Chapter 10: Digital Electronics II – Sequential Logic

In static logic you present a set of inputs, e.g. a byte of data, and you get an output that depends on the input. It does not depend on a time ordering of events. In sequential logic, the output can depend on the time ordering of the inputs. This sequential ordering is based on propagation delays in the gates.

A. Propagation Delays

If the input to an inverter goes from Hi to Low, the output will go from Low to Hi, but it may take several nanoseconds for the output to change after the input is changed. It will look something like the drawing at the right if you show both traces on an oscilloscope. If each tick on the time scale is 5ns, this is about the delay you would expect from an ALSTTL gate, about 4 to 5 ns. The propagation delays may not be quite the same for the high to low transitions as for the low to high transitions, but they are usually similar. This allows one to make an oscillator, called a ring oscillator, by connecting three inverters together as shown in fig. 10.2. This connection represents a logical contradiction, but because of the propagation delays the system will oscillate at a frequency that has a period of about six propagation delays. This will work for any odd number of inverters greater than one.

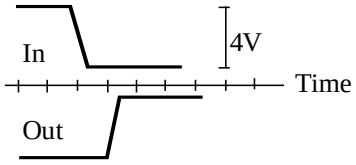


Fig. 10.1 Propagation Delays

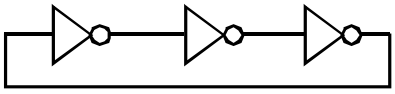


Fig. 10.2 Ring Oscillator

The period of the oscillation will be about $2n \times T_{p.d.}$, where $T_{p.d.}$ is the propagation delay for each device and n is the number of gates in the "ring". You can use this to measure the propagation delay. If the above oscillator has a frequency of 30MHz for three inverters, $T_{p.d.} = 5.5$ns.

These propagation delays limit how fast a logic system, e.g. a computer's CPU, can operate, and they are ultimately due to limitations on how fast a transistor can switch on and off.

B. SR Latch (flip-flop)

An SR latch is made of two NAND gates or two NOR gates. I will describe the operation of the NAND gate version. The diagram is shown at the right. Note that the top NAND gate is labeled 1 and the bottom is labeled 2. Also, the wire from the output of gate 1 to the input to gate 2 does not connect to the wire from the output of gate 2 that goes to the input to gate 1. The SR stands for a Set-Reset. The outputs are labeled Q and $\overline{Q}$ because they are "normally" inverses of each other. I usually call this a latch, but it is also called a flip-flop.

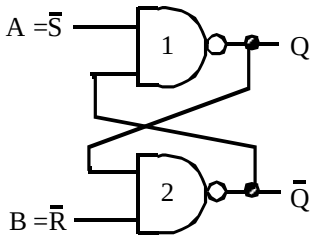


Fig. 10.3 An SR Latch (flipflop)

The interesting thing about an SR latch is that the state of the outputs can depend on what the inputs were, not just on what they are now. It has memory. If both inputs are 0, both outputs are 1. If A now becomes 1, Q = 0 but $\overline{Q}$ =1. If B then goes to 1, the outputs stay the same. However, if A goes to 0, then Q becomes 1 and $\overline{Q}$ goes to 0. Now if A goes to 1 again, there is no change. So if A=B=1, the output depends on which one of the two was last 0.

This is shown in the table at the right, where the inputs go through the sequence of values from top to bottom. If you should try to go from the A=0, B=0 state to the A=1, B=1 state, the output will depend on which input crossed the Low to Hi threshold first. Because of this, that transition is said to produce a "race"

A=$\overline{S}$	B=$\overline{R}$	Q	$\overline{Q}$
0	0	1	1
1	0	0	1
1	1	0	1
0	1	1	0
1	1	1	0

condition and the output is not "predictable". The SR latch usually operates by transitions among the last four rows of the table above. The A=B=1 condition is called the latched condition because the output holds the result of the previous state of the inputs. In a sense this is the basis of memory, and static random access memory (static RAM or SRAM) typically uses cross coupled transistors that mimic this behavior.

One can make a gated SR latch and ultimately a gated data latch, or a gated D latch. The gated SR latch is shown in fig. 10.4 and the gated data latch in 10.5.

