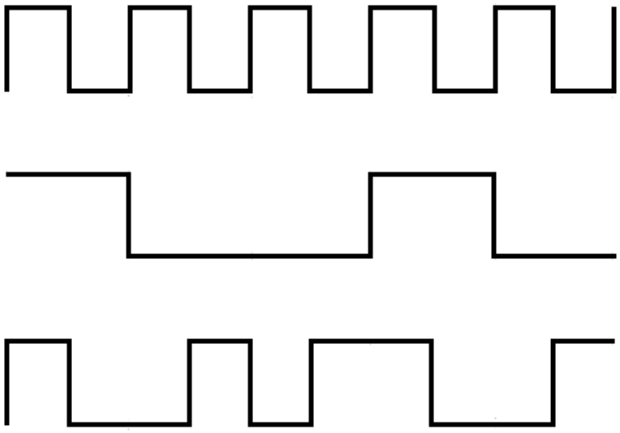
adafruit learning system

Digital Circuits 4: Sequential Circuits

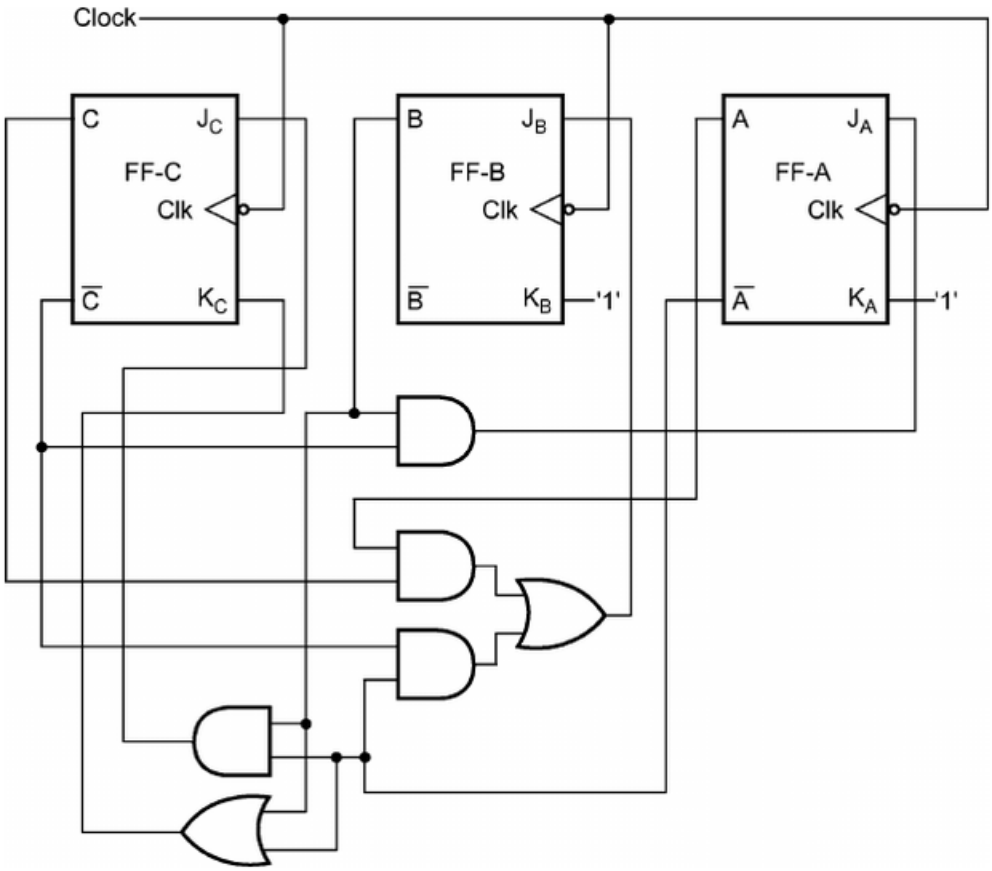
Created by Dave Astels



Last updated on 2018-08-22 04:07:00 PM UTC

Guide Contents

Overview



Sequential Circuits

In the last part (https://adafru.it/BJm) we saw that combinational circuits are combinations of logic gates that operate in a fully functional manner (meaning that for a given configuration of inputs, there is a corresponding set of outputs which always result from those inputs). Notably, the circuit has no state, it always works the same way.

