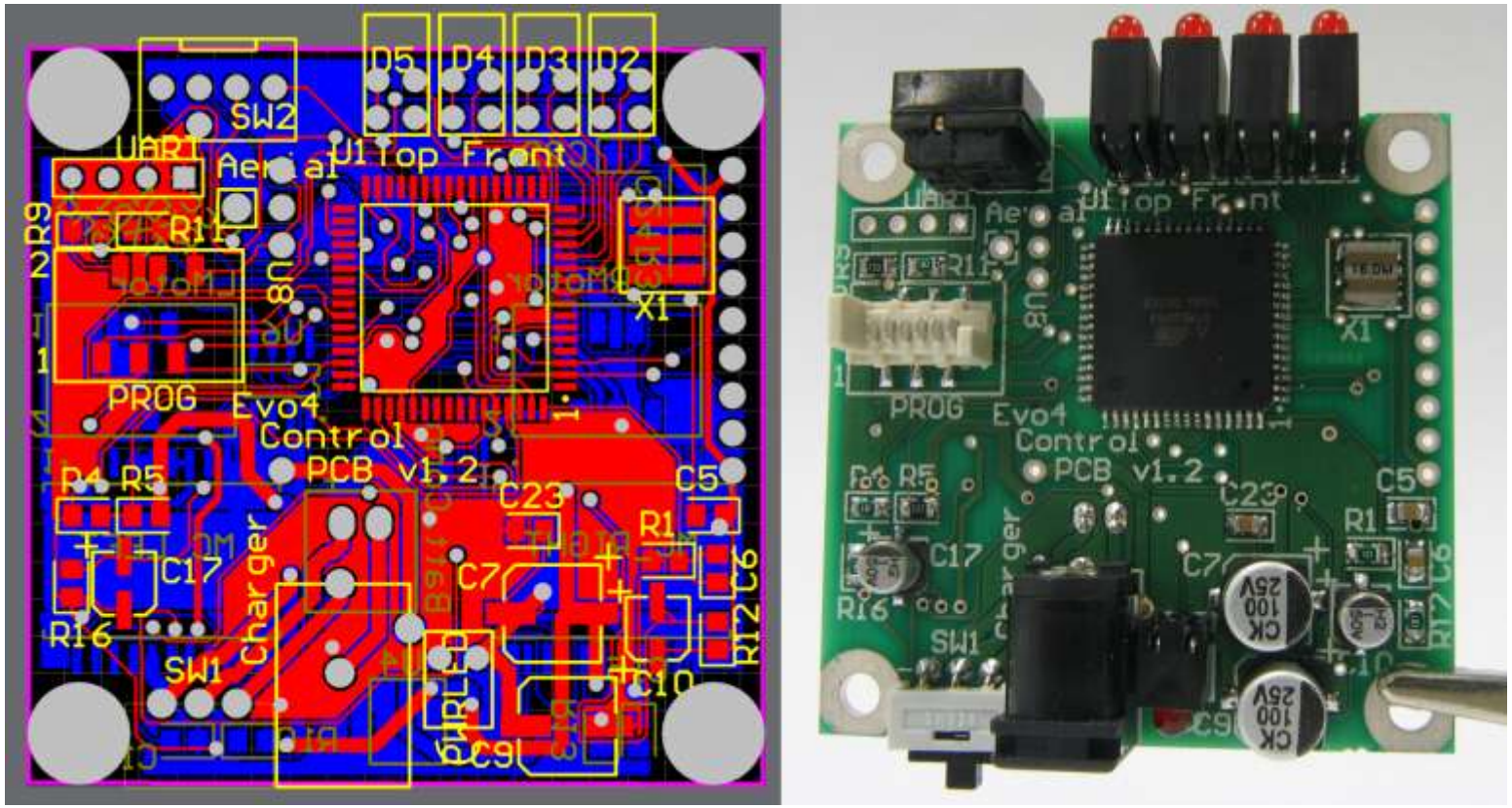
Adders, multiplexers, latches, flip-flops



Overview

- Combinatorial circuits
  - Build up a toolbox of useful circuits
    - Adder, multiplexer, demultiplexer
- Sequential circuits
  - Latches and flip-flops
  - Basis for computer memory

