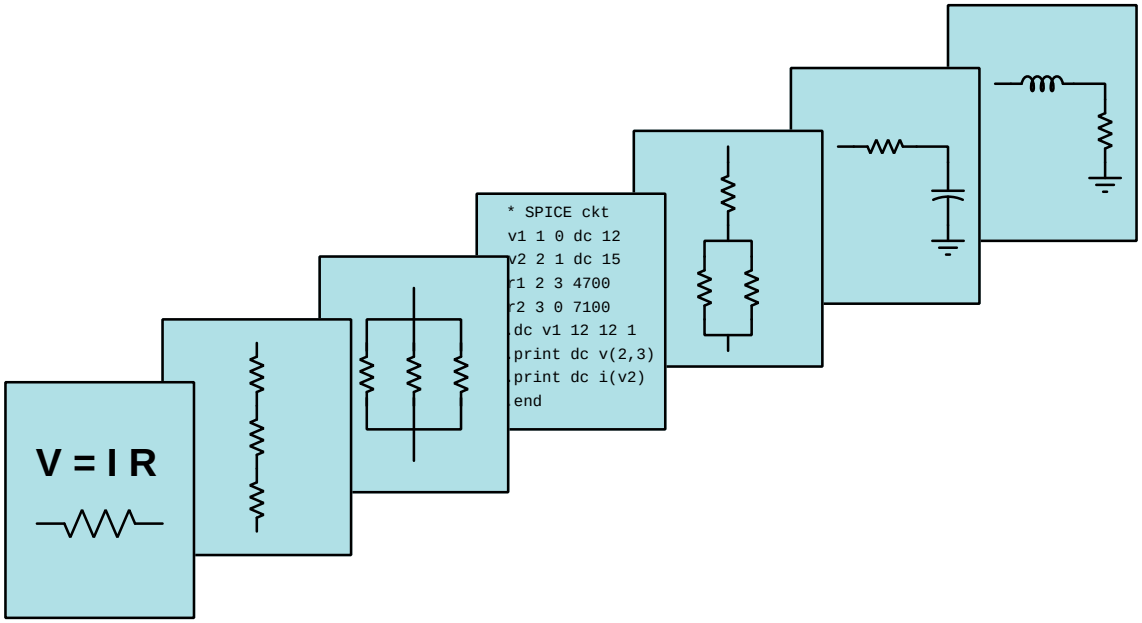
Modular Electronics Learning (ModEL) project



Shift Registers

Last update = 30 November 2023

Contents

Chapter 1

Introduction

Shift registers are very useful digital circuits, especially for converting between parallel (all bits at once) and serial (one bit at a time) forms of digital data. These circuits "shift" data bits from input to output lines on flip-flops, using the flip-flop units as single-bit memory devices. Combinational logic, usually in the form of AND and OR gates, may be added to the flip-flops to selectively "steer" discrete signals in different directions, thus giving more functionality to the shift register. The ultimate realization of this is the *universal* shift register, able to serially shift in either direction as well as shift data in parallel form.

Important concepts related to shift registers include **logic functions**, **truth tables**, **set** versus **reset** states, **flip-flops** versus **latches**, **edge-triggering**, **timing diagrams**, **registers**, **stacks**, **serial** versus **parallel** data, **steering network**, **ring counter**, **frequency division**, and **asynchronous** inputs.

Here are some good questions to ask of yourself while studying this subject:

- How might an experiment be designed and conducted to measure the minimum necessary set-up time for a flip-flop? What hypothesis (i.e. prediction) might you pose for that experiment, and what result(s) would either support or disprove that hypothesis?
- How might an experiment be designed and conducted to measure the minimum necessary hold time for a flip-flop? What hypothesis (i.e. prediction) might you pose for that experiment, and what result(s) would either support or disprove that hypothesis?
- How might an experiment be designed and conducted to test the proper operation of a shift register IC? What hypotheses (i.e. predictions) might you pose for that experiment, and what result(s) would either support or disprove those hypotheses?
- What distinguishes a flip-flop from a latch?
- What distinguishes a JK flip-flop from an SR flip-flop?
- What does a "register" circuit do?
- How do multiple flip-flops work together to form a register?

- What are some different types of registers?
- What does it mean to communicate digital data in *serial* fashion as opposed to *parallel*?
- What does it mean to *shift* data inside of a register?
- What does it mean to *load* data into a register?
- How may a register be used as a ring counter?
- What are some different types of ring counter circuits?
- How do AND and OR gates work to "steer" digital signals in certain types of shift register circuits?
- How may a ring counter be used as a frequency divider?
- How may multiple frequency dividers be cascaded to achieve large division ratios?
- What do the Preset and Clear inputs do on a flip-flop or latch?

Chapter 2

Tutorial

2.1 Latch and flip-flop review

Latch circuits are an important category of digital logic. A "latch" is a logic function designed to retain its last output state under certain input conditions. Like a toggle switch able to remain in either of its two possible states in the absence of any motivating force, latch circuits do the same at the command of electrical input signals.

The critical feature of latch circuits granting them their ability to "remember" previous states is *feedback*[1]: where the output of one or more logic gates is "fed back" to one or more inputs of other logic gates so that the circuit has a natural tendency to drive itself into one of two different states. Feedback is clearly evident in the logic gate diagrams of the *Set-Reset* latch, two versions of which are shown in the following illustration, along with a truth table describing the latch's function:

Set-Reset (SR) latch

S	R	Q	$\overline{Q}$
0	0	*Latch*	
0	1	0	1
1	0	1	0
1	1	0	0

S	R	Q	$\overline{Q}$
0	0	1	1
0	1	1	0
1	0	0	1
1	1	*Latch*	

SR latches based on NOR gates latch whenever both inputs are low, whereas SR latches based

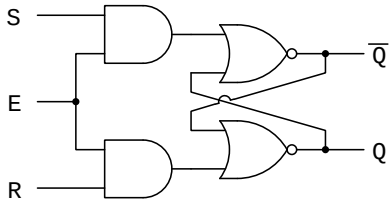
[1] Specifically, latch circuits employ *positive* (a.k.a. *regenerative*) feedback, so named because the effect of the fed-back signal is to reinforce the system's existing condition.

on NAND gates latch whenever both inputs are high. A latch is considered to be "set" when Q is high and $\overline{Q}$ is low. The "reset" state is just the opposite: Q low and $\overline{Q}$ high. If ever a latch's two outputs are found in the same state, it is considered *invalid*.

When packaged as integrated circuits (rather than built up from individual logic gates) SR latches are typically represented by "box" symbols, as shown in the following illustration. Note how the NOR-based SR latch has *active-high* inputs while the NAND-based SR latch has *active-low* inputs[2]:

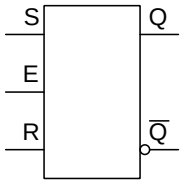
A more useful variation of the SR latch is the *enabled SR latch* with a third input line (E). This additional input line's state either enables or disables the set (S) and reset (R) inputs' functions. When enabled, this latch behaves as any normal SR latch; when disabled, this latch circuit remains in its "latched" state regardless of either the S or R input states:

Enabled SR latch internal schematic



E	S	R	Q	$\overline{Q}$
0	0	0	*Latch*	
0	0	1	*Latch*	
0	1	0	*Latch*	
0	1	1	*Latch*	
1	0	0	*Latch*	
1	0	1	0	1
1	1	0	1	0
1	1	1	0	0

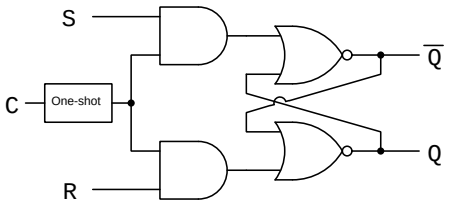
Enabled SR latch symbol



[2] An input's "active" state is the logical state necessary for that input to force a certain output condition. An SR latch's inputs are considered to both be inactive when the circuit is in its "latch" state. If you examine the truth table on the previous page, you will see how the NOR-based SR latch requires a 1 (high) input state to either set or reset, while the NAND-based SR latch requires the input to be 0 (low) to force either a set or reset state. This, in turn, is based on the truth tables of NOR and NAND gates, respectively. Any 1 (high) state input to a NOR gate forces its output low regardless of the other input state(s). Likewise, with NAND gates it is a 0 (low) input state which forces the output high regardless of the other input condition(s).

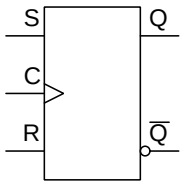
Another variation on this theme is to equip the latch circuit with a "one-shot"[3] circuit designed to detect either the rising or falling edge of a square-wave pulse signal, enabling the latch only during that brief transition from low to high (positive edge) or from high to low (negative edge) depending on the detection network. With this addition, the latch becomes a *flip-flop*. An internal diagram of a NOR-based SR flip-flop appears in the following diagram:

SR flip-flop internal schematic



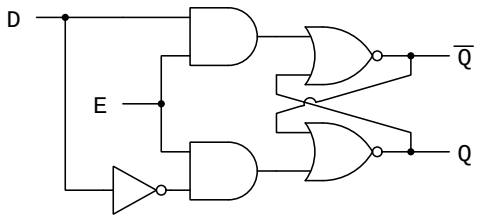
C	S	R	Q	$\overline{Q}$
X	0	0	*Latch*	
X	0	1	*Latch*	
X	1	0	*Latch*	
X	1	1	*Latch*	
⎍	0	0	*Latch*	
⎍	0	1	0	1
⎍	1	0	1	0
⎍	1	1	0	0

SR flip-flop symbol



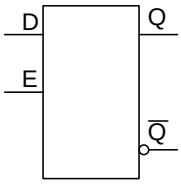
The D-type latch and D-type flip-flop are simplified versions of the SR latch and flip-flop, respectively, having just a single "data" (D) input rather than separate set (S) and reset (R) inputs. With this alteration, the device no longer has an "invalid" state:

D latch internal schematic

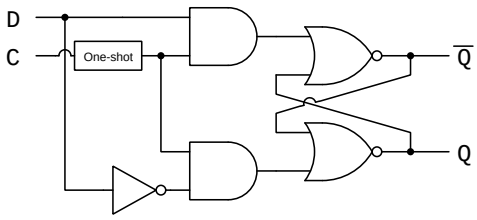


E	D	Q	$\overline{Q}$
0	0	*Latch*	
0	1	*Latch*	
1	0	0	1
1	1	1	0

D latch symbol

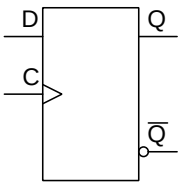


D flip-flop internal schematic



C	D	Q	$\overline{Q}$
X	0	*Latch*	
X	1	*Latch*	
⎍	0	0	1
⎍	1	1	0

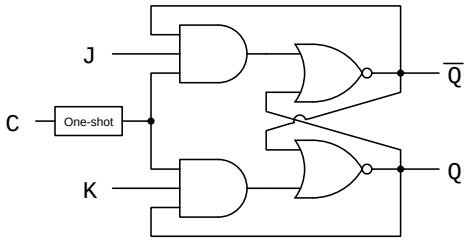
D flip-flop symbol



[3] The term "one-shot" refers to the fact that the edge-detecting circuit outputs a single pulse for each transition of the input signal. Even if the input signal transitions to its new state and remains in that new state for a long time, the one-shot responds only with a pulse at the transition time and nothing afterward.

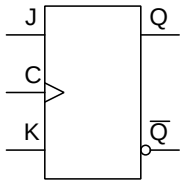
Perhaps the most useful type of flip-flop is the *JK* form. This uses an additional layer of feedback to give it a novel mode called *toggle*. When both J and K inputs are simultaneously active, and the clock pulse signal transitions, the "toggle" mode results in the Q and $\overline{Q}$ output states reversing. This feature is very useful for creating larger-scale digital circuits such as frequency dividers and counters:

JK flip-flop internal schematic



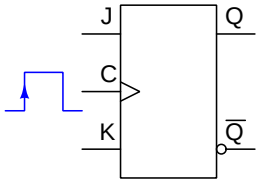
C	J	K	Q	$\overline{Q}$
X	0	0	*Latch*	
X	0	1	*Latch*	
X	1	0	*Latch*	
X	1	1	*Latch*	
⎍	0	0	*Latch*	
⎍	0	1	0	1
⎍	1	0	1	0
⎍	1	1	*Toggle*	

JK flip-flop symbol



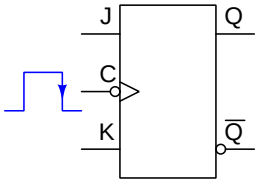
As with other flip-flops, JK flip-flops are designed in both *positive-edge-* and *negative-edge-triggered* versions. If a flip-flop's clock input happens to be a negative-edge style, an inversion "bubble" will be shown at that input terminal of the device. Both versions of a JK flip-flop appear in the following illustration:

Positive edge-triggering



J and K input states are recognized by the flip-flop only when the clock signal *rises*

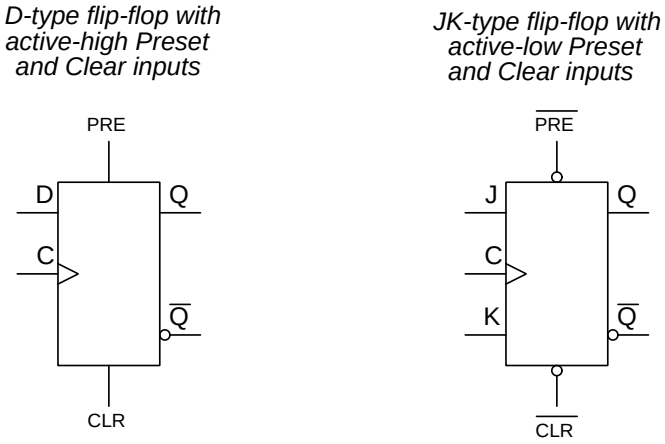
Negative edge-triggering



J and K input states are recognized by the flip-flop only when the clock signal *falls*

Latches and flip-flops alike require their input states to be stable for a certain amount of time prior to the reception of a clock (or enable) pulse, and for a certain amount of time following. These minimum signal times are called *set-up time* and *hold time*, respectively. Failure to abide by these limits may result in inconsistent operation. In some applications these set-up and hold times are viewed as limiting factors, particularly in high-speed digital logic circuitry where clock frequencies are so high that achieving the necessary set-up and hold times may be challenging. In other applications these minimum times are an exploitable feature, particularly in the case of *synchronous counter* and *shift register* circuits where the need for set-up time in particular halts the progression of data from one cascaded flip-flop to the next even though all the flip-flops receive the exact same clock pulse at the exact same times.

Some flip-flops provide one or more additional inputs designed to force the output lines to particular states, overriding the other input(s). These additional inputs are called *Preset* and *Clear*, the purpose of the Preset input being to force the flip-flop to a "set" state ($Q = 1$ and $\overline{Q} = 0$) and the purpose of the Clear input being to force the flip-flop to a "reset" state ($Q = 0$ and $\overline{Q} = 1$). Below we see schematic diagram symbols of a D-type flip-flop that happens to have active-high Preset and Clear inputs, as well as a JK flip-flop that happens to have active-low Preset and Clear inputs:

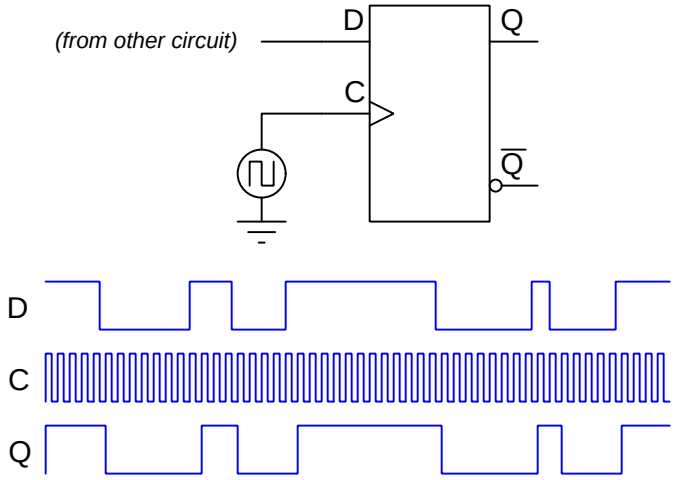


In the case of the D flip-flop, an active Preset or Clear input will override the state of the D input. In the case of the JK flip-flop, an active Preset or Clear input will override both J and K inputs.

In addition to active-high versus active-low varieties for Preset and Clear inputs, another important distinction is *synchronous* versus *asynchronous* Preset and Clear inputs. The term "synchronous" refers to events happening at the same time, and so a *synchronous* Preset or Clear input will not have any effect until the clock pulse arrives – i.e. its effect is always synchronized with the clock signal. In contrast, an *asynchronous* Preset or Clear input effects the Q and $\overline{Q}$ outputs *immediately* without waiting for a clock pulse. In other words, asynchronous Preset or Clear inputs function like the S and R inputs on a plain (non-enabled) SR latch, affecting the output states immediately when activated. One cannot tell whether Preset and/or Clear inputs are synchronous or asynchronous from the schematic symbols – only the datasheet for that particular integrated circuit will show you!

2.2 Registers

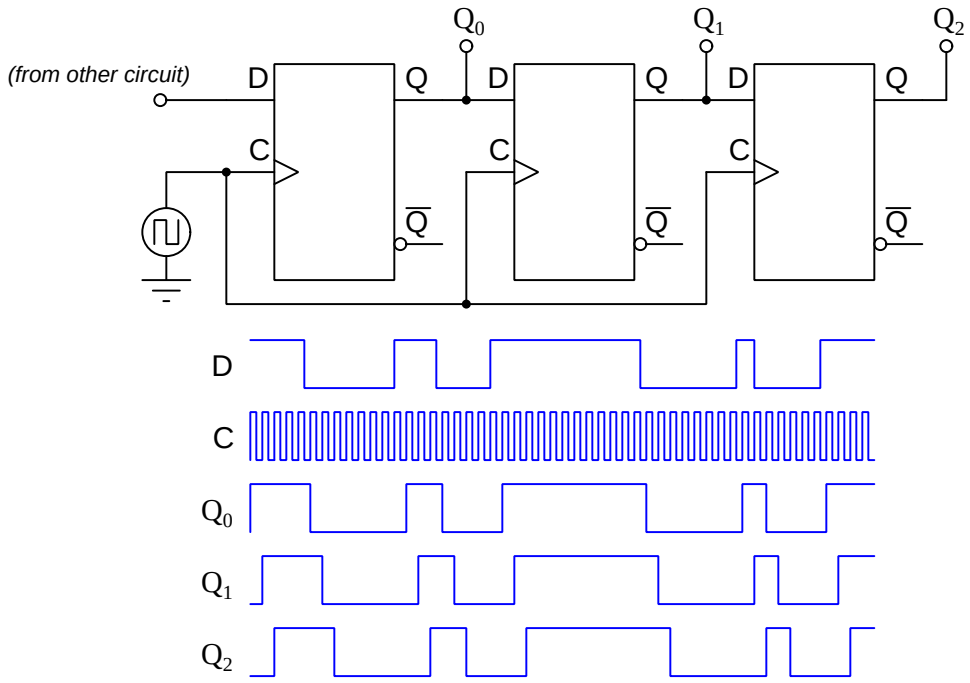
Latches and flip-flops alike function as single-bit *memory* devices, retaining discrete logic states representative of past input conditions when disabled (or unclocked). If we examine the response of a rapidly-clocked D-type flip-flop to a random stream of data, we see that the Q output of the flip-flop represents the input logic state during the *last* clock pulse:



Another way to think of this is to consider Q as being an *older* version of D: the data appearing at Q reflects what the data at D used to be one clock cycle ago. It is as though the flip-flop repeats the data given to it at the D input, *shifting that data over time* to appear at the Q output. Yet another way to think of this flip-flop's behavior is as a *sampling system*, periodically taking a "sample" of the data at D and holding that sampled data state until the next clock pulse when it samples again.