Sequential circuits, on the other hand, do have state. They typically have an input (or inputs) that can cause the state to change.

Onward

Similar to combinational logic, we'll start with the building blocks of sequential logic: the flip-flop in its various forms. Once we go over the basics we'll look at some ways we can use them in larger circuits.

Flip-Flops

Flip-flops are the basic piece of sequential logic. They effectively store a single binary digit of state. There are a variety of flip-flops available that differ on how that state is manipulated.

Since a flip-flop stores a binary digit it must, by definition, have 2 states. Furthermore it is bistable, which means it is stable in each state: when is put in a specific state, it will stay in that state until something causes it to change to the other state.

R-S Flip Flop

This Flip-Flop has two inputs that change its state: **R**eset and **S**et.

When R goes low, Q goes low and /Q goes high.

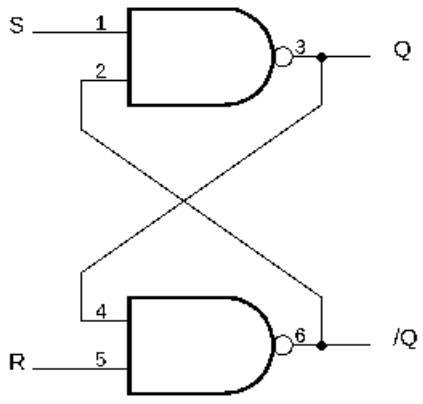
When S goes low, Q goes high and /Q goes low.

When both R and S are high the Flip-Flop is stable and doesn't change.

R and S can not be both low at the same time. Both Q and /Q would be high... something about ripped space-time and imploding realities...

Operation	S	R	Q_{n+1}
Stable	1	1	Q_n
Set	0	1	1
Reset	1	0	0
Forbidden	0	0	–

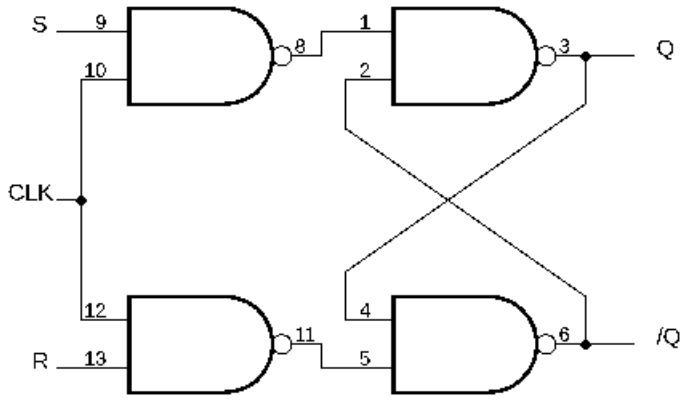
An R-S Flip-Flop is simple. We can build one from half of a 7400 NAND gate chip. The circuit below shows how. Go ahead, build it and play with it. I'll wait. The pulse input and LED output boards from part 2 (https://adafru.it/BJI) are handy for providing pulse input and watching the output of this, but you can hand wire some push-buttons and LEDs (with resistors!). Note that debouncing isn't a concern in this application, since multiple low input pulses have no effect, it's just the first one that counts. Using the I/O boards just means less wiring to do for a quick experiment.



Both R and S are, by default, high. If R goes low, the output of it's NAND gate goes high regardless of the other input (attached to Q). So /Q is high. That means that both inputs of the other NAND gate are high (since S will be high), making it's output (Q) low. That low value feeds back to the R gate so that when R returns to it's resting high value, it's gate's output stays high. If S goes low, the opposite happens with Q going high, making /Q low (since R is also high). The low /Q will hold the output of the S gate high.

Since things happen with R or S are low, they are *active-low* inputs. If we wanted the opposite, i.e. active-high inputs, an inverter would be added to each input.

This Flip-Flop is a sequential circuit. It is not, however, a synchronous circuit. The state change happens whenever R or S go low. We can make it synchronous quite simply. What we have to do is *gate* R and S using another input: the clock. We can use the other two NAND gates for this.



