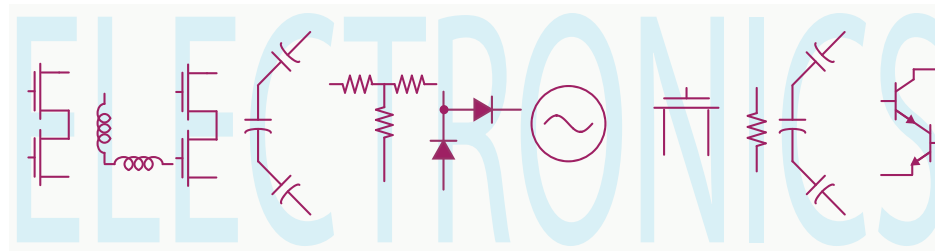


EE 42/100
Lecture 24: Latches and Flip Flops



Rev B 4/21/2010 (2:04 PM)

Prof. Ali M. Niknejad

University of California, Berkeley

Copyright © 2010 by Ali M. Niknejad

Gate Propagation Delay

- In a logic gate, the output does not change instantaneously when the inputs change. This delay between the output and input is caused by the finite speed of light (time it takes signals to propagate along wires). More commonly, the delay is set by the non-zero capacitance and non-zero resistance of the transistors, which means there is an RC delay associated with the gate input/output
- The maximum time delay from when an input changes to the output reaching its final state is known as the *propagation delay*, or t_{pd} .

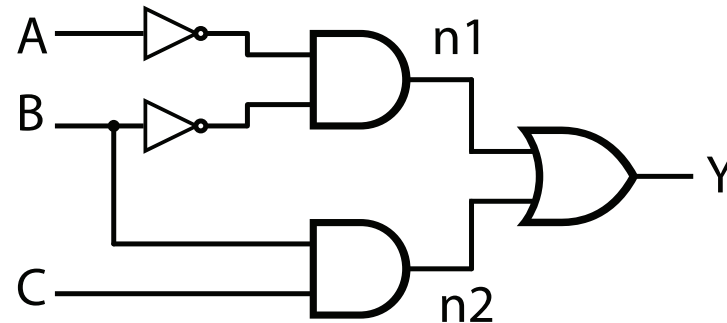
Contamination Delay

- The *contamination delay*, t_{cd} , is the minimum time from when an input changes until any output changes (not necessarily going to the steady value).

Example

- Consider the multi-level digital gate shown below. Notice that for inputs A and B there is a long path from the input to the output which has to go through three gates. The longest path from the input to the output is known as the *critical path*, because it sets the maximum speed at which we can run a circuit.
- The shortest path from the input to the output is known as the *short path*, and it sets the contamination delay for the circuit.

Glitches



- Glitches can occur in a logic circuit if the output transitions to the wrong value but then finally settles to the correct value. The intermediate output is known as a glitch. Note that if the circuit is run sufficiently slow, determined by the propagation delay, no errors will result. But this extra transition is often unwanted (causes extra power dissipation) and may lead to problems.
- In this example, suppose $A = 0$, and $B = C = 1$. Next suppose B transitions from $1 \rightarrow 0$. Due to the propagation delay of the inverter, the $B = 0$ will initially cause both OR inputs to go to zero, which causes the output to drop to 0, but then after the B signal is inverted and applied to the AND, the output is restored.
- The circuit can be redesigned to avoid the glitch (homework problem?)

Bistable Circuits

- A bistable circuit has two possible stable states. If the circuit is put into a stable state, it will remain there. The circuit has memory. If the stable state is changed (flipped – there are only two states), then it will remain in the new state.
- Two inverters feeding back on each other, or cross coupled (inputs/outputs interchanged), form a bistable circuit. Note that when we power on the circuit we have no idea which state the bistable circuit will be in !
- There is also a metastable state where each output is at the midpoint. But if there is any noise or disturbance in the circuit, it will leave the metastable state.

SR Latch

- The SR latch is essentially the same as the cross-coupled inverters, except we now have a way to set or reset it into a given state.
- To understand the operation, assume that we have $R = 1, S = 0$. Then the first NOR produces a 0 output (it only needs one 1 input to do so) and so $Q = 0$. But then the second NOR is fed by 0 on both inputs, so its output is 1, or $\overline{Q} = 1$.
- By symmetry, if we apply $R = 0$ and $S = 1$, then $Q = 1$ and $\overline{Q} = 0$.
- What if $R = S = 1$? Then both NOR gates produce a 0 output and $Q = \overline{Q} = 0$.

