Chapter 6:

Digital Logic Basics

Topics:

Combinational logic gates
Registers and memory
Flip-flops and finite state machines

Reading: P&H Appendix B

Logisim examples:

http://unixlab.sfsu.edu/~whsu/csc256/LABS/DOCS/LogisimExamples/

Basic Logic Functions

x0	x1	not x0	x0 and x1	x0 or x1	x0 nand x1	x0 nor x1	y xor z
0	0	1	0	0	1	1	0
0	1	1	0	1	1	0	1
1	0	0	0	1	1	0	1
1	1	0	1	1	0	0	0

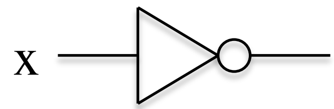
Digital computer systems are constructed from components that implement these functions.

Voltage on a pin/wire: +5 V or 0 V
Logic value: 1 or 0

3 ways to play with logic circuits:
1) “pencil and paper”
2) build circuits! (not hard, but need lab etc; see Engr 356/357)
3) use a logic simulator; our choice is *logisim*:

http://www.cburch.com/logisim/index.html

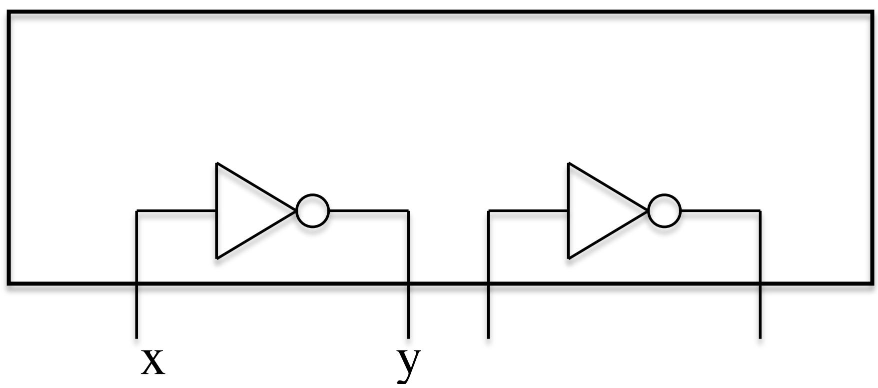
Not gate:



