Sequential Circuits

Types of Circuits

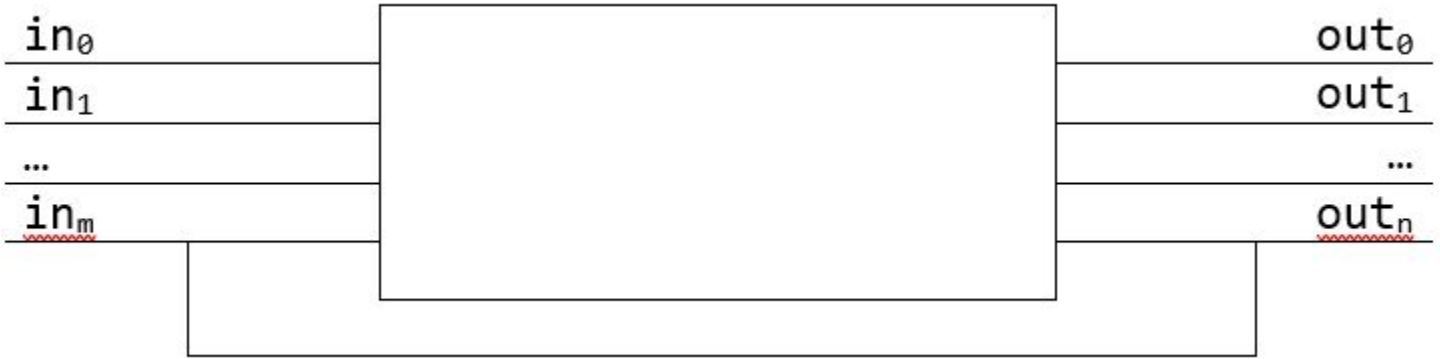
- There are two types of circuits
  - Combinatorial Circuits (Previously Covered)
    - outputs are determined exclusively by its current inputs
  - Sequential Circuits
    - outputs may be determined with previous outputs

Sequential Circuits

- A circuit is sequential if its outputs are determined by its current and previous inputs.
- In other words, a sequential circuit has a state.
- As we will see, this is accomplished by introducing feedback.

Sequential Circuits

- Diagram: block diagram of sequential circuit with m inputs and n outputs
  - **Note:** the combinational component with feedback loop through memory elements



Sequential Circuits

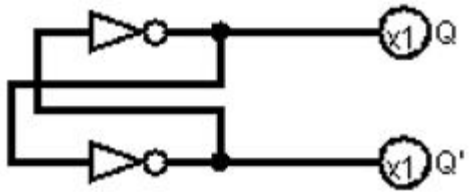
- An exhaustive study of sequential logic analysis and design is beyond the scope of this course
- Luckily, in order to implement a basic computer, only a small number of fundamental sequential components are needed, including:
  - registers
  - shift registers
  - counters
- These are built out of flip-flops, which are built out of latches

Bistability

- Feedback is when output is connected back to input.
- Not all feedback is useful, or stable (e.g. an inverter connected to itself).
- What is the result of doing this?
- Feedback the output from an inverter to itself will result in an oscillator

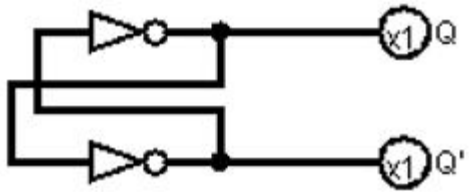
Simple Stable Sequential Circuit

- The simplest stable sequential circuit is:
- In fact, this circuit is bistable, meaning it has two stable states:
    1. Q=1 and Q’=0
    2. Q=0 and Q’=1
- Note this is positive (instead of negative) feedback



Simple Stable Sequential Circuit

- If you build this circuit and then supply power, it will settle into one of them.
- ... But which one?
- It is indeterminable, but one side will start before the other and then drive the circuit to that state



Transfer Functions

- Diagram: transfer functions for two inverters in bistable feedback loop
  - note two stable states and one metastable point

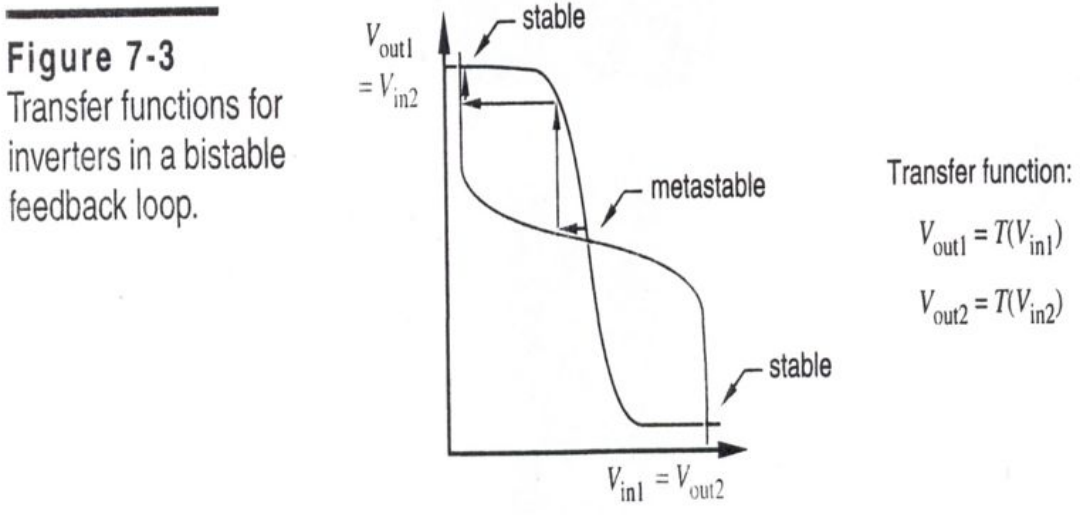


Figure 7-3
Transfer functions for inverters in a bistable feedback loop.