Building an adder

- Goal: $x + y = z$ for 4-bit integers
  - Input: $x = x_3x_2x_1x_0$
  - Input: $y = y_3y_2y_1y_0$
  - Input: c_0 (carry-in to first column)
  - Output: $z = z_3z_2z_1z_0$
  - Output: c_4 (carry-out last column)

				c_0
	x_3	x_2	x_1	x_0
+	y_3	y_2	y_1	y_0
c_4	z_3	z_2	z_1	z_0

Building an adder

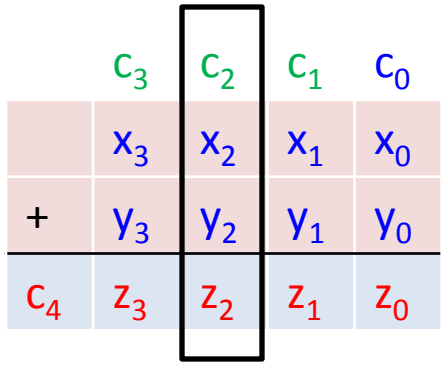
- Goal: x + y = z for 4-bit integers
- First attempt:
  - Build a truth table
  - 9 input variables to 5 output variables

				c_0
	x_3	x_2	x_1	x_0
+	y_3	y_2	y_1	y_0
c_4	z_3	z_2	z_1	z_0

c_0	x_3	x_2	x_1	x_0	y_3	y_2	y_1	y_0	c_4	z_3	z_2	z_1	z_0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	1	0
0	0	0	0	0	0	0	1	1	0	0	0	1	1
.	.	.	.	.	.	.	.	.	.	.	.	.	.
1	1	1	1	1	1	1	1	1	1	1	1	1	1

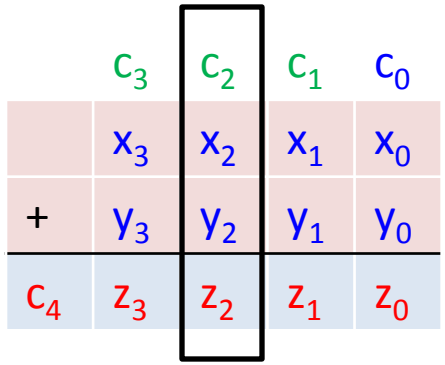
Divide and conquer!

- Let's figure out one bit-at-a-time
- Input:
  - 1 bit from x
  - 1 bit from y
  - 1 bit carry-in
- Output:
  - 1 bit for sum
  - 1 bit for carry-out



Divide and conquer!

- Let's figure out one bit-at-a-time
- Input:
  - 1 bit from x
  - 1 bit from y
  - 1 bit carry-in
- Output:
  - 1 bit for sum
  - 1 bit for carry-out



x_i	y_i	c_i	z_i
0	0	0	0
0	0	1	**1**
0	1	0	**1**
0	1	1	0
1	0	0	**1**
1	0	1	0
1	1	0	0
1	1	1	**1**

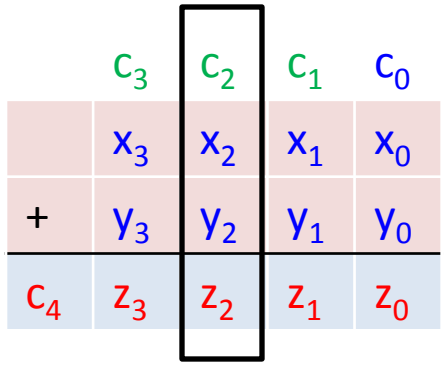
Sum bit

x_i	y_i	c_i	c_{i+1}
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	**1**
1	0	0	0
1	0	1	**1**
1	1	0	**1**
1	1	1	**1**

Carry-out bit

Divide and conquer!

- Let's figure out one bit-at-a-time
- Input:
  - 1 bit from x
  - 1 bit from y
  - 1 bit carry-in
- Output:
  - 1 bit for sum
  - 1 bit for carry-out



x_i	y_i	c_i	z_i	ODD
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	0
1	0	0	1	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Sum bit

x_i	y_i	c_i	c_{i+1}	MAJ
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	1
1	0	0	0	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

Carry-out bit

ODD and MAJ

x_i	y_i	c_i	z_i	ODD
0	0	0	0	0
0	0	1	**1**	**1**
0	1	0	**1**	**1**
0	1	1	0	0
1	0	0	**1**	**1**
1	0	1	0	0
1	1	0	0	0
1	1	1	**1**	**1**