$y = \text{not}(x) = \overline{x}$

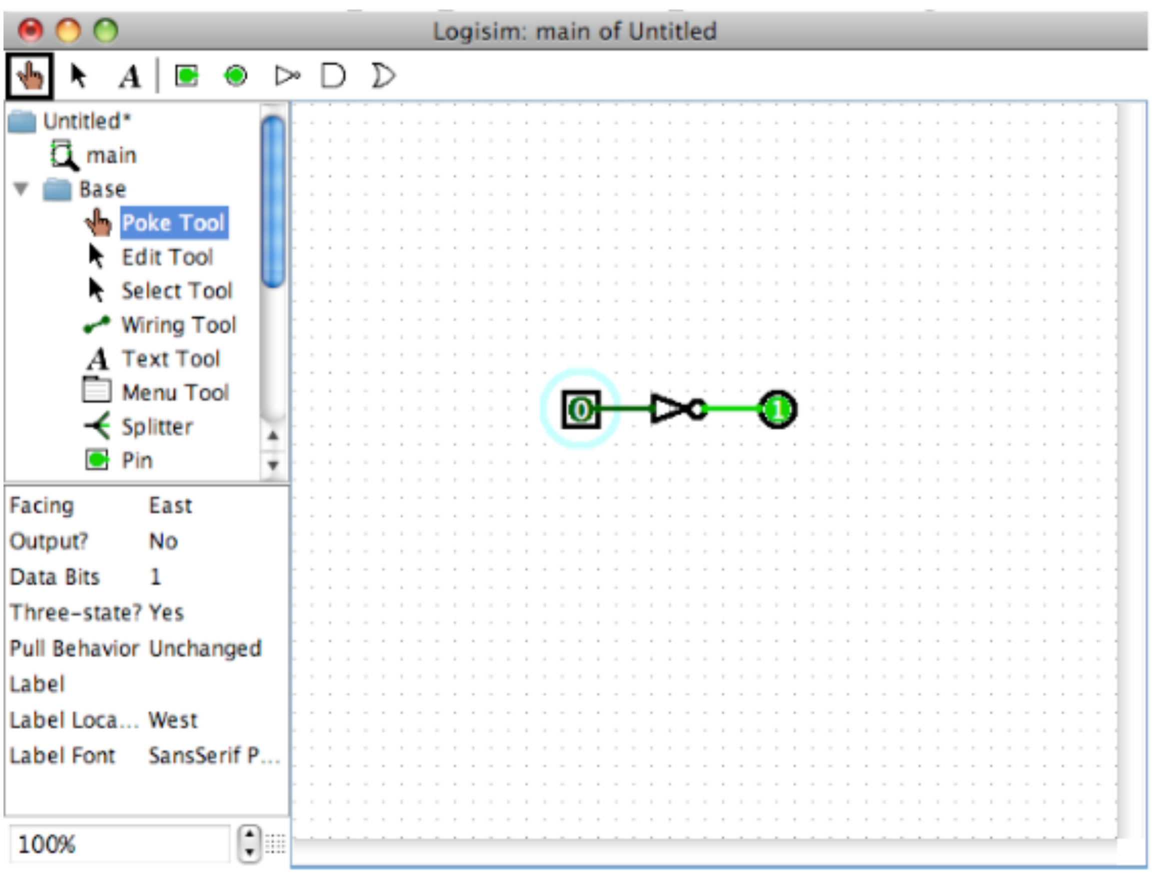
x is input, y is output

Usually have several not gates on a chip:



Set x to 0: connect x pin to *ground (0 volts)*
    y pin will show a value of 1 (+5 volts)
Set x to 1: connect x pin to +5 volts
    y pin will show a value of 0 (0 volts)

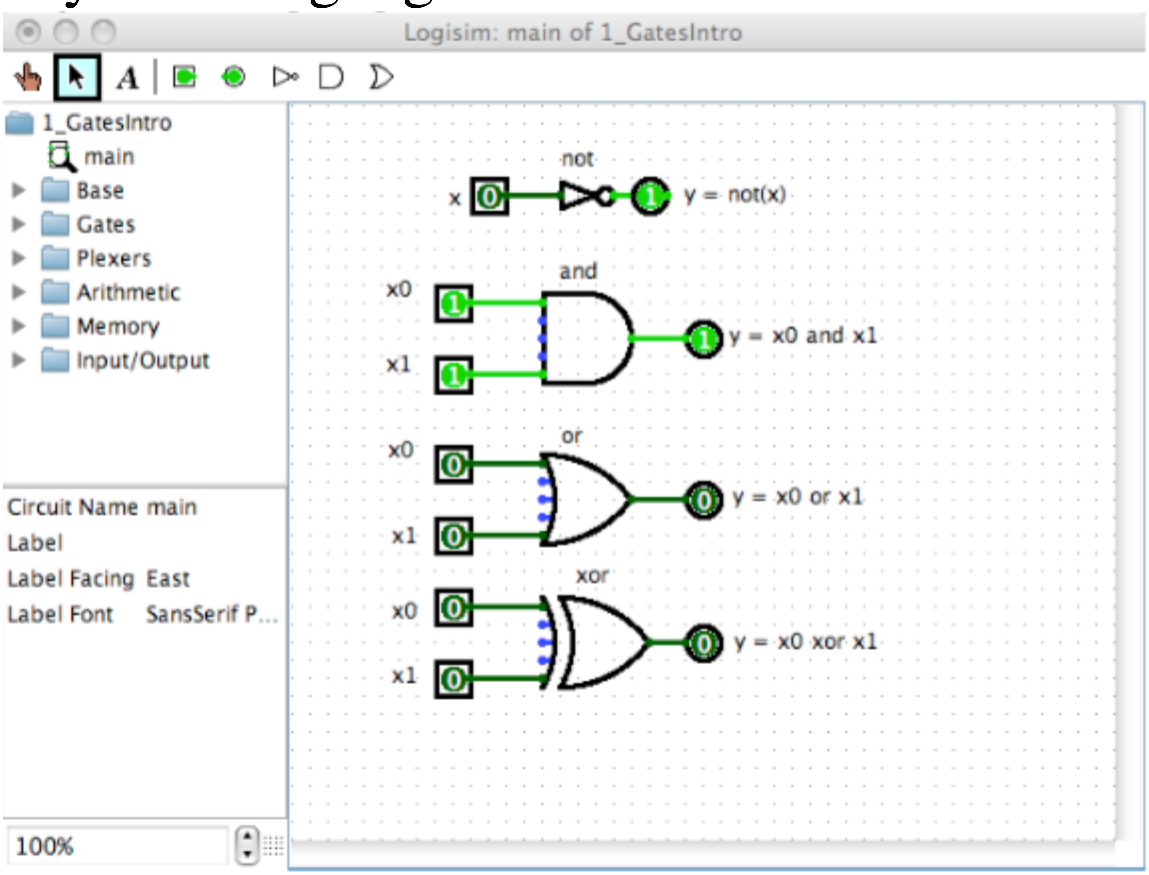
In Logisim:
1) click on not gate at top of window
2) click in dotted *editing area*
3) click on input pin, place in editing area, near input of not gate
4) click on output pin, place in editing area, near output of not gate
5) select *Wiring tool* (under Base menu)
6) connect input pin to input of not gate
7) connect output pin to output of not gate