From Digital Design Principles and Practices 4th Ed. John F. Wakerly, p524

Transfer Functions - Metastable Point

- A metastable point is an equilibrium point, which will be driven to a stable state by random noise
- Analogy: a ball balanced at the top of a hill
- Drop the ball on the hill and it will probably roll one way or the other.
- It is possible that it will balance at the top. This is NOT oscillating, but rather balancing between 0 and 1
- If the circuit starts anywhere except the metastable point, feedback will drive it into the nearest stable state

Transfer Functions - Metastable Point

- If the circuit above starts in the metastable state, and then noise pushes it slightly to one side, feedback will drive it to stabilize fully on that side
- Once in a stable state, noise (unless extreme) is insufficient to destabilize it

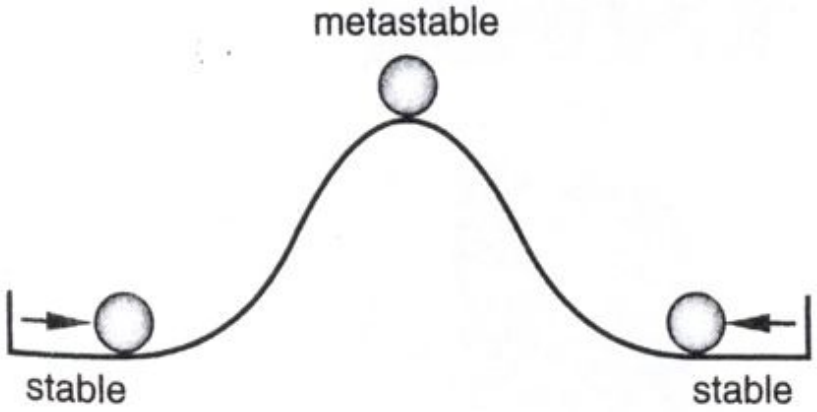


Figure 7-4
Ball and hill analogy for metastable behavior.

Transfer Functions - Metastable Point

- **Note:**
  - all sequential circuits can enter a metastable state under certain circumstances (discussed below)
  - once in this state, there is no theoretical limit to its duration (but the likelihood decreases exponentially with time)
  - when it stabilizes, the result will be due to random noise
- Note: Logisim won't simulate the above circuit because its algorithm requires that there be inputs

Transfer Functions - Metastable Point

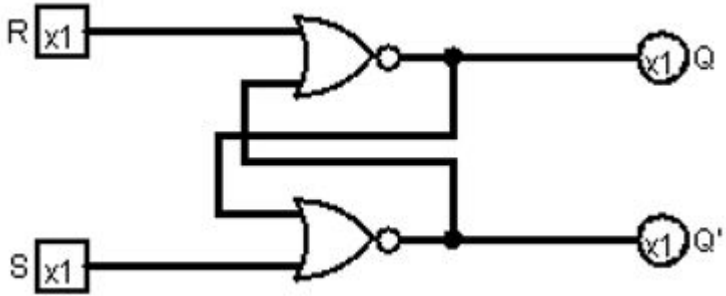
- Entering a Metastable state
- As we have just discussed it is possible to enter on startup – this is the reason that on startup a memory check is done. This drives memory cells into a stable state.
- Asynchronous input in a synchronous system. If input gets cut off at a time change then if the signal is:
  - too weak stay in the same state
  - too strong move to opposite state
  - just right become metastable

SR Latch

- A latch is a sequential circuit that watches its inputs continuously and can change its outputs at any time.
- It would be very useful if we could cause the bistable circuit above to flip states on command. Why?
- We could store a bit!
- An SR Latch is a **S**et/**R**eset Latch

SR Latch

- The previous circuit can be generalized by replacing the inverters with NOR gates:
- This is called an SR latch. Notice where S and R are located.



SR Latch

- **Diagram:** SR Latch Block Diagram
- It is possible that there will be a circle on the Q’ line outside the block

SR Latch

- When S=R=0 the circuits are equivalent, and the current state is held stable
- A signal on R (reset) forces Q=0 and Q’=1
  - … and then this signal can be removed!
- A signal on S (set) forces Q=1 and Q’=0
  - … and then this signal can be removed!
- What happens if Q=0 and we assert R? How about if Q=1 and we assert S?
  - In both cases there is NO change. Q and Q’ will remain unchanged

SR Latch

- Simultaneous signals on S and R will force both outputs to 0. We usually try to avoid this.
- What happens if S and R are both asserted, and then de-asserted? What will be the resulting Q?
- Asserting both will result in both Q and Q’ going to 0.
    - Q will oscillate and then be undefined

Propagation Delay

- All real circuits entail propagation delay:
  - it takes time for a signal to travel between an output and the next input
  - it takes time for a gate's output to change in response to an input change (gate delay)
- This means there is a minimum pulse requirement for S and R
  - If too short, the circuit can become metastable

Propagation Delay

- The analogy of rolling a ball over a hill with:
  1. Enough force
     - Changes state
  2. Far too little force
     - Remains in the original state
  3. Just too little force
     - Makes it to the top of the hill and teeters. It will fall into an unknown state