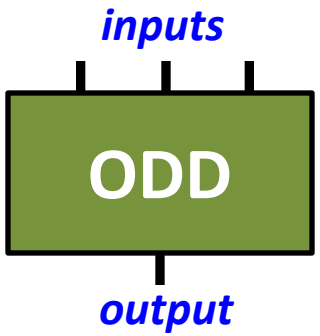
x_i	y_i	c_i	c_{i+1}	MAJ
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	**1**	**1**
1	0	0	0	0
1	0	1	**1**	**1**
1	1	0	**1**	**1**
1	1	1	**1**	**1**

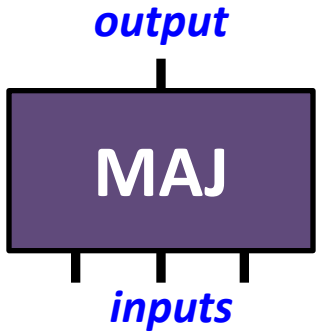
Sum bit

$$z_i = x_i'y_i'c_i + x_i'y_ic_i' + x_iy_i'c_i' + x_iy_ic_i$$
$$= x_i \text{ XOR } y_i \text{ XOR } c_i$$

Carry-out bit

$$c_{i+1} = yc_i + xc_i + xy$$

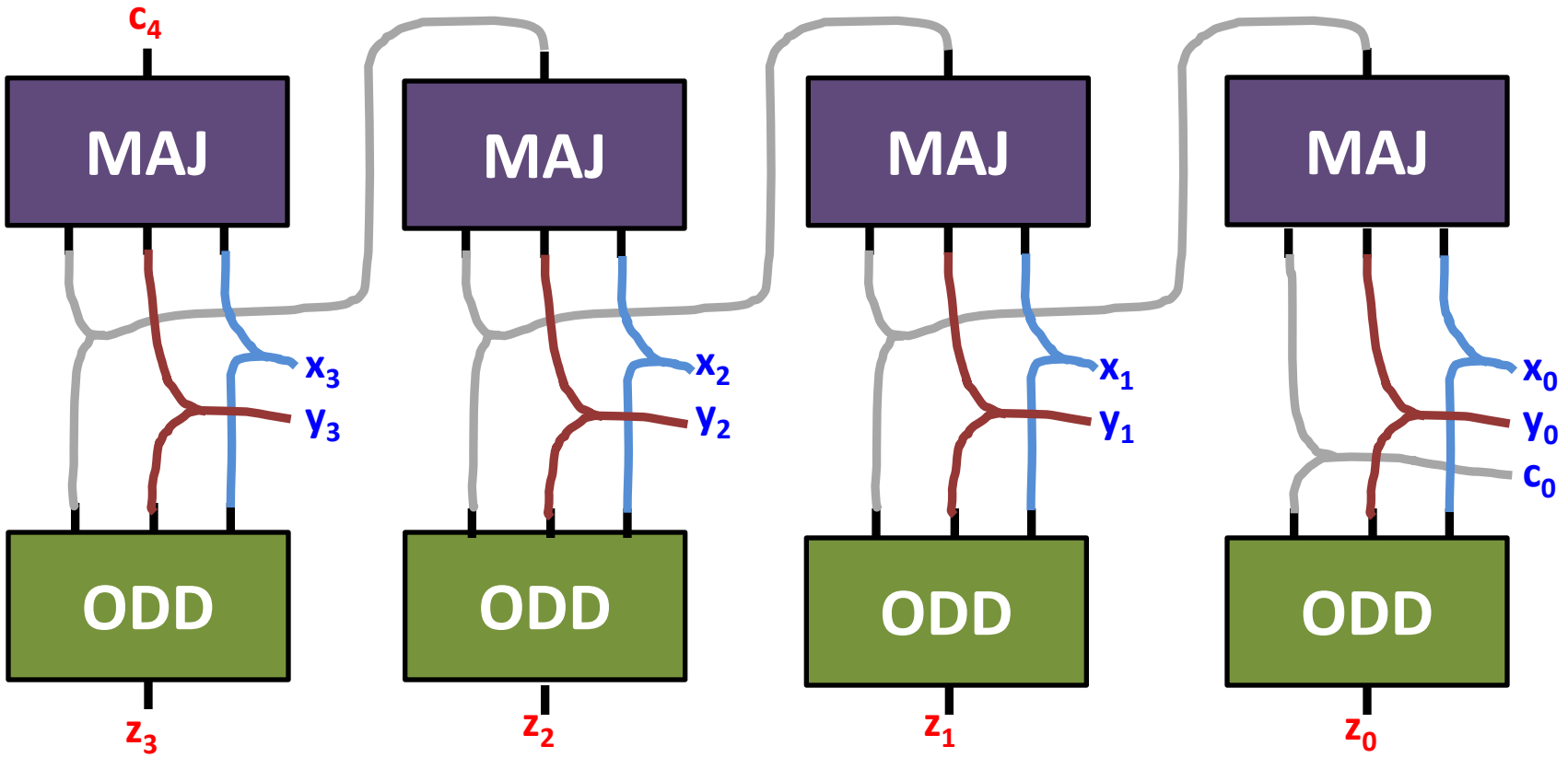