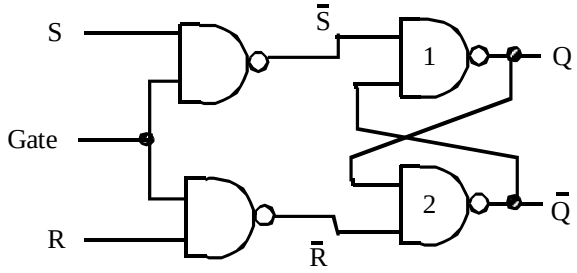


Fig. 10.4 Gated SR Latch

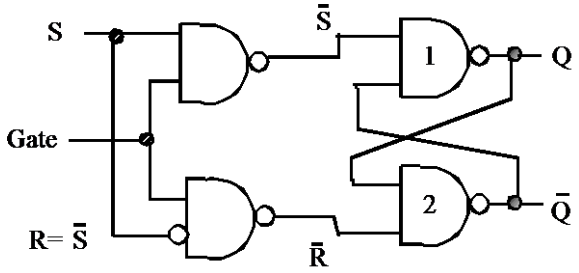


Fig. 10.5 Gated Data Latch

The gated SR latch operates just like the SR latch when the Gate = G = 1, except the S and R inputs are inverted to $\overline{S}$ and $\overline{R}$. When the Gate goes to 0, $\overline{S}$ and $\overline{R}$ are both forced to 1 and the latch is "latched". The difficulty is that if S and R both equal 1 just before G goes to 0, $\overline{S}$ and $\overline{R}$ are both 0 and when G goes to 0 there is a race condition. The output depends on which one of $\overline{S}$ and $\overline{R}$ gets to 1 first. The Gated D Latch solves this problem by making the R input the inverse of S so that they cannot both be 1 at the same time. (The bubble at the input to the NAND gate indicated the inversion.) The effect of this is to transfer the signal at S into Q when

the gate is 1 and to latch that when the gate goes to 0. This "saves" the value of S in Q, i.e. Q equals what S was the instant before the gate went to 0. That is why this is called a Gated Data Latch. Often a Gated D Latch is given the symbol shown at the right, with just the D = S input, the gate and the Q's shown as connections. However this stands for the circuit shown in fig. 10.5. Sometimes the gate is called a clock, or CK, since it is often connected to a clock line. The Gate, G, and Data, D, are inputs to

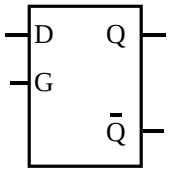


Fig. 10.6 Gated D Latch

the latch. The Q's are outputs. When G = 1, the value at D is transferred to Q, and the latch is said to be transparent, since changes at the D input go though to Q. When the gate or clock goes to 0 the latch is latched and Q retains the value D had just before the clock went to 0.

Two gated D latches can be cascaded to form a master-slave flip-flop as shown at the right. Note that the clock for latch 1, CK1 is the inverse of CK2. The two inputs are D_{in}, or data input, and the master clock. Changes at CK1 occur one propagation delay after the change at CK2 because of the inverter.

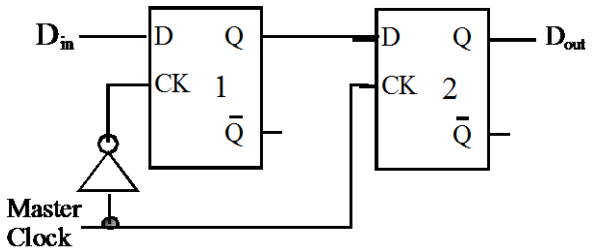


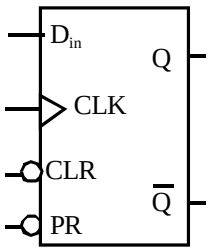
Fig. 10.7 Master-Slave Flip-flop

1. If the master clock is <u>initially</u> high, latch 1 is latched (CK1 = Gate 1 =0). Changes at D_{in} will not transfer through to D_{out} = Q1.
2. When the master clock goes to 0, the second latch is latched (CK2 goes to 0) so that D_{out} still does not change, but retains the value Q1 = D2 <u>had</u> as the clock went to 0.
3. A propagation delay later, CK1 goes to 1 and the first latch becomes transparent and the Q1 = D_{in}. As long as the master clock stays 0, no changes occur at D_{out}.
4. When the master clock goes back Hi, CK2 goes to 1 before CK1 and latch 2 becomes transparent; so D2=Q1=D_{in} is transferred to D_{out} and now D_{out} changes to whatever D_{in} was just as the clock was going Hi. (Technically, it changes to what D_{in} was just before the clock went to 1.)
5. A propagation delay later, CK1 goes Low and the first latch latches, and Q1 is fixed which implies D_{out} does not change after this until the master clock goes from 0 to 1 again.