When the CLK input is low, both of the gating NANDs output high. When CLK is high, the S and R inputs are passed through the gate NANDs and inverted. So if S is high, the set input to the Flip-Flop will be low, setting it. Similarly with the R input.

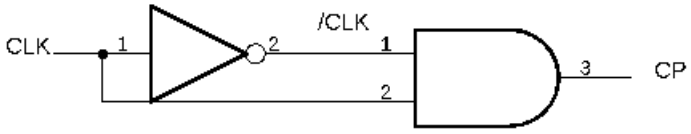
Note that because of the inversion done by the gating NANDs, the inputs to the circuit as a whole are now *active-high*.

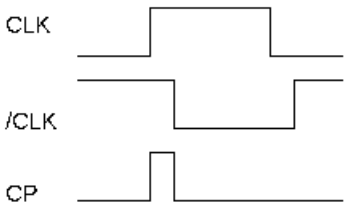
Level vs Edge Triggered

The above clocked R-S Flip-Flop is level triggered; this the CLK input being high that is relevant. As long as CLK is high, the R and S inputs can change the state of the Flip-Flop. Sometimes this is fine, but often we want that window of change to be limited to the instant CLK transitions from low to high or as close as possible to it. If CLK stays high, R and S have no effect after the initial change of CLK. This is what we mean when we say the circuit is *edge triggered*. It's the rising edge of CLK that is relevant. Specifically, it's the values of R and S at the instant the rising edge occurs.

Rising/positive Edge Detector

We can convert the above level triggered circuit into an edge triggered one with an inverter and an AND gate.





Remember when we first talked about gates. There was mention of the fact that for a gate it takes some small amount of time for the output to respond to a change in inputs(s). That's usually seen as a bad thing, and much work has been

done to make this time smaller and smaller (one result of this work is faster computers). This circuit takes advantage of the small delay in the inverter. When CLK goes high, it takes one input of the AND gate high. Now it takes that small amount of time for the output of the inverter to catch up and switch to low (now that its input is high). In that brief time, both inputs of the AND gate are high, and so it's output goes high. When the inverter catches up its output goes low, and so does the output of the AND gate. The result? A brief high pulse on the output of the AND gate whenever the CLK signal switch from low to high.

If we now slap this edge detector on the CLK input of the level triggered Flip-Flop, its R and S inputs are used to effect its state during that brief pulse. We now have an edge triggered R-S Flip-Flop.

Falling/negative edge detector

If we look at the timing diagram above, we see that the pulse is high for the short time both CLK and /CLK are high. A similar thing happens when CLK switches back to low. The /CLK signal lags slightly so there is a period of time when both are low. If we switch the AND gate for a NOR gate we get a short pulse then. Thus we can make a falling edge detector as well. Given that, we have the choice of triggering out R-S Flip-Flop on the rising or falling edge of CLK.

J-K Flip-Flop

Firstly, it's clocked.

Secondly, instead of the case where both inputs are active being illegal, it causes the state to toggle.

Operation	*Clock*	J	K	Q_{n+1}
Stable	⌐_	1	1	Q_n
Set	⌐_	0	1	1
Reset	⌐_	1	0	0
Toggle	⌐_	0	0	$\overline{Q_n}$

Set and Clear

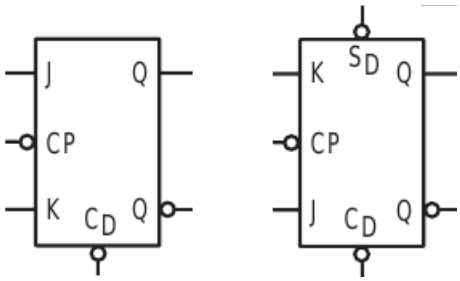
It's common for a flip-flop to have a way to set Q to either high or low independent of all other inputs. Set will make Q high (and /Q low) regardless of whatever else might be going on. Conversely Clear will set Q to low (and /Q to high). Having Set and Clear both active at the same time is not a valid situation... that thing about holes in space-time and imploding realities...