Sequential Circuits

- Truth tables aren't used to describe sequential circuits. Why?
  - Because truth tables deal with fixed inputs – sequential circuits need to deal with feedback values
- There are two common methods of analyzing sequential circuits:
  - A characteristic table provides information about what the next state of the circuit will be on specific input
  - An excitation table shows the minimum inputs that are necessary to generate a particular next state
    https://en.wikipedia.org/wiki/Excitation_table

SR Latch

- Create a table containing all of the possible combinations of S, R and Q(curr), assume that Q'(curr) is the opposite

S	R	Q(curr)	Q(next)	Q'(next)	Comments
0	0	0			
0	0	1			
1	0	0			
1	0	1			
0	1	0			
0	1	1			
1	1	0			
1	1	1			

SR Latch

- Create a table containing all of the possible combinations of S, R and Q(curr), assume that Q'(curr) is the opposite

S	R	Q(curr)	Q(next)	Q'(next)	Comments
0	0	0	0	1	**Q(curr) = Q(next) Hold**
0	0	1	0	1	
1	0	0	1	0	**Q(next)=1 Set**
1	0	1	1	0	
0	1	0	0	1	**Q(next)=0 Reset**
0	1	1	0	1	
1	1	0	0	0	**Q=Q'=0 Bad!**
1	1	1	0	0	

SR Latch

- The “characteristic table” for the SR latch is:

S	R	Q(Next)
0	0	Q(curr)
0	1	0
1	0	1
1	1	0

Note: S=R=1 → Q(next)=Q’(next)=0

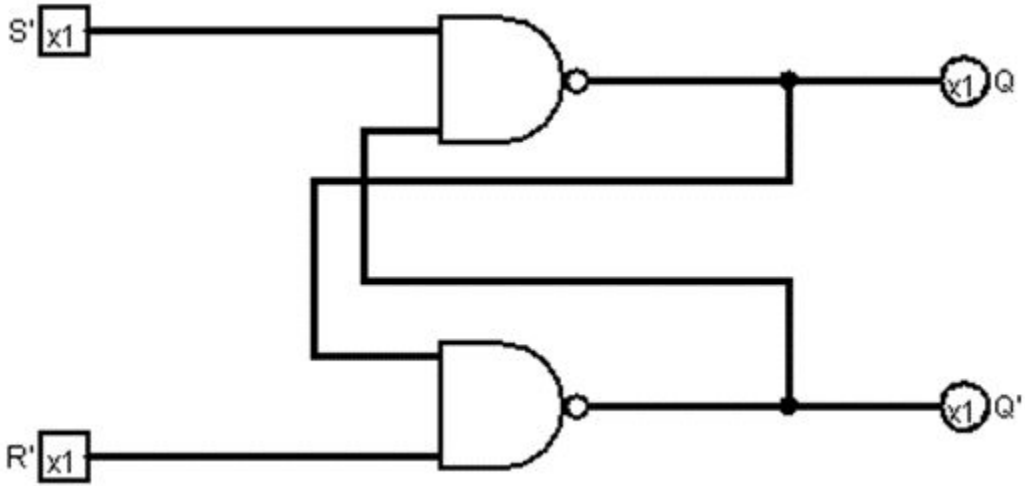
- The “excitation table”, which says which changes require which inputs, is:

Q(curr)	Q(next)	S	R
0	0	0	x
0	1	1	0
1	0	0	1
1	1	x	0

Note: x means it does not matter

SR Latch

- A version of the SR latch can also be implemented with NAND gates



SR Latch

- Create a table containing all of the possible combinations of S, R and Q(curr), assume that Q'(curr) is the opposite

S	R	Q(curr)	Q(next)	Q'(next)	Comments
0	0	0			
0	0	1			
1	0	0			
1	0	1			
0	1	0			
0	1	1			
1	1	0			
1	1	1			

SR Latch

- Since there are two possibilities for Q(curr) for every S-R pair determine the result for the pair and determine which setting is the stable situation
    - Typically the stable state is where both inputs are 0 or 1
- Then using the existing Q(curr) and Q’(curr) values and working with the S or R input determine which changes Q(next)

SR Latch

- Determine the values for Q(next) and Q'(next)

S	R	Q(curr)	Q(next)	Q'(next)	Comments
0	0	0	1	1	
0	0	1	1	1	
1	0	0	0	1	
1	0	1	0	1	
0	1	0	1	0	
0	1	1	1	0	
1	1	0	0	1	
1	1	1	1	0	

SR Latch

- Determine the values for Q(next) and Q'(next)

S	R	Q(curr)	Q(next)	Q'(next)	Comments
0	0	0	1	1	**Q=Q'=1 Bad!**
0	0	1	1	1	
1	0	0	0	1	**Q(next) = 0 Reset**
1	0	1	0	1	
0	1	0	1	0	**Q(next) = 1 Set**
0	1	1	1	0	
1	1	0	0	1	**Q(next) = Q(curr) Hold**
1	1	1	1	0	

SR Latch

- Characteristic Table – create by looking at the table on previous slide
  - Basically compress the two S-R lines into one

S	R	Q(next)
0	0	1
0	1	1
1	0	0
1	1	Q(curr)

