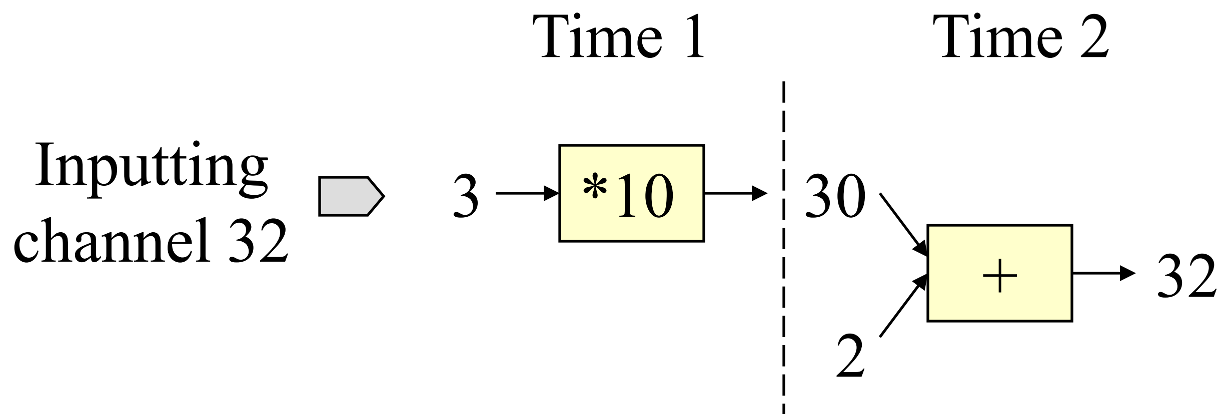
Whether or not the light is on is only dependent on the current position of the switches

© Mark Redekopp, All rights reserved

Sequential Example: Remote Control



The channel is a function of the first button pressed and the second (we must remember the 3 and then use it with the 2)



© Mark Redekopp, All rights reserved

Sequential Logic

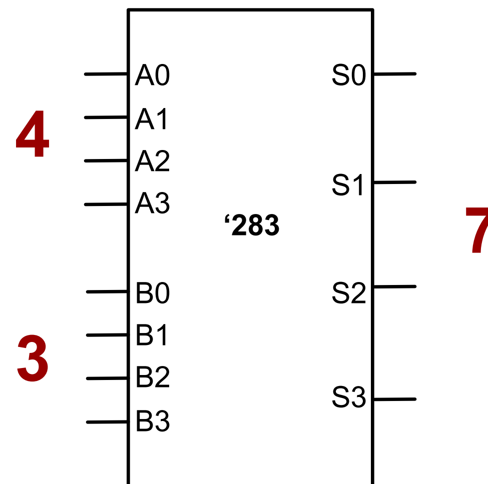
- All logic is categorized into 2 groups
  - Combinational logic:
    - Outputs = f(current inputs)
  - Sequential Logic
    - Outputs = f(current inputs, previous inputs)
    - With sequential logic there is the idea of “memory”

© Mark Redekopp, All rights reserved

Sequential Logic

- With *combinational logic* the outputs only depend on what the inputs are right now

4



7

3

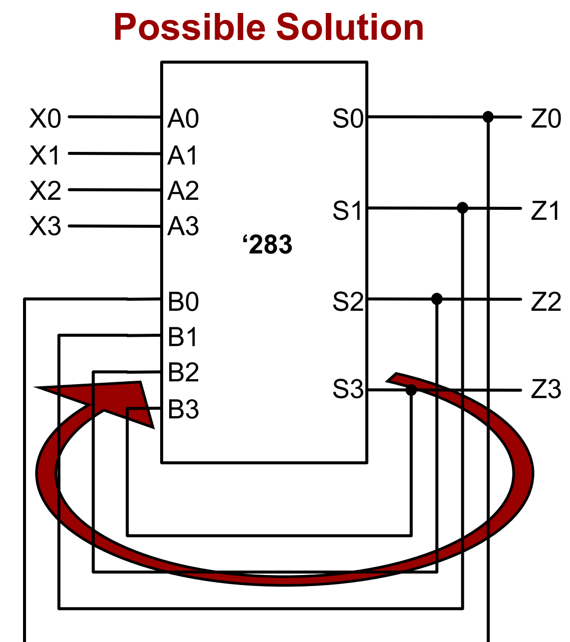
It doesn't matter what the inputs were previously

© Mark Redekopp, All rights reserved

Sequential Logic

- Suppose we have a sequence of input numbers on X[3:0] that are entered over time that we want to sum up
- Possible solution: Route the outputs back to the inputs so we can add the current sum to the input X
- Problem 1: No way to initialize sum
- Problem 2: Outputs can race around to inputs and be added more than once per input number