The amount of time delay separating data at Q from data at D depends entirely on the timing of the clock pulse. Note in the timing diagram shown for the single flip-flop circuit that the data at Q never lags the data at D by more than one clock cycle. The exact amount of time lag is not consistent, either: if you carefully compare the D and Q waveforms you will see how the width of each pulse on Q does not always match the width of respective pulses on D, and this is due to Q only being able to change state at the rising edge of the clock signal waveform while D (as an input) is free to change states at any time and thus is not dependent upon this circuit's clock pulse.

A more complex (and useful) variation on this theme is to cascade multiple D-type flip-flops so that each successive stage receives the time-shifted data from the one previous:



If Q_0 could be considered an "older" version of D, then Q_1 is an older version of Q_0, and Q_2 is an older version of Q_1. The utility of such a circuit becomes clear if the sequence of logical states sent to the D input is not random but instead has some meaning. In such a case, the states retained at the Q outputs become a record of D over time.

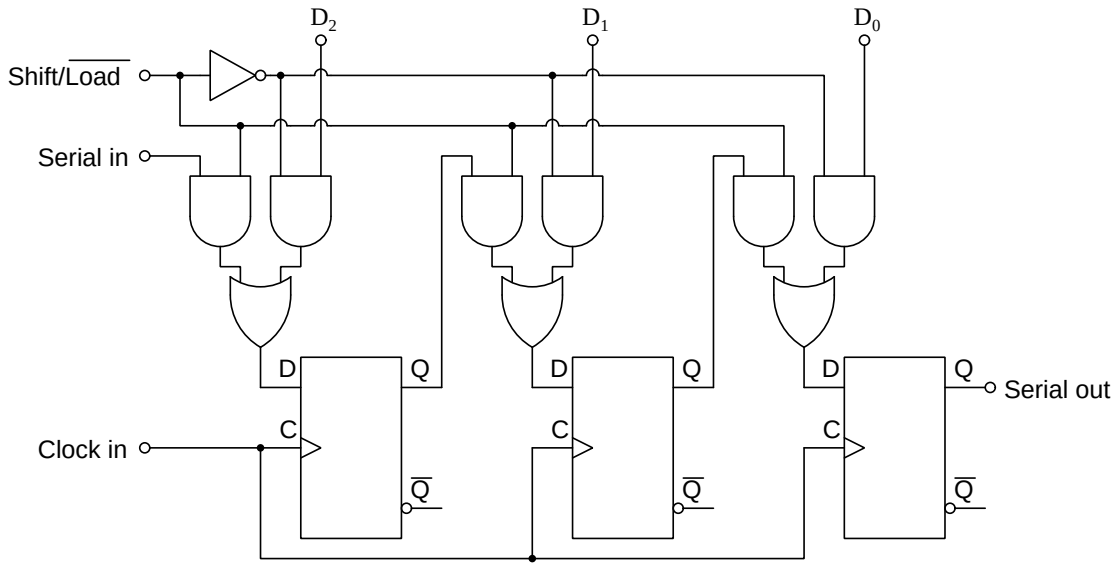
A digital circuit designed to retain a set of bit states is called a *register*. Register design typically exploits the retentive behavior of latches or flip-flops to achieve this purpose. Specifically, we refer to this circuit as a *shift register* because each pulse of the clock signal causes the logical states to shift position (e.g. from Q_0 to Q_1, from Q_1 to Q_2).

One practical application of such a circuit is to convert *serial* digital data into *parallel* digital data. "Serial" data is a set of bits conveyed one at a time over the same data line, whereas "parallel" data is a set of bits conveyed over individual lines (one line per bit). If the clock pulse frequency matches[4] the serial data's bit rate (i.e. the number of bits per second), then each Q output – over time – will come to represent those serially-communicated bits. A flip-book animation in section 4.1 (starting on page 26) demonstrates how this might work for a serially-coded set of eight bits, shifted one at a time into the register and finally presented in parallel form at the Q output lines.

[4] This is a critically important criterion, as without some degree of synchronism between the serial data rate and the shift register's clock the task of reliably translating data from serial into parallel form becomes impossible. Various methods exist to achieve this synchronization of clock and data rates, some more sophisticated than others, and is one of the topics of consideration in any study of serial data communication.

2.3 Shift register variations

Likewise, we may use a set of flip-flops to take digital data in parallel form and then "shift" the bits out of the register to create a stream of serial data: one bit at a time from the original data "word" transmitted by the shift register at a rate established by the clock pulse. The following shift register circuit performs this very function:



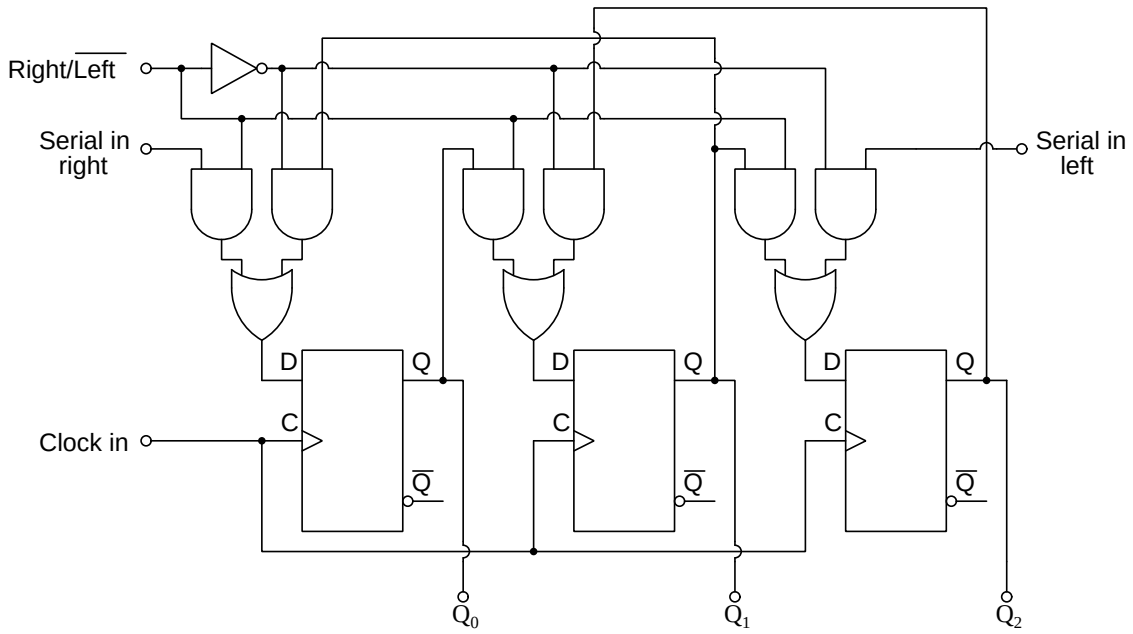
The AND/OR gate combinational networks feeding each flip-flop's D input act to steer data from either the preceding flip-flop or from one of the parallel data input lines (D_0 through D_2). When the $Shift/\overline{Load}$ input is made high, the left-hand AND gates become "enabled" to pass data through to each flip-flop D input in serial form, that data becoming "shifted" from left to right with every clock pulse. In other words, while in "Shift" mode this digital circuit acts as a *serial-in, serial-out* shift register. However, if the $Shift/\overline{Load}$ input goes low, the right-hand AND gates are now "enabled" to pass data from the parallel D_0-D_2 input lines through to all flip-flops as soon as a clock pulse arrives. Proper parallel-to-serial data conversion with this circuit, therefore, requires a short "load" period (just one clock cycle in duration) to read the parallel data followed by a longer "shift" period (as many clock cycles as there are bits stored in the register) to send that data out in serial form.

A flip-book animation in section 4.2 (starting on page 45) demonstrates how this might work for a parallel-coded set of eight bits, shifted once into the register in parallel format and then shifted one at a time out of the register in serial format.

A feature of this shift register is its "serial in" line. Left in the "shift" state, this register will simply shift serial data in at the "serial in" line and send it out the "serial out" line, functioning as a *serial-in, serial-out* shift register. At first this may not seem particularly useful, until you consider the fact that this is precisely the kind of feature we would need if this shift register were packaged as its own integrated circuit and we wished to cascade more than one to form a longer shift register

array. In that case, the "serial in" line of this shift register would need to connect to the "serial out" line of the preceding shift register.

Similar AND/OR "steering" networks may be employed to switch the register's shift direction, allowing it to shift right or shift left on command:

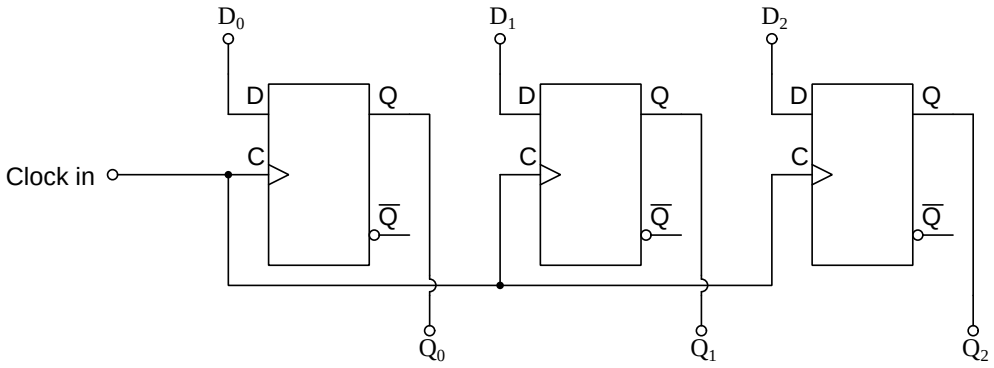


Maintaining the $Right/\overline{Left}$ input in the high state and pulsing the clock input line causes any logic states present at the *Serial in right* input to be shifted to Q_0, then to Q_1, and so on in a rightward direction. Switching $Right/\overline{Left}$ to the low state causes data present at the $Serial\,in\,left$ line to be shifted to Q_2, then to Q_1, then to Q_0 (i.e. leftward) with each clock pulse. In each case, the "enabled" AND gate steers signals from either the source on the left or the source on the right to the input of each flip-flop.

A further extension of shift register design is to combine parallel input and output with reversible shift direction to form a *universal* shift register. Instead of a single mode-control line called $Shift/\overline{Load}$ or $Right/\overline{Left}$, universal shift registers typically have two mode-control lines providing four different shift options (e.g. `00` = hold ; `01` = shift right ; `10` = shift left ; `11` = parallel load)[5].

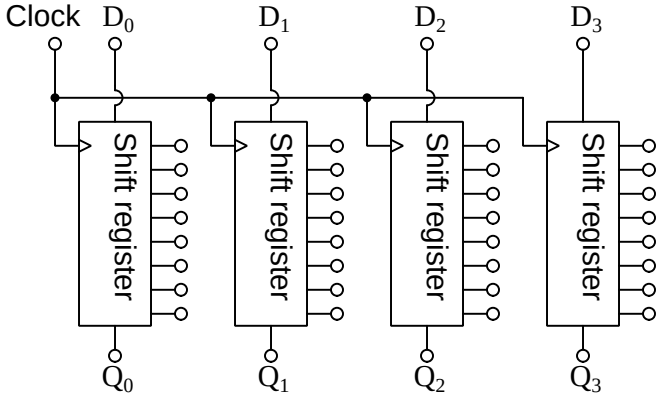
[5]The way this works internal to the shift register is to use a 2-line to 4-line decoder to decode the two mode-control inputs into one of four active lines. These four lines then control an enhanced array of AND/OR gates (three two-input AND gates feeding into a three-input OR gate, one such array per flip-flop) to steer signals either rightward, leftward, or from parallel input lines to the D inputs of the flip-flops. When the "hold" mode simply disables all flip-flop clock inputs to prevent any shifting.

Another type of shift register, much simpler than the serial-shifting designs, is the *parallel-in, parallel-out* shift register. Its purpose is to capture and retain all bit-states of a parallel data "word" at the active edge of every clock pulse:



Such shift registers are very useful for "staging" a set of binary bits for the input of some other digital circuit such as a memory array or an arithmetic logic unit or a digital-to-analog converter, to ensure the states of those bits remain stable between clock pulses.

Just as individual D-type flop-flops may be "paralleled" in this manner to produce a parallel-in, parallel-out shift register, banks of serial-shifting registers may also be paralleled to form shift registers operating on multi-bit words instead of operating on single bits. Schematic diagrams are almost too detailed to be helpful here, and so block diagrams work well to illustrate the concept:



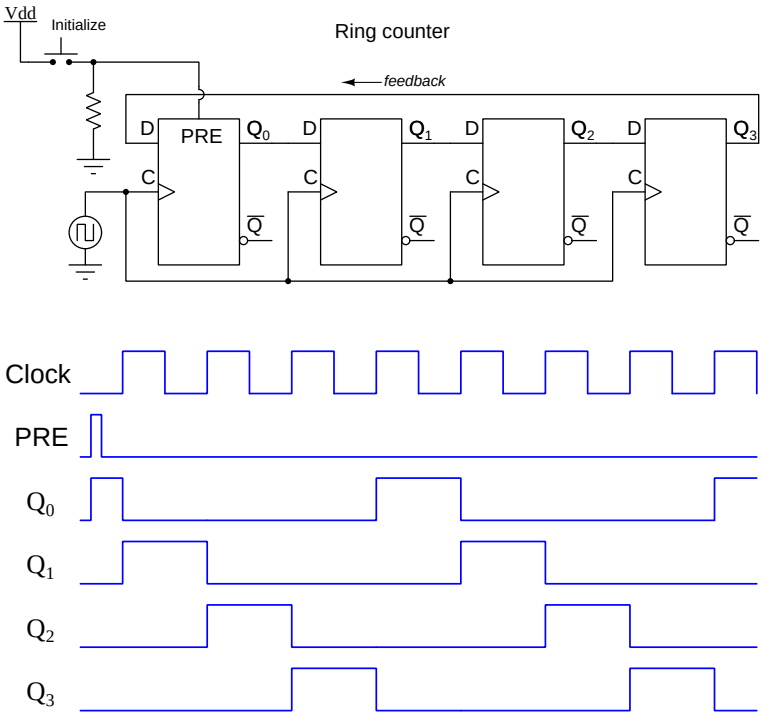
At each clock pulse, all four D data bits are read by the register array, and with each successive clock pulse those data bits propagate through the steps of the register array. This idea – of digital words stored in a shiftable memory array – is called a *stack*, and finds wide application in digital computing. Every time the array shifts data "down" (from the D inputs toward the Q outputs), data is being "pushed" onto the stack. If the shift direction is reversible, data may be "popped" off

of the stack in reverse order from that in which it was pushed, so that the last entry to the stack becomes the first entry read back from it[6].

[6]This idea of last-in-first-out (LIFO) stack is particularly useful in microprocessor programming, where the normal operation of a program must be interrupted to perform some other task. When the microprocessor receives an "interrupt" request, it pushes data for its current operation onto the stack before it switches to service the interrupt. Then, after the interrupt has been serviced, it "pops" the data off the stack to receive direction on how to resume normal operation from where it left off. This is analogous to a person being frequently interrupted by requests from other people, but just before acting on the latest request the person writes a note to themselves explaining where they left off in the last task, then places that note on top of a pile of other notes. As soon as each task is complete, the person reads the top note on the pile, discards it, and then proceeds to complete the task described by that note, eventually depleting the pile of notes and returning to their original work. The beauty of a multi-layer stack is that this process of interruption may occur several times (called *nested interrupts*) before returning to the original flow of execution, and so long as the stack is "deep" enough to remember all those points where it left off, the LIFO protocol ensures each of the tasks is picked up in the proper order.

2.4 Ring counters

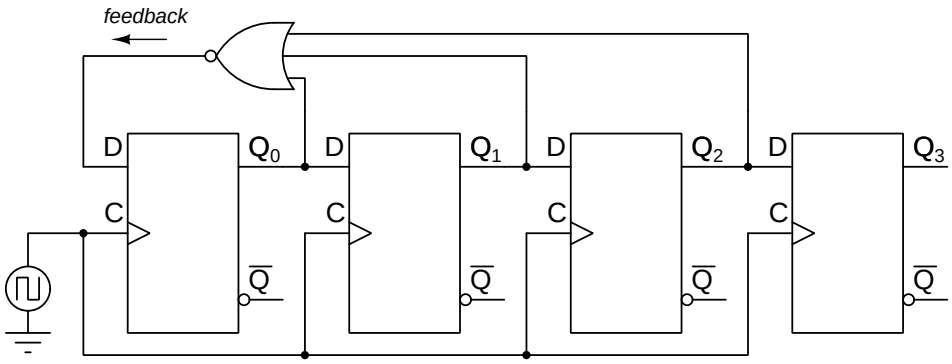
An interesting application of serial-in, serial-out shift registers is to generate pulse sequences based on the "recirculation" of shifted bits into and out of the register. Such a digital circuit is called a *ring counter*. Consider the following ring counter circuit, where the serial output line connects around to the register's own serial input line. The timing diagram below the schematic shows how this circuit would respond if the "Initialize" button were pressed prior to the first clock pulse:



As we have seen elsewhere, adding feedback to any system generally results in interesting new properties. Shift registers are no exception to this rule, and what we see in this case is that the shift register will circulate any pattern of high and low states entered serially via the "Initialize" pushbutton switch as the clock pulse continues its oscillations. If the register happens to hold a single "high" state (i.e. one flip-flop at a time is set while all others in the register are reset, sometimes referred to as a *one-hot* ring counter) as shown by the example timing diagram, then the pulse sequence measured at any Q output will have a frequency of n times slower than the clock, where n is the number of flip-flops in the serial shift register. Thus, a "one-hot" ring counter may serve as a digital *frequency divider*, the divisor limited only by the practicality of building a circuit with n number of flip-flops.

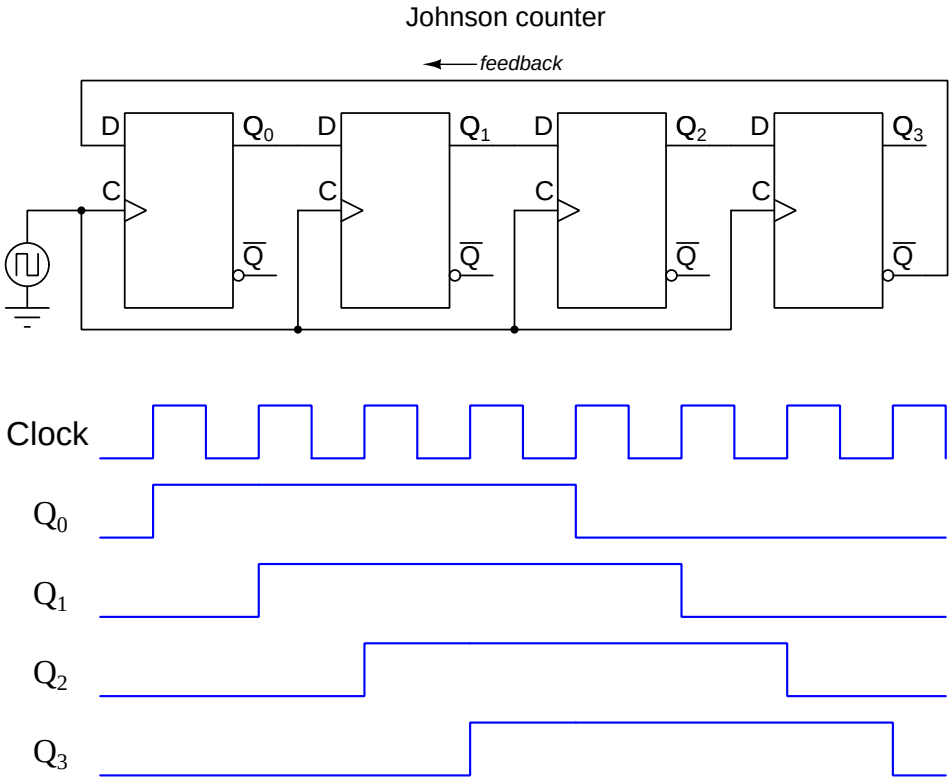
A significant limitation of the simple ring counter previously shown is that it must be "initialized" at every power-up by someone pressing the pushbutton switch for just the right amount of time. This is obviously not practical for anything but a demonstration circuit.