$\overline{Set}$	$\overline{Clear}$	*Clock*	J	K	Q_{n+1}	$\overline{Q_{n+1}}$
0	1	X	X	X	1	0
1	0	X	X	X	0	1
0	0	X	X	X	–	–
1	1	⌐_	0	0	Q_n	$\overline{Q_n}$
1	1	⌐_	1	0	1	0
1	1	⌐_	0	1	0	1
1	1	⌐_	1	1	$\overline{Q_n}$	Q_n

One way to think about Set and Clear is as a reset that puts the flip-flop into a known initial condition (0 or 1 depending on which signal is used). It can be used and abused in other ways as well, typically to reset a group of flip-flops when certain conditions occur.

As these flip-flops get more complex, we seldom draw out the gate level circuit. Also, flip-flops are easily available packaged into ICs so it is natural to drop them into a design as a unit. The 7473A and 7476A are two example of J-K

flip-flops. the '73 has a clear input, while the '76 has set and clear. Be careful of the '73... it has power & ground pins in unusual locations. The 74107 has the same functionality with power and ground in the usual locations.



The Universal Flip-Flop

The J-K flip-flop has the distinction that it can be used to construct any other flip-flop, much like NAND gates can be used to construct any other type of gate (and by extension, any digital circuit). Because of this, the J-K is sometimes called a *universal flip-flop*.

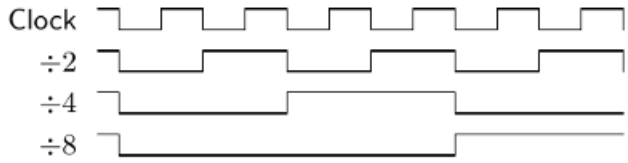
T Flip-Flop

This is a simple one. On the active edge of the T input (rising or falling) the flip-flop's state and Q (and /Q) output toggles.

T	Q_n	Q_{n+1}
⌐_	0	1
⌐_	1	0

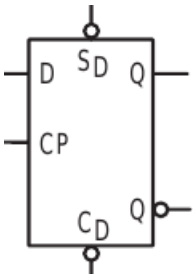
The T flip-flop isn't usually found as is. You can easily make one using a J-K and making both J and K active: connect them to Vcc if they are active high, or to ground if they are active low. In fact, that's usually how you'll find a T flip-flop.

One neat thing about a T flip-flop is that it will divide whatever clock signal you apply to it by 2. You can chain several T flip-flops together to divide the incoming clock by 2, 4, 8, etc.



D Flip-Flop

The D flip-flop is basically a single bit storage cell. In this respect it is little different than any of the other flip-flops we've looked at; it is differentiated by its simplicity. It has a single input D that is used to set the state on the appropriate clock edge. As usual, Q and /Q reflect that state. That's all there is to it. The 7474 is the canonical D flip-flop.



It provides Set and Clear inputs as described above, and otherwise the state of the flip-flop only changes on the falling clock edge.

$\overline{Set}$	$\overline{Clear}$	$Clock$	D	Q_{n+1}	$\overline{Q_{n+1}}$
0	1	X	X	1	0
1	0	X	X	0	1
0	0	X	X	–	–
1	1	0	X	Q_n	$\overline{Q_n}$
1	1	1	X	Q_n	$\overline{Q_n}$
1	1	⌝⌞	0	0	1
1	1	⌝⌟	1	1	0

Continuing

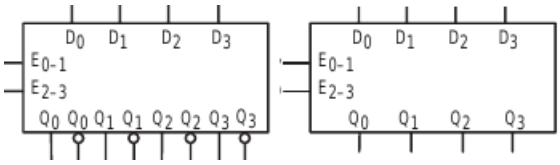
Now that we know all about flip-flops, we can look at some things we can use them for.

Latches

Latches are a lot like D flip-flops, except that instead of a clock they have an enable. When the enable is active, whatever is on the D input is transferred to the internal state and the Q output. When the enable goes inactive, the state will remain what it was the last instant enable was active and the value of D has no effect. So it's very fair to say that data is latched in by the enable.