9, 3, 2

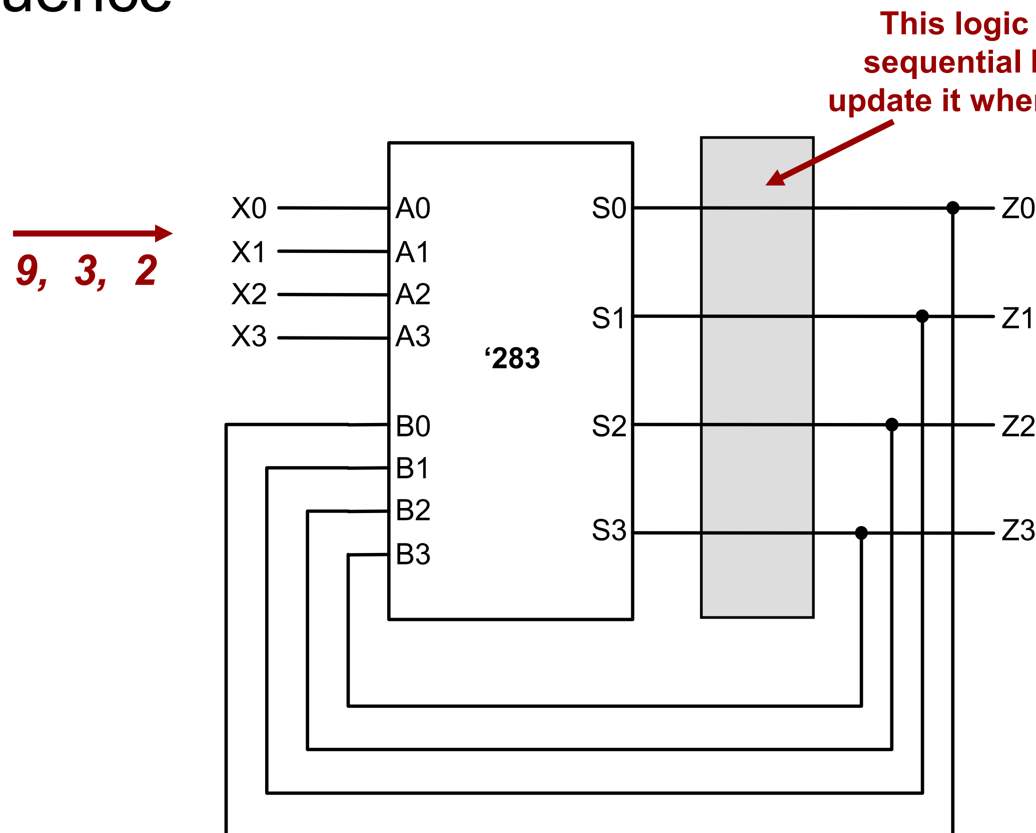


Outputs can feedback to inputs and update them sum more than once per input

© Mark Redekopp, All rights reserved

Sequential Logic

- Add logic at outputs to just capture and remember the new sum until we're ready to input the next number in the sequence



The data can still loop around and add up again (2+2=4) but if we just hold our output = 2 then the feedback loop will be broken

We remember initial sum of 2 until input 3 arrives at which point we'd capture & remember the sum 5.

© Mark Redekopp, All rights reserved

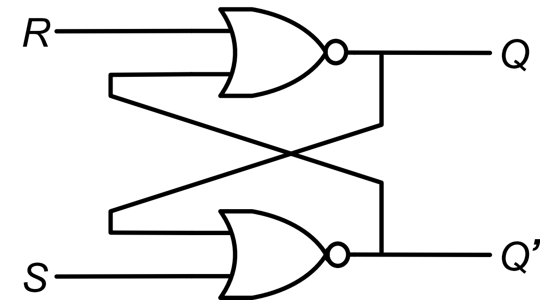
Sequential Logic

- Our first goal will be to design a circuit that can remember one bit of information

© Mark Redekopp, All rights reserved

RS (SR) Bistable

- Cross-Connected NOR gates (outputs feed back to inputs)
- When Set = 1, Q should be forced to 1
- When Reset = 1, Q should be forced to 0
- When neither are 1, Q should remain at its present value