The net effect is that when the master clock goes from 0 to 1, D_{out} becomes what D_{in} was just before the master clock changed, and D_{out} will retain this value until the master clock goes from 0 to 1 again. It is the CHANGE from 0 to 1 in the master clock that transfers D_{in} to D_{out}. The rest of the time D_{out} is stable or latched. This type of behavior is called edge triggering and such a device is called an edge triggered flip-flop. This type of flip-flop would be called a positive edge triggered D flip-flop because it is on the positive going pulse, 0 → 1, that the changes occur.

C. Edge Triggered D Flip-Flop

Commercial edge triggered D flip-flops are not made this way, but they function in this manner. They also have a couple of other features. They have a Clear function that sets Q to 0 when the Clear pin is taken Low or to 0. They also have a preset which will set Q to 1 when the preset is 0. The "symbol" for such a device is show at the right. The lines coming out represent possible external connections. The bubbles at the CLR and PR inputs (Clear and Preset) indicate that they are active Low or at logic 0. The triangle at the clock input indicated that the device is edge triggered. A 7474, 74LS74 or 74ALS74 is a TTL IC that contains two positive edge triggered flip-flops like the one in fig. 10.8.



10.8 Edge Triggered D Flip-flop

A common use of a flip-flop like this is to detect if something has occurred. For instance, if you are trying to detect a particle moving past a detector and the particle goes by very rapidly, i.e. in a very short time, your computer needs a signal that something has happened so it will go and check to see just what happened. If the only signal is a Hi signal when the particle passes through, the computer may not be looking at the device at that instant, so you need some electronics that with register the passage with a Hi signal that STAYS Hi until the computer sees that signal and then clears it when it is through checking what happens. A flip-flop can do just

this. The circuit at the right will set Q Hi when the short pulse from the passage of the particle past the detector reaches the Clock line. This Q will signal the computer that the event has occurred, perhaps via and interrupt, and when the computer has finished its work, it can reset, or clear, the flip-flop by taking the CLR line low and then Hi again. Note that the preset (PR) pin must be held Hi. Do not leave it unconnected or it could "accidentally" be taken Low if an adjacent pin has a signal that changes from Hi to Low. (There is a little capacitance between the pins and this AC coupling can make the PR pin go Low if it is left unconnected.)

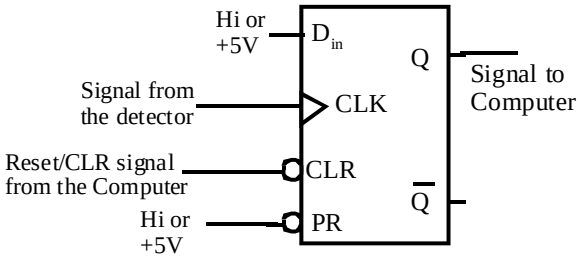


fig. 10.9

One of the most common uses of flip-flops is counting. This is achieved by connecting the $\overline{Q}$ output back to the Data input, D_{in}. (Of course the Preset and Clear should be held Hi unless you want to clear or preset the flip-flop.) In this case, the Q of the flip-flop with change or toggle every time the clock input goes from Low to Hi. If you connect the Q of the first flip-flop to the Clock input of a second flip-flop that is similarly configured you will notice that the sequence of clock pulses at the first flip-flop causes the outputs to count down from 3, assuming they start with both Q's initially 0. In this case Q_1 and Q_2 will follow the sequence shown at the right. If Q_1 is the LSB and Q_2 is the MSB then the count sequence is 3→2→1→0 , or 11, 10, 01, 00 in binary. The sequence will repeat if the pulses continue to come in. (Note that in fig. 10.10 the two lines that cross are not connected – there is no dot to indicate an electrical connection!)