Note: S=R=0 → Q(next)=Q'(next)=1

SR Latch

- Excitation Table – ignore the bad situation when creating this table

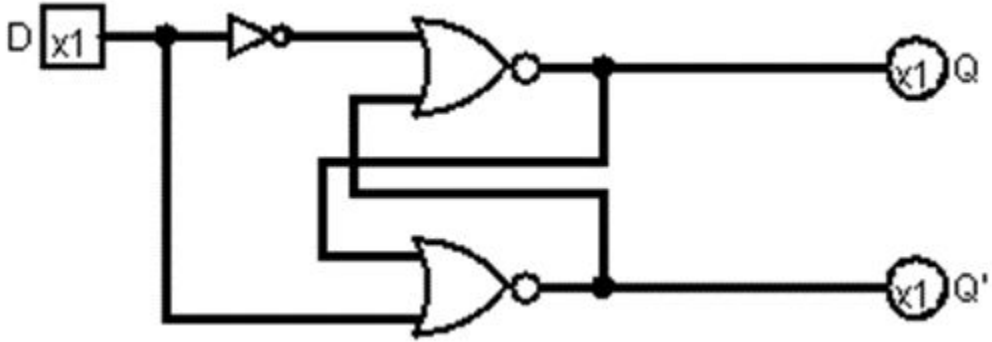
Q(curr)	Q(next)	S	R
0	0	1	x
0	1	0	1
1	0	1	0
1	1	x	1

Note: The NOR and NAND implementations have different (opposite) positive and negative logic

- The NOR sets a value of 1, whereas the NAND sets a value of 0

D Latch (With Enable and Asynchronous Reset)

- It is good to avoid the possibility of S = R = 1
  - Especially if we just want to store a bit
- The **D** Latch accomplished this
  - D stands for Data



D Latch

- **Exercise:** Write the characteristic and excitation tables for the D latch

D	Q(next)
0	0
1	1

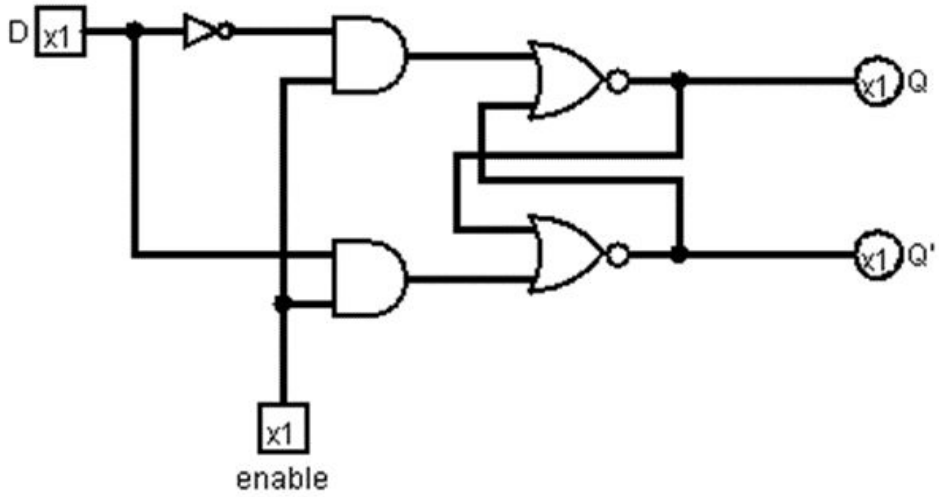
Q(curr)	Q(next)	D
0	0	0
0	1	1
1	0	0
1	1	1

D Latch

- Problem: even though it stores D, it always outputs Q = D.
  - Stated in a different fashion Q always follows D immediately
- What's the issue?
  - Q always follows D immediately!
- This is simply a replication of the SR latch, but without the issue of a Bad state
- The issue is controlling WHEN the latch changes
- Ideally the latch will be connected to a bus line and the bus line will be continuously changing

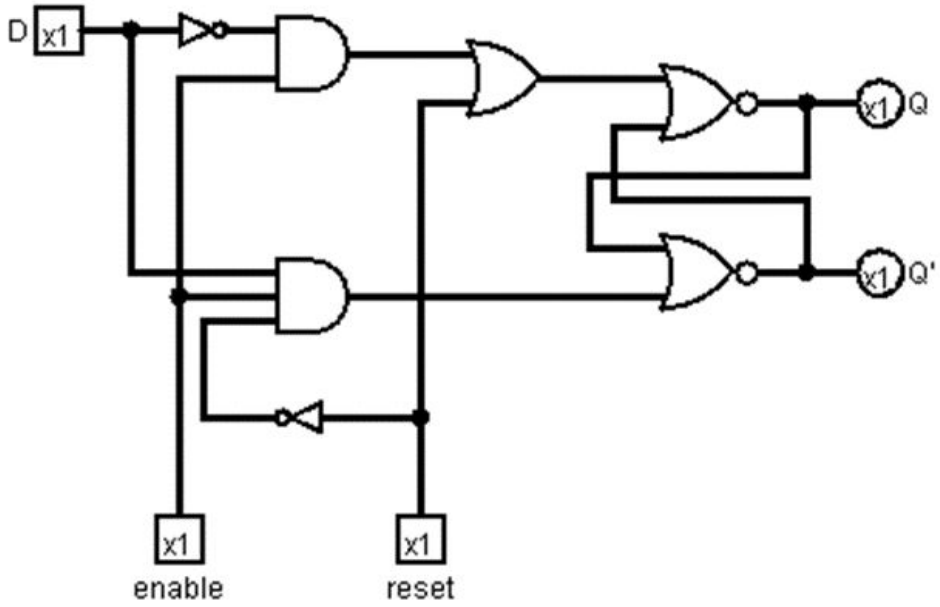
D Latch

- We can add an enable input so that D is ignored and Q is held until the circuit is told to load D
- The enable input can be connected to a clock signal so that Q can be updated regularly, but only during one half of the clock cycle
- **Synchronous:** in time with a common clock signal.