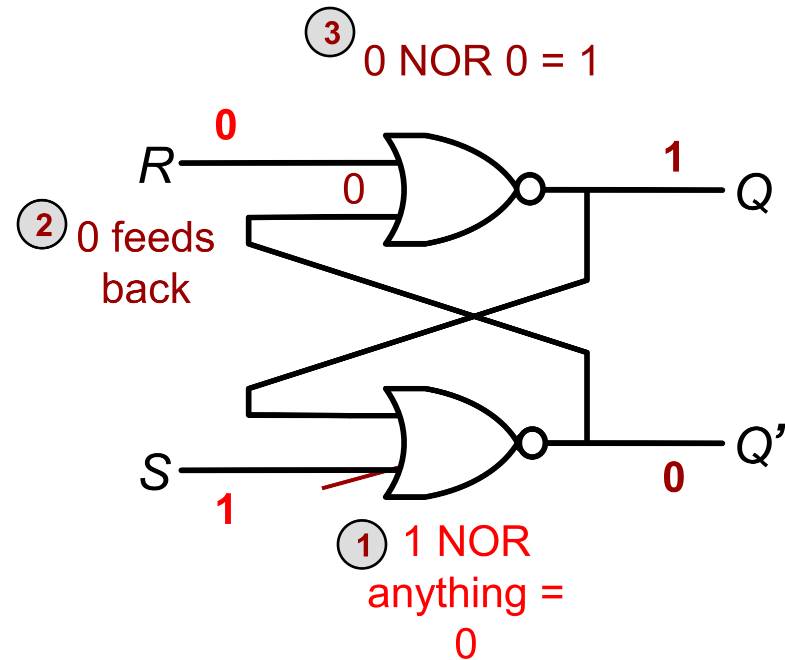


S	R	Q	Q'
0	0	Q_0	Q_0'
1	0	1	0
0	1	0	1
1	1	0 (illegal)	0 (illegal)

© Mark Redekopp, All rights reserved

RS (SR) Bistable

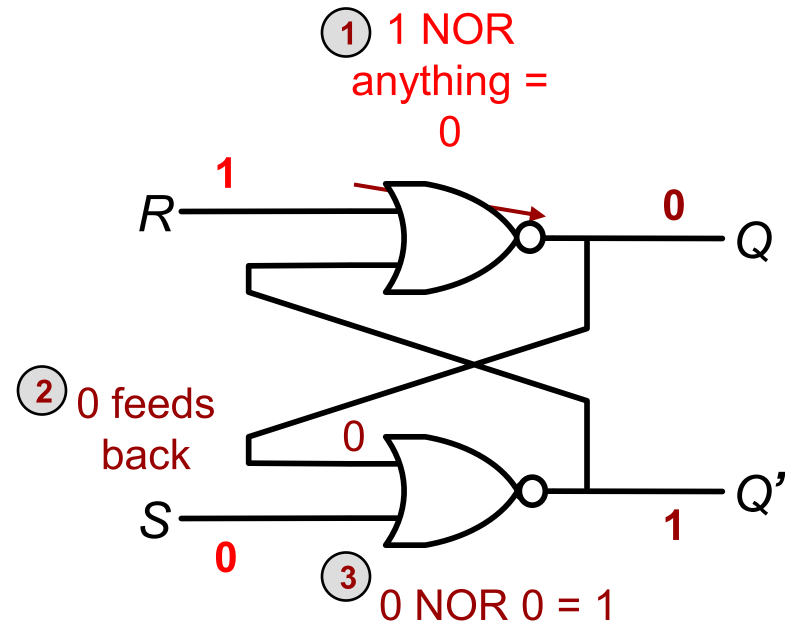
S	R	Q	Q'
0	0	Q_0	Q_0'
1	0	1	0
0	1	0	1
1	1	0 (illegal)	0 (illegal)



© Mark Redekopp, All rights reserved

RS (SR) Bistable

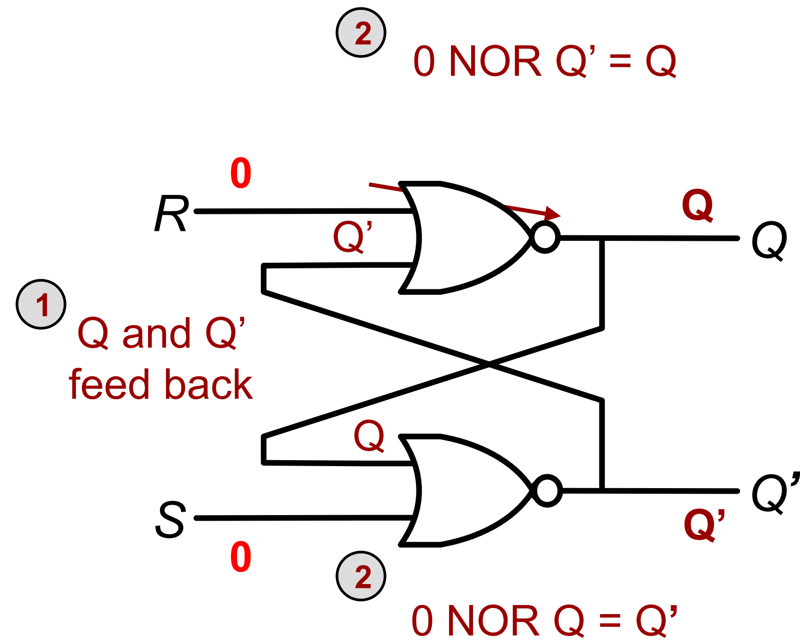
S	R	Q	Q'
0	0	Q_0	Q_0'
1	0	1	0
0	1	0	1
1	1	0 (illegal)	0 (illegal)