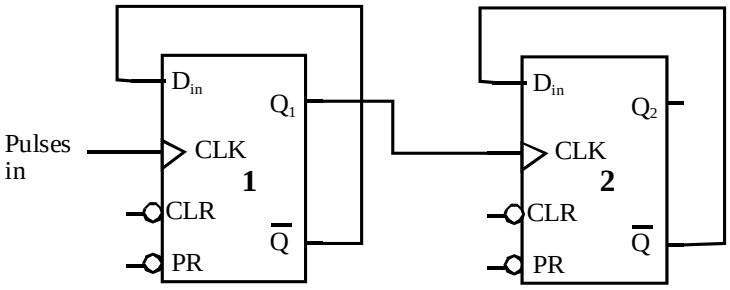


fig. 10.10 Two-bit Down Counter

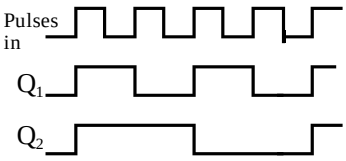


fig. 10.11 Count Sequence

Note that the propagation delays in transferring D_{in} to Q mean that Q and $\overline{Q}$ will not change until a few nanoseconds after the clock rises. As a result it is the value of $\overline{Q}$ just before the clock rises that gets transferred into Q when the clock rises. The "new" $\overline{Q}$ does not appear at D_{in} until a few nanoseconds after the clock has risen and the internal latches have latched.

One can make this into a three bit down counter by connecting a third D flip-flop with its $\overline{Q}$ connected to its D input and its clock connected to the Q_2 output from flip-flop 2. By connecting n D flip-flops this way, you can make an n bit down counter. If you want to turn it into an up counter, where the count sequence is 0, 1, 2,…, you can simply connect the $\overline{Q}$ output of the previous stage to the clock input instead of the Q output. You should verify that this works. For an n bit counter, the sequence repeats after 2^n clock pulses.

This type of counter is called a ripple counter because the second flip-flop clock pulse comes after the pulse to the first flip-flop. The Q_1 output typically changes a few propagation delays after the pulse

that produced it. (For a 74LS74 the typical delay is 13ns to go High and 25ns if the output is going Low.) This means that the changes in the outputs "ripple" down the chain of flip-flops, each one occurring slightly after the previous one. If you have an eight bit counter going from the state 1111 1111 to the state 0000 0000, the eighth flip-flop will change about 200ns after the clock pulse that originally started the change at the clock input to flip-flop one. This may not always be acceptable behavior. If the pulses coming in have a frequency of 10MHz, two new pulses would have come into flip-flop one before the output of flip-flop eight changed state in response to the original pulse 200ns earlier. If you are trying to read the count on the flip-flops, you want all of them to display the result simultaneously, not at different times. otherwise you might not read the correct count. The way around this is to make counters from flip-flops that all change on the SAME clock pulse. These are called synchronous counters. You can make these from a slightly different type of flip-flop, a JK flip-flop.

D. JK Flip-Flops

A JK flip flop is shown at the right. Instead of a D input, its has two other inputs, J and K. The bubble at the clock indicates that it is triggered on the negative or falling edge of the clock input. Usually there are Clear and Preset inputs as well, but I haven't shown them. They are usually active Low, so you would have to tie them High to inactivate them. The truth tables for this flip-flop and the D flip-flop are shown below. This is what you would see on a Data Sheet for these devices.

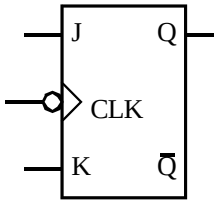


Fig. 10.12 JK Flip-Flop

D Flip-Flop

Preset	Clear	Clock	D	Q	$\overline{Q}$
L	H	X	X	H	L
H	L	X	X	L	H
L	L	X	X	H*	H*
H	H	↑	H	H	L
H	H	↑	L	L	H
H	H	n.c.	X	Q_o	$\overline{Q}_o$

JK Flip-Flop

Preset	Clear	Clock	J	K	Q	$\overline{Q}$
L	H	X	X		H	L
H	L	X	X		L	H
L	L	X	X		H*	H*
H	H	↓	L	L	Q_o	$\overline{Q}_o$
H	H	↓	H	L	H	L
H	H	↓	L	H	L	H
H	H	↓	H	H	Toggle	Toggle
H	H	n.c.	X		Q_o	$\overline{Q}_o$

H stands for High, L for Low, X means that it doesn't matter what the state of the input is, n.c. means no change, a rising arrow ↑ indicates are rising edge or Low to High transition, and ↓ indicates a

falling edge. Q_o means that there is no change in the state of Q from its previous value, much like n.c. Toggle means that it changes from what it was, e.g. if it was High, it will become Low. The H* indicates that the state is unstable. Preset and Clear both Low forces both Q and $\overline{Q}$ High, but they are unstable and will change if both Preset and Clear go High. (It will be a "race" condition.)