To set input pin x to 0 or 1, use the Poke Tool (also in the Base menu).
Click on input to watch it change!

Note that the input pin takes on 3 values: 0, 1, and x.
x means *don't care;* can be either 0 or 1.
(More on this later…)

Try other logic gates:



Boolean algebra notation:

x0 and x1	x0	$\bullet$	x1
x0 or x1	x0	+	x1
x0 xor x1	x0	$\oplus$	x1

More complicated functions are implemented with more complex circuits.
A Boolean function can be specified:
1) in Boolean algebra notation
2) as an equivalent digital logic circuit
3) as a truth table; for each combination of inputs, list its output

Example: given Boolean expression

$$y = x0 \bullet ((\ \text{not}(x1) + x1 \bullet x2\ ))$$

Show an equivalent digital logic circuit and truth table.

Truth table:

x2	x1	x0	T0=x1•x2	T1=T0+~x1	y=x0•T1
0	0	0	0	1	0
0	0	1	0	1	1
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	0	1	0
1	0	1	0	1	1
1	1	0	1	1	0
1	1	1	1	1	1

Build circuit in Logisim, check result…

Note: Logisim's *Analyze Circuit* feature will do this automatically!

1) Go to Menu item Project-> Analyze Circuit
2) Click on Inputs: enter names of inputs
3) Click on Outputs: enter name(s) of outputs
4) Click on Expression: enter expression
5) Click on Build Circuit (check also Table!)

If we're given a truth table to start with, we can also build a digital logic circuit, or derive a Boolean expression that is equivalent.

How to do this? Simple procedure:
1) look at outputs that are 1
2) *and* together the input combinations
3) *or* together all the combinations in Step 2
(This is called sums of products…)

Lets go through this with the truth table from previous example…
1) look at outputs that are 1
2) *and* together input combinations

$$\overline{x2}\ \overline{x1}\ x0 \qquad x2\ \overline{x1}\ x0 \qquad x2\ x1\ x0$$

3) *or* together all combinations in Step 2

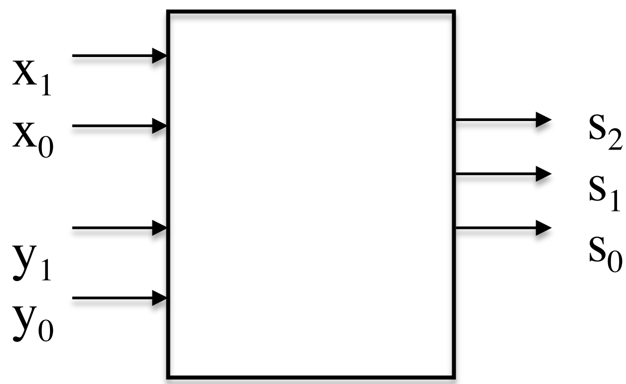
$$y = \overline{x2}\ \overline{x1}\ x0 + x2\ \overline{x1}\ x0 + x2\ x1\ x0$$

Note that more than one circuit/Boolean expression corresponds to the same truth table! (Can also enter truth table in Logisim!)