© Mark Redekopp, All rights reserved

RS (SR) Bistable

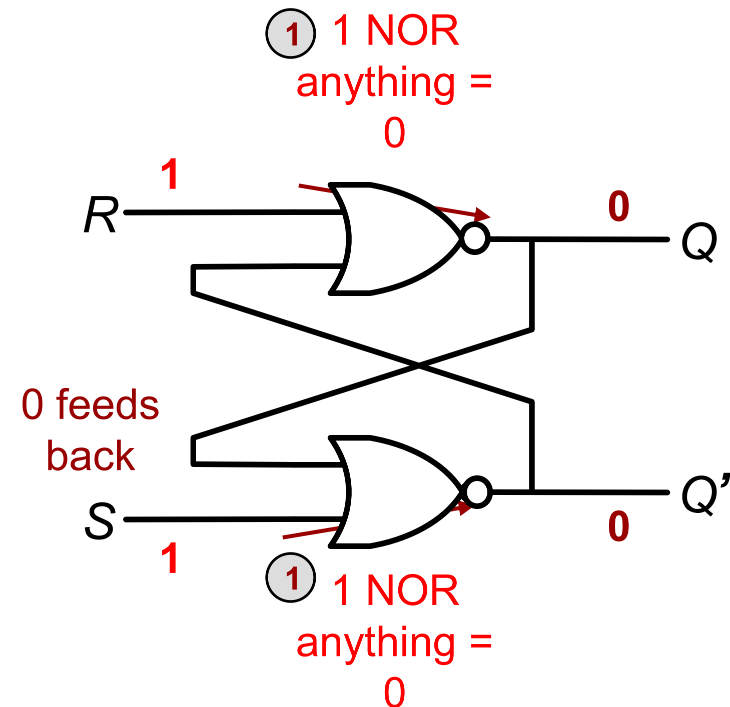
S	R	Q	Q'
0	0	Q_0	Q_0'
1	0	1	0
0	1	0	1
1	1	0 (illegal)	0 (illegal)



© Mark Redekopp, All rights reserved

RS (SR) Bistable

S	R	Q	Q'
0	0	Q_0	Q_0'
1	0	1	0
0	1	0	1
1	1	0 (illegal)	0 (illegal)



- 1,1 combination violates the Q, Q' relationship
- It cannot be "remembered"…meaning as soon as R or S goes to 0 then it will set and reset; if R and S goto 0 at the same instant, then we will have unpredictable behavior

© Mark Redekopp, All rights reserved

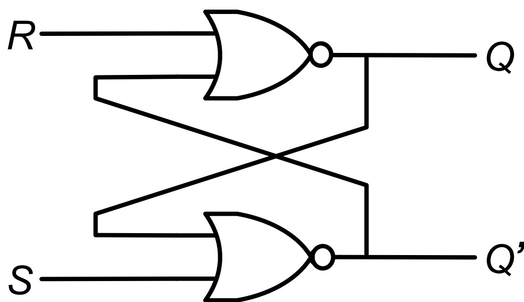
Bistables

- There are 3 criteria for a circuit to be considered a bistable:
  1. Must be able to be Set (output set to 1)
  2. Must be able to be Reset (output reset to 0)
  3. Must be able to remember the current state of the output (output remembered and not changed)

© Mark Redekopp, All rights reserved

Bistables

- Cross-Connected NOR gates form a valid bistable (in this case an SR bistable)
- When Set = 1, output forced to 1
- When Reset = 1, output forced to 0
- When both are 0, Q at its present value

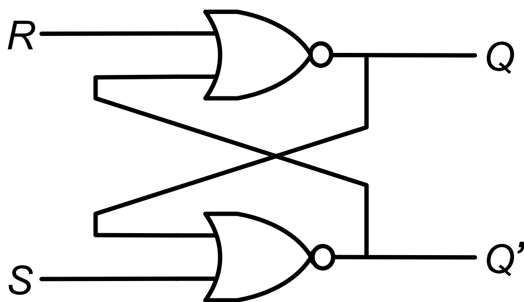


S	R	Q	Q'
0	0	Q_0	Q_0'
1	0	1	0
0	1	0	1
1	1	0 (illegal)	0 (illegal)