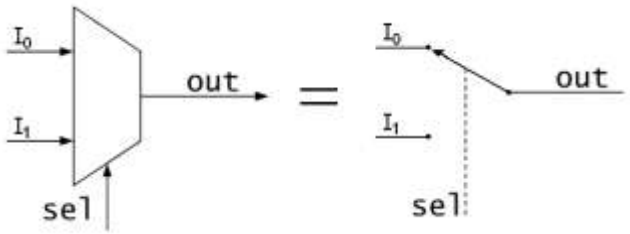
Building a full adder

- Goal: x + y = z for 4-bit integers

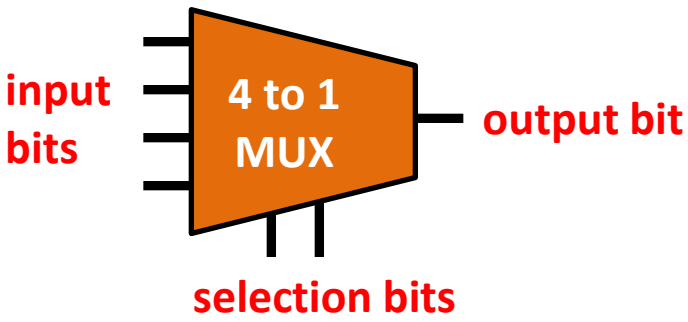
				c_0
	x_3	x_2	x_1	x_0
+	y_3	y_2	y_1	y_0
c_4	z_3	z_2	z_1	z_0

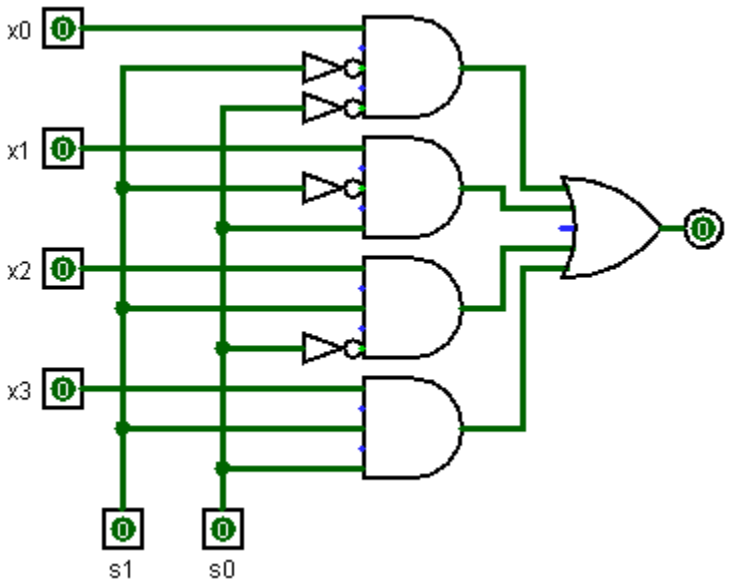


Multiplexer

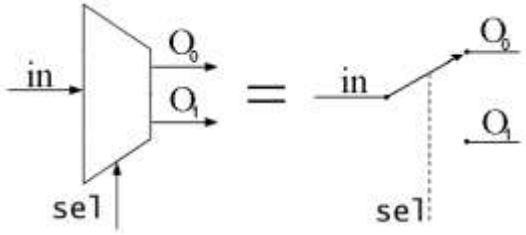


- 2^n-to-1 multiplexer
  - Copies selected input bit to output bit, controlled switch
  - Input: n selection bits, 2^n input bits
  - Output: 1 output bit