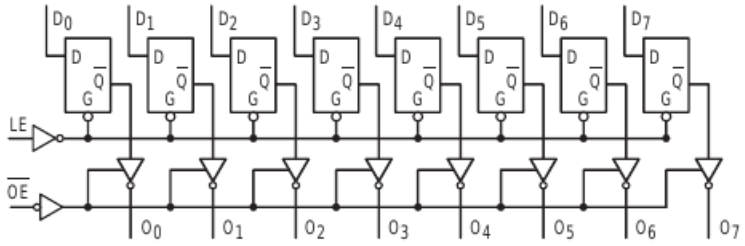
$\overline{E}$	D	Q_{n+1}	$\overline{Q_{n+1}}$
1	X	Q_n	$\overline{Q_n}$
0	0	0	1
0	1	1	0

Some latches omit the /Q outputs as they are not generally required when a latch is needed. Also, omitting them allows the use of a smaller IC package. The 7475 and 7477 are a classic example of this. Each contains 4 single-bit D latches. Notice that enable inputs are shared between pairs of latches.



Wider Latches

We've seen a 8-bit latch in part 2 on the logic level input board (https://adafru.it/BJx). On that board a 74373 was used to grab the switch values when the latch button was pressed. That button activated the enable input of the latch. Below is the internal structure of the '373. Note the 8 D latches, and the common active high LE (aka latch enable).



Tri-state Outputs

There's one more thing to notice that we haven't talked about yet. Look at those odd inverters on the outputs. They each have a second input. Wait? Wut?

These are *tri-state* inverters: in addition to the 0 and 1 states we are familiar with, their outputs have a third state.

A logic 0 is essentially connected to ground (through a transistor of some sort usually) and a logic 1 is similarly connected to Vcc. The third state (sometimes called high-impedance) is essentially when the output isn't connected to anything. As such it has no impact on anything else connected to it. This is especially useful when you have a bus with multiple data sources connected to it. Only one can put data on the bus at any time; all the others need their outputs to be in high-impedance mode, i.e. unconnected. In our input board we don't need to worry about that situation so the /OE (output enable) input is connected to ground, enabling the outputs permanently.

We've also seen tri-states on the output board (https://adafru.it/BJy) where we buffered the signals using a 74244. We just used it as a buffer to isolate the circuit being observed from the LEDs, but each buffer is tri-state, controlled in

groups of 4.

Shift Registers

Parallel: all the bits at the same time.

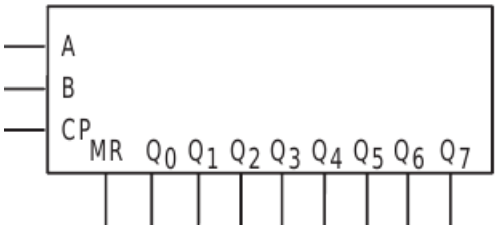
Serial: one bit at a time.

A shift register typically is used to convert between parallel and serial, in one direction or another. Either data is loaded in parallel and shifted out one bit at a time, or it's shifted in one at a time and output in parallel. Some shift registers can do both.

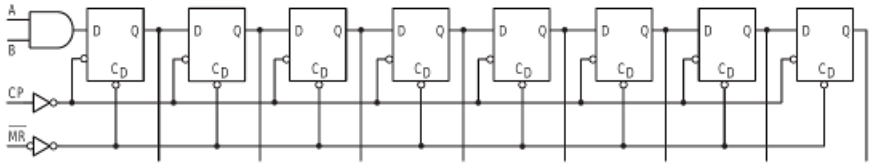
Serial-in Parallel-out

A bit is shifted into the register on each clock pulse. The values that have been shifted in are available on the outputs. Often (but not always) the register will be filled (i.e. 8 bits shifted into an 8-bit register) before the output is used.

The 74164 is a register like this. Instead on a single data input, it has two which are ANDed together internally. The result of that AND is shifted into the register.

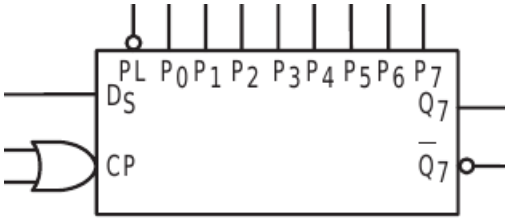


It's instructive to look at the internals of this chip. As we see below, it's simply a series of D flip-flops. That change state synchronously, aka they all change state at the same time in response to a common clock signal.