The figure at the right shows two JK flip-flops connected to form a synchronous two bit up counter. The basic idea is that the first flip-flop (1) will toggle with each negative clock edge, but the second one will only toggle if the first one's Q is High. If they start out in the 00_b state, the first negative edge of the clock will make flip-flop 1 toggle so the count is 01_b. The next negative edge makes both flip-flops toggle so the state goes to 10_b. The next negative edge makes the first one, but not the second one toggle to 11_b. The fourth negative edge makes both toggle to the 00_b state again and the count starts over. Of course it is assumed that the Preset and Clear inputs are tied High so that they are inactive, so they are not shown. Showing all the connections, e.g. the power supplies, makes the figure too cluttered.

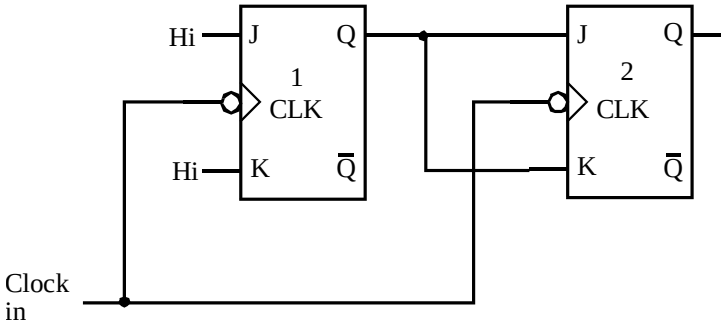


Fig. 10.13 Synchronous Two Bit Counter

JK flip-flops are more versatile than D flip-flops, but they are slightly more complicated to use. However, they are a good test of how logically you can think. Typically you can find single chips that have many of these flip-flops inside configured as various types of counters. You usually use those if you really need more than a two bit counter. Chips containing two negative edge triggered JK flip-flops are 74LS112, 74LS113 and 74LS114.

E. Shift Registers

Another common use of flip-flops is as a shift register. Shift registers are often used to convert a data word (e.g. 16 bits in parallel) to a serial sequence of bits for serial data transmission. They are also used for serial to parallel conversions. In a computer data is normally transmitted as a set of bits on parallel "wires" or lines called the data bus. Data busses usually consist of 8, 16, 32 or 64 parallel lines. A 16-bit bus would transfer a 16 bits of information at a time. One line would carry the LSB, another line carries the next most LSB etc. up to the MSB. A 16-bit bus would transfer two bytes every clock cycle. (Many devices take more clock cycles to transfer a byte of word of data, especially if the memory cannot operate as fast as the processor.) Since all 16 bits are transferred simultaneously in one cycle, we say they are transferred in parallel. This means that the bus must have 16 separate lines to do this. When transferring data over long distances or outside the computer, it is often inconvenient to have that many lines connecting the two devices. As a result the data is often transferred one bit at a time. In effect one bit per clock cycle. This only requires one data line plus a ground or common. This process needs a device that will convert a 16 bit number into a sequence of data bits to be transferred one per cycle and a device that will receive these and convert them back into parallel data. A shift register can do both of these tasks.

The device at the right shows a three bit shift register consisting of three edge triggered D flip-flops, labeled A, B and C. The Q of A is connected to the D of B and the Q of B is connected to the D of C. It is assumed that the Clears and Presets are Inactive (tied Hi). The value of D_{in} at the rising edge of the clock is transferred to Q_A. At that time the "previous" value of Q_A that it had just before the clock went Hi is transferred to Q_B and the previous value of Q_B is transferred into Q_C. As a result, if the serial sequence of values at D_{in} is 1 then 1 then 0 at the beginning of each clock cycle, the values in (Q_A, Q_B, Q_C) are (011) after three clock cycles. These can now be transferred in parallel onto a data bus. Of course an actual shift register that does this would be 8, 16 or 32 bits long instead of 3, but the idea is the same. (It takes too much space to draw 8 of them.) Propagation delays insure that the Q_A that gets transferred into Q_B is the "previous" value of D_{in}, since it takes a couple of propagation delays for the value at D to be transferred to Q in a flip-flop.