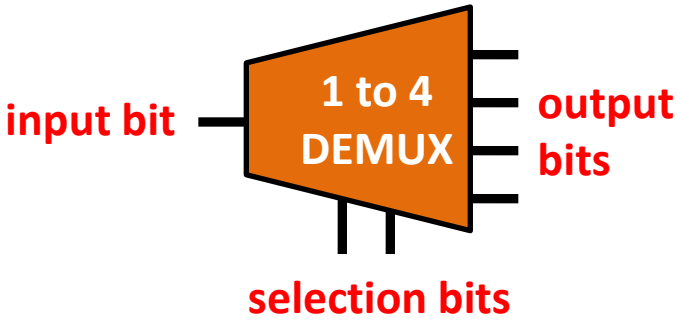


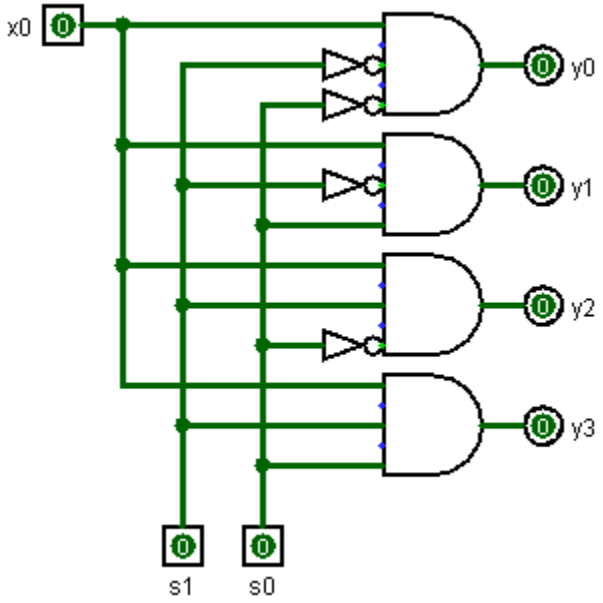


Demultiplexor



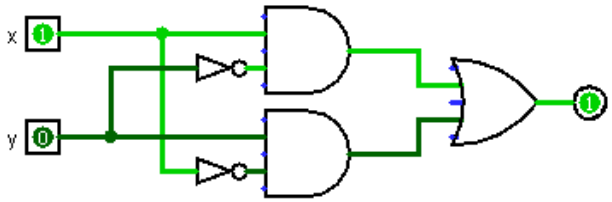
- 1-to-2^n demultiplexer
  - Copies single input bit to one of 2^n output bits
  - Input: n selection bits, 1 input bits
  - Output: 2^n output bits



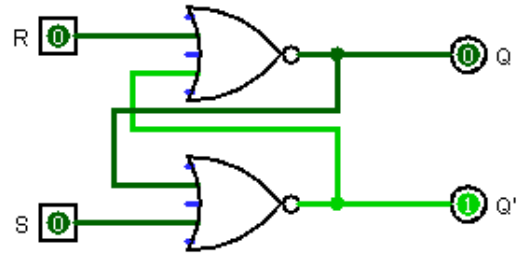


Combinational vs. Sequential circuits

- Combinational circuits
  - Output determined only by inputs
  - Can draw with signals going strictly left-to-right



- Sequential circuits
  - Output determined by inputs and previous outputs
  - Feedback loop!