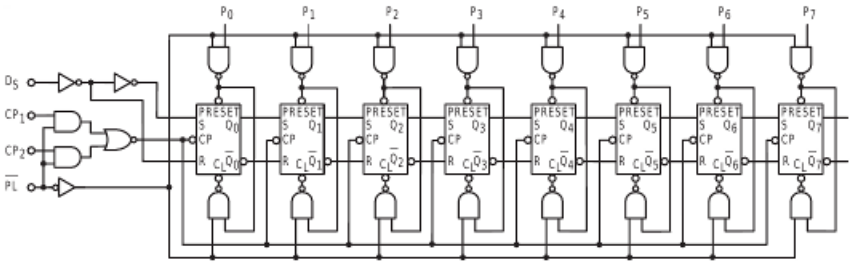


Parallel-in, Serial-out

The 74165 is the other case: parallel in, serial out. Notice it also has the ability to shift serial data in. It accepts 2 clock inputs that are ORed; when either makes the transition from 0 to 1 while the other remains at 0 the register is clocked, the value on Ds is shifted in, while the next value in line is shifted out. When /PL is low the clock is ignored and the values of P0-P7 and loaded into the internal flip-flops. Those values will then get shifted out on the next 8 rising clock edges.

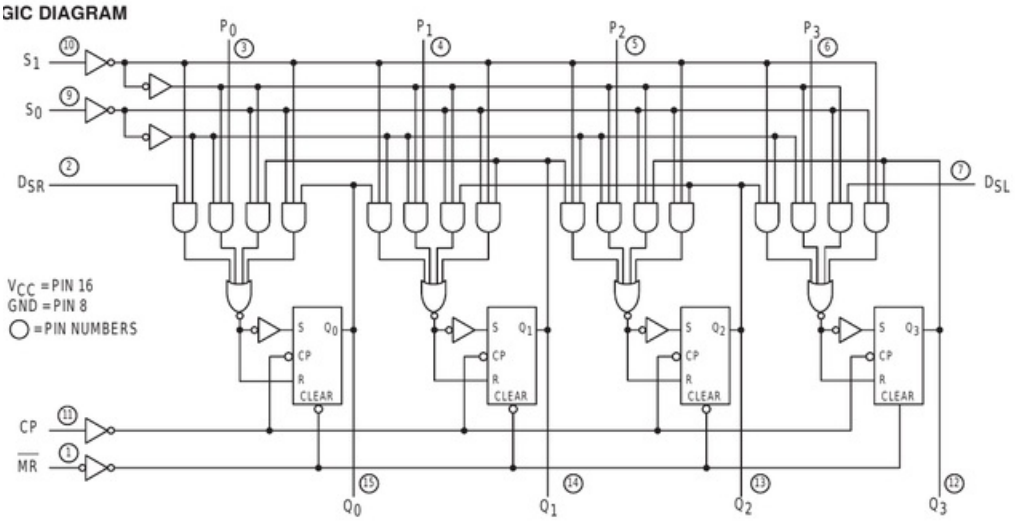


The internal details of the '165 are shown below. Note how the R-S flip-flops are chained Q->S and /Q->R, as well as how the Ds input and it's inverted form are used to drive the R and S of the initial flip-flop. See how the /PL input is used to disable the clock signals by gating them with AND gates. Finally, see how the parallel data is loaded into the flip-flops by utilising their Set and Clear inputs. It's an interesting bit of design. Be glad it's available on a single IC so you don't need to build it each time. This was/is commonly done for useful circuits.



Other configurations

Beyond these two basic types of shift register we get into various combinations of parallel-in, parallel-out, serial-in, and serial-out. Some can shift in both directions. The 74194 is a prime example: a bidirectional universal 4-bit shift register. As expected, it's fairly complex inside.



The P inputs are used to load parallel data, Q outputs are the parallel outputs. Dsr and Dsl are serial input from the right and left, respectively (allowing easy chaining to build wider shift registers). Q0 and Q3 double as serial out signals to the right and left, respectively. /MR resets all the states to 0. CP is the rising edge clock. Finally, The S inputs select how the register operates.