D Latch

- **Asynchronous:** independent of a *common* clock
- It is useful to add an asynchronous reset signal so that Q can be reset to 0 independently of enable:

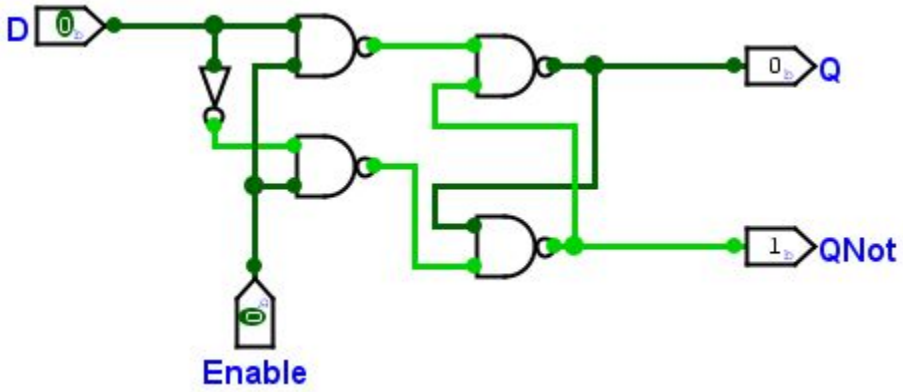


D Latch

- What are the Boolean functions for the S and R inputs to the core latch?
  - S = D • enable • reset'
  - R = D'• enable + reset
- Note: commercial latches sometimes provide the Q' output for convenience, but it can be ignored

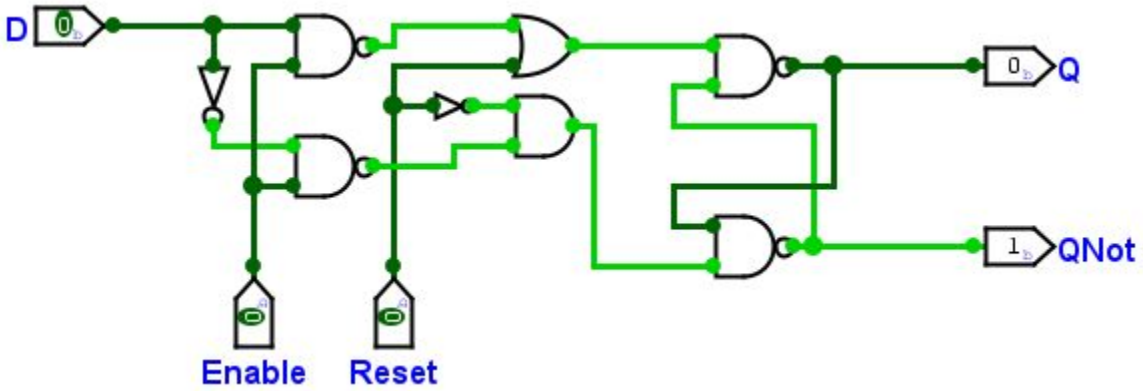
D Latch

- D Latch with Enable NAND gates



D Latch

- D Latch with Enable and Reset using NAND gates

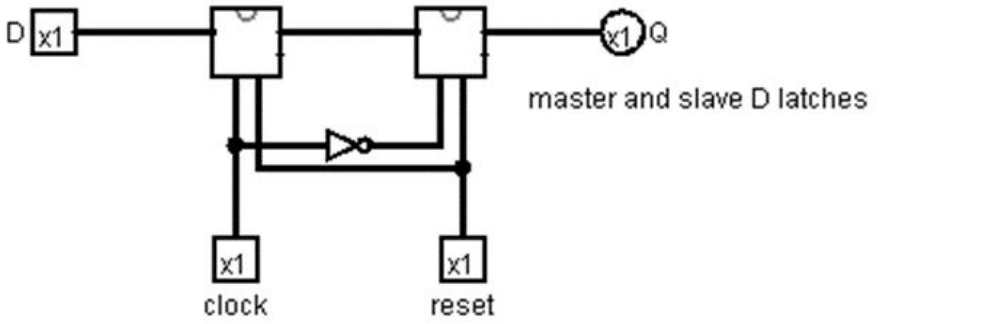


Edge Triggered D Flip-Flop

- The above are “transparent” latches: state changes follow input changes immediately (assuming enable = 1 and reset = 0).
- This is a problem. Why?
- Due to feedback, a change in a memory unit’s output could very quickly cause a change in its input, resulting in an unstable, run-away circuit state.
- When enabled the final Q value depends on D which can change at any time.
- Analogy: designing an exit door for a space station. Closed = no problem. Open = big problem!