SR Latch Truth Table

- Finally, if $R = S = 0$, then the first NOR has an input of 0 and \overline{Q} . The second NOR has an input of 0 and Q . Assume that $Q = 1$ so that the second NOR then produces a 0 output, which then feeds the first NOR with both 0 inputs, and the output is 1, which is the original value of Q we assumed. This state is then stable and $Q = 1$ is maintained and regenerated.
- Now if we assume $Q = 0$, then the second NOR is fed with 0 on both inputs, so it produces a $\overline{Q} = 1$, which then produces a zero back on Q again (first NOR has at least one input of 1). So this state is also stable and regenerated.
- From this analysis, we construct the truth table of the SR latch. Note that the if $R = 1(S = 0)$, the latch resets. If $S = 1(R = 0)$, the latch sets. If both $S = R = 0$, the latch maintains its previous value. The state $R = S = 1$ is particularly useless and it should be avoided!

D Latch

- The SR latch leaves something to be desired. First it requires us to avoid the “bad input” $R = S = 1$. Note that although we labeled our outputs as complementary, if we have $R = S = 1$ the outputs are both 0 (not complements).
- The D latch solves this problem by always generating complementary inputs to the SR latch. We also “gate” the latch so that it only changes state when the CLK is high.
- This is very important because we can design circuits that only change states during a positive clock cycle, and during the 0 of the clock the circuit retains its state.

D Latch Truth Table

- The truth table for the D latch is easily constructed. When $CLK = 1$, we say the latch is *transparent*. When $CLK = 0$, we say the latch is *opaque*.

D Flip Flop (FF)

- If we connect two latches back to back, as shown, with the clock inversion between the first and second, we obtain a flip-flop (FF).
- Sometimes the terms flip-flop and latch are used interchangeably, but there is a distinction. A latch is transparent during a positive clock, whereas a FF is only transparent during a brief interval during the clock transition (edge). To see why, let's analyze the circuit.
- When clock is low, the first latch is in transparent mode and it's sampling the input. When the clock transitions to high, the first latch goes into opaque mode and the second latch reads the last value of the input right before the clock rising edge.
- Hence we say the D FF is a positive edge triggered FF. You can also design a negative edge triggered FF (homework!).

Registers

- A parallel bank of FF form a register bank. They are all clocked together and read in a word of data (or other bit widths) on the clock edge.

Enabled FF

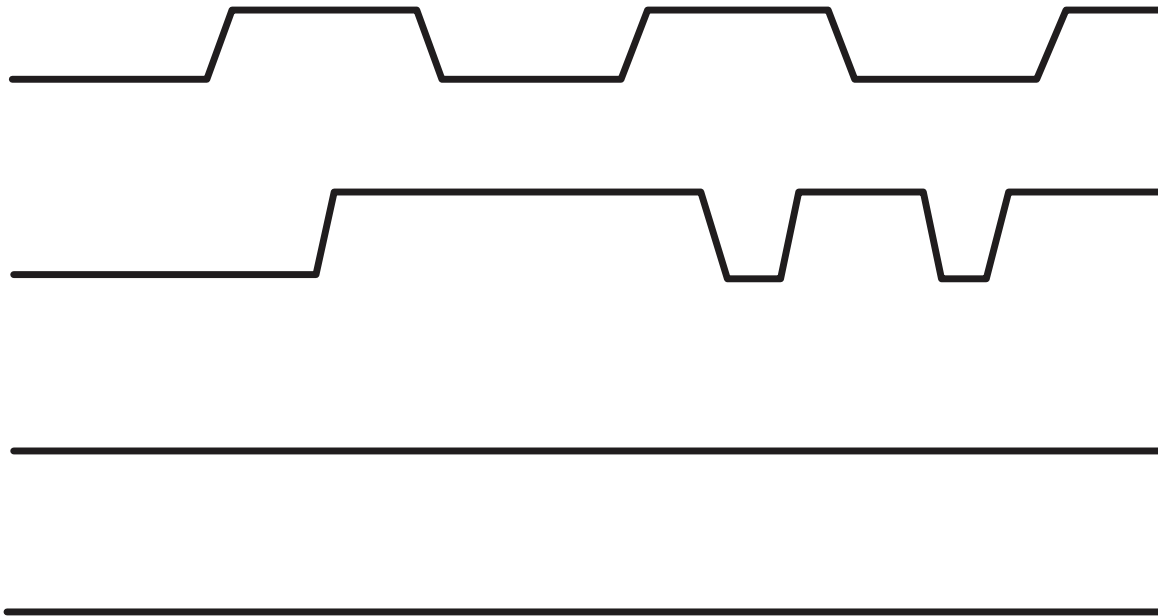
- Sometimes it's useful to enable a FF by adding an extra input. Then if the enable is low, the FF cannot read a new value. If enable is high, then the FF can read a value.
- This can be done by gating the clock with the enable signal, but gating the clock is a bad idea because it delays the clock, which means that the FF read is not perfect in sync with the clock (and perhaps the rest of the circuitry).
- A better solution is to MUX the input as shown.