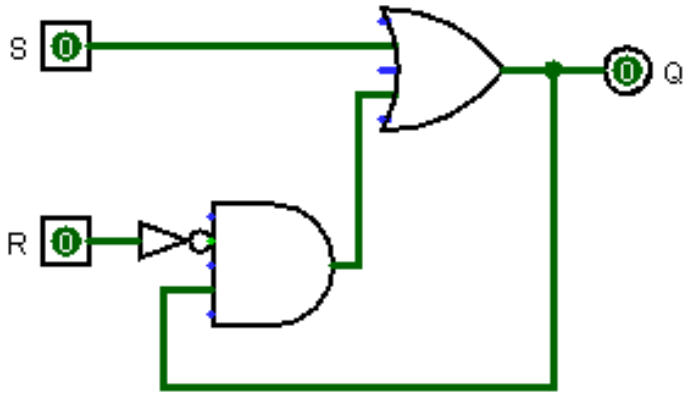


SR latch

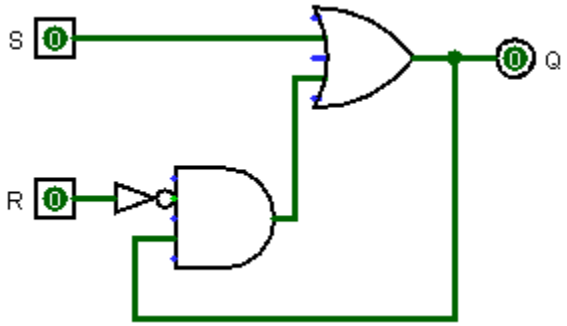
- SR Latch
  - Basic sequential circuit for storing a bit
  - Input bit: S = SET (turn the bit on)
  - Input bit: R = RESET (turn the bit off)

S (set)	R (reset)	Q(t)	Q(t+1)
0	0	x	x
0	1	x	0
1	0	x	1
1	1	x	not used

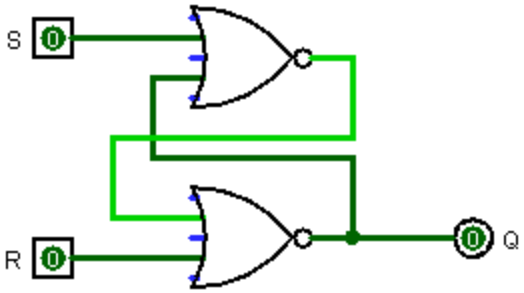
Excitation table for SR latch.
x = value (0 or 1) at time Q(t)



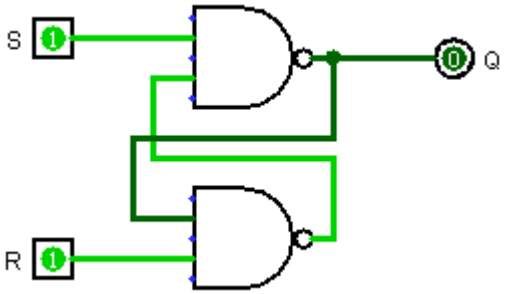
SR latch variants



SR latch
Set and reset on high signal



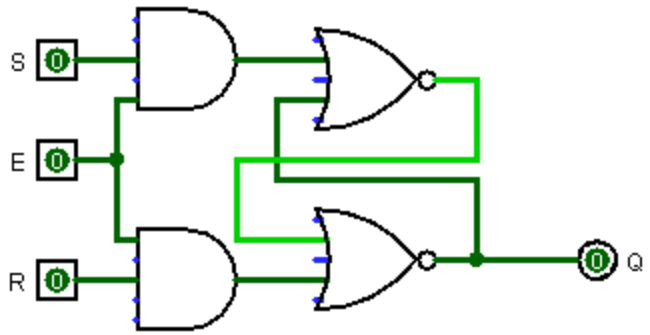
SR NOR latch
Set and reset on high signal



S'R' NAND latch
Set and reset on low signal

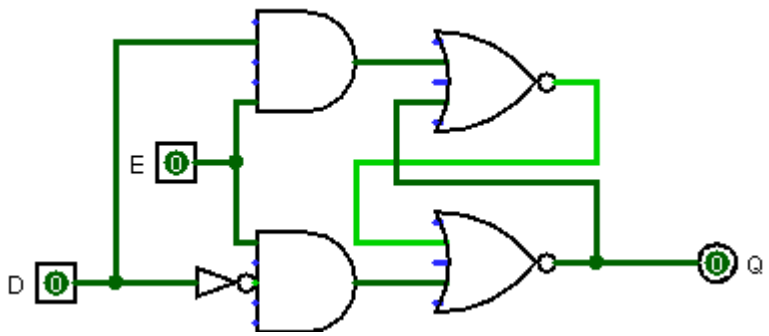
SR latch with enable

- Often we want to control when latching occurs
- Add enable line
  - Must be high to SET / RESET



D latch with enable

- D latch (D for "data")
  - Eliminates invalid input: S=R=1
  - Whenever enable high, memorizes D bit