<u>Building a simple adder circuit</u>

Suppose we want to build a circuit that adds two 2-bit integers, x (x_1x_0) and y (y_1y_0), to compute a sum s ($s_2s_1s_0$).

Basic idea:



Truth table:

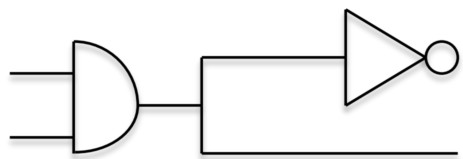
X1	X0	Y1	Y0	S2	S1	S0
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	1	1
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	0	1	1
0	1	1	1	1	0	0
1	0	0	0	0	1	0
1	0	0	1	0	1	1
1	0	1	0	1	0	0
1	0	1	1	1	0	1
1	1	0	0	0	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	1
1	1	1	1	1	1	0

Follow the procedure for each digit of the sum:

```
~x1 ~x0 ~y1 y0
+ ~x1 ~x0 y1 y0
+ ~x1 x0 ~y1 ~y0
+ ~x1 x0 y1 ~y0 …
```

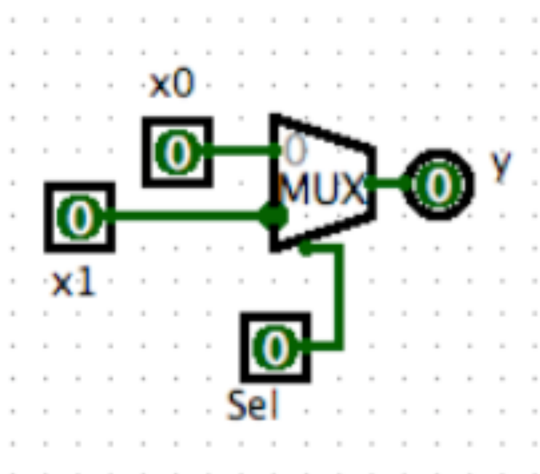
Making selections

It's fine to connect one pin to several inputs:



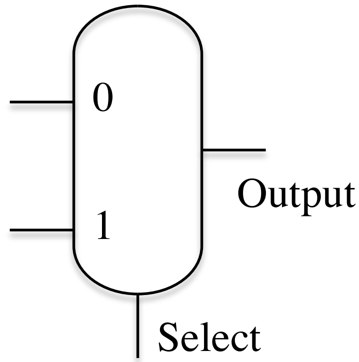
But it's *not* ok to connect more than one output to one pin! (Like assigning two values to the same variable; only one value will be result.)

To select one of 2 signals: use a *multiplexor*

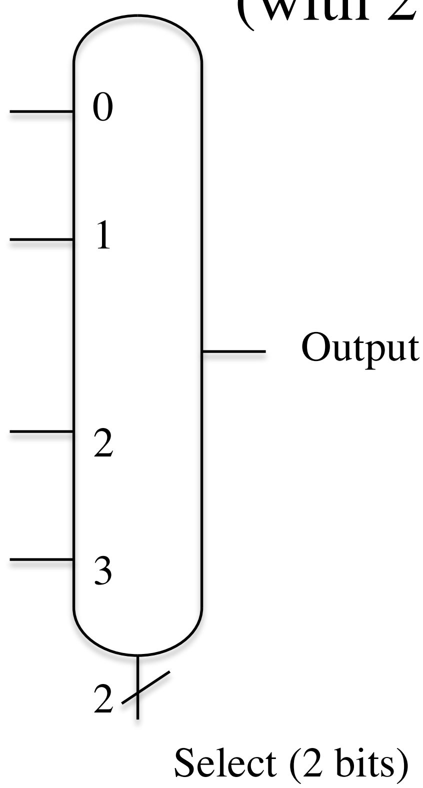


Inputs are numbered 0-1 (top to bottom)
Sel signal selects the input that goes to output
If (Sel == 0) y = x0 else y = x1