Resettable FF

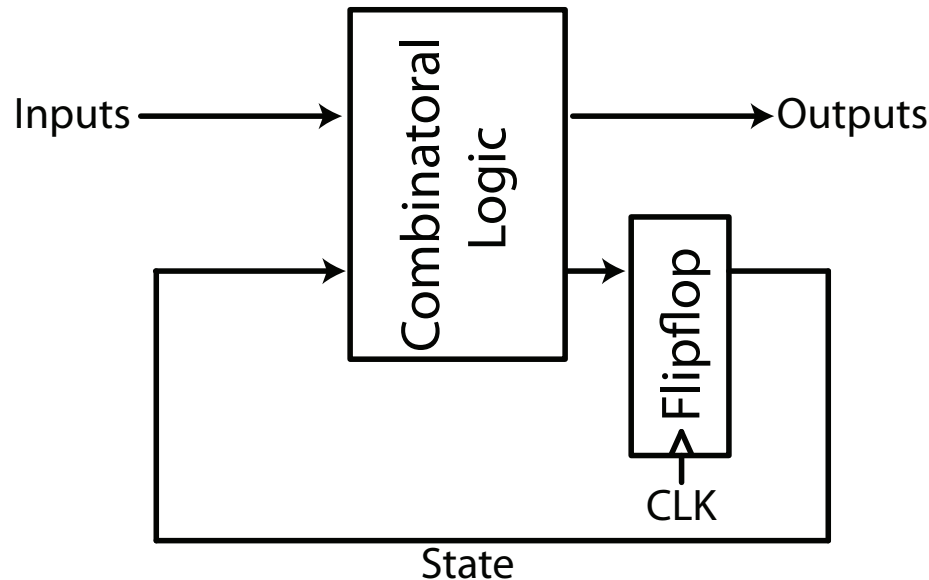
- A reset option is very handy if we want to put our FF into a known state. The reset signal as shown is active low, which then puts a 0 into the FF. This is an synchronous reset, meaning that the FF will be reset only on the rising edge of the next clock. We can also design an asynchronous reset (homework), which means that independent of the clock the FF is reset.

FF versus Latch

- The following timing diagram shows the difference between a latch and a FF. Note that the FF can only change on the positive edge of the clock.



Finite State Machine (FSM)



- A Finite State Machine (FSM) uses latches to hold the current state of the system. At each clock cycle, the output of the FSM depends on the current input and the current state, which is encoded by the registers.
- Examples of FSMs: A vending machine, traffic signals, elevator controls, automatic door opener (homework), ... , a CPU!

State Transition Diagram

- Consider a light controller using a touch interface. Basically there are three four light levels, OFF, LOW, MEDIUM, and HIGH. Every time we touch the circuit, it should cycle through the states.
- The above state transition diagram shows the basic functionality. Each circle represents a state of the system. The arrows leaving the circle have input labels. An arrow can point back to the same state if no transition is necessary.

Light Switch – More Functionality

- Suppose that it's dark and we have a simple light sensor (lab!). Then we can add another input to the system and light up the light control so that the user can find it in the dark.
- Here's the new diagram. Note that we only need to turn on the light when the light is in the OFF state.

State Transition Logic

- We need to now translate the state transition diagram into a truth table. This table has rows corresponding to the inputs and the current state, and the next state. It's a simple matter to fill this table.

Circuit Realization

- From the state transition table, we can write Boolean expressions and design the equivalent circuits. Note that the current state will be read from a register set implemented with flip-flops.