© Mark Redekopp, All rights reserved

Bistables

- Cross-Connected NOR gates form a valid bistable (in this case an SR bistable)
- When Set = 1, output forced to 1
- When Reset = 1, output forced to 0
- When both are 0, Q at its present value

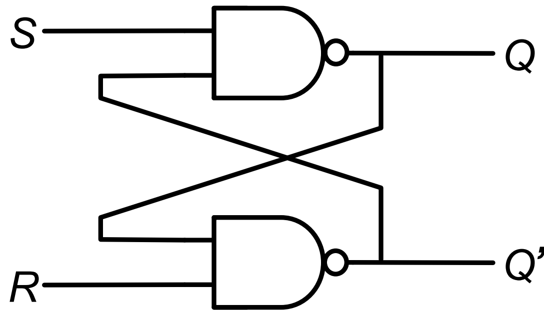


S	R	Q	Q'
0	0	Q_0	Q_0'
1	0	1	0
0	1	0	1
1	1	0 (illegal)	0 (illegal)

© Mark Redekopp, All rights reserved

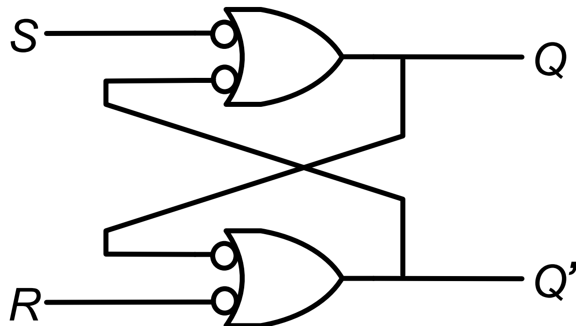
Bistable

- A bistable with active-lo inputs can be built with cross-connected NAND gates



To "view" it as active-lo we can redraw our NAND gates

=



S	R	Q	Q'
1	1	Q_0	Q_0'
0	1	1	0
1	0	0	1
0	0	1 (illegal)	1 (illegal)

Notice the active-lo inputs
(A 0 on S, sets Q=1)

© Mark Redekopp, All rights reserved

Criteria for a Bistable

1. Able to set (preset) => Force Q=1
2. Able to reset (clear) => Force Q=0
3. Able to remember (hold) => $Q = Q_0$

© Mark Redekopp, All rights reserved

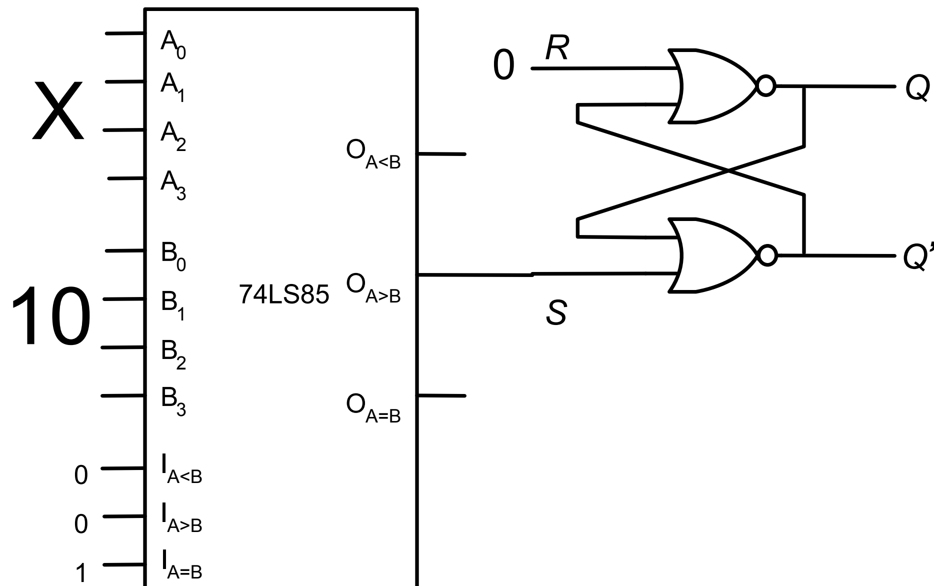
Checking for Bistable Validity

- Need to check the 3 criteria
  - Can it set, reset, remember
- Method for checking:
  - Find the passive input values of bistable inputs (the values that allow Q to cycle around the cross-connected loop)
    - Those passive input values identify the inactive values of the inputs…thus we can know whether the inputs are active hi or low
  - Activate each input separately to find if it is the Set or Reset input
    - Just see what happens to Q when you activate each input separately