S_1	S_0	$Mode$
0	0	hold
0	1	shift right
1	0	shift left
1	1	parallel load

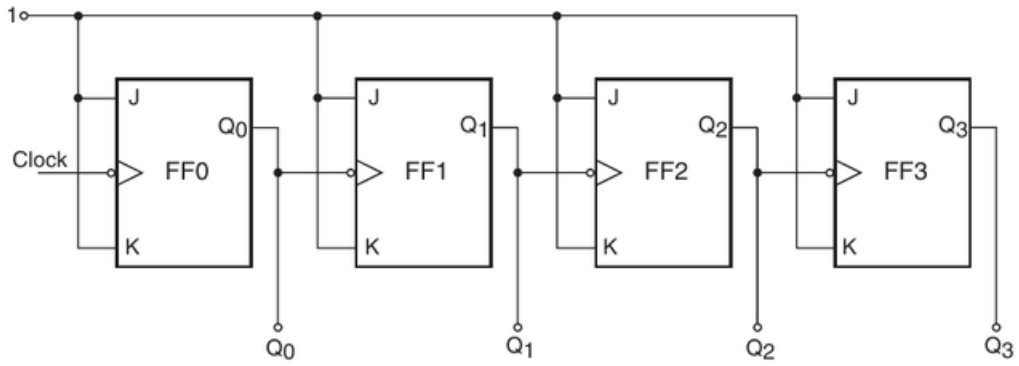
Regardless of which mode is selected, action only happens on the rising edge of CP.

Counters

Counters are a fundamental class of sequential circuitry. There can be used anywhere a series of steps is required. E.g. to drive a hardware state machine.

Ripple Counters

Unlike shift registers that move bits from one flip-flop to another, counters go through a sequence of numbered states; a 4-bit counter will count 0000, 0001, 0010, 0011, 0100, 0110, 0111, and so on. I.e. they count. The simplest way to accomplish this is to chain T flip-flops together:



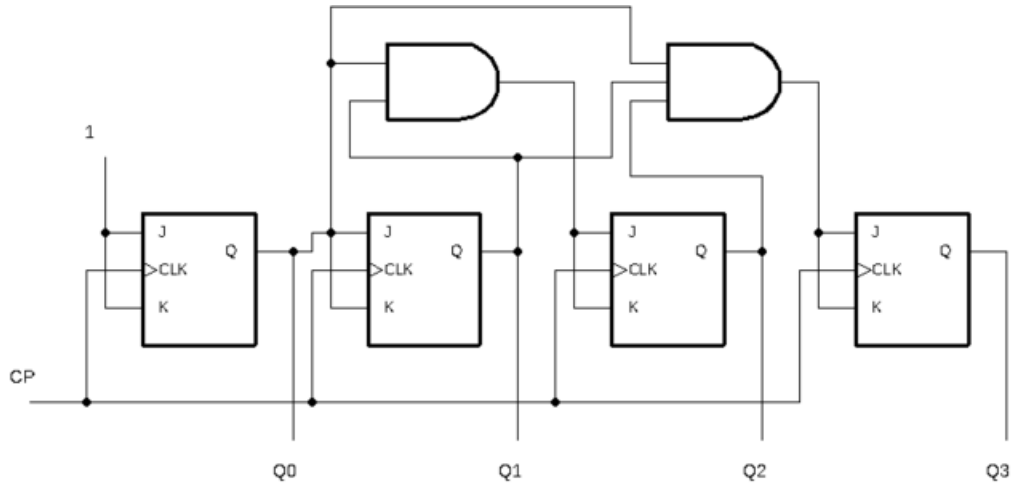
This will count from 0000 to 1111, and loop back to 0000.

The problem with this design is that each flip-flop, like all digital circuits, has a delay from the active edge of the clock to its output changing. So each clock pulse coming in causes a change that ripples through the chain of flip-flops. This is why we call these *ripple counters*. Observed on a human time scale, this ripple looks instantaneous, but at the circuitry level, those delays could have unpleasant effects since the values on all the outputs are not consistent until the final one has updated. They are simple, but the ripple effect can be problematic, especially as the chain of flip-flops gets longer.

That said, they are simple and effective. In cases where the ripple effect isn't a problem, they can be a good solution.

Synchronous Counters

To avoid the latency inherent in the design of a ripple counter, we need to have all the flip-flops update at the same time. That means having them all use the same clock signal. As usual, solving a problem isn't without cost. In this case (indeed in many cases in digital circuit design) this takes the form of more circuitry. Since all flip-flops are being clocked at the same time, rather than the clock rippling through, we need to add some logic to control when each flip-flop toggles. Below is a 4-bit synchronous counter. Compare it to the 4-bit ripple counter above.