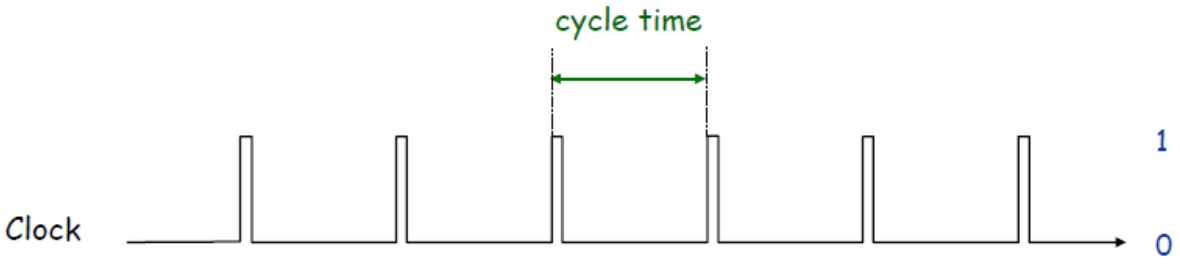


D (data)	E (enable)	Q(t)	Q(t+1)
0	0	x	x
0	1	x	0
1	0	x	x
1	1	x	1

Excitation table for D latch with enable.
x = value (0 or 1) at time Q(t)

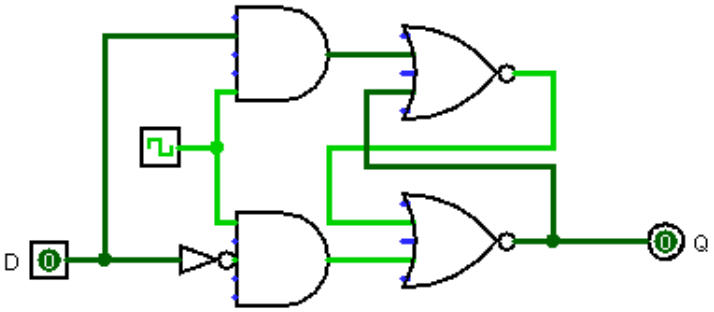
Clock

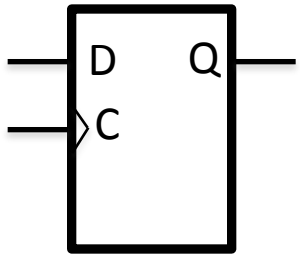
- Clock
  - Fundamental component of computers
  - Synchronizes different circuit elements
  - Regular on-off pulse
    - Pulse less often than worst case internal propagation delay of any circuit
  - 1 GHz clock = 1 billion pulses per second



Clocked D flip-flop

- D flip-flop
  - Hook the enable line to a clock
  - Clocked latch = flip-flop
  - State change has to wait for next clock cycle



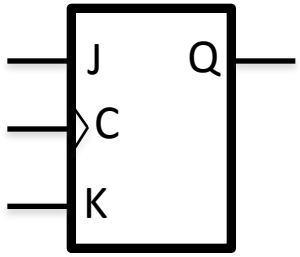


D (data)	C (clock)	Q(t)	Q(t+1)
0	high	x	0
1	high	x	1

JK flip-flop

- JK flip-flop
  - Refinement of SR flip-flop
  - SET: J=1, K=0
  - RESET: J=0, K=1
  - TOGGLE: J=1, K=1 (flips the stored bit)

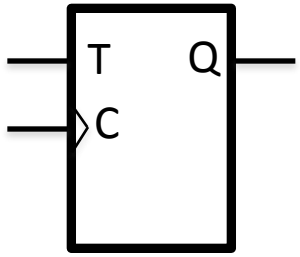
J	K	C (clock)	Q(t)	Q(t+1)
0	0	high	x	x
0	1	high	x	0
1	0	high	x	1
1	1	high	x	x’



T flip-flop

- T flip-flop "toggle"
  - Like JK but connect J and K together
  - LEAVE: T=0
  - TOGGLE: T=1 (flips the stored bit)

T	C (clock)	Q(t)	Q(t+1)
0	high	x	x
1	high	x	x'



Summary

- Combinational circuits
  - How we compute things
  - Built from basic gates like {AND, OR, NOT}
  - Examples: MAJ, ODD, adder, (de)multiplexer
- Sequential circuits
  - How we remember things
  - Clock needed to synchronize things
  - Latches + clock = flip-flop
  - Examples: SR latch, D flip-flop, JK flip-flop, T flip-flop