Edge Triggered D Flip-Flop

- A bistable circuit that samples its input only at a certain time during a clock cycle
  - e.g. at each negative edge transition, is called a flip-flop
- One way to accomplish this is by chaining “master” and “slave” latches:



negative edge-triggered D flip-flop with asynchronous reset (note: Q' discarded)

Edge Triggered D Flip-Flop

- Aside: there is an optimized version which uses fewer gates, but we will not study it (it's harder to analyze)
- https://www.falstad.com/circuit/e-masterslaveff.html
- Again, the reset signal is asynchronous. Commercial flip-flops sometimes also provide an asynchronous set, or preset
- Consider generic sequential circuits which feedback through memory elements
    - It is the synchronous nature of the flip-flops which keeps them from "running away": the state evolves in time with the clock signal

Edge Triggered D Flip-Flop

- Aside: actually, there are asynchronous sequential circuits. We won't discuss them.
- For this circuit to work, we must ensure that D is held stable around the active edge of the clock signal! What can happen if D is changing too near the active edge?
    - The circuit can become metastable

Edge Triggered D Flip-Flop

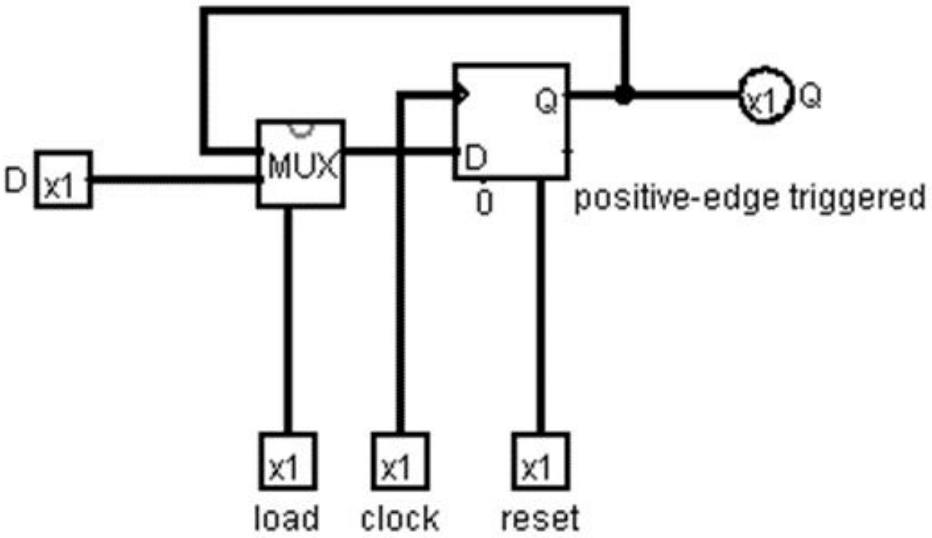
- Flip-flops have:
  - a setup time
  - a hold time
- The circuit which drives D must ensure D stabilizes early enough before the active edge, and that it remains stable late enough afterwards.
- So ...
- The clock period must be slow enough to allow D to stabilize (due to propagation delay) and to allow sufficient setup time.

Edge Triggered D Flip-Flop

- The propagation delay leading up to D must be great enough to allow sufficient hold time (having zero delay would be bad)!
- Note the difference in the block diagrams for a latch and a FF. A latch is just a normal chip, whereas a FF has a clock signal. Include each from Logisim

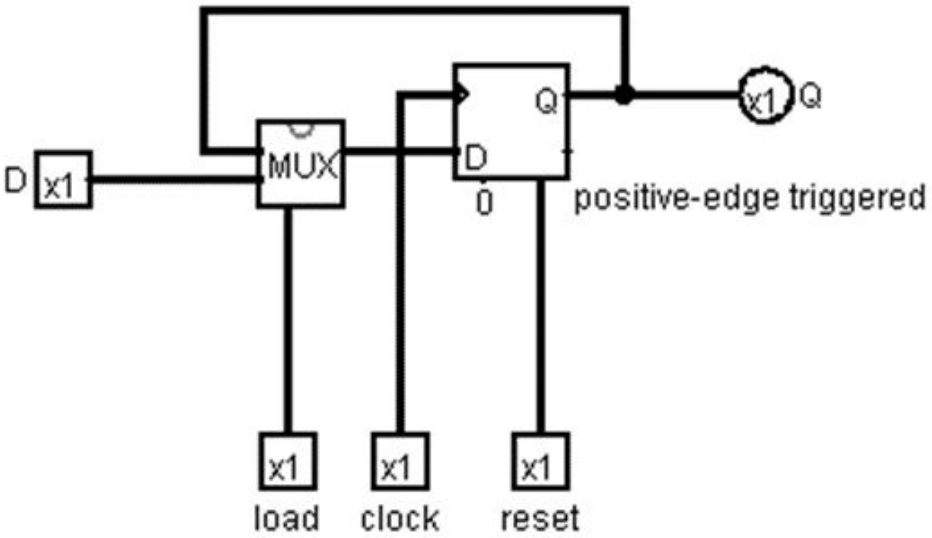
Registers

- It is easy to build a 1-bit register out of a D flip-flop:
- Notice that the flip-flop loads itself on every active edge!
- It loads either with its current Q or the supplied D