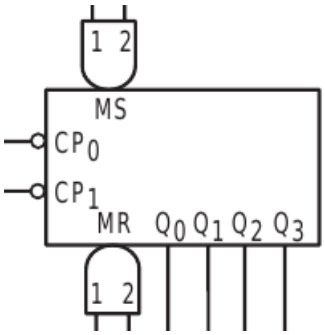


Each flip-flop only toggles when all the flip-flops to the left (i.e. lower place values) have a state of 1.

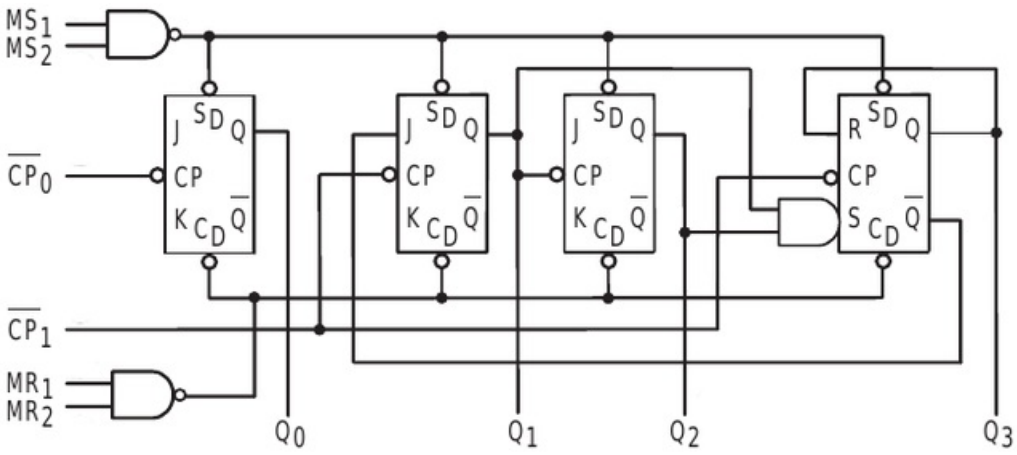
Counting to different values

Counting in binary is great, but sometimes we just don't want to count to a power of two before looping. We may want to count 0-9 and repeat. We know that will need 4 bits, but we need to go back to zero before the counter naturally would.

The 7490 is one counter that we can use to make a 0-9 counter. The logic symbol makes it look simple enough.



There are a couple each of active high set (MS) and reset/clear (MR) inputs that are ANDed (e.g. both MR inputs have to be high to clear the counter). As expected, there are 4 Q outputs. The odd bit is the two clock inputs. We need to take a look inside to see what that's all about.



As you can see, the set and reset connects to all 4 flip-flops. But other than that, the first/left-most/lowest-place flip-flop is separate from the other three. To make a 0-9 counter, Q0 is connected to /CP1, and /CP0 is the clock input. Notice that some of the J and K inputs aren't connected in the diagram; they can be assumed to be always at logic 1.

The final interesting thing is that the rightmost flip-flop is an R-S, unlike the other J-Ks. And it's clock is syncronous with the Q1 flip-flop, while clocks to the Q1 and Q2 flip-flops ripple (assuming Q0 is connected to /CP1).

Exercise

Assuming all flip-flops start with a 0 state, trace through the circuit and figure out what happens on each of the next 10 clock pulses. Record the values of the various signals at each state in a truthtable like the one below. All clocks are active on their falling edge.

Q_0	Q_1	Q_2	Q_3	J_1	R_3	S_3
0	0	0	0	1	0	0

Next Time

That's pretty much it for basic digital circutis. Next time we'll have a look at computer memory. Following that we'll combine everything so far and through the design, construction, and testing of a project or two before continuing on.

Series Index

1. Binary, Boolean, and Logic (https://adafru.it/BJk)
2. Some Tools (https://adafru.it/BJl)
3. Combinational Circuits (https://adafru.it/BJm)
4. Sequential Circuits (https://adafru.it/BJn)
5. Memories (https://adafru.it/Bji)
6. An EPROM Emulator (https://adafru.it/BIT)
7. MCUs... how do they work? (https://adafru.it/BJo)