© Mark Redekopp, All rights reserved

Problem w/ Bistables

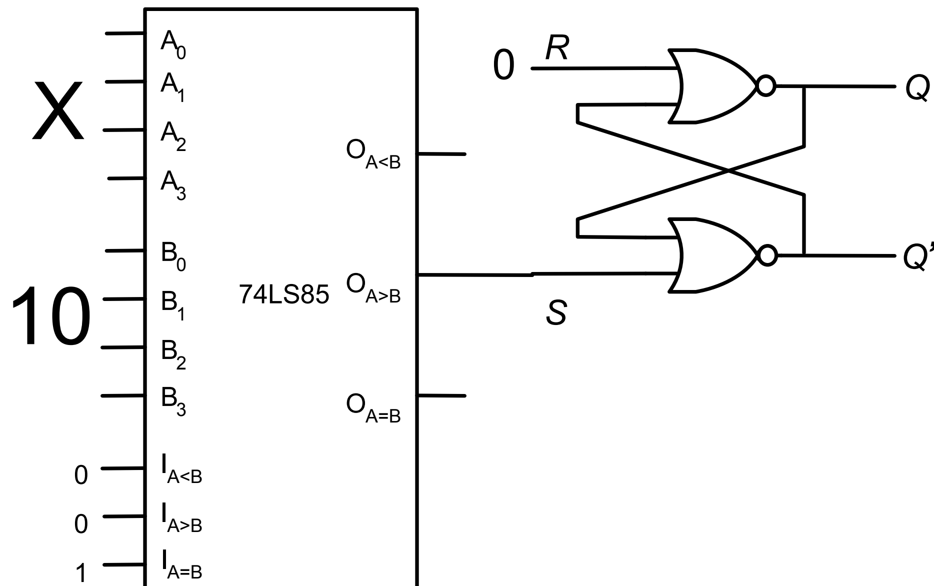
- Bistables will remember input values whether we want them to or not
- Imagine we connect the Set input to the output of a comparator to check if any number in a sequence is > 10 and then remember that



© Mark Redekopp, All rights reserved

Problem w/ Bistables

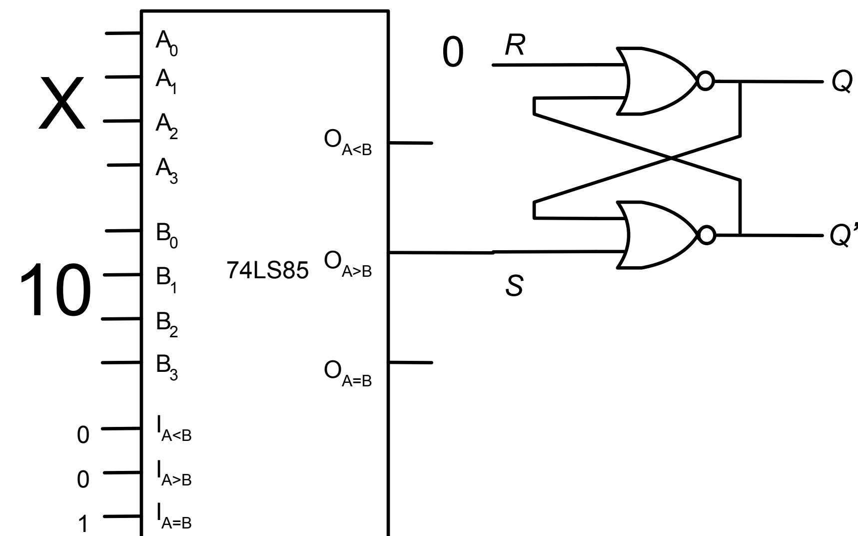
- When inputs change in a combinational circuit, the outputs may transition back and forth between 1 and 0
- This is called a "glitch" and is caused due to the propagation delay of the combinational logic

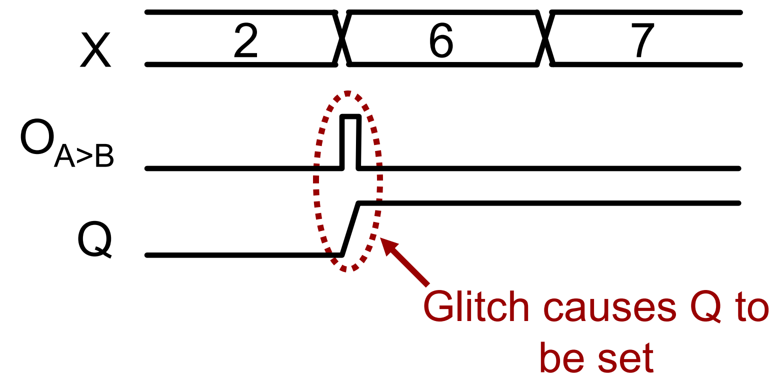


© Mark Redekopp, All rights reserved

Problem w/ Bistables

- Suppose we get a sequence: 2,6,7
- At the end Q should still = 0 since no numbers > 10
- However, if when the inputs change a small glitch occurs on A>B, the bistable will remember that and set Q = 1