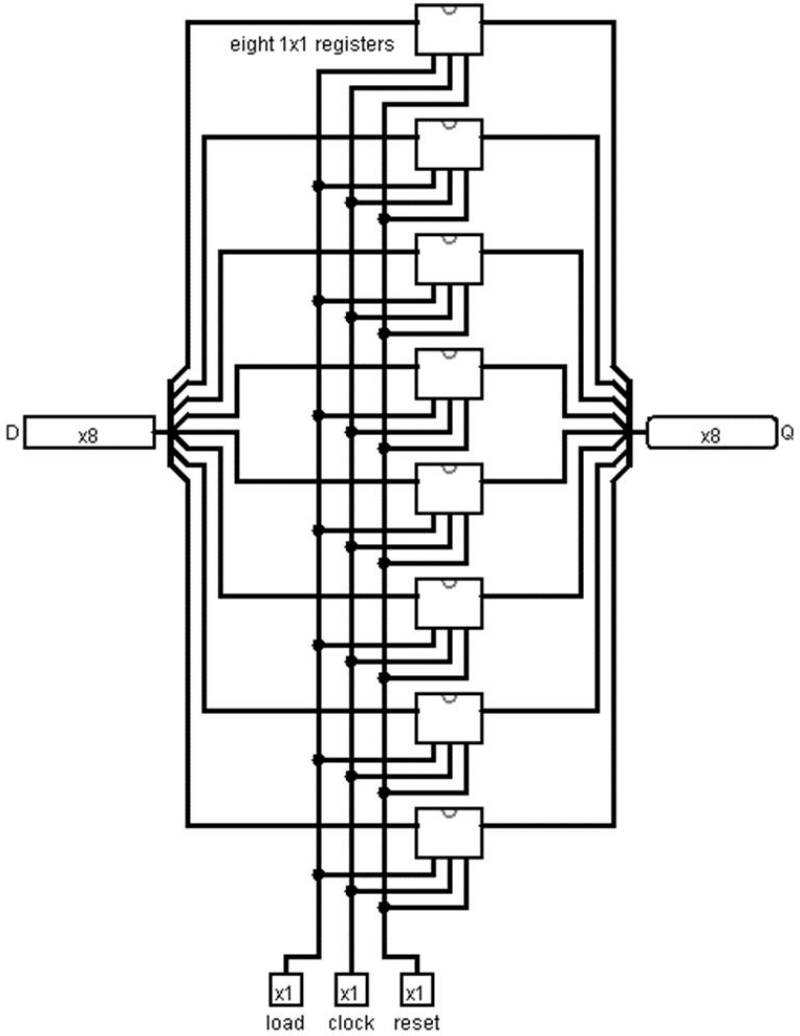
Registers

- Load selects the source to put through the 2x1 MUX
  - 0 → Reload with the same value, i.e. hold
  - 1 → load from an external source, i.e. the bus
- Important: we don't gate the clock! This leads to "clock skew" and destroys synchronization



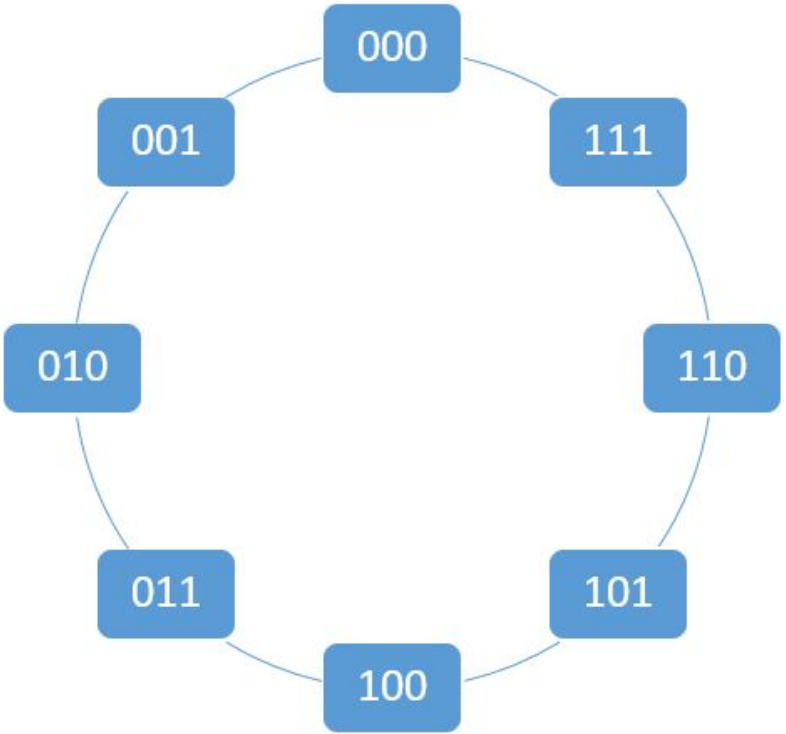
Register

- The reset is asynchronous. This flip-flop also has an asynchronous set (which is hardwired to 0).
- It is now easy to build an n-bit register with “parallel load”.
- E.g. an 8-bit version:



Counters

- A sequential circuit that goes through a prescribed sequence of states upon the application of an input pulse



Counter

- Counters are registers whose contents can be incremented and/or decremented, which is very useful (why?)
  - PC / SP / index register
  - Counting the # of occurrences of event
  - Generating a timing sequence
- Here, we design a design an 8-bit up/down counter with parallel load
- A counter is simply a modification to the Register that we previously created