One solution to the initialization problem is to make the feedback more complex. Instead of coupling the shift register's serial output line directly to its serial input line, we drive the input with a NOR gate set up to output a high state only when outputs Q_0 through Q_2 are all low:



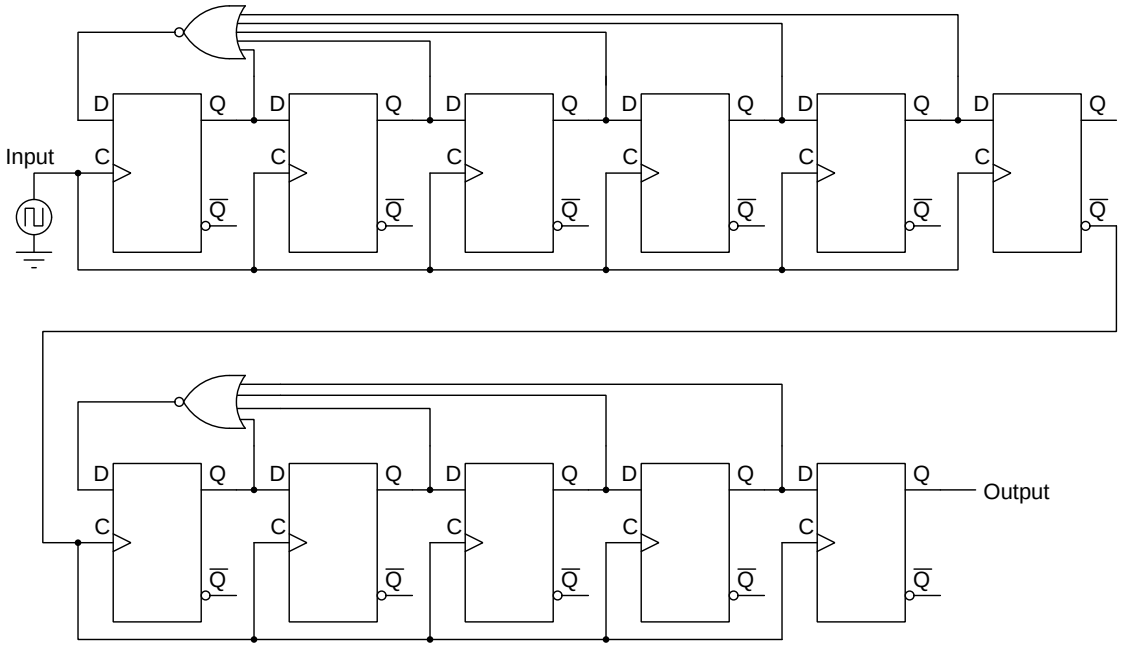
No matter what states these flip-flops happen to be in immediately following power-up, by the time four clock pulses have elapsed the circuit will have settled to a "one-hot" condition where only one flip-flop is set and all others are reset. Once a "one-hot" pattern is established, the frequency-division ratio for this ring counter will be $\frac{1}{n}$ just like any other "one-hot" ring counter containing n flip-flops: i.e. the frequency of any Q or $\overline{Q}$ flip-flop output will be precisely $\frac{1}{n}$ the frequency of the clock signal driving the entire counter.

A special type of ring counter called a *Johnson counter* works by feeding back the *complement* of the serial output. It is extremely simple to implement given the fact that every flip-flop in the shift register has both Q and $\overline{Q}$ outputs:



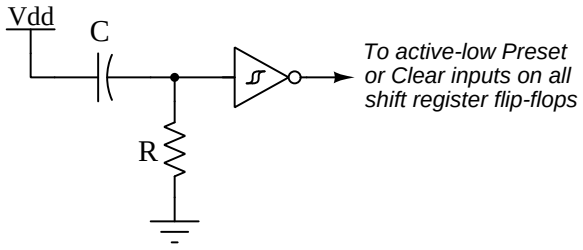
The frequency of any output signal (i.e. sensed at any Q output) is slower than the clock's frequency by a factor of $2n$, where n is the number of flip-flops. Furthermore, the frequency-divided signal has a nice symmetrical duty cycle of 50%, unlike the "narrow" pulses output by a standard ring counter.

If extremely large divisions of frequency are desired, multiple ring counters may be cascaded: connecting the clock pulse input of one ring counter to the serial output line of the preceding counter. The result of this cascading will be that the over-all frequency division ratio is the *product* of the numbers of flip-flop stages within each ring counter. For example, a ring counter with 6 flip-flops cascading its output signal to the clock input of another ring counter having 5 flip-flops would create a 30:1 frequency reduction ratio, as shown in the schematic below:



2.5 Auto-initialization

Initializing a Johnson counter is a simpler task than initializing a regular ring counter which requires *one* of its flip-flop be set and all others reset. To initialize a Johnson counter, we merely need to set or clear *all* flip-flops simultaneously. One easy method for doing so automatically upon power-up uses a resistor-capacitor network and Schmitt trigger[7] gate to generate a momentary pulse on power-up, suitable for connection to the asynchronous[8] Preset or Clear inputs of all flip-flops inside the shift register:



If the shift register in question is built of individual flip-flops, the output of this initializing-pulse network will need to connect to every $\overline{PRE}$ or $\overline{CLR}$ input. If an integrated-circuit shift register is used instead of individual flip-flops, there is usually a master "clear" or "preset" input available to do the same. Incidentally, this same power-up initialization circuit may be used to ensure a reliable start-up condition for any form of state-based digital logic (e.g. latches, flip-flops, shift registers, counters, etc.).

[7] To review, a *Schmitt trigger* is a special type of digital gate input designed to exhibit hysteresis: it does not register a "high" state until the input voltage rises above a certain threshold, and does not register a "low" state until the voltage falls below another (lower) threshold. This hysteretic action eliminates problems which might otherwise arise from a logic gate receiving a decaying (analog) voltage signal from the RC network. An ordinary logic gate may act erratically if the input voltage falls between the standard "low" and "high" threshold values. A Schmitt trigger will not be erratic: its output remains solidly at its last state until the input voltage clears either of the threshold values.

[8] An "asynchronous" input is one that has immediate effect on the circuit's output state(s) even when the other inputs are disabled by virtue of the clock pulse not transitioning at that time. In other words, the Preset and Clear inputs do not have to wait for the clock pulse to take effect.

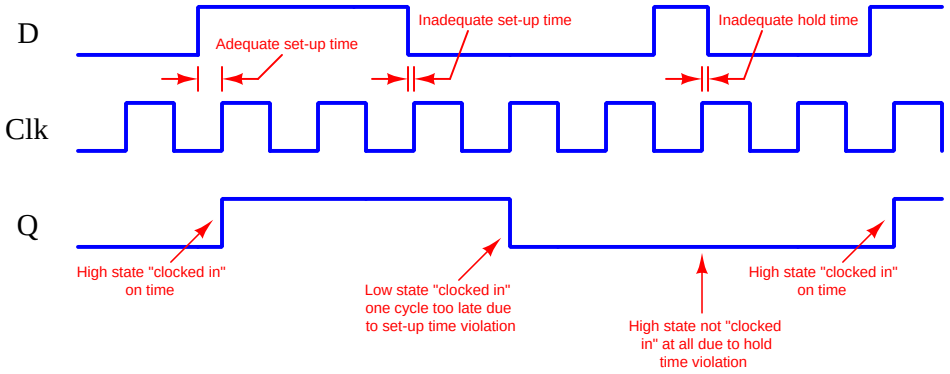
Chapter 3

Derivations and Technical References

This chapter is where you will find mathematical derivations too detailed to include in the tutorial, and/or tables and other technical reference material.

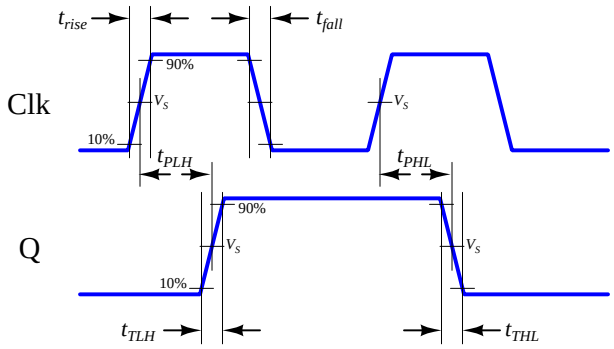
3.1 Digital pulse criteria

Clock-synchronized digital logic circuits such as counters, shift registers, and microprocessors require their input signals to be at stable states immediately before and immediately after the clock pulse arrives. For example, the following timing diagram shows input and output states for a D-type flip-flop circuit (positive-edge triggered), showing the effects of some signal timing violations:



Datasheets for digital circuits often provide timing diagrams showing criteria related to pulse signal timing and logic states. These diagrams don't typically show ideal square-edged pulses, but rather *trapezoidal* pulse profiles intended to exaggerate realistic features such as rise and fall times, propagation delays, and minimum set-up/hold times. Such diagrams usually confuse students who are accustomed to seeing square-edged pulses in their textbook timing diagrams. This technical reference will show some typical timing diagrams and explain what they represent.

For example, consider this timing diagram for a positive-edge-triggered JK flip-flop having both its J and K inputs tied high so as to maintain the circuit in its "toggle" mode. As such we would expect its output (Q) to change state with every rising edge of the clock pulse:



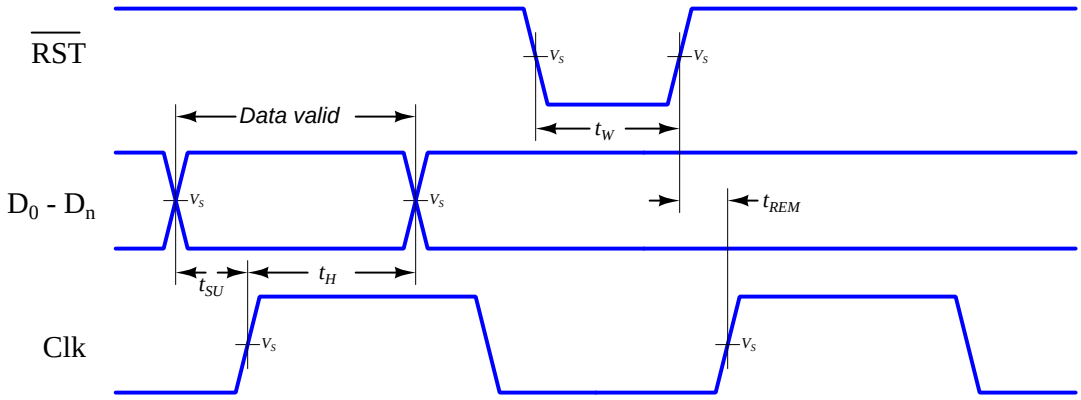
Each of the labels found in this diagram is defined as follows:

- t_{rise} = Rise time of input signal, typically measured from 10% of signal amplitude to 90% of signal amplitude
- t_{fall} = Fall time of input signal, typically measured from 90% of signal amplitude to 10% of signal amplitude
- t_{TLH} = Low-to-High transition time of output signal, typically measured from 10% of signal amplitude to 90% of signal amplitude (the same concept as rise time, but applied to the output signal instead of the input signal)
- t_{THL} = High-to-Low transition time of output signal, typically measured from 90% of signal amplitude to 10% of signal amplitude (the same concept as fall time, but applied to the output signal instead of the input signal)
- t_{PLH} = Propagation delay time of output signal when switching from low to high
- t_{PHL} = Propagation delay time of output signal when switching from high to low
- V_S = Switching threshold voltage, typically defined as 50% of signal amplitude

This timing diagram shows how a digital logic circuit reacts to a single input signal, in this case the clock pulse. Although this example happens to be for a JK flip-flop in toggle mode, the same type of timing diagram with its exaggerated rise/fall times and propagation delays could be applied to any digital logic gate whose output state depended solely on the state of a single input.

For synchronous digital logic circuits where input signals must coordinate with the clock pulse signal in order to be properly accepted by the circuit, we typically find timing diagrams comparing these input states to each other, often without showing the output(s) at all. Instead of showing us how the digital logic circuit will react to an input signal, this sort of timing diagram shows what the digital logic circuit *expects* of its multiple input signals.

The example is shown here for a positive-edge-triggered D register[1] having multiple data lines (D_0 through D_n), one asynchronous[2] reset line ($\overline{RST}$), and one clock input. The arbitrary logic levels of the multiple data lines are shown as a pair of complementary-state pulse waveforms, the only relevant features being the *timing* of the data and not the particular voltage levels of the data signals:



Labels shown in this diagram refer to *minimum* time durations the logic circuit requires for reliable operation:

- t_{SU} = Minimum set-up time before the arrival of the next clock pulse
- t_H = Minimum hold time following the last clock pulse
- t_W = Minimum width (duration) of the asynchronous reset pulse
- t_{REM} = Minimum removal time before the arrival of the next clock pulse

Violations of any of these minimum times may result in unexpected behavior from the logic circuit, and is an all-too-common cause of spurious errors in high-speed digital circuit designs. The assessment of digital pulse signals with regard to reliable circuit operation is generally known as *digital signal integrity*.

[1] In this case, a "D register" is synonymous with multiple D-type flip-flops sharing a common clock input, passing data through from each D input to each corresponding Q output synchronously with each clock pulse.

[2] To review, a *synchronous* input depends on a clock pulse while an *asynchronous* input is able to affect the circuit independent of the clock pulse.

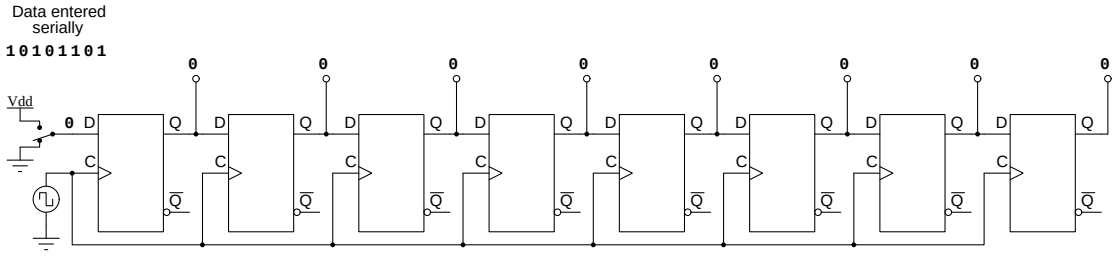
Chapter 4

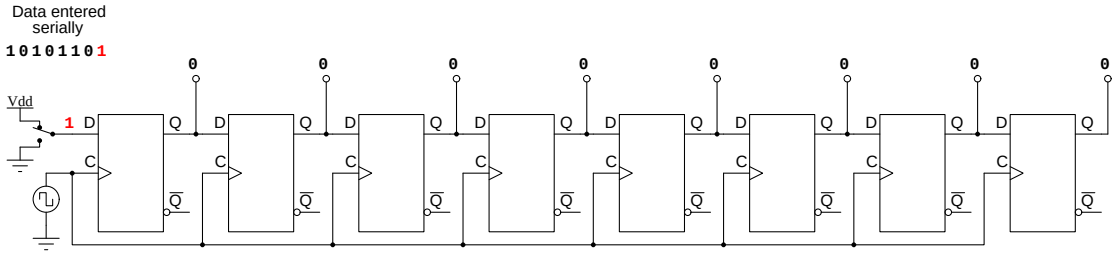
Animations

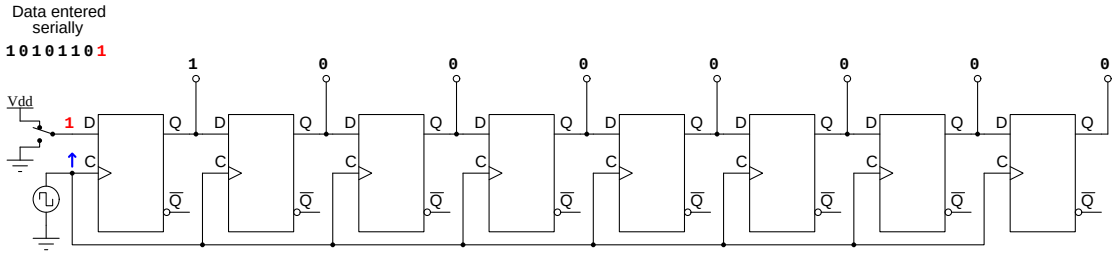
Some concepts are much easier to grasp when seen in *action*. A simple yet effective form of animation suitable to an electronic document such as this is a "flip-book" animation where a set of pages in the document show successive frames of a simple animation. Such "flip-book" animations are designed to be viewed by paging forward (and/or back) with the document-reading software application, watching it frame-by-frame. Unlike video which may be difficult to pause at certain moments, "flip-book" animations lend themselves very well to individual frame viewing.

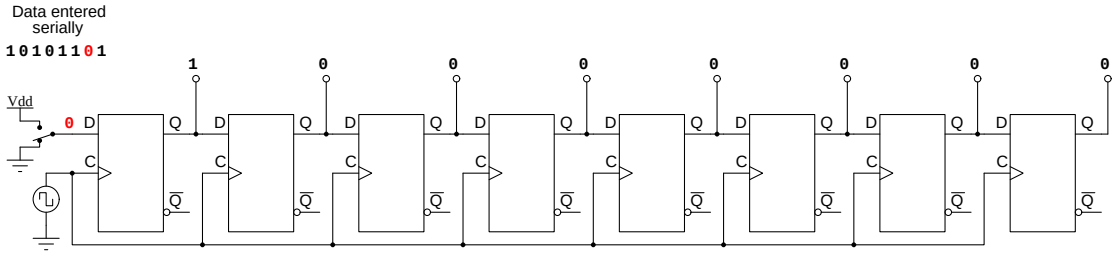
4.1 Animation of serial-in, parallel-out shift register

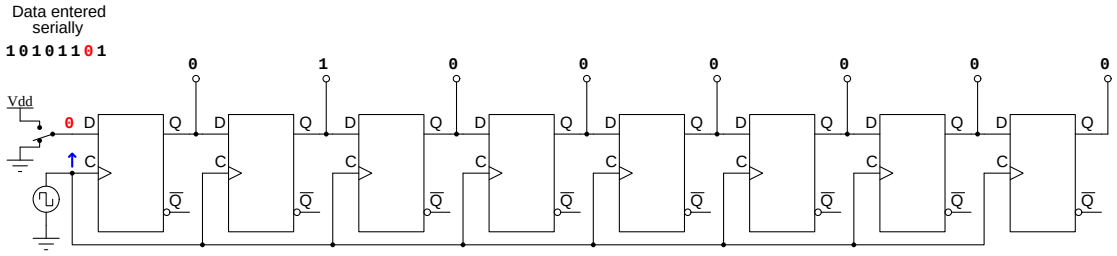
The following animation shows an eight-bit serial-in, parallel-out shift register accepting a serial stream of eight bits and one-by-one shifting them to its eight output lines for parallel reading.