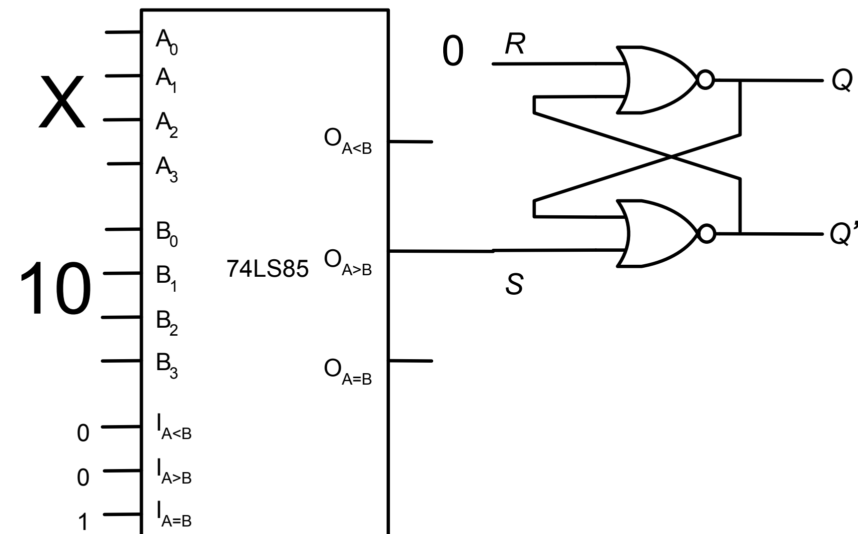


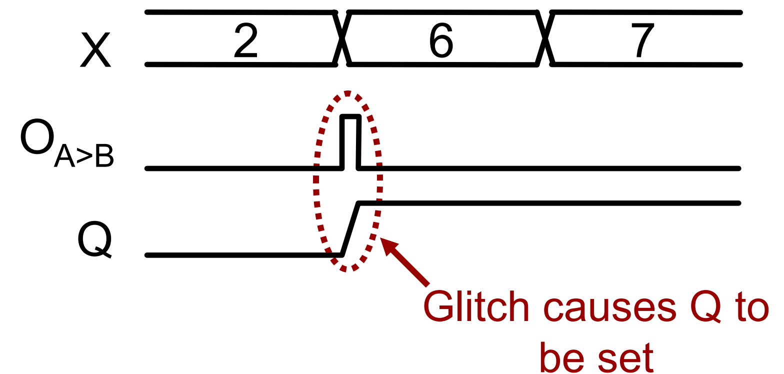


© Mark Redekopp, All rights reserved

Problem w/ Bistables

- Output should have been 0 at end of sequence
- Problem: Glitch was remembered
- Need some way to ignore inputs until they are stable and valid