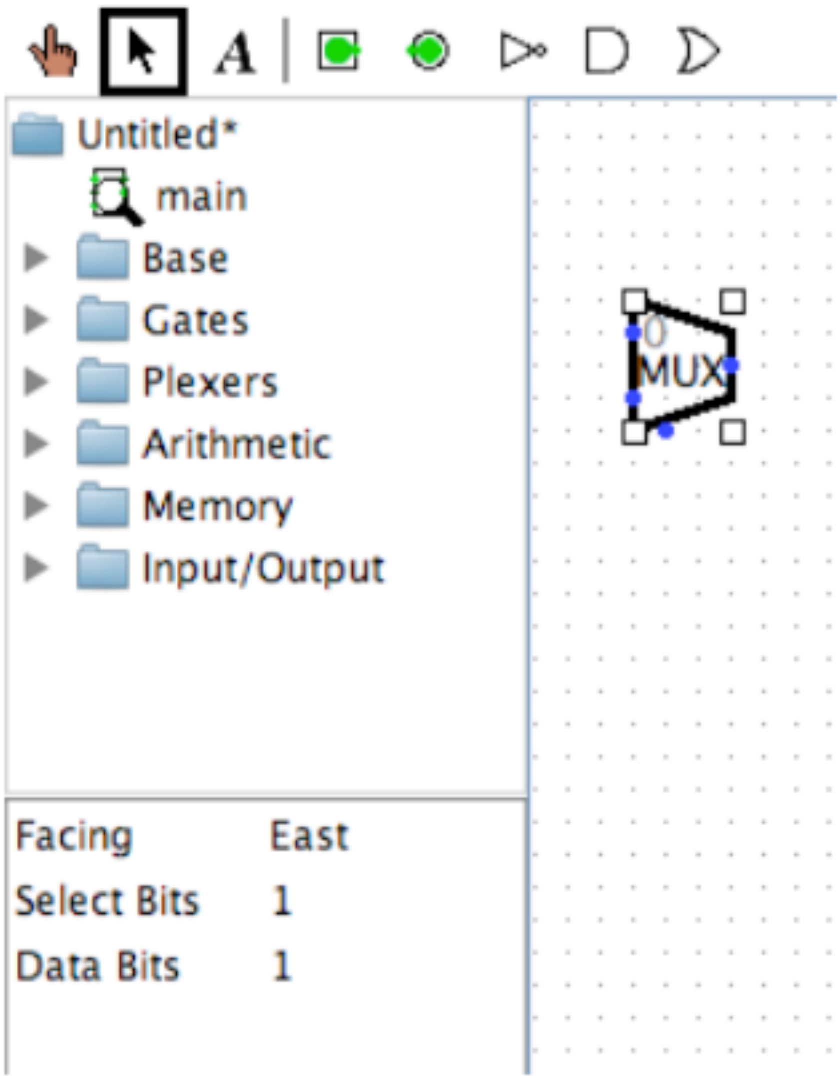
2 x 1 multiplexors are usually drawn like this:



4 x 1 multiplexor selects one of four inputs
(with 2 select bits, 00 01 10 11):

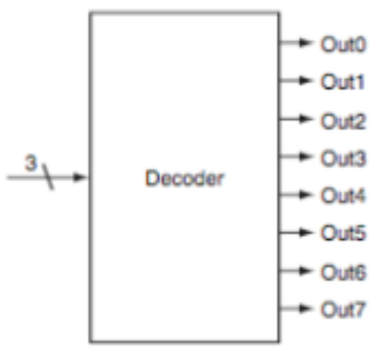


In general, a 2^N x 1 multiplexor has N select bits.In Logisim, can change size of multiplexor in lower left *attributes* pane:



Multiplexors select one of 2^N inputs, sends it to the output, based on N select bits.

Decoders are like multiplexors in reverse! There are N input bits, only one of 2^N output bits is set to 1.



a. A 3-bit decoder

Inputs			Outputs							
I2	I1	I0	Out7	Out6	Out5	Out4	Out3	Out2	Out1	Out0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

b. The truth table for a 3-bit decoder

FIGURE C.3.1 A 3-bit decoder has 3 inputs, called I2, I1, and I0, and 2^3 = 8 outputs, called Out0 to Out7. Only the output corresponding to the binary value of the input is true, as shown in the truth table. The label 3 on the input to the decoder says that the input signal is 3 bits wide.

stopped 10/27/16

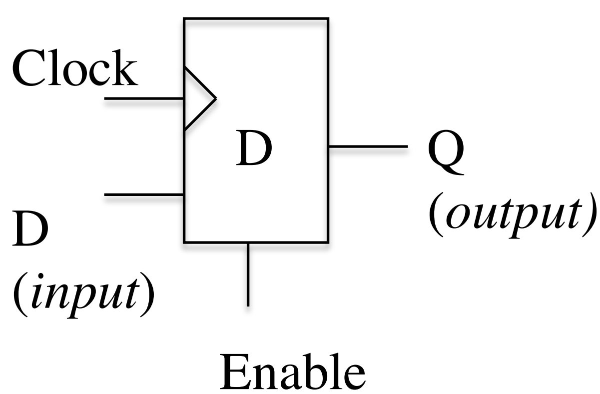
Flip-flops and Registers

So far, we talked about *combinational logic components* and *circuits*.
AND, OR, multiplexors etc have no *memory* or storage; when the inputs change, the outputs change as well.

A flip-flop is a digital logic component that is a *memory element*; it can store information.

There are several types of flip-flops; we'll only cover D (for *Delay*) flip-flops.

A D flip-flop stores one bit:



The stored bit (or *state* of the flip-flop) can be "read" on the Q (output) pin.

The state of the flip-flop will change only if 1) Enable is set to 1, and 2) the *clock* pin goes from 0 to 1. (This is called a *rising edge*.)

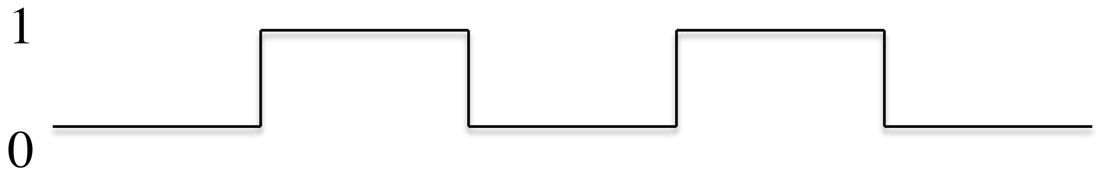
If there is a rising edge on the clock pin, the state Q will be assigned the value on the D input pin. (If there is no rising edge on the clock pin, Q will *never change*.)

(See Logisim example: 2_DFlipFlop)

Enable = 0: flip-flop is *disabled*; does not change state

Enable = 1: flip-flop is *enabled*; when clock has rising edge, value at D is stored (assigned to state)

Clock signal keeps changing 0->1->0->1…:



Simple timing diagram (C is clock):

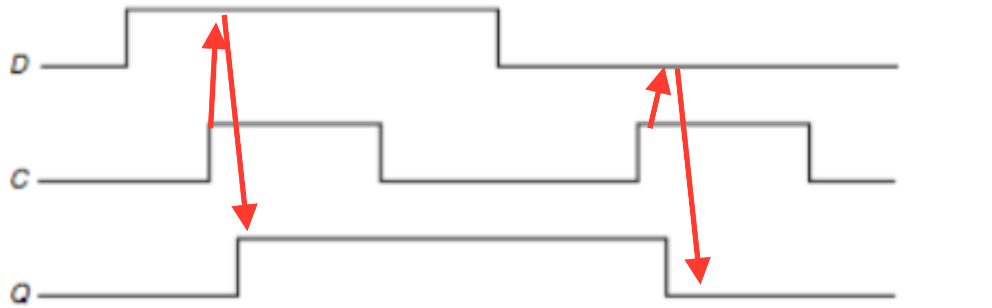


FIGURE C.8.3 Operation of a D latch, assuming the output is initially deasserted. When the clock, *C*, is asserted, the latch is open and the *Q* output immediately assumes the value of the *D* input.

Note 1: state (Q) does not change until rising edge of C.

Note 2: there is a small *latency* or time-lag between the rising edge of the clock, and the change of state Q. (Signals take a little time to travel through wires and gates!)

All circuits in a system "see" the same clock signal (*synchronous* system). State changes occur at about the same time. (Otherwise chaos!)