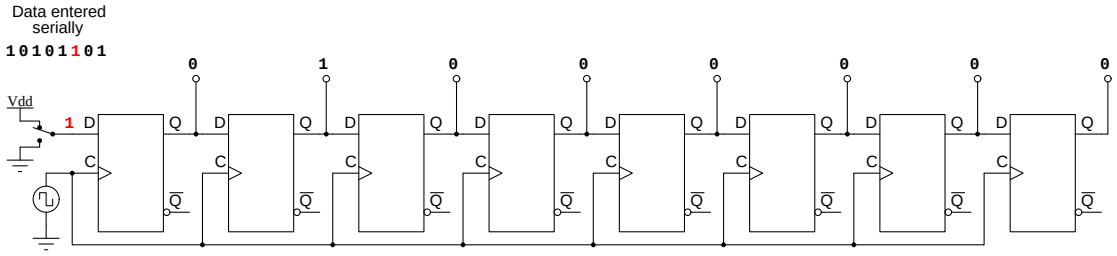

Data entered
serially
10101101
Vdd
0
D
Q
C
$\overline{Q}$

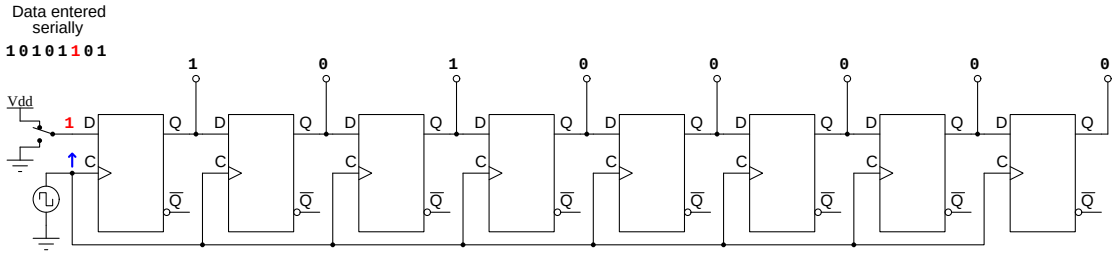

Data entered
serially
10101101
Vdd
1
D
Q
C
Q
0

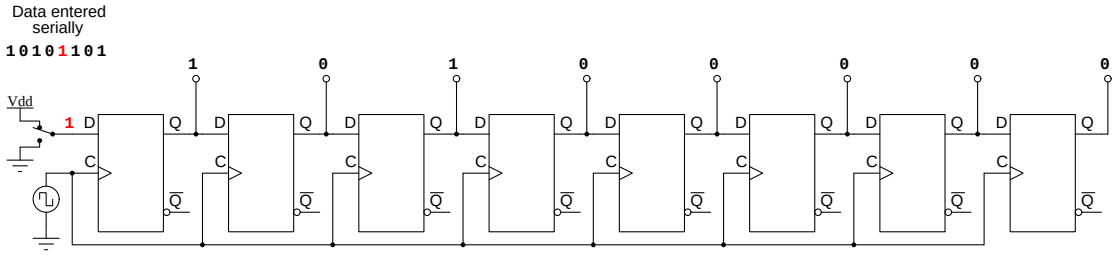

Data entered serially
10101101
Vdd
1
D
C
Q
$\overline{Q}$
1
0
0
0
0
0
0
0

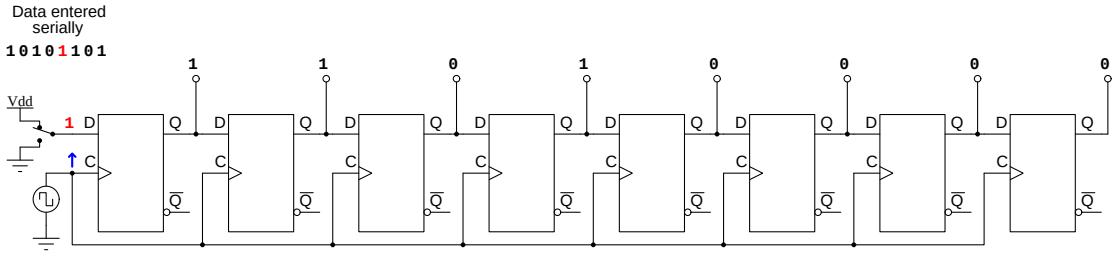

Data entered
serially
1 0 1 0 1 1 0 1
Vdd
0
1
0
0
0
0
0
0
D
Q
C
Q

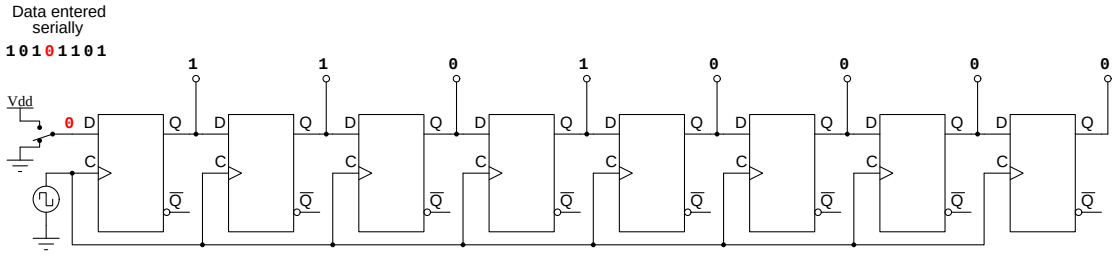

Data entered serially
1 0 1 0 1 1 0 1
Vdd
0
D
C
Q
$\overline{Q}$
0
1
0
0
0
0
0
0

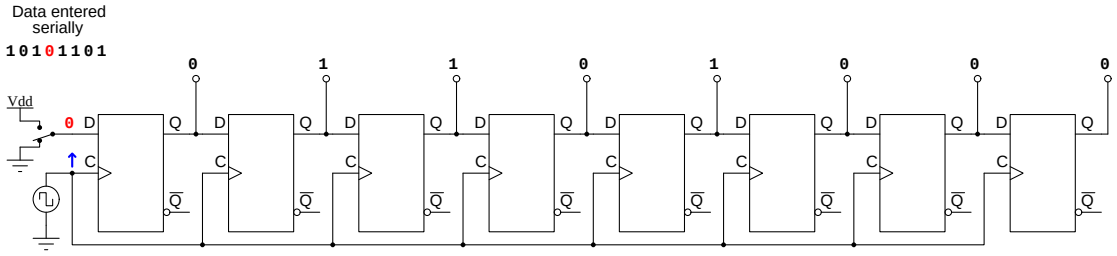

Data entered
serially
10101101
Vdd
1
D
Q
C
Q
0
1
0
0
0
0
0
0

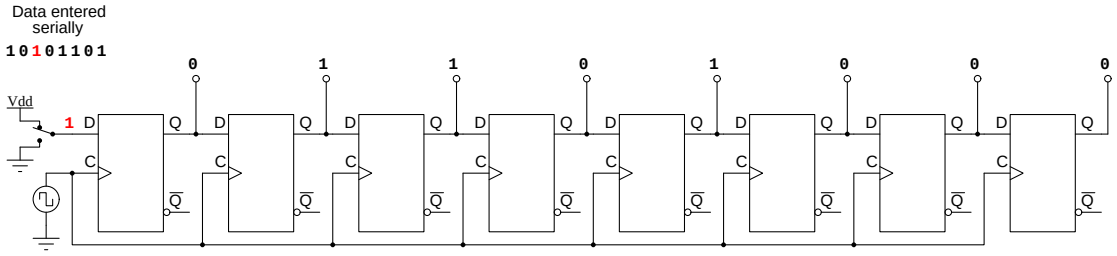

Data entered serially
10101101
Vdd
1
D
C
Q
$\overline{Q}$
1
0
1
0
0
0
0
0

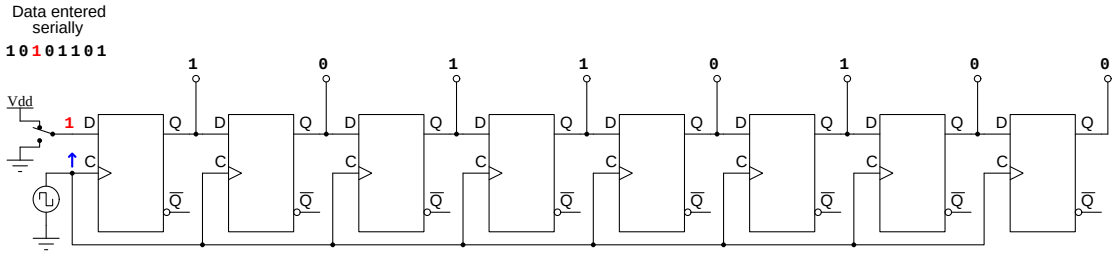

Data entered
serially
10101101
Vdd
1
D
Q
C
Q
1
0
1
0
0
0
0
0

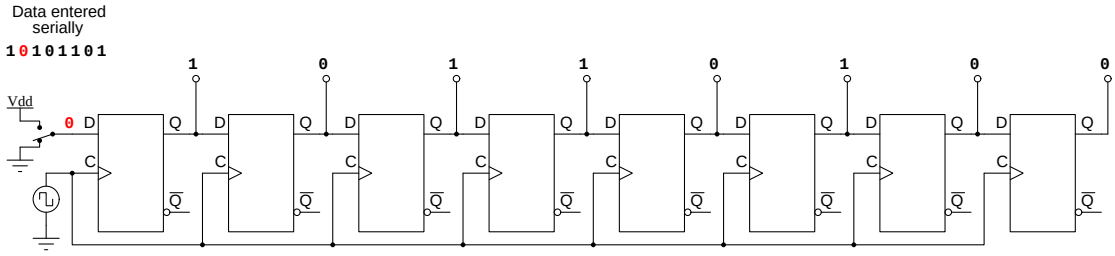

Data entered serially
1 0 1 0 1 1 0 1
Vdd
1
D
C
Q
Q̄
1
1
0
1
0
0
0
0

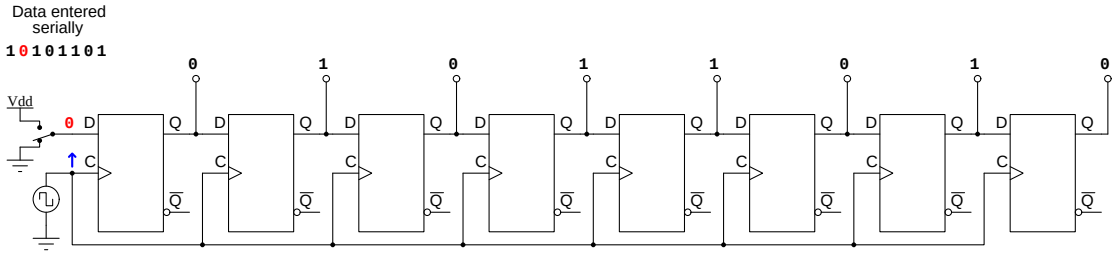

Data entered
serially
10101101
Vdd
0
D
Q
C
Q
1
1
0
1
0
0
0
0

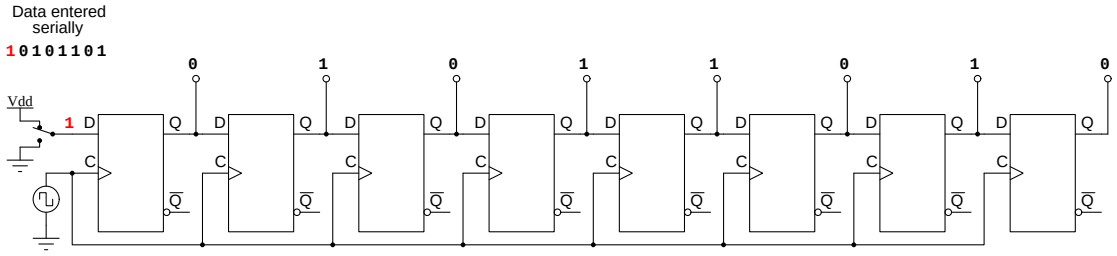

Data entered
serially
10101101
Vdd
0
D
C
Q
Q
0
1
1
0
1
0
0
0

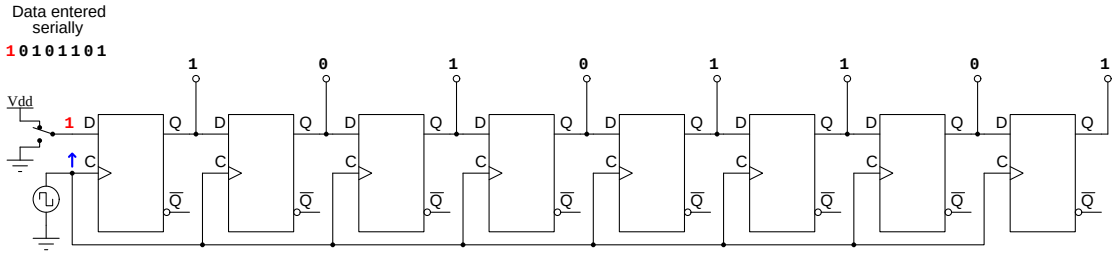

Data entered
serially
10101101
Vdd
1
D
Q
C
Q
0
1
1
0
1
0
0
0

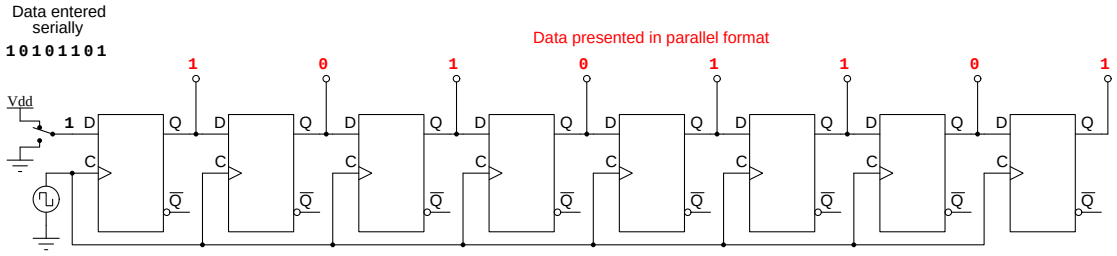

Data entered serially
10101101
Vdd
1
D
C
Q
Q
1
0
1
1
0
1
0
0


Data entered
serially
10101101
Vdd
0
D
Q
C
Q
1
0
1
1
0
1
0
0


Data entered serially
10101101
Vdd
0
D
C
Q
$\overline{Q}$
0
1
0
1
1
0
1
0

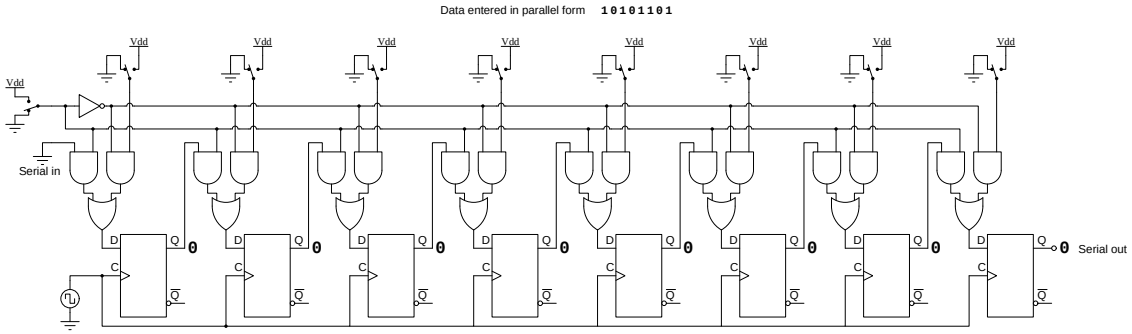

Data entered
serially
10101101
Vdd
1
D
Q
C
Q
0
1
0
1
1
0
1
0

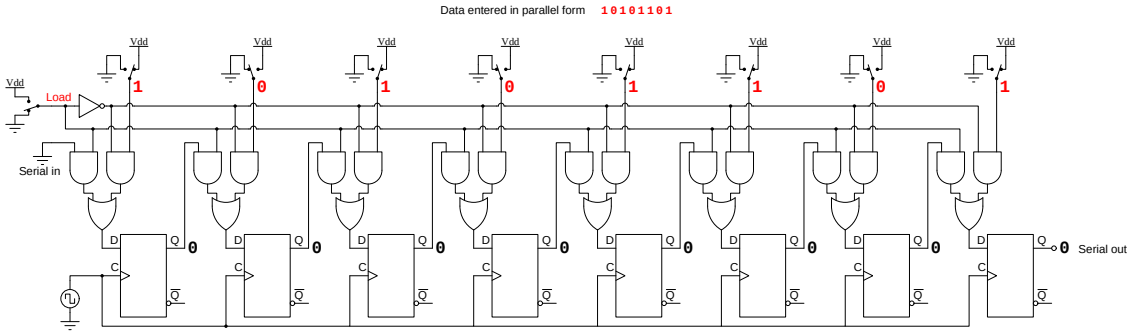

Data entered
serially
10101101
Vdd
1
D
C
Q
$\overline{Q}$
1
0
1
0
1
1
0
1

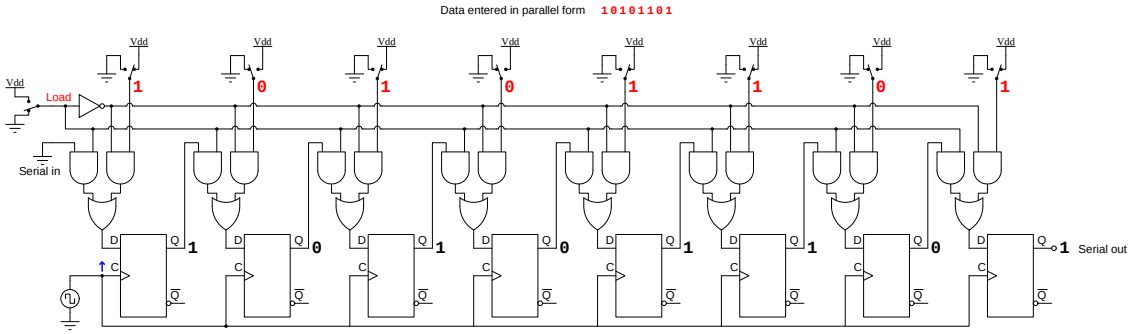

Data entered serially
10101101
Data presented in parallel format
Vdd
1
D
Q
C
1
0
1
0
1
1
0
1

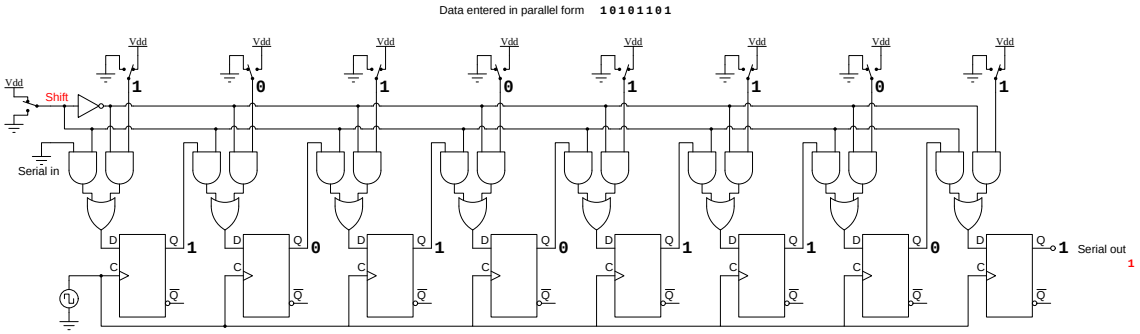
4.2 Animation of parallel-in, serial-out shift register

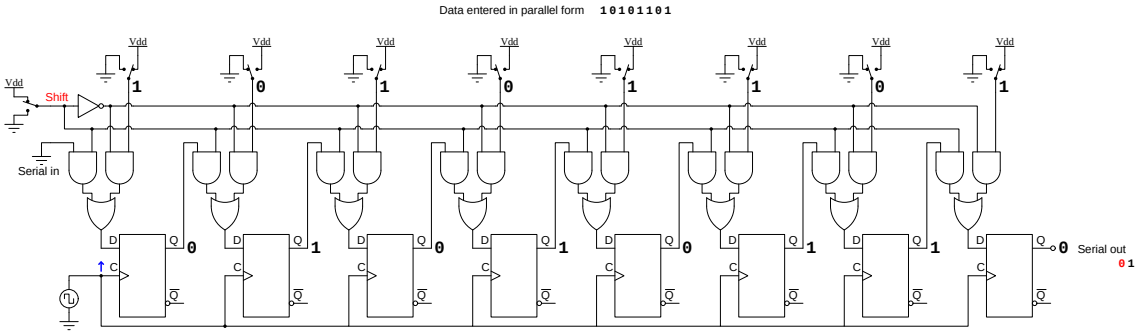
The following animation shows an eight-bit parallel-in, serial-out shift register accepting a parallel batch of of eight bits and one-by-one shifting them to its eight output lines for serial transmission to some other digital circuit.

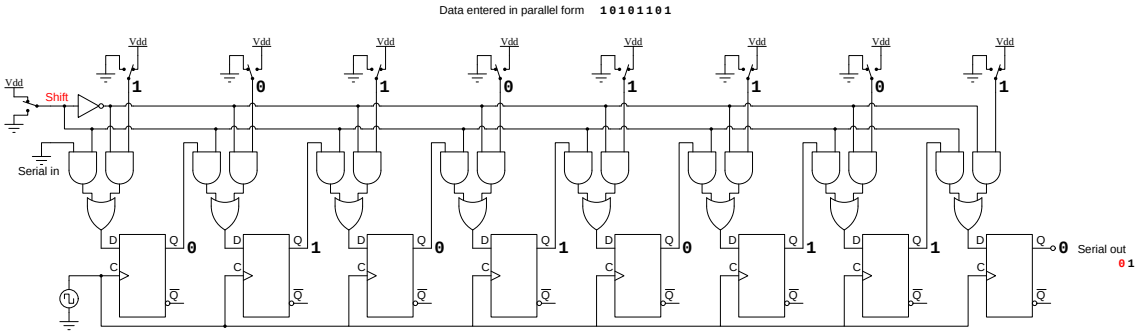

Data entered in parallel form 10101101
Vdd
Serial in
D
Q
C
Q
0
Serial out

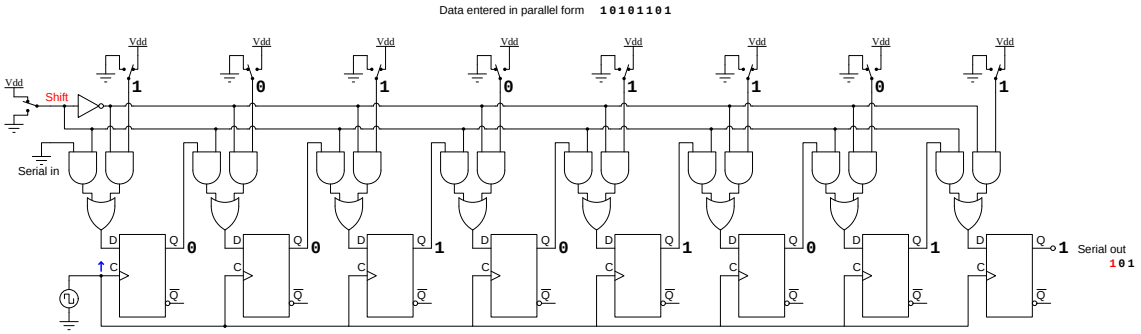

Data entered in parallel form 10101101
Vdd
Load
Serial in
Serial out
D
Q
C
1
0

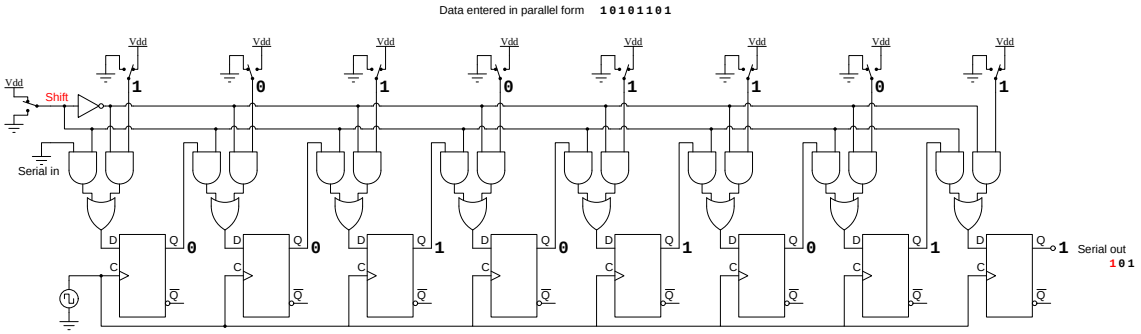

Data entered in parallel form 10101101
Vdd
Load
Serial in
Serial out
D
Q
C
1
0

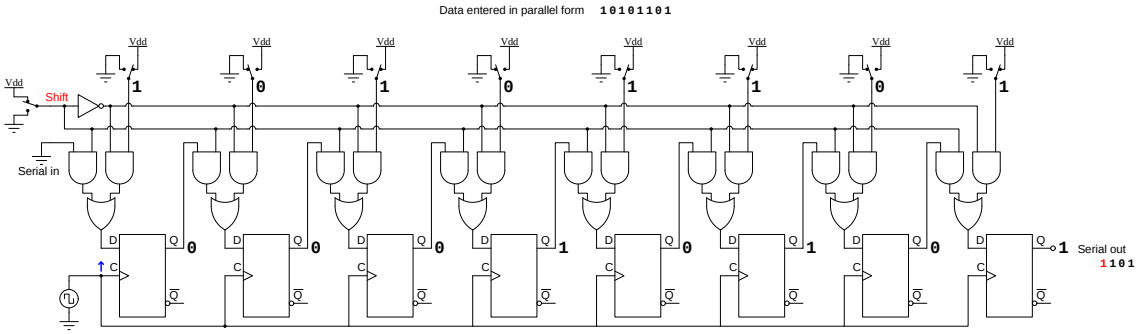

Data entered in parallel form 10101101
Vdd
Shift
Serial in
1
0
1
0
1
1
0
1
D
Q
C
Q
1
0
1
0
1
1
0
1
Serial out
1

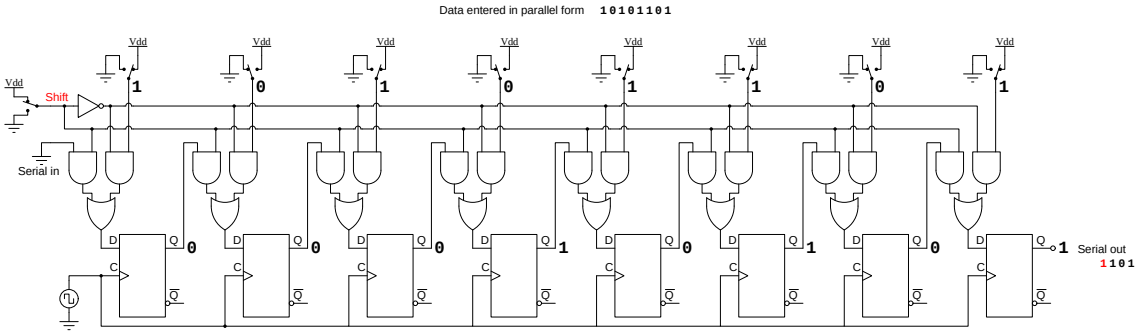

Data entered in parallel form 10101101
Vdd
Shift
Serial in
Serial out
01
D
Q
C
Q̄

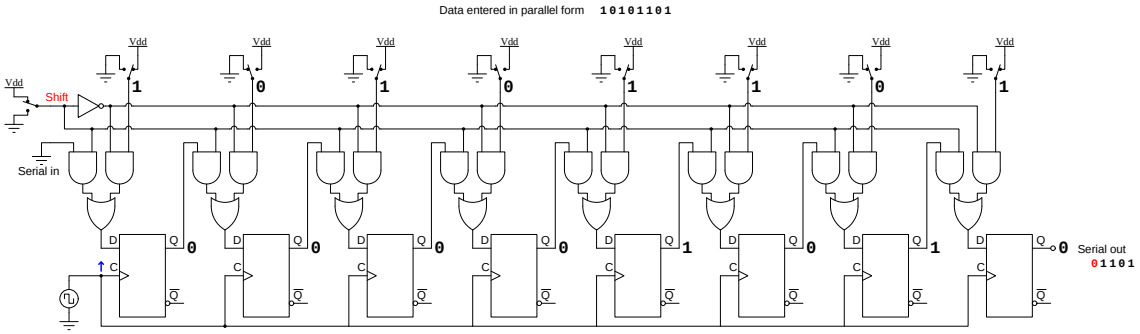

Data entered in parallel form 10101101
Vdd
Shift
Serial in
1
0
1
0
1
1
0
1
D
Q
C
Q
0
1
0
1
0
1
1
0
Serial out
01

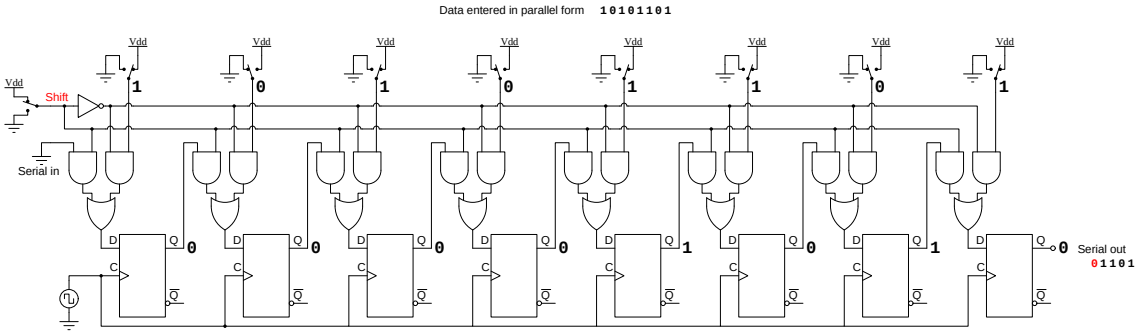

Data entered in parallel form 10101101
Vdd
Shift
Serial in
Serial out
101
D
Q
C
Q̄

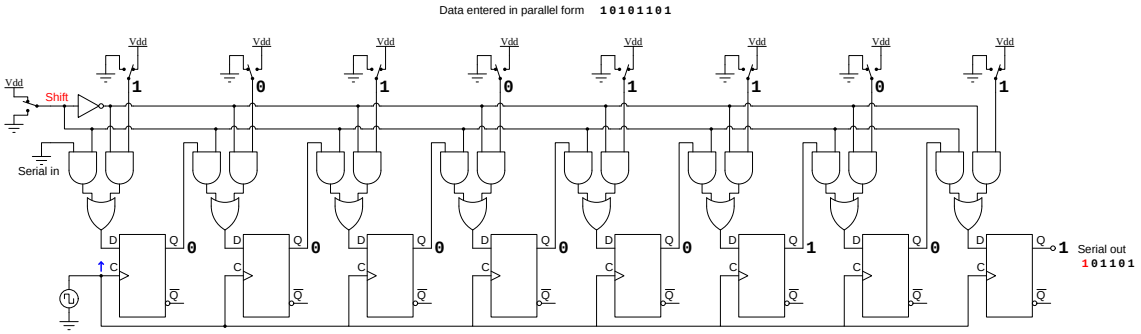

Data entered in parallel form 10101101
Vdd
Shift
Serial in
1
0
1
0
1
1
0
1
D
Q
C
Q
0
0
1
0
1
0
1
1
Serial out
101

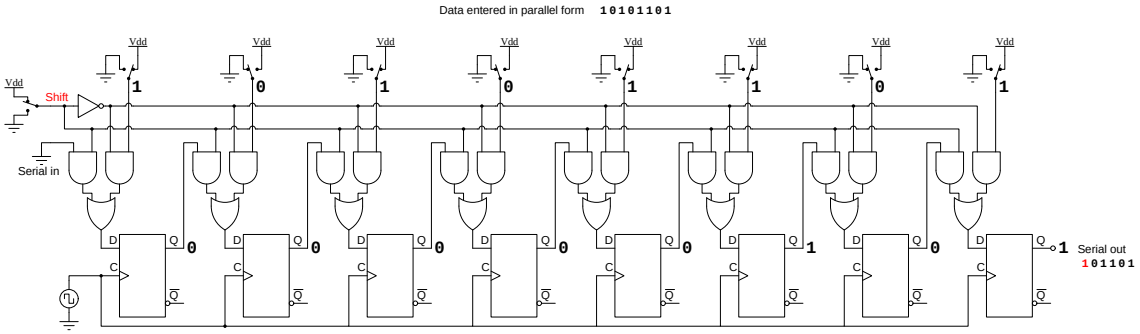

Data entered in parallel form 10101101
Vdd
Shift
Serial in
1
0
1
0
1
1
0
1
D
Q
C
Serial out
1101

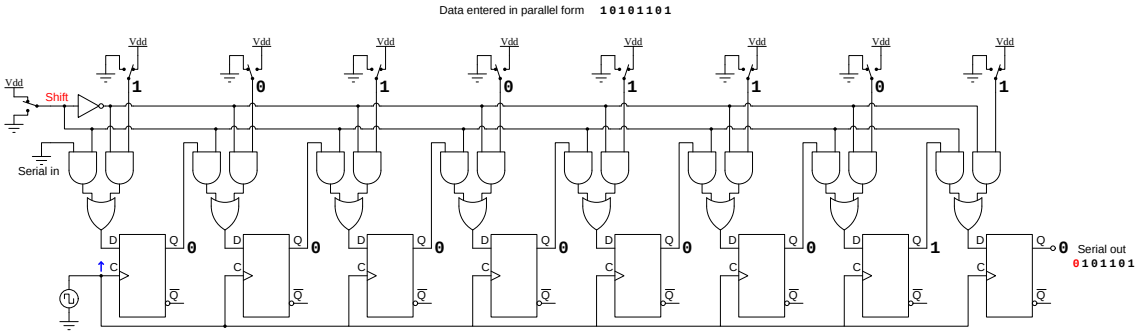

Data entered in parallel form 10101101
Vdd
Shift
Serial in
1
0
1
0
1
1
0
1
D
Q
C
Q
0
0
0
1
0
1
0
1
Serial out
1101

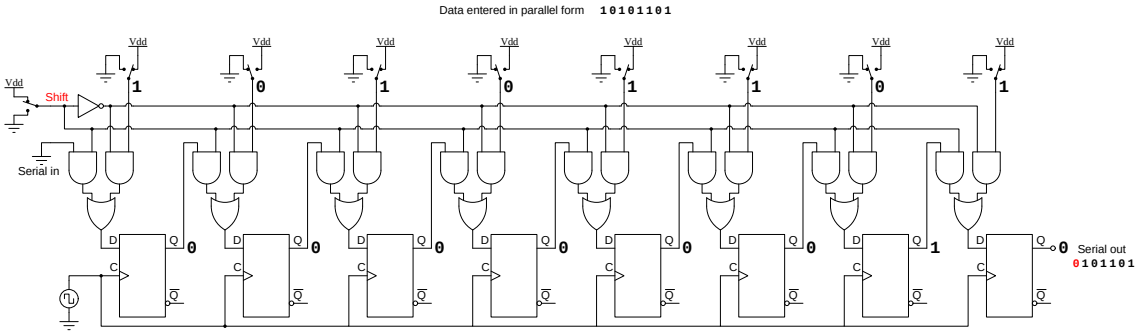

Data entered in parallel form 10101101
Vdd
Shift
Serial in
1
0
1
0
1
1
0
1
D
Q
C
Serial out
01101

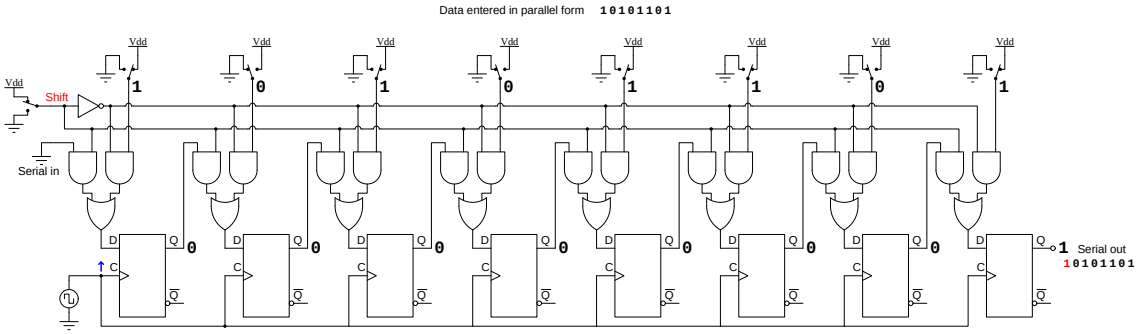

Data entered in parallel form 10101101
Vdd
Shift
Serial in
1
0
1
0
1
1
0
1
D
Q
C
Q
0
0
0
0
1
0
1
0
Serial out
01101

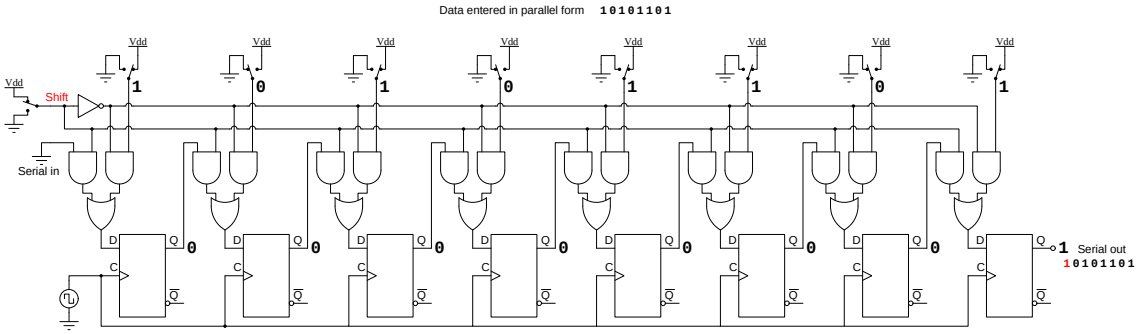

Data entered in parallel form 10101101
Vdd
Shift
Serial in
1
0
1
0
1
1
0
1
D
Q
C
Serial out
101101


Data entered in parallel form 10101101
Vdd
Shift
Serial in
1
0
1
0
1
1
0
1
D
Q
C
Q
0
1
Serial out
101101


Data entered in parallel form 10101101
Vdd
Shift
Serial in
1
0
1
0
1
1
0
1
D
Q
C
Serial out
0101101


Data entered in parallel form 10101101
Vdd
Shift
Serial in
1
0
1
0
1
1
0
1
D
Q
C
Q
0
1
Serial out
0101101


Data entered in parallel form 10101101
Vdd
Shift
Serial in
1
0
1
0
1
1
0
1
D
Q
C
Serial out
10101101


Data entered in parallel form 10101101
Vdd
Shift
Serial in
1
0
1
0
1
1
0
1
D
Q
C
Q
0
1
Serial out
10101101

Chapter 5

Questions

This learning module, along with all others in the ModEL collection, is designed to be used in an inverted instructional environment where students independently read[1] the tutorials and attempt to answer questions on their own *prior* to the instructor's interaction with them. In place of lecture[2], the instructor engages with students in Socratic-style dialogue, probing and challenging their understanding of the subject matter through inquiry.

Answers are not provided for questions within this chapter, and this is by design. Solved problems may be found in the Tutorial and Derivation chapters, instead. The goal here is *independence*, and this requires students to be challenged in ways where others cannot think for them. Remember that you always have the tools of *experimentation* and *computer simulation* (e.g. SPICE) to explore concepts!

The following lists contain ideas for Socratic-style questions and challenges. Upon inspection, one will notice a strong theme of *metacognition* within these statements: they are designed to foster a regular habit of examining one's own thoughts as a means toward clearer thinking. As such these sample questions are useful both for instructor-led discussions as well as for self-study.

[1] Technical reading is an essential academic skill for any technical practitioner to possess for the simple reason that the most comprehensive, accurate, and useful information to be found for developing technical competence is in textual form. Technical careers in general are characterized by the need for continuous learning to remain current with standards and technology, and therefore any technical practitioner who cannot read well is handicapped in their professional development. An excellent resource for educators on improving students' reading prowess through intentional effort and strategy is the book textitReading For Understanding – How Reading Apprenticeship Improves Disciplinary Learning in Secondary and College Classrooms by Ruth Schoenbach, Cynthia Greenleaf, and Lynn Murphy.

[2] Lecture is popular as a teaching method because it is easy to implement: any reasonably articulate subject matter expert can talk to students, even with little preparation. However, it is also quite problematic. A good lecture always makes complicated concepts seem easier than they are, which is bad for students because it instills a false sense of confidence in their own understanding; reading and re-articulation requires more cognitive effort and serves to verify comprehension. A culture of teaching-by-lecture fosters a debilitating dependence upon direct personal instruction, whereas the challenges of modern life demand independent and critical thought made possible only by gathering information and perspectives from afar. Information presented in a lecture is ephemeral, easily lost to failures of memory and dictation; text is forever, and may be referenced at any time.

General challenges following tutorial reading

- Summarize as much of the text as you can in one paragraph of your own words. A helpful strategy is to explain ideas as you would for an intelligent child: as simple as you can without compromising too much accuracy.
- Simplify a particular section of the text, for example a paragraph or even a single sentence, so as to capture the same fundamental idea in fewer words.
- Where did the text make the most sense to you? What was it about the text's presentation that made it clear?
- Identify where it might be easy for someone to misunderstand the text, and explain why you think it could be confusing.
- Identify any new concept(s) presented in the text, and explain in your own words.
- Identify any familiar concept(s) such as physical laws or principles applied or referenced in the text.
- Devise a proof of concept experiment demonstrating an important principle, physical law, or technical innovation represented in the text.
- Devise an experiment to disprove a plausible misconception.
- Did the text reveal any misconceptions you might have harbored? If so, describe the misconception(s) and the reason(s) why you now know them to be incorrect.
- Describe any useful problem-solving strategies applied in the text.
- Devise a question of your own to challenge a reader's comprehension of the text.

General follow-up challenges for assigned problems

- Identify where any fundamental laws or principles apply to the solution of this problem, especially before applying any mathematical techniques.
- Devise a thought experiment to explore the characteristics of the problem scenario, applying known laws and principles to mentally model its behavior.
- Describe in detail your own strategy for solving this problem. How did you identify and organized the given information? Did you sketch any diagrams to help frame the problem?
- Is there more than one way to solve this problem? Which method seems best to you?
- Show the work you did in solving this problem, even if the solution is incomplete or incorrect.
- What would you say was the most challenging part of this problem, and why was it so?
- Was any important information missing from the problem which you had to research or recall?
- Was there any extraneous information presented within this problem? If so, what was it and why did it not matter?
- Examine someone else's solution to identify where they applied fundamental laws or principles.
- Simplify the problem from its given form and show how to solve this simpler version of it. Examples include eliminating certain variables or conditions, altering values to simpler (usually whole) numbers, applying a limiting case (i.e. altering a variable to some extreme or ultimate value).
- For quantitative problems, identify the real-world meaning of all intermediate calculations: their units of measurement, where they fit into the scenario at hand. Annotate any diagrams or illustrations with these calculated values.
- For quantitative problems, try approaching it qualitatively instead, thinking in terms of "increase" and "decrease" rather than definite values.
- For qualitative problems, try approaching it quantitatively instead, proposing simple numerical values for the variables.
- Were there any assumptions you made while solving this problem? Would your solution change if one of those assumptions were altered?
- Identify where it would be easy for someone to go astray in attempting to solve this problem.
- Formulate your own problem based on what you learned solving this one.

General follow-up challenges for experiments or projects

- In what way(s) was this experiment or project easy to complete?
- Identify some of the challenges you faced in completing this experiment or project.

- Show how thorough documentation assisted in the completion of this experiment or project.
- Which fundamental laws or principles are key to this system's function?
- Identify any way(s) in which one might obtain false or otherwise misleading measurements from test equipment in this system.
- What will happen if (component X) fails (open/shorted/etc.)?
- What would have to occur to make this system unsafe?

5.1 Conceptual reasoning

These questions are designed to stimulate your analytic and synthetic thinking[3]. In a Socratic discussion with your instructor, the goal is for these questions to prompt an extended dialogue where assumptions are revealed, conclusions are tested, and understanding is sharpened. Your instructor may also pose additional questions based on those assigned, in order to further probe and refine your conceptual understanding.

Questions that follow are presented to challenge and probe your understanding of various concepts presented in the tutorial. These questions are intended to serve as a guide for the Socratic dialogue between yourself and the instructor. Your instructor's task is to ensure you have a sound grasp of these concepts, and the questions contained in this document are merely a means to this end. Your instructor may, at his or her discretion, alter or substitute questions for the benefit of tailoring the discussion to each student's needs. The only absolute requirement is that each student is challenged and assessed at a level equal to or greater than that represented by the documented questions.

It is far more important that you convey your *reasoning* than it is to simply convey a correct answer. For this reason, you should refrain from researching other information sources to answer questions. What matters here is that *you* are doing the thinking. If the answer is incorrect, your instructor will work with you to correct it through proper reasoning. A correct answer without an adequate explanation of how you derived that answer is unacceptable, as it does not aid the learning or assessment process.

You will note a conspicuous lack of answers given for these conceptual questions. Unlike standard textbooks where answers to every other question are given somewhere toward the back of the book, here in these learning modules students must rely on other means to check their work. The best way by far is to debate the answers with fellow students and also with the instructor during the Socratic dialogue sessions intended to be used with these learning modules. Reasoning through challenging questions with other people is an excellent tool for developing strong reasoning skills.

Another means of checking your conceptual answers, where applicable, is to use circuit simulation software to explore the effects of changes made to circuits. For example, if one of these conceptual questions challenges you to predict the effects of altering some component parameter in a circuit, you may check the validity of your work by simulating that same parameter change within software and seeing if the results agree.

[3] *Analytical* thinking involves the "disassembly" of an idea into its constituent parts, analogous to dissection. *Synthetic* thinking involves the "assembly" of a new idea comprised of multiple concepts, analogous to construction. Both activities are high-level cognitive skills, extremely important for effective problem-solving, necessitating frequent challenge and regular practice to fully develop.

5.1.1 Reading outline and reflections

"Reading maketh a full man; conference a ready man; and writing an exact man" – Francis Bacon

Francis Bacon's advice is a blueprint for effective education: reading provides the learner with knowledge, writing focuses the learner's thoughts, and critical dialogue equips the learner to confidently communicate and apply their learning. Independent acquisition and application of knowledge is a powerful skill, well worth the effort to cultivate. To this end, students should read these educational resources closely, write their own outline and reflections on the reading, and discuss in detail their findings with classmates and instructor(s). You should be able to do all of the following after reading any instructional text:

☑ Briefly OUTLINE THE TEXT, as though you were writing a detailed Table of Contents. Feel free to rearrange the order if it makes more sense that way. Prepare to articulate these points in detail and to answer questions from your classmates and instructor. Outlining is a good self-test of thorough reading because you cannot outline what you have not read or do not comprehend.

☑ Demonstrate ACTIVE READING STRATEGIES, including verbalizing your impressions as you read, simplifying long passages to convey the same ideas using fewer words, annotating text and illustrations with your own interpretations, working through mathematical examples shown in the text, cross-referencing passages with relevant illustrations and/or other passages, identifying problem-solving strategies applied by the author, etc. Technical reading is a special case of problem-solving, and so these strategies work precisely because they help solve any problem: paying attention to your own thoughts (metacognition), eliminating unnecessary complexities, identifying what makes sense, paying close attention to details, drawing connections between separated facts, and noting the successful strategies of others.

☑ Identify IMPORTANT THEMES, especially GENERAL LAWS and PRINCIPLES, expounded in the text and express them in the simplest of terms as though you were teaching an intelligent child. This emphasizes connections between related topics and develops your ability to communicate complex ideas to anyone.

☑ Form YOUR OWN QUESTIONS based on the reading, and then pose them to your instructor and classmates for their consideration. Anticipate both correct and incorrect answers, the incorrect answer(s) assuming one or more plausible misconceptions. This helps you view the subject from different perspectives to grasp it more fully.

☑ Devise EXPERIMENTS to test claims presented in the reading, or to disprove misconceptions. Predict possible outcomes of these experiments, and evaluate their meanings: what result(s) would confirm, and what would constitute disproof? Running mental simulations and evaluating results is essential to scientific and diagnostic reasoning.

☑ Specifically identify any points you found CONFUSING. The reason for doing this is to help diagnose misconceptions and overcome barriers to learning.

5.1.2 Foundational concepts

Correct analysis and diagnosis of electric circuits begins with a proper understanding of some basic concepts. The following is a list of some important concepts referenced in this module's full tutorial. Define each of them in your own words, and be prepared to illustrate each of these concepts with a description of a practical example and/or a live demonstration.

Truth table

Logic function

NOR function

NAND function

Positive feedback

Active-low versus Active-high

Latch behavior

Flip-flop behavior

Set-up time

Hold time

Memory

Register

Bit-shifting

Serial versus Parallel data

Stack

Frequency division

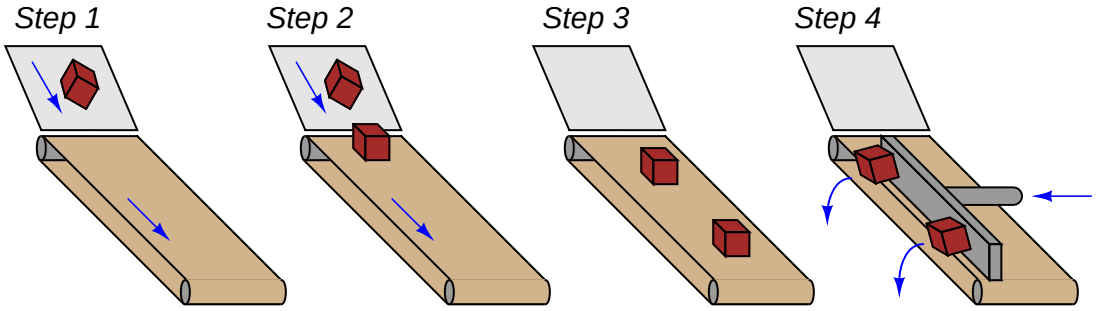
Initialization

Schmitt trigger

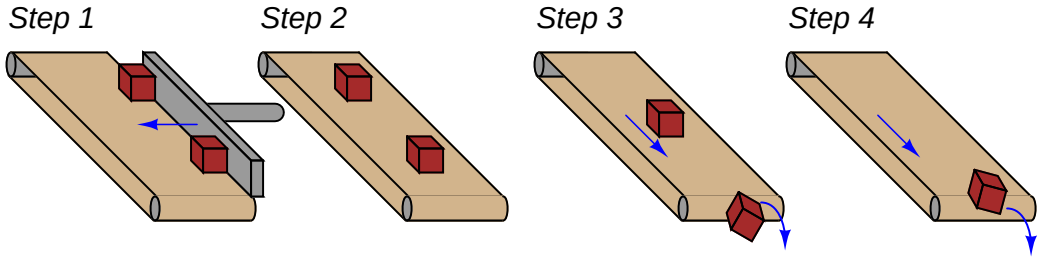
5.1.3 Conveyor belt analogies

A helpful analogy for a shift register is a *conveyor belt*. Examine the following illustrations, each one showing a single conveyor belt at four different times, and determine which of the following shift register operations (e.g. serial-in, serial-out, serial-in, parallel-out, etc.) these sequences represent:

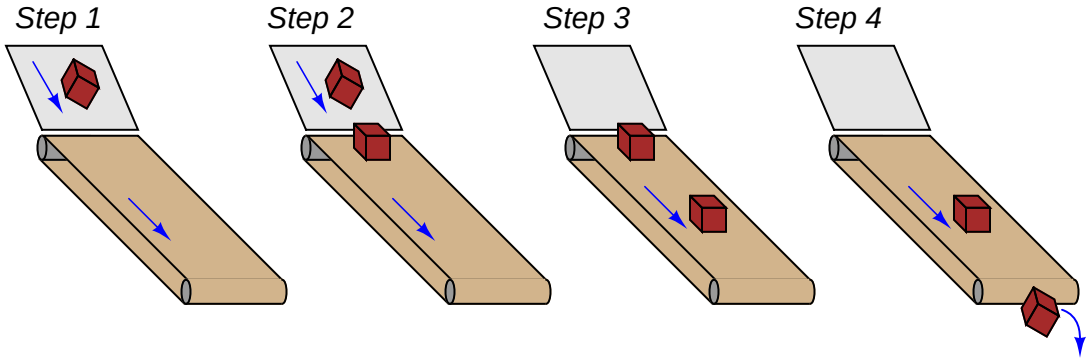
Conveyor #1



Conveyor #2



Conveyor #3



- Parallel-in, serial-out

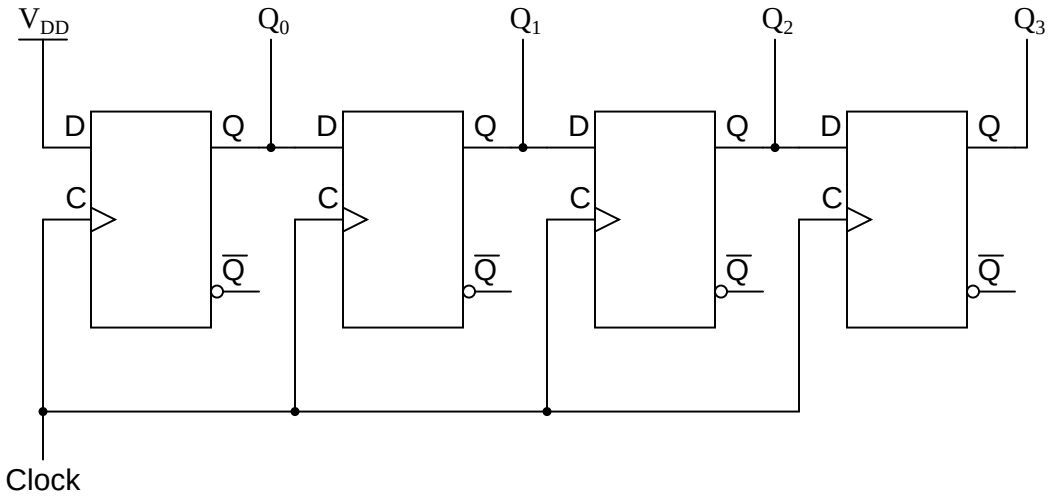
- Parallel-in, parallel-out
- Serial-in, serial-out
- Serial-in, parallel-out

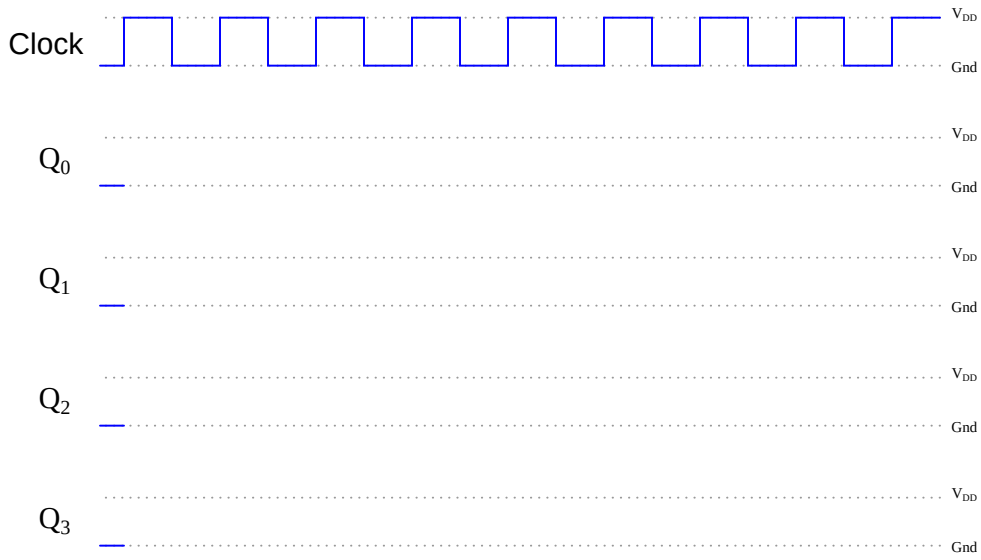
Challenges

- Identify a practical purpose for serial data transmission, in lieu of parallel.

5.1.4 Timing diagram for a simple shift register

Complete the timing diagram for this circuit, assuming all Q outputs begin in the low state:





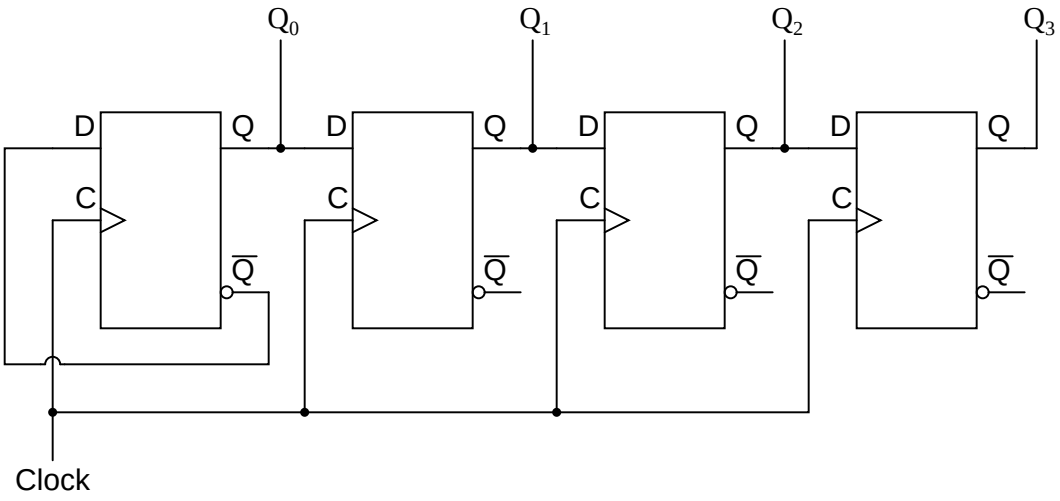
Challenges

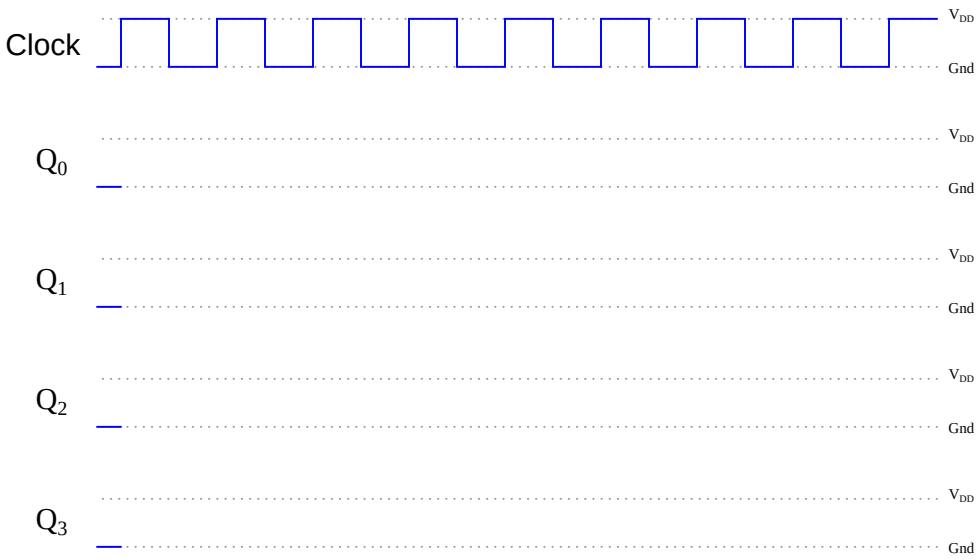
- How would the timing diagram differ if all flip-flops began in the *set* state?

- Explain why the "high" state at the D input of the first flip-flop does not ripple through *all* the flip-flops at the first clock pulse.

5.1.5 Timing diagram for another simple shift register

Complete the timing diagram for this circuit, assuming all Q outputs begin in the low state:



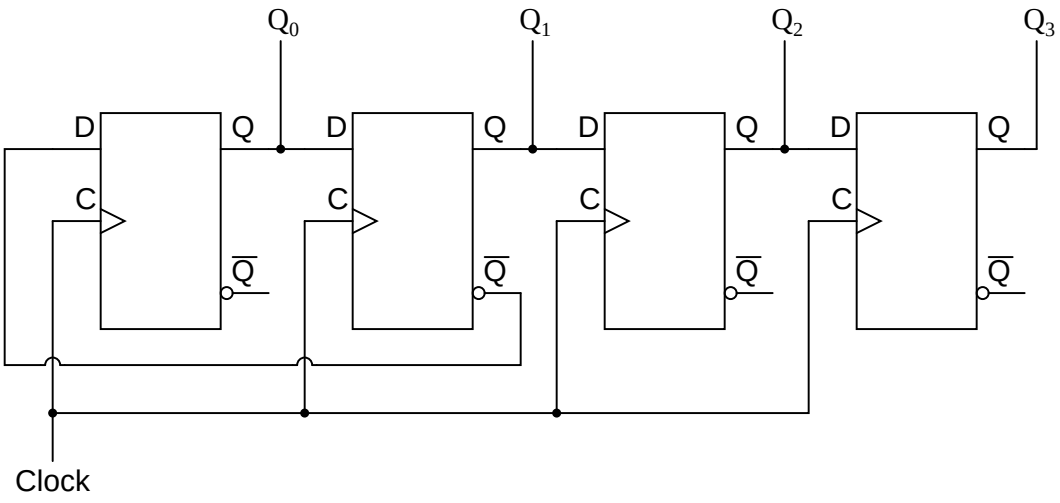


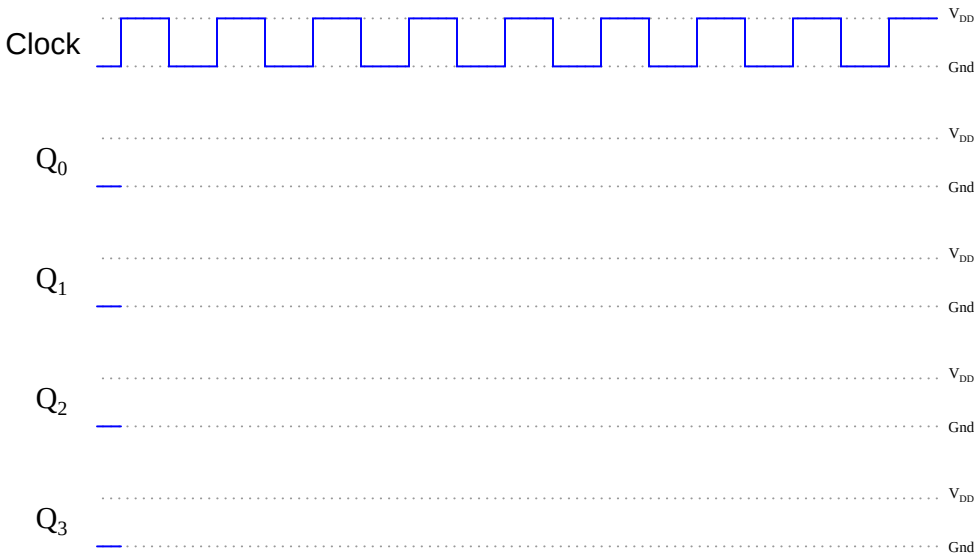
Challenges

- How would the timing diagram differ if all flip-flops began in the *set* state?

5.1.6 Timing diagram for yet another simple shift register

Complete the timing diagram for this circuit, showing propagation delays for all flip-flops (delay times much less than the width of a clock pulse), assuming all Q outputs begin in the low state:





Challenges

- How would the timing diagram differ if all flip-flops began in the *set* state?

- Explain how this circuit is similar to, yet not exactly the same as, a Johnson counter.

5.1.7 Schematic diagram for a five-bit shift register

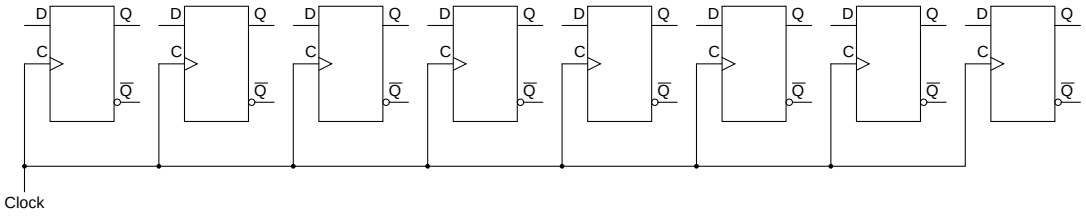
Draw the schematic diagram for a five-bit serial-in/serial-out shift register circuit, and be prepared to give a brief explanation of how it functions.

Challenges

- If we wished to output data from this shift register circuit in parallel form, where would we make the connections?
- Identify how this shift register circuit could be equipped with a *master clear* pushbutton.

5.1.8 Altering shift direction

Draw the necessary connecting wires between flip-flops so that serial data is shifted from *right to left* instead of left to right as you may be accustomed to seeing in a shift register schematic:



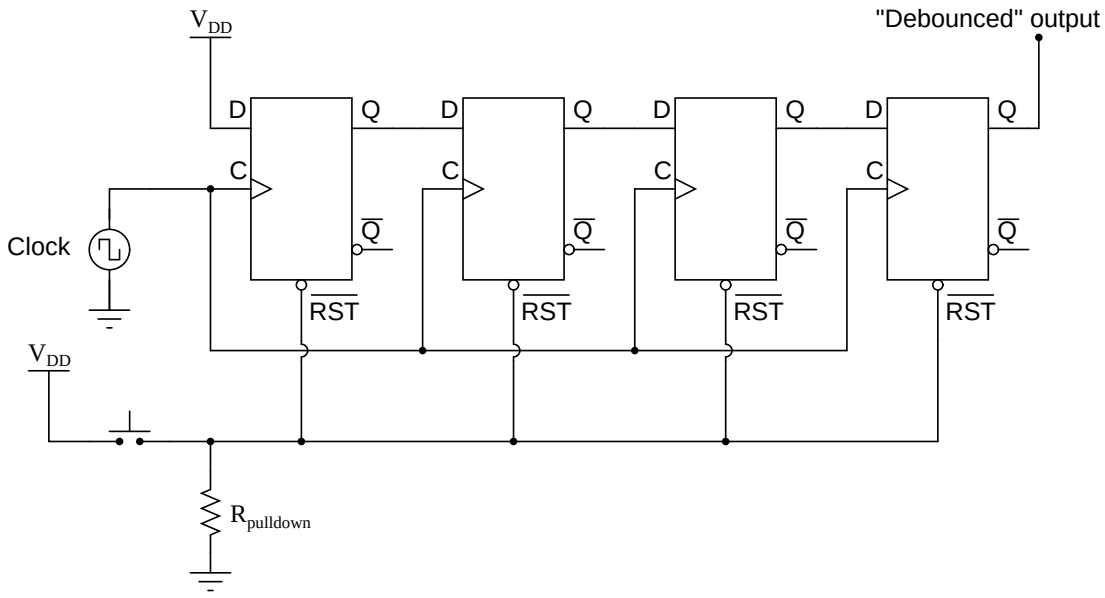
Be sure to also note where data enters this shift register, and where data exits.

Challenges

- If we wished to output data from this shift register circuit in parallel form, where would we make the connections?
- Identify how this shift register circuit could be equipped with a *master clear* pushbutton.

5.1.9 Switch debouncer

Describe the phenomenon of *switch bounce*, and then explain how this circuit eliminates it:



Challenges

- How would you empirically test an appropriate clock frequency for this circuit?
- How could you predict an appropriate clock frequency for this circuit prior to building it?

5.1.10 Sequenced water fountain

A mechanically inclined friend of yours wishes to build an automated water fountain, where ten water jets are turned on in sequence, one at a time, for artistic effect. Each water jet is controlled by a solenoid valve, energized by 120 volt AC line power.

Your friend understands how to wire up the solenoid valves and build all the plumbing to make the fountain work. He also understands how to interpose power to the solenoid valve coils using small relays, so a digital control circuit operating at a low DC supply voltage will be able to energize the valves. The only problem is, this friend of yours does not know how to build a circuit to do the sequencing. How do you turn on one out of ten outputs at a time, in sequence?

Challenges

- Suppose the solenoid valves were rated for 24 VDC instead of 120 VAC. How would this alter your design, if at all?
- Explain how to obtain more than one sequence pattern with your sequencing circuit.

5.2 Quantitative reasoning

These questions are designed to stimulate your computational thinking. In a Socratic discussion with your instructor, the goal is for these questions to reveal your mathematical approach(es) to problem-solving so that good technique and sound reasoning may be reinforced. Your instructor may also pose additional questions based on those assigned, in order to observe your problem-solving firsthand.

Mental arithmetic and estimations are strongly encouraged for all calculations, because without these abilities you will be unable to readily detect errors caused by calculator misuse (e.g. keystroke errors).

You will note a conspicuous lack of answers given for these quantitative questions. Unlike standard textbooks where answers to every other question are given somewhere toward the back of the book, here in these learning modules students must rely on other means to check their work. My advice is to use circuit simulation software such as SPICE to check the correctness of quantitative answers. Refer to those learning modules within this collection focusing on SPICE to see worked examples which you may use directly as practice problems for your own study, and/or as templates you may modify to run your own analyses and generate your own practice problems.

Completely worked example problems found in the Tutorial may also serve as "test cases[4]" for gaining proficiency in the use of circuit simulation software, and then once that proficiency is gained you will never need to rely[5] on an answer key!

[4] In other words, set up the circuit simulation software to analyze the same circuit examples found in the Tutorial. If the simulated results match the answers shown in the Tutorial, it confirms the simulation has properly run. If the simulated results disagree with the Tutorial's answers, something has been set up incorrectly in the simulation software. Using every Tutorial as practice in this way will quickly develop proficiency in the use of circuit simulation software.

[5] This approach is perfectly in keeping with the instructional philosophy of these learning modules: *teaching students to be self-sufficient thinkers*. Answer keys can be useful, but it is even more useful to your long-term success to have a set of tools on hand for checking your own work, because once you have left school and are on your own, there will no longer be "answer keys" available for the problems you will have to solve.

5.2.1 Miscellaneous physical constants

Note: constants shown in **bold** type are *exact*, not approximations. Values inside of parentheses show one standard deviation (σ) of uncertainty in the final digits: for example, the magnetic permeability of free space value given as $1.25663706212(19) \times 10^{-6}$ H/m represents a center value (i.e. the location parameter) of $1.25663706212 \times 10^{-6}$ Henrys per meter with one standard deviation of uncertainty equal to $0.0000000000019 \times 10^{-6}$ Henrys per meter.

Avogadro's number (N_A) = **6.02214076** $\times 10^{23}$ **per mole** (mol^{-1})

Boltzmann's constant (k) = **1.380649** $\times 10^{-23}$ **Joules per Kelvin** (J/K)

Electronic charge (e) = **1.602176634** $\times 10^{-19}$ **Coulomb** (C)

Faraday constant (F) = **96,485.33212...** $\times 10^{4}$ **Coulombs per mole** (C/mol)

Magnetic permeability of free space (μ_0) = $1.25663706212(19) \times 10^{-6}$ Henrys per meter (H/m)

Electric permittivity of free space (ϵ_0) = $8.8541878128(13) \times 10^{-12}$ Farads per meter (F/m)

Characteristic impedance of free space (Z_0) = 376.730313668(57) Ohms (Ω)

Gravitational constant (G) = $6.67430(15) \times 10^{-11}$ cubic meters per kilogram-seconds squared ($m^3/kg\text{-}s^2$)

Molar gas constant (R) = **8.314462618... Joules per mole-Kelvin** (J/mol-K) = 0.08205746(14) liters-atmospheres per mole-Kelvin

Planck constant (h) = **6.62607015** $\times 10^{-34}$ **joule-seconds** (J-s)

Stefan-Boltzmann constant (σ) = **5.670374419...** $\times 10^{-8}$ **Watts per square meter-Kelvin**^{4} ($W/m^2 \cdot K^4$)

Speed of light in a vacuum (c) = **299,792,458 meters per second** (m/s) = 186282.4 miles per second (mi/s)

Note: All constants taken from NIST data "Fundamental Physical Constants – Complete Listing", from `http://physics.nist.gov/constants`, National Institute of Standards and Technology (NIST), 2018 CODATA Adjustment.

5.2.2 Introduction to spreadsheets

A powerful computational tool you are encouraged to use in your work is a *spreadsheet*. Available on most personal computers (e.g. Microsoft Excel), *spreadsheet* software performs numerical calculations based on number values and formulae entered into cells of a grid. This grid is typically arranged as lettered columns and numbered rows, with each cell of the grid identified by its column/row coordinates (e.g. cell `B3`, cell `A8`). Each cell may contain a string of text, a number value, or a mathematical formula. The spreadsheet automatically updates the results of all mathematical formulae whenever the entered number values are changed. This means it is possible to set up a spreadsheet to perform a series of calculations on entered data, and those calculations will be re-done by the computer any time the data points are edited in any way.

For example, the following spreadsheet calculates average speed based on entered values of distance traveled and time elapsed:

	A	B	C	D
1	Distance traveled	46.9	Kilometers	
2	Time elapsed	1.18	Hours	
3	Average speed	*= B1 / B2*	km/h	
4				
5				

Text labels contained in cells `A1` through `A3` and cells `C1` through `C3` exist solely for readability and are not involved in any calculations. Cell `B1` contains a sample distance value while cell `B2` contains a sample time value. The formula for computing speed is contained in cell `B3`. Note how this formula begins with an "equals" symbol (`=`), references the values for distance and speed by lettered column and numbered row coordinates (`B1` and `B2`), and uses a forward slash symbol for division (`/`). The coordinates `B1` and `B2` function as *variables*[6] would in an algebraic formula.

When this spreadsheet is executed, the numerical value `39.74576` will appear in cell `B3` rather than the formula `= B1 / B2`, because 39.74576 is the computed speed value given 46.9 kilometers traveled over a period of 1.18 hours. If a different numerical value for distance is entered into cell `B1` or a different value for time is entered into cell `B2`, cell `B3`'s value will automatically update. All you need to do is set up the given values and any formulae into the spreadsheet, and the computer will do all the calculations for you.

Cell `B3` may be referenced by other formulae in the spreadsheet if desired, since it is a variable just like the given values contained in `B1` and `B2`. This means it is possible to set up an entire chain of calculations, one dependent on the result of another, in order to arrive at a final value. The arrangement of the given data and formulae need not follow any pattern on the grid, which means you may place them anywhere.

[6] Spreadsheets may also provide means to attach text labels to cells for use as variable names (Microsoft Excel simply calls these labels "names"), but for simple spreadsheets such as those shown here it's usually easier just to use the standard coordinate naming for each cell.

Common[7] arithmetic operations available for your use in a spreadsheet include the following:

- Addition (`+`)
- Subtraction (`-`)
- Multiplication (`*`)
- Division (`/`)
- Powers (`^`)
- Square roots (`sqrt()`)
- Logarithms (`ln()` , `log10()`)

Parentheses may be used to ensure[8] proper order of operations within a complex formula. Consider this example of a spreadsheet implementing the *quadratic formula*, used to solve for roots of a polynomial expression in the form of $ax^2 + bx + c$:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

	A	B
1	x_1	= (-B4 + sqrt((B4^2) - (4*B3*B5))) / (2*B3)
2	x_2	= (-B4 - sqrt((B4^2) - (4*B3*B5))) / (2*B3)
3	a =	9
4	b =	5
5	c =	-2

This example is configured to compute roots[9] of the polynomial $9x^2 + 5x - 2$ because the values of 9, 5, and -2 have been inserted into cells `B3`, `B4`, and `B5`, respectively. Once this spreadsheet has been built, though, it may be used to calculate the roots of *any* second-degree polynomial expression simply by entering the new a, b, and c coefficients into cells `B3` through `B5`. The numerical values appearing in cells `B1` and `B2` will be automatically updated by the computer immediately following any changes made to the coefficients.

[7] Modern spreadsheet software offers a bewildering array of mathematical functions you may use in your computations. I recommend you consult the documentation for your particular spreadsheet for information on operations other than those listed here.

[8] Spreadsheet programs, like text-based programming languages, are designed to follow standard order of operations by default. However, my personal preference is to use parentheses even where strictly unnecessary just to make it clear to any other person viewing the formula what the intended order of operations is.

[9] Reviewing some algebra here, a *root* is a value for x that yields an overall value of zero for the polynomial. For this polynomial ($9x^2 + 5x - 2$) the two roots happen to be $x = 0.269381$ and $x = -0.82494$, with these values displayed in cells `B1` and `B2`, respectively upon execution of the spreadsheet.

Alternatively, one could break up the long quadratic formula into smaller pieces like this:

$$y = \sqrt{b^2 - 4ac} \qquad z = 2a$$

$$x = \frac{-b \pm y}{z}$$

	A	B	C
1	x_1	= (-B4 + C1) / C2	= sqrt((B4^2) - (4*B3*B5))
2	x_2	= (-B4 - C1) / C2	= 2*B3
3	a =	9	
4	b =	5	
5	c =	-2	

Note how the square-root term (y) is calculated in cell C1, and the denominator term (z) in cell C2. This makes the two final formulae (in cells B1 and B2) simpler to interpret. The positioning of all these cells on the grid is completely arbitrary[10] – all that matters is that they properly reference each other in the formulae.

Spreadsheets are particularly useful for situations where the same set of calculations representing a circuit or other system must be repeated for different initial conditions. The power of a spreadsheet is that it automates what would otherwise be a tedious set of calculations. One specific application of this is to simulate the effects of various components within a circuit failing with abnormal values (e.g. a shorted resistor simulated by making its value nearly zero; an open resistor simulated by making its value extremely large). Another application is analyzing the behavior of a circuit design given new components that are out of specification, and/or aging components experiencing drift over time.

[10] My personal preference is to locate all the "given" data in the upper-left cells of the spreadsheet grid (each data point flanked by a sensible name in the cell to the left and units of measurement in the cell to the right as illustrated in the first distance/time spreadsheet example), sometimes coloring them in order to clearly distinguish which cells contain entered data versus which cells contain computed results from formulae. I like to place all formulae in cells below the given data, and try to arrange them in logical order so that anyone examining my spreadsheet will be able to figure out *how* I constructed a solution. This is a general principle I believe all computer programmers should follow: *document and arrange your code to make it easy for other people to learn from it.*

5.2.3 Shifting binary integers

Suppose a binary integer number is parallel-loaded into a shift register. The shift register is then commanded to "shift right" for one clock pulse. How does the value of the shifted integer compare to the number originally loaded in, assuming that the MSB is on the far-left flip-flop of the shift register?

Challenges

- Explain how a shift register could be used to *quadruple* the value of a binary integer.
- Microprocessors typically offer an instruction to shift the contents of a register either right or left. Explain how this operation might prove useful in programming arithmetic functions.

5.2.4 Frequency divider design

Design a shift register circuit to divide the frequency of a clock signal by a factor of 15.

Challenges

- Explain why a Johnson counter could *not* be used for this purpose.

5.3 Diagnostic reasoning

These questions are designed to stimulate your deductive and inductive thinking, where you must apply general principles to specific scenarios (deductive) and also derive conclusions about the failed circuit from specific details (inductive). In a Socratic discussion with your instructor, the goal is for these questions to reinforce your recall and use of general circuit principles and also challenge your ability to integrate multiple symptoms into a sensible explanation of what's wrong in a circuit. Your instructor may also pose additional questions based on those assigned, in order to further challenge and sharpen your diagnostic abilities.

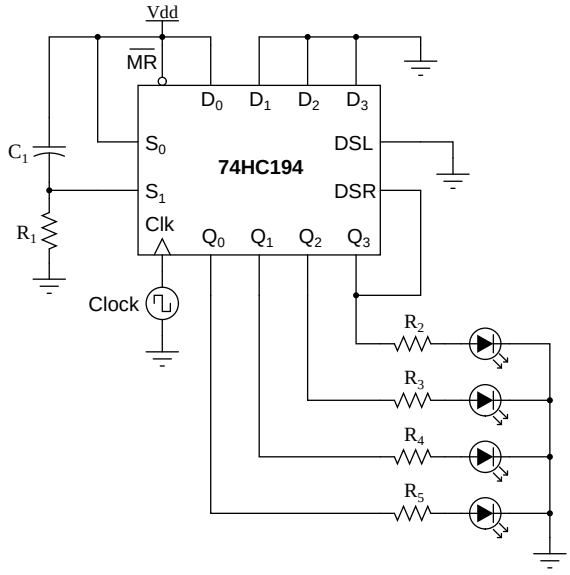
As always, your goal is to fully *explain* your analysis of each problem. Simply obtaining a correct answer is not good enough – you must also demonstrate sound reasoning in order to successfully complete the assignment. Your instructor's responsibility is to probe and challenge your understanding of the relevant principles and analytical processes in order to ensure you have a strong foundation upon which to build further understanding.

You will note a conspicuous lack of answers given for these diagnostic questions. Unlike standard textbooks where answers to every other question are given somewhere toward the back of the book, here in these learning modules students must rely on other means to check their work. The best way by far is to debate the answers with fellow students and also with the instructor during the Socratic dialogue sessions intended to be used with these learning modules. Reasoning through challenging questions with other people is an excellent tool for developing strong reasoning skills.

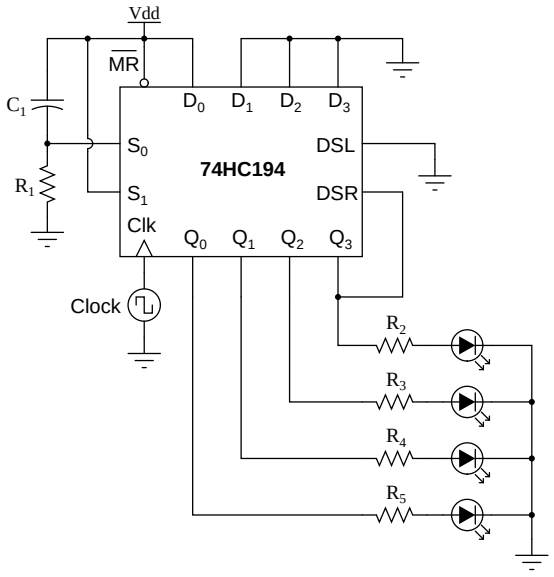
Another means of checking your diagnostic answers, where applicable, is to use circuit simulation software to explore the effects of faults placed in circuits. For example, if one of these diagnostic questions requires that you predict the effect of an open or a short in a circuit, you may check the validity of your work by simulating that same fault (substituting a very high resistance in place of that component for an open, and substituting a very low resistance for a short) within software and seeing if the results agree.

5.3.1 Faulty LED sequencer design

This shift register circuit energizes one LED at a time (beginning with the bottom LED at power-up), in a rotating pattern that moves at the pace of the clock:



A technician decides to reverse the direction of pattern motion, and alters the circuit as such:



Unfortunately, this does not work as planned. Now, the bottom LED blinks once upon power-up, then all LEDs remain off. What did the technician do that was incorrect? What needs to be done

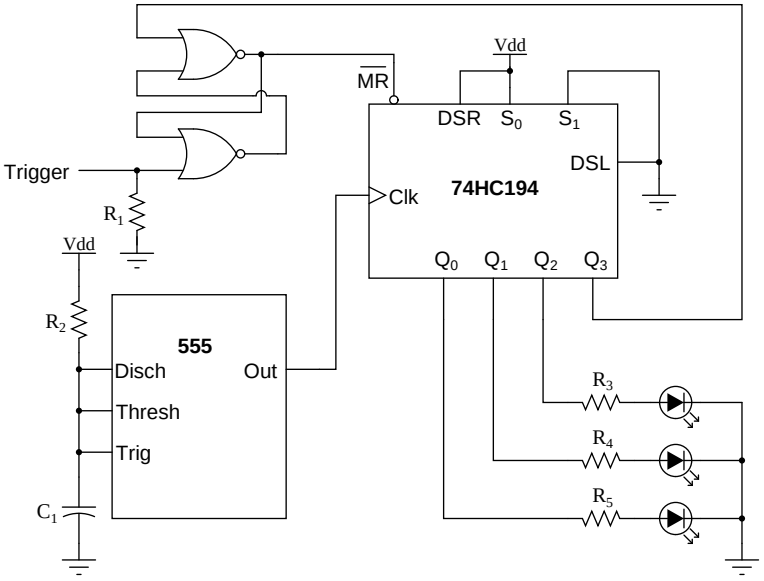
to fix the problem?

Challenges

- Modify this circuit design to drive 120 VAC incandescent lamps instead of LEDs.

5.3.2 Sequential tail-light blinker

This shift register circuit produces a sequential light pattern reminiscent of the old Mercury Cougar tail-lights: first one LED energizes, then two LEDs energize, and then all three LEDs energize before all de-energizing and repeating the sequence. The 74HC194 shift register circuit is set to always operate in the "shift right" mode with the shift-right serial input (DSR) tied high, the master reset ($\overline{MR}$) input used to set all output lines to a low state at the end of each cycle:



The sequential light pattern is supposed to begin whenever the "Trigger" input momentarily goes high. Unfortunately, something has failed in this circuit which is preventing any of the LEDs from energizing. No blinking light sequence ensues, no matter what the state of the "Trigger" input.

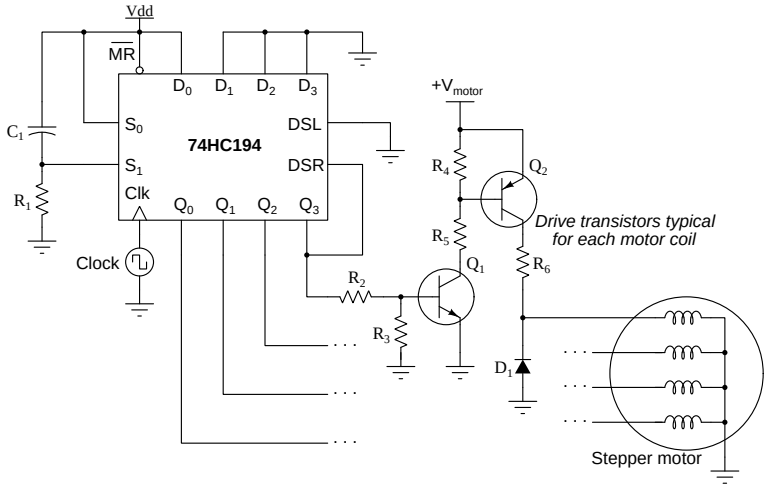
Identify some likely failures in this circuit that could cause this to happen, other than a lack of power supply voltage. Explain why each of your proposed faults would cause the problem, and also identify how you would isolate each fault using test equipment.

Challenges

- Identify multiple ways to speed up the LED's blinking sequence.
- Modify this circuit design to drive 120 VAC incandescent lamps instead of LEDs.

5.3.3 Troubleshooting a failed stepper motor drive

This shift register circuit drives the four coils of a unipolar stepper motor, one at a time, in a rotating pattern that moves at the pace of the clock. The drive transistor circuitry (Q_1, Q_2, and resistors R_2 through R_6) are shown only for one of the four coils. The other three shift register outputs have identical drive circuits connected to the respective motor coils:



Suppose this stepper motor circuit worked just fine for several years, then suddenly stopped working. Explain where you would take your first few measurements to isolate the problem, and why you would measure there.

Challenges

- Identify the purpose of D_1.
- Identify multiple ways to speed up the stepper motor's rotation.
- Modify this circuit design to incorporate a start/stop toggle switch.
- Modify this circuit design to incorporate MOSFETs instead of BJTs.

Appendix A

Problem-Solving Strategies

The ability to solve complex problems is arguably one of the most valuable skills one can possess, and this skill is particularly important in any science-based discipline.

- Study principles, not procedures. Don't be satisfied with merely knowing how to compute solutions – learn *why* those solutions work.
- Identify what it is you need to solve, identify all relevant data, identify all units of measurement, identify any general principles or formulae linking the given information to the solution, and then identify any "missing pieces" to a solution. Annotate all diagrams with this data.
- Sketch a diagram to help visualize the problem. When building a real system, always devise a plan for that system and analyze its function *before* constructing it.
- Follow the units of measurement and meaning of every calculation. If you are ever performing mathematical calculations as part of a problem-solving procedure, and you find yourself unable to apply each and every intermediate result to some aspect of the problem, it means you don't understand what you are doing. Properly done, every mathematical result should have practical meaning for the problem, and not just be an abstract number. You should be able to identify the proper units of measurement for each and every calculated result, and show where that result fits into the problem.
- Perform "thought experiments" to explore the effects of different conditions for theoretical problems. When troubleshooting real systems, perform *diagnostic tests* rather than visually inspecting for faults, the best diagnostic test being the one giving you the most information about the nature and/or location of the fault with the fewest steps.
- Simplify the problem until the solution becomes obvious, and then use that obvious case as a model to follow in solving the more complex version of the problem.
- Check for exceptions to see if your solution is incorrect or incomplete. A good solution will work for *all* known conditions and criteria. A good example of this is the process of testing scientific hypotheses: the task of a scientist is not to find support for a new idea, but rather to *challenge* that new idea to see if it holds up under a battery of tests. The philosophical

principle of *reductio ad absurdum* (i.e. disproving a general idea by finding a specific case where it fails) is useful here.

- Work "backward" from a hypothetical solution to a new set of given conditions.
- Add quantities to problems that are qualitative in nature, because sometimes a little math helps illuminate the scenario.
- Sketch graphs illustrating how variables relate to each other. These may be quantitative (i.e. with realistic number values) or qualitative (i.e. simply showing increases and decreases).
- Treat quantitative problems as qualitative in order to discern the relative magnitudes and/or directions of change of the relevant variables. For example, try determining what happens if a certain variable were to increase or decrease before attempting to precisely calculate quantities: how will each of the dependent variables respond, by increasing, decreasing, or remaining the same as before?
- Consider limiting cases. This works especially well for qualitative problems where you need to determine which direction a variable will change. Take the given condition and magnify that condition to an extreme degree as a way of simplifying the direction of the system's response.
- Check your work. This means regularly testing your conclusions to see if they make sense. This does *not* mean repeating the same steps originally used to obtain the conclusion(s), but rather to use some other means to check validity. Simply repeating procedures often leads to *repeating the same errors* if any were made, which is why alternative paths are better.

Appendix B

Instructional philosophy

"The unexamined circuit is not worth energizing" – Socrates (if he had taught electricity)

These learning modules, although useful for self-study, were designed to be used in a formal learning environment where a subject-matter expert challenges students to digest the content and exercise their critical thinking abilities in the answering of questions and in the construction and testing of working circuits.

The following principles inform the instructional and assessment philosophies embodied in these learning modules:

- The first goal of education is to enhance clear and independent thought, in order that every student reach their fullest potential in a highly complex and inter-dependent world. Robust reasoning is *always* more important than particulars of any subject matter, because its application is universal.
- Literacy is fundamental to independent learning and thought because text continues to be the most efficient way to communicate complex ideas over space and time. Those who cannot read with ease are limited in their ability to acquire knowledge and perspective.
- Articulate communication is fundamental to work that is complex and interdisciplinary.
- Faulty assumptions and poor reasoning are best corrected through challenge, not presentation. The rhetorical technique of *reductio ad absurdum* (disproving an assertion by exposing an absurdity) works well to discipline student's minds, not only to correct the problem at hand but also to learn how to detect and correct future errors.
- Important principles should be repeatedly explored and widely applied throughout a course of study, not only to reinforce their importance and help ensure their mastery, but also to showcase the interconnectedness and utility of knowledge.

These learning modules were expressly designed to be used in an "inverted" teaching environment[1] where students first read the introductory and tutorial chapters on their own, then individually attempt to answer the questions and construct working circuits according to the experiment and project guidelines. The instructor never lectures, but instead meets regularly with each individual student to review their progress, answer questions, identify misconceptions, and challenge the student to new depths of understanding through further questioning. Regular meetings between instructor and student should resemble a Socratic[2] dialogue, where questions serve as scalpels to dissect topics and expose assumptions. The student passes each module only after consistently demonstrating their ability to logically analyze and correctly apply all major concepts in each question or project/experiment. The instructor must be vigilant in probing each student's understanding to ensure they are truly *reasoning* and not just *memorizing*. This is why "Challenge" points appear throughout, as prompts for students to think deeper about topics and as starting points for instructor queries. Sometimes these challenge points require additional knowledge that hasn't been covered in the series to answer in full. This is okay, as the major purpose of the Challenges is to stimulate analysis and synthesis on the part of each student.

The instructor must possess enough mastery of the subject matter and awareness of students' reasoning to generate their own follow-up questions to practically any student response. Even completely correct answers given by the student should be challenged by the instructor for the purpose of having students practice articulating their thoughts and defending their reasoning. Conceptual errors committed by the student should be exposed and corrected not by direct instruction, but rather by reducing the errors to an absurdity[3] through well-chosen questions and thought experiments posed by the instructor. Becoming proficient at this style of instruction requires time and dedication, but the positive effects on critical thinking for both student and instructor are spectacular.

An inspection of these learning modules reveals certain unique characteristics. One of these is a bias toward thorough explanations in the tutorial chapters. Without a live instructor to explain concepts and applications to students, the text itself must fulfill this role. This philosophy results in lengthier explanations than what you might typically find in a textbook, each step of the reasoning process fully explained, including footnotes addressing common questions and concerns students raise while learning these concepts. Each tutorial seeks to not only explain each major concept in sufficient detail, but also to explain the logic of each concept and how each may be developed

[1] In a traditional teaching environment, students first encounter new information via *lecture* from an expert, and then independently apply that information via *homework*. In an "inverted" course of study, students first encounter new information via *homework*, and then independently apply that information under the scrutiny of an expert. The expert's role in lecture is to simply *explain*, but the expert's role in an inverted session is to *challenge*, *critique*, and if necessary *explain* where gaps in understanding still exist.

[2] Socrates is a figure in ancient Greek philosophy famous for his unflinching style of questioning. Although he authored no texts, he appears as a character in Plato's many writings. The essence of Socratic philosophy is to leave no question unexamined and no point of view unchallenged. While purists may argue a topic such as electric circuits is too narrow for a true Socratic-style dialogue, I would argue that the essential thought processes involved with scientific reasoning on *any* topic are not far removed from the Socratic ideal, and that students of electricity and electronics would do very well to challenge assumptions, pose thought experiments, identify fallacies, and otherwise employ the arsenal of critical thinking skills modeled by Socrates.

[3] This rhetorical technique is known by the Latin phrase *reductio ad absurdum*. The concept is to expose errors by counter-example, since only one solid counter-example is necessary to disprove a universal claim. As an example of this, consider the common misconception among beginning students of electricity that voltage cannot exist without current. One way to apply *reductio ad absurdum* to this statement is to ask how much current passes through a fully-charged battery connected to nothing (i.e. a clear example of voltage existing without current).

from "first principles". Again, this reflects the goal of developing clear and independent thought in students' minds, by showing how clear and logical thought was used to forge each concept. Students benefit from witnessing a model of clear thinking in action, and these tutorials strive to be just that.

Another characteristic of these learning modules is a lack of step-by-step instructions in the Project and Experiment chapters. Unlike many modern workbooks and laboratory guides where step-by-step instructions are prescribed for each experiment, these modules take the approach that students must learn to closely read the tutorials and apply their own reasoning to identify the appropriate experimental steps. Sometimes these steps are plainly declared in the text, just not as a set of enumerated points. At other times certain steps are implied, an example being assumed competence in test equipment use where the student should not need to be told *again* how to use their multimeter because that was thoroughly explained in previous lessons. In some circumstances no steps are given at all, leaving the entire procedure up to the student.

This lack of prescription is not a flaw, but rather a feature. Close reading and clear thinking are foundational principles of this learning series, and in keeping with this philosophy all activities are designed to *require* those behaviors. Some students may find the lack of prescription frustrating, because it demands more from them than what their previous educational experiences required. This frustration should be interpreted as an unfamiliarity with autonomous thinking, a problem which must be corrected if the student is ever to become a self-directed learner and effective problem-solver. Ultimately, the need for students to read closely and think clearly is more important both in the near-term and far-term than any specific facet of the subject matter at hand. If a student takes longer than expected to complete a module because they are forced to outline, digest, and reason on their own, so be it. The future gains enjoyed by developing this mental discipline will be well worth the additional effort and delay.

Another feature of these learning modules is that they do not treat topics in isolation. Rather, important concepts are introduced early in the series, and appear repeatedly as stepping-stones toward other concepts in subsequent modules. This helps to avoid the "compartmentalization" of knowledge, demonstrating the inter-connectedness of concepts and simultaneously reinforcing them. Each module is fairly complete in itself, reserving the beginning of its tutorial to a review of foundational concepts.

This methodology of assigning text-based modules to students for digestion and then using Socratic dialogue to assess progress and hone students' thinking was developed over a period of several years by the author with his Electronics and Instrumentation students at the two-year college level. While decidedly unconventional and sometimes even unsettling for students accustomed to a more passive lecture environment, this instructional philosophy has proven its ability to convey conceptual mastery, foster careful analysis, and enhance employability so much better than lecture that the author refuses to ever teach by lecture again.

Problems which often go undiagnosed in a lecture environment are laid bare in this "inverted" format where students must articulate and logically defend their reasoning. This, too, may be unsettling for students accustomed to lecture sessions where the instructor cannot tell for sure who comprehends and who does not, and this vulnerability necessitates sensitivity on the part of the "inverted" session instructor in order that students never feel discouraged by having their errors exposed. *Everyone* makes mistakes from time to time, and learning is a lifelong process! Part of the instructor's job is to build a culture of learning among the students where errors are not seen as shameful, but rather as opportunities for progress.

To this end, instructors managing courses based on these modules should adhere to the following principles:

- Student questions are always welcome and demand thorough, honest answers. The only type of question an instructor should refuse to answer is one the student should be able to easily answer on their own. Remember, *the fundamental goal of education is for each student to learn to think clearly and independently.* This requires hard work on the part of the student, which no instructor should ever circumvent. Anything done to bypass the student's responsibility to do that hard work ultimately limits that student's potential and thereby does real harm.

- It is not only permissible, but encouraged, to answer a student's question by asking questions in return, these follow-up questions designed to guide the student to reach a correct answer through their own reasoning.

- All student answers demand to be challenged by the instructor and/or by other students. This includes both correct and incorrect answers – the goal is to practice the articulation and defense of one's own reasoning.

- No reading assignment is deemed complete unless and until the student demonstrates their ability to accurately summarize the major points in their own terms. Recitation of the original text is unacceptable. This is why every module contains an "Outline and reflections" question as well as a "Foundational concepts" question in the Conceptual reasoning section, to prompt reflective reading.

- No assigned question is deemed answered unless and until the student demonstrates their ability to consistently and correctly apply the concepts to *variations* of that question. This is why module questions typically contain multiple "Challenges" suggesting different applications of the concept(s) as well as variations on the same theme(s). Instructors are encouraged to devise as many of their own "Challenges" as they are able, in order to have a multitude of ways ready to probe students' understanding.

- No assigned experiment or project is deemed complete unless and until the student demonstrates the task in action. If this cannot be done "live" before the instructor, video-recordings showing the demonstration are acceptable. All relevant safety precautions must be followed, all test equipment must be used correctly, and the student must be able to properly explain all results. The student must also successfully answer all Challenges presented by the instructor for that experiment or project.

Students learning from these modules would do well to abide by the following principles:

- No text should be considered fully and adequately read unless and until you can express every idea *in your own words, using your own examples.*
- You should always articulate your thoughts as you read the text, noting points of agreement, confusion, and epiphanies. Feel free to print the text on paper and then write your notes in the margins. Alternatively, keep a journal for your own reflections as you read. This is truly a helpful tool when digesting complicated concepts.
- Never take the easy path of highlighting or underlining important text. Instead, *summarize* and/or *comment* on the text using your own words. This actively engages your mind, allowing you to more clearly perceive points of confusion or misunderstanding on your own.
- A very helpful strategy when learning new concepts is to place yourself in the role of a teacher, if only as a mental exercise. Either explain what you have recently learned to someone else, or at least *imagine* yourself explaining what you have learned to someone else. The simple act of having to articulate new knowledge and skill forces you to take on a different perspective, and will help reveal weaknesses in your understanding.
- Perform each and every mathematical calculation and thought experiment