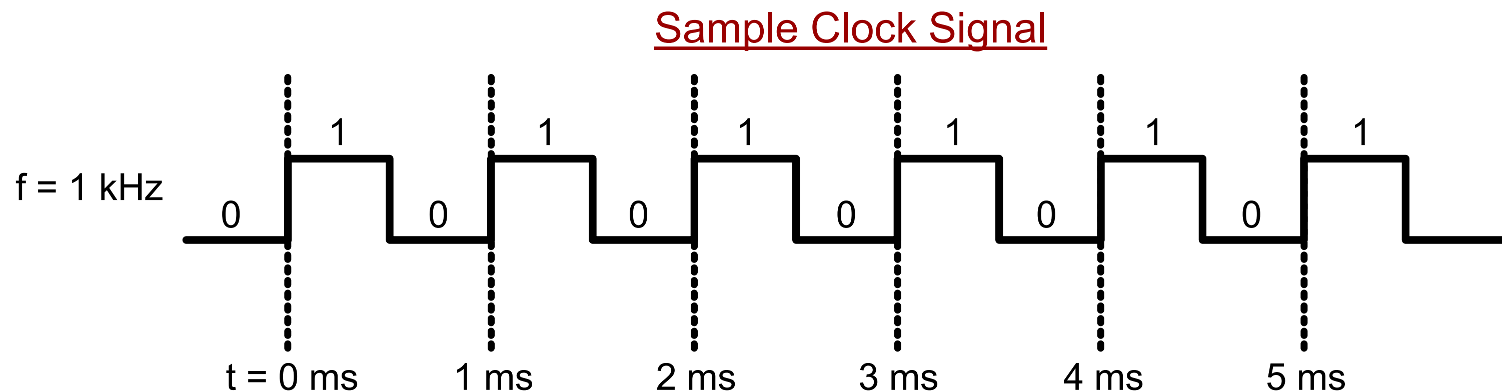




© Mark Redekopp, All rights reserved

Clock Signals

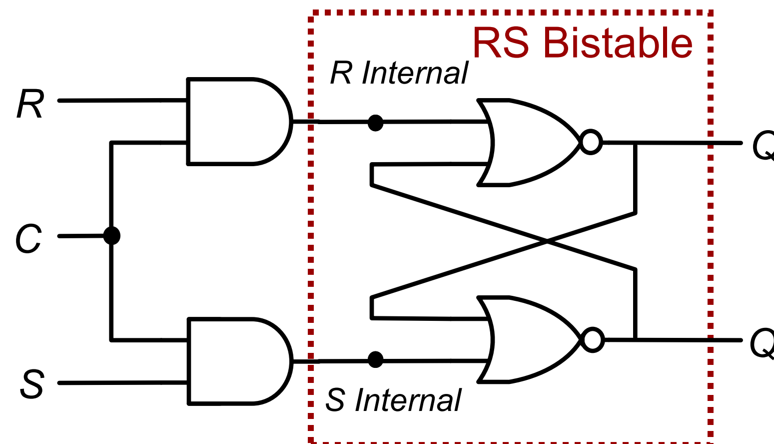
- A clock signal is an alternating sequence of 1's and 0's
- It can be used to help ignore the inputs of a bistable when there might be glitches or other invalid values
- Idea:
  - When clock is 0, ignore inputs
  - When clock is 1, respond to inputs



© Mark Redekopp, All rights reserved

Latches

- Latches are bistables that include a new ***clock input***
- The clock input will tell the latch when to ignore the inputs (when C=0) and when to respond to them (when C=1)

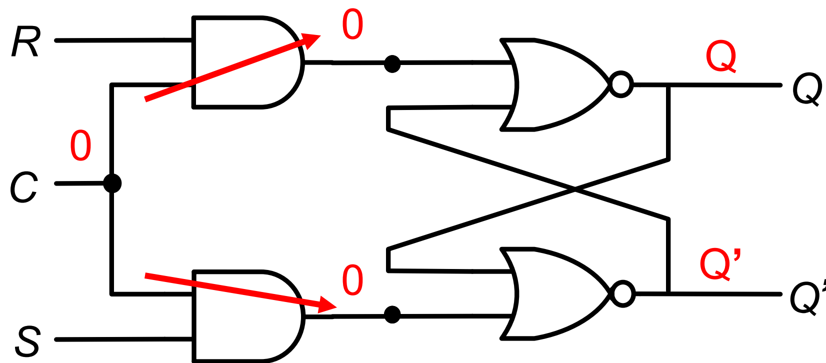


RS Latch

© Mark Redekopp, All rights reserved

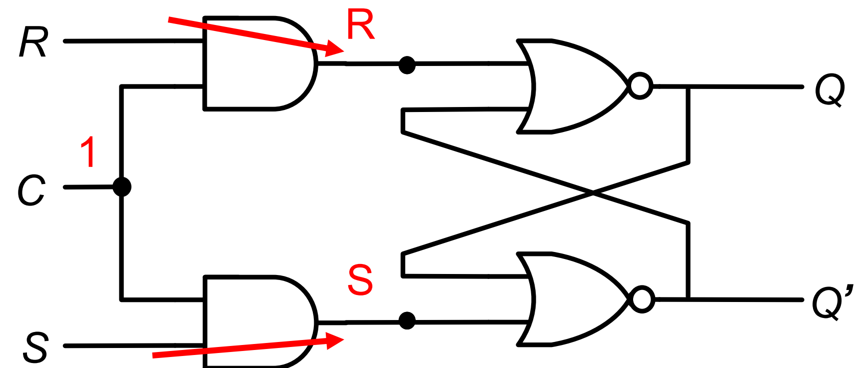
Latches

RS Latch
(C=0)



C=0 causes S=R=0 and thus Q and Q' remain unchanged

RS Latch
(C=1)



C=1 allows S,R to pass and thus Q and Q' are set, reset or remain unchanged based on those inputs

© Mark Redekopp, All rights reserved

Latches

- Rule
  - When clock = 0, inputs don't matter, outputs remain the same
  - When clock = 1, inputs pass to the inner bistable and the outputs change based on those inputs

© Mark Redekopp, All rights reserved