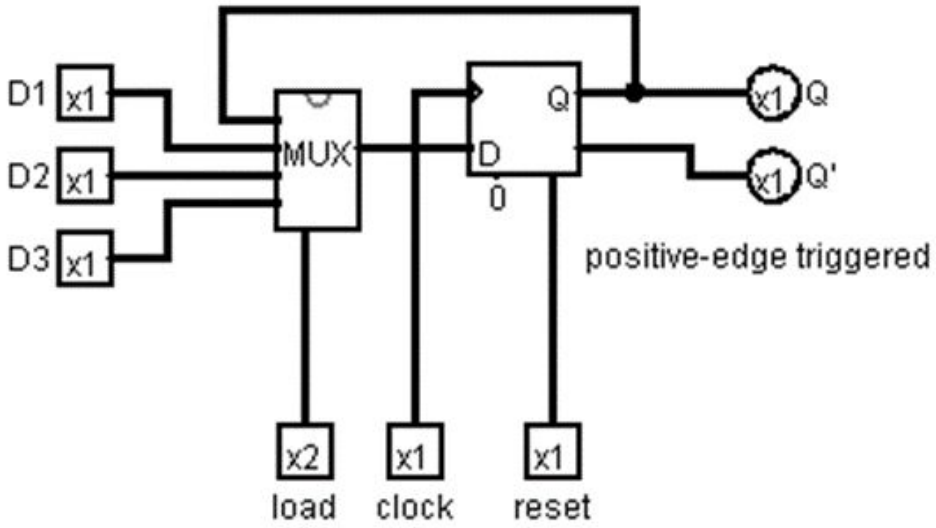
Counter

- The current register is limited to 2 inputs – hold and load.
- This is not fixed in stone - could have more.
- If instead we wanted 3 inputs HOW would this be done?
- Modify the MUX – make it bigger, i.e. a 4x1 MUX, but just won't use one input sources so it becomes a 3x1
    - i.e. a "3×1" register – a 1-bit register which can load from 1 of 3 sources

Counter

- Simplify the counter to only 1-bit and then the desired pattern is:

```
0
1
0
1
0
1
etc...
```



Counter

- How can we easily generate this pattern?
- There are many possibilities, but the simplest is:
  - If Q is the current state – then Q’ is the future state if toggling.
- We use this to build a 1-bit up/down counter with parallel load (note: develop first as up only) …

Counter

- If such a counter kept counting up, it would produce:

```
0
1
0
1
0
1
etc...
```

- Let’s use the 3-input register and set it up to either hold, load a given bit, or load Q' (the third input will be unused and wired to 0)

Counter

- The 2-bit load signal, which selects the load source, is:
  - 00 = hold
  - 01 = load given bit
  - 10 = load Q'
  - 11 = (unused – or synchronous reset)
- Can separate the two signals!
- LSB is load from external source
- MSB is toggle

Counter

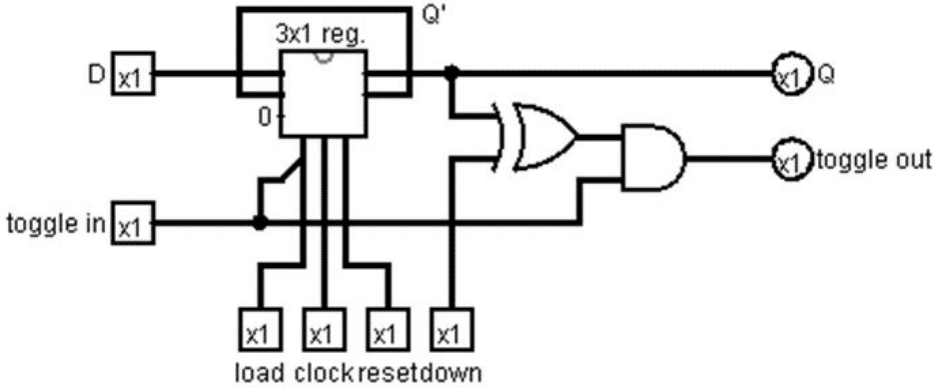
- Wait! We eventually want to chain these together to build larger counters. For example 2-bit:
  - 00
  - 01
  - 10
  - 11
  - 00
  - etc...

Counter

- Question: if the circuit represents a single bit within an n-bit counter, when does it need to be toggled?
- Answer:
  - the LSB is always toggled
  - each bit tells the next bit to toggle when it is being told to toggle and it is a 1

Counter

- Question: how can we make the 1-bit circuit count up and down?
- Answer: the same way, except each bit tells the next bit to toggle when it is being told to toggle and it is a 0
- The final 1-bit up/down counter with parallel load is:



Shift Registers

- Another very useful circuit is the shift register: the contents can be shifted left and/or right on command.
- It is easy to build an n-bit bidirectional shift register with parallel load out of n 3×1 registers
- … but, we must decide how to handle the incoming bit. The options are:
  - Always shift in 0 (or 1?) → shift left, logical shift right
  - Replicate the previous bit → arithmetic shift right
  - Replicate the outgoing bit → rotate
  - From an external source → Rx register in serial communications

Shift Register

- We must also decide what to do about the outgoing bit
  - Does it get dropped **or**
  - Is it provided as an extra output
    (e.g. Tx. register in serial communications)

Shift Register

- What are the actions for the load bits?

```
00  hold
01  load (from external source)
10  shift right
11  shift left
```

- This leads to the following questions:
  - The low order bit goes where?
  - The high order bit goes where?

Shift Registers

- For shifting right →
  - The low order bit goes where? Simply output
  - The high order bit is what? 0 for logical shift
- For shifting left
  - The low order bit is what? 0 for logical shift
  - The high order bit is what? Simply output
- Exercise: design a 3-bit bidirectional shift register with parallel load which performs logical shifting

Serial vs. Parallel

- There are two fundamental ways to transfer data between registers (or units, or computers):
  - Parallel n outputs of Tx reg. connected to n inputs of Rx reg.
  - Serial 1 bit at a time is transferred between Tx and Rx regs.
- Diagram: two 8-bit registers (Tx and Rx) connected for parallel transfer
- **Note:** Rx must support parallel load, so that the transfer completes in 1 clock cycle.

Serial vs. Parallel

- Diagram: two 8-bit shift registers (Tx and Rx) connected for serial transfer
- Note:
  - If they both shift right, Tx shifts out of its LSB and Rx shifts into its MSB (can also shift left)
  - It takes 8 clock cycles to complete the transfer
  - The operation is destructive for Tx (unless it rotates)