CS99S
Laboratory 4 Preparation

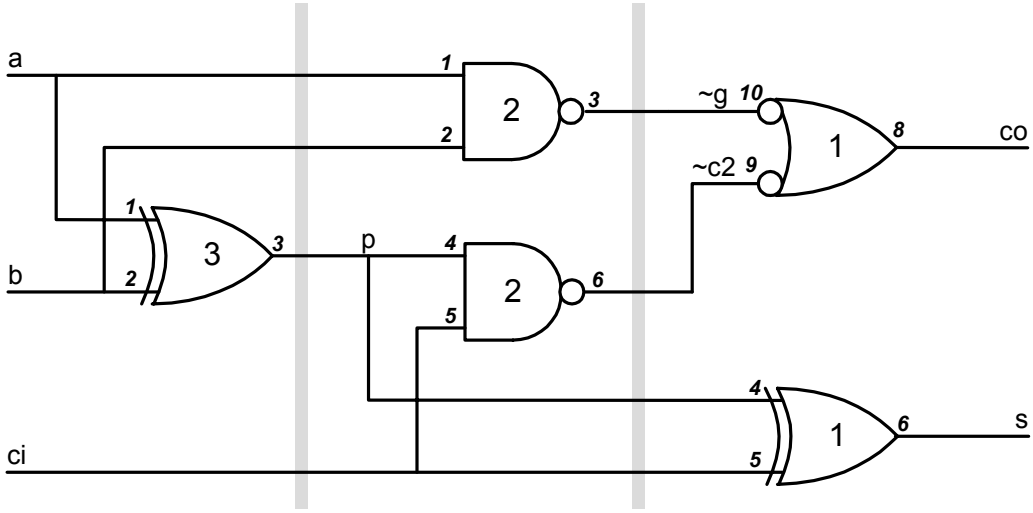
October 22, 2001

Objectives:

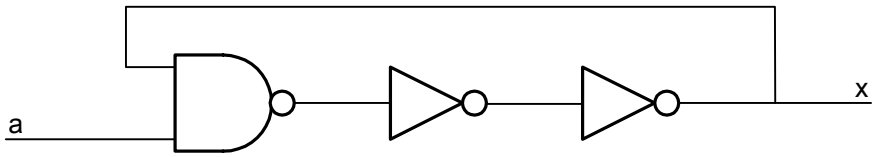
1. Understand principles of sequential logic
2. Build a latch and a flip-flop from gates
3. (Optional) Experiment with dynamic storage

Feedback and State

So far the circuits we have been building are *combinational* circuits in which the output depends only on the present state of the inputs. There are no feedback paths in combinational circuits: signals flow in one direction from the inputs to the outputs with no loops. Because there is no feedback you can divide a combinational circuit into *levels* as illustrated below for the full adder circuit from lab 3. Start by labeling gates that drive an output with a "1". Then label each gate for which all of the gates it drives are labeled with one more than the maximum label. Continue until all gates are labeled. You can evaluate the value generated by a logic circuit by evaluating each gate exactly once – in the reverse order of this labeling. The first rank of gates depends only on the inputs, the second rank depends only on the first rank, and so on. The value eventually computed for the output depends only on the value of the input and the connection of the gates. It is not affected by the state of the circuit before the inputs are applied.

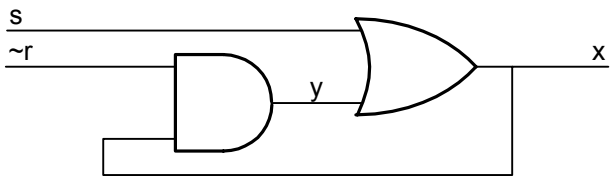


If we add feedback to a logic circuit, one of two things happens. If the feedback is unstable, it causes an oscillation. On the other hand, if the feedback is stable it holds the state of the circuit after the inputs are removed. In either case, we call the circuit a *sequential* logic circuit because the output now depends on the *sequence* or history of inputs and not just on the present *combination* of the inputs.



a	x	next x
0	*	1
1	0	1
1	1	0

The circuit above (called a gated ring oscillator) is an example of a circuit with unstable feedback. When input a is high the circuit will oscillate rapidly producing alternating 1s and 0s on output x. To see that the circuit is unstable, cut the feedback loop from x to the NAND gate. The truth table for the circuit with the feedback cut is shown above x corresponds to the input of the NAND gate and next x corresponds to the output. If the upper input of the NAND is 1, it will cause the output to go to 0, so we say that x=1 leads to next x = 0. Similarly, if the upper input of the NAND is 0, it will cause the output to go to 1. Thus x will not settle to a stable value, but will oscillate as long as a is high. In general, a feedback loop with inversion around the loop is unstable. We can see this in the truth table (called a state table) as two or more rows that lead to one another in a cycle. In this case, the last two rows lead to one another.



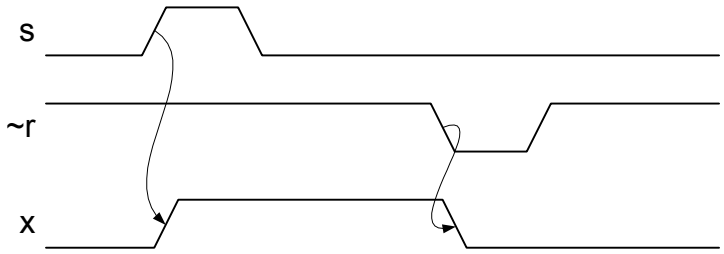
The circuit above (called an RS flip-flop) has stable feedback. To see this, break the feedback loop from x to the input of the AND gate. When s is low and ~r is high a 1 on the input of the AND will give a 1 on x, and a 0 on the input of the AND will give a 0 on x. The output is stable, but depends on the present state of the circuit, the values of x and y, in addition to the inputs. We say that x is a *state variable* of this circuit. We can write a truth table for the output of the circuit in terms of its inputs and its state variables by breaking the feedback loop at the state variable. The truth table for this particular RS flip-flop, using * to denote don't care, is:

s	~r	x	next x
0	0	*	0
0	1	0	0
0	1	1	1
1	0	*	illegal
1	1	*	1

We see that the flip-flop is reset (x set to 0) when ~r = 0 and is set (x set to 1) when s = 1 otherwise it holds its previous state. Setting ~r=0 and s=1 is not allowed.

Note that we could just as easily have chosen y as a state variable of the circuit. However, we can't choose both x and y as state variables since the circuit only holds one bit of state. The values of x and y depend on one another. If we pick one as an independent state variable, the other then becomes a dependent variable.

We can illustrate the operation of a circuit by drawing *waveforms* for its inputs and outputs (sometimes called a *timing diagram*). Waveforms for our RS flip-flop are shown below. Time increases from left to right and for each signal voltage increases from bottom to top, so a 0 is at the bottom and a 1 at the top. We start with x in the low state. A high pulse on s sets the flip-flop and then a low pulse on ~r clears the flip-flop. The arrows indicate cause and effect for each output transition. The rising edge on s causes the rising edge on x and the falling edge on ~r causes the falling edge on x.



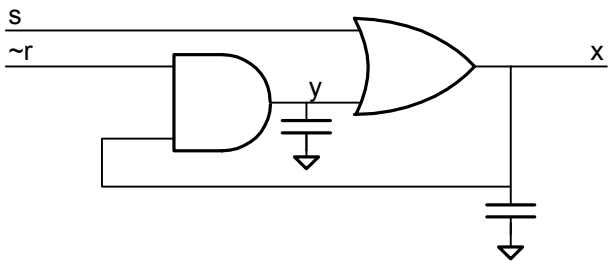
Prep Question 1: Draw the truth table (state table) for the sequential circuit with one input, a, and two outputs x and y described by the following logic equation. What does it do? Sketch the waveforms generated on x and y in response to alternating 1s and 0s on a.

$x = ((a \vee x) \wedge \sim y) \vee (\sim a \wedge x)$

$y = ((a \vee y) \wedge x) \vee (a \wedge y)$

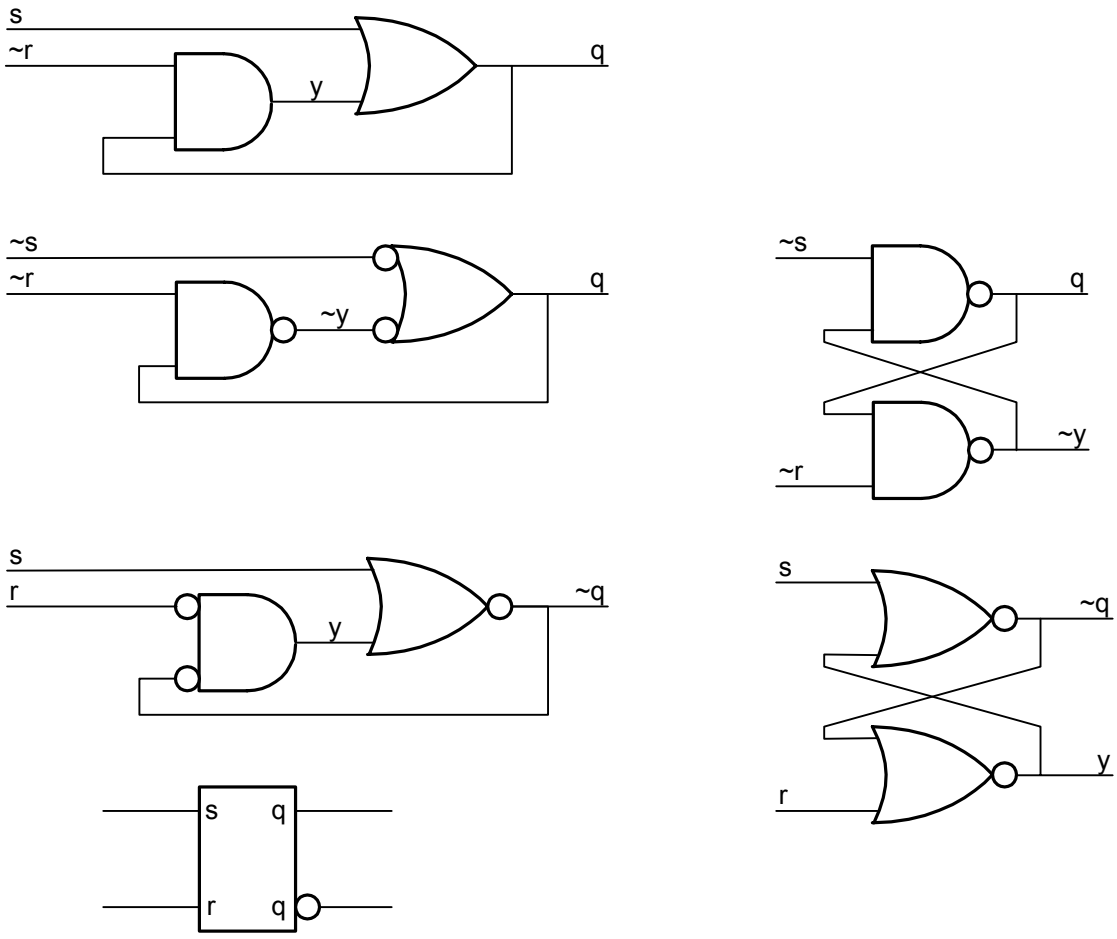
Where is the bit stored?

Where is the bit physically stored in our RS flip-flop when ~r = 1 and s=0? It is the capacitance of the circuit that physically stores the bit. The circuit, showing the parasitic capacitors that store the bit is shown below. For the storage to be stable, it has to resist changing state when a small disturbance is applied to the circuit. Suppose that x = y = 1. A disturbance that discharges either (or both) of the capacitors by less than half of their value will leave the state unchanged and the positive feedback will act to charge the capacitors back up – restoring the state to a full 1. Without the capacitance, a small disturbance could flip the state of x or y, changing the stored bit.



The RS Flip-Flop

Using DeMorgan's Law, we can redraw the RS flip-flop in a number of ways. Some of the most popular are shown below. While the circuits look very different, this is logically the same circuit (with some inversions of inputs and outputs) drawn five different ways.

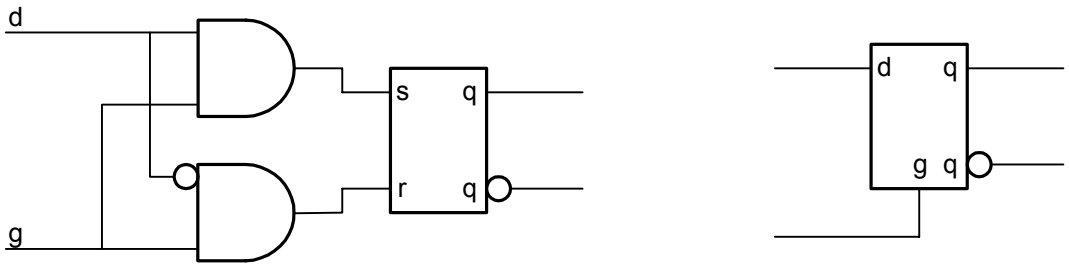


The top circuit is our original non-inverting RS flip-flop redrawn. The second row shows this circuit realized with NAND gates. Both the s and the r inputs are low-true in this version as is the dependent variable y. We can redraw this NAND flip-flop as shown on the right side to make it look more symmetrical. Similarly, the third row shows the flip-flop realized with NOR gates. Here both inputs are high true and the output q is inverted. The NOR circuit is also redrawn at the right in a more symmetrical manner. For shorthand we use the symbol at the bottom to represent the RS flip-flop (in this case a NOR flip-flop since both inputs are high true)[1].

[1] In the NOR flip flop ~q and y are not strictly complements of one another (since s wins fights on ~q and r wins fights on y). However, since s = r = 1 is an illegal input anyway, we ignore this subtlety and label the outputs q and ~q.

The Latch

A latch is a sequential circuit that passes its input to its output when an enable signal, g, is high and holds its previous state when the enable signal, g, is low. We can build a latch from our RS flip-flop by gating the data signal into the r and s inputs of the flop as follows:

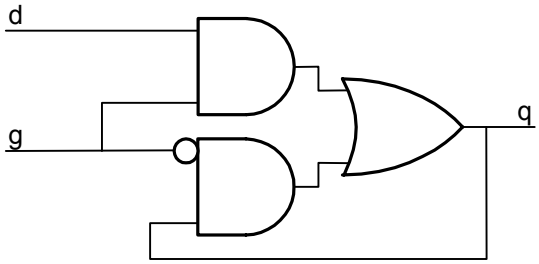


When the g input is high a high input on d sets the flop and a low input on d resets the flop. When the g input is low, both s and r are low and the flop holds its previous state. For shorthand, we represent a latch with the symbol on the right. A latch may or may not have both polarities of output.

Prep question 2: Draw a schematic for a latch using only NAND gates. Label the chip numbers and pin numbers for this schematic because you are going to build it using 74AC00 chips.

A Latch that doesn't work

If you draw the state table for the following circuit you might think that it's a latch. When g is high d is passed to the output q, and when g is low the value on q is recirculated with positive feedback. In fact, it's a faulty latch. It doesn't always work.

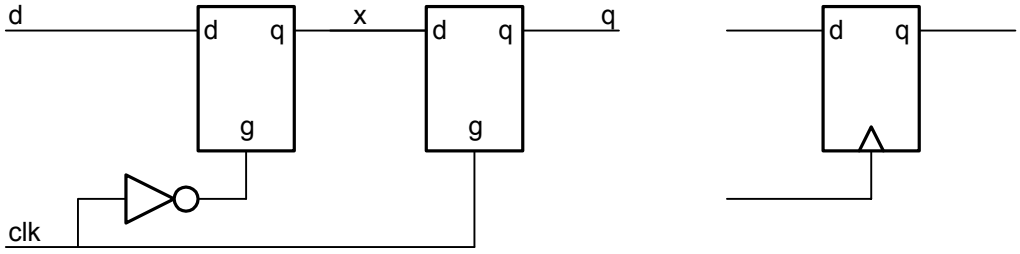


Prep question 3 (optional challenge): Explain why the latch above doesn't work and fix it. (Hint: consider what happens when g changes from high to low if the upper AND gate is much faster than the lower AND gate).

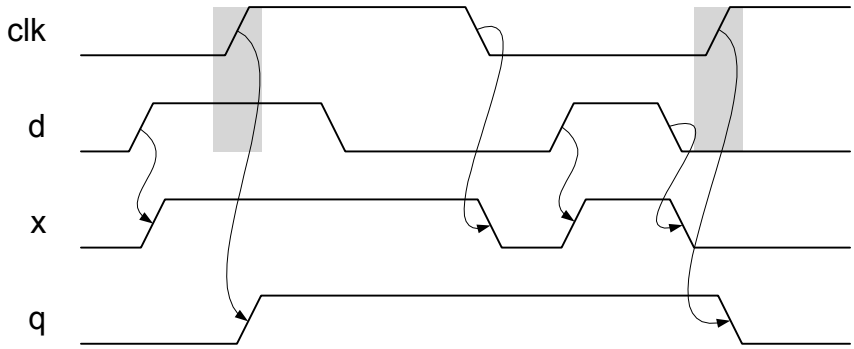
The D Flip-Flop

If we put two latches back-to-back enabled by opposite polarities of a clock signal, as shown below, we can sample the data on the rising *edge* of the clock. This type of flip-flop is often called a *master-slave* flip-flop. The first latch is considered the *master* and

the second latch the *slave.* When the clock is low, the master latch passes the data to the intermediate signal, x. However the new data cannot update the output q because the enable to the slave latch is low. When the clock rises, the enable to the master latch goes low – holding the value of d at the rising edge of the clock on the intermediate signal x, and the enable to the second latch goes high, passing this sampled value to the output q. For shorthand, we represent an edge-triggered D flip-flop with the symbol on the right. The triangle indicates an edge-sensitive clock input.



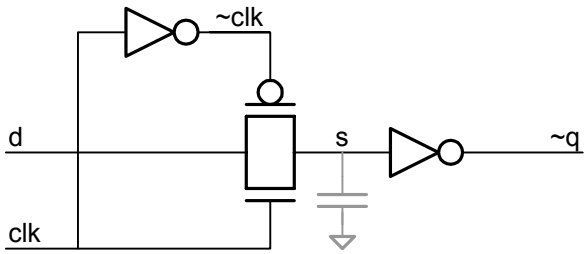
The waveforms below illustrate operation of a master-slave edge-triggered D flip-flop. This figure is a timing diagram. Time advances from left to right. For each signal, the voltage (high or low) of that signal is shown as a function of time. The arrows between signals show cause and effect for each transition. When the clock is low, internal signal x follows the data input but then freezes when clock goes high – delaying the falling edge on x until the clock goes low again. The output q follows x when the clock is high. Since x is frozen at this point, the flip-flop in effect samples input d on each rising edge of the clock.



To operate properly the d input to an edge-triggered input must be stable from a short time (called the *setup time*) before the clock rises until a short time after the clock rises (called the *hold time*). This region during which d must be stable is illustrated as a shaded region in the figure. If d changes during this region, the flip-flop may malfunction.

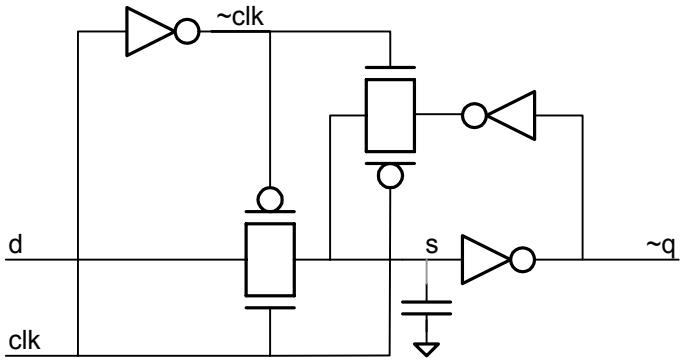
Prep Question 4: Starting with your schematic of a latch, draw a schematic of a D flip-flop using 74AC00 NAND gates. Number your schematic to facilitate construction.

Dynamic Storage (Optional)



Some CMOS circuits use *dynamic* storage to reduce device count and area. A dynamic latch is shown above. When the clock is high, the input *pass gate* (the NFET and PFET) is on connecting input d to storage node s. If d is high, s will also be high and if d is low, s will be low. Output ~q will always be in the opposite state as storage node s. When the clock falls, the pass gate is turned off and node s is isolated from the input. Whatever value was stored on node s on the falling edge of the clock remains stored on the *parasitic* capacitance of node s. This capacitance, illustrated in gray in the figure is not a separate component but rather is the capacitance of the inverter and metal lines connecting the components. After the clock falls, the dynamic latch will hold its value until the charge on node s *leaks* back through the pass gate (which is never completely off). The clock needs to go high before this happens to *refresh* the stored value or replace it with a new value.

A dynamic RAM or DRAM also stores a bit by putting a charge on a capacitor but uses a different form of readout to eliminate the need for the capacitor in the figure. Also, most DRAMs use only an NFET in the pass gate eliminating the PFET. Hence they only need one transistor (and one capacitor) per bit cell.



We can gain some of the efficiency of a CMOS pass-gate circuit without requiring refresh by building a *static* CMOS latch as shown above. This circuit is identical to the dynamic latch with the addition of a feedback inverter and pass-gate. When the clock goes low, the input pass gate turns off and the feedback pass gate turns on refreshing storage node s with a fully restored copy of its value.

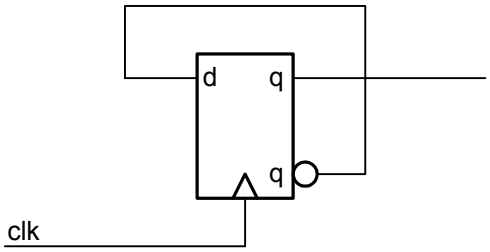
The dynamic latch requires only four transistors (if we ignore the clock inverter which can be shared) compared to 18 transistors for a NAND latch and 8 transistors for a static pass-gate latch so it results in considerable area and power savings. This is at the expense of having to refresh the latch before the charge leaks off. For example, we can't run the clock too slow or the charge will leak off during a long low period.

Prep Question 5 (optional): Draw a schematic of a dynamic latch using an NFET and PFET from your 4007 and two inverters from a 74AC04. Make sure to label all unit numbers and pin numbers.

The Lab

For the lab itself you should do the following:

1. Wire up your latch from Prep Question 2. Verify that you can store a 1 and a 0 in your latch and that it holds this value when the g input is low.
2. Using your latch as a starting point, wire up your D flip-flop from Prep Question 4. Verify that you can store a 1 and a 0 into the D flip-flop and that it holds its value except at the rising edge of the clock.
3. Connect your D flip-flop as a divide-by-two circuit as shown below. Connect the clock to a debounced push button (from Lab 1). Each time you push the button, the output, which you should connect to an LED, should toggle between a 1 and a 0.



4. (Optional) Wire up a dynamic latch from prep question 5. Store a 1 into the latch and then set d=0. How long does the output remain in the 0 state when the clock is low? Repeat the experiment storing a 0 in the latch and then setting d=1. How long does the output remain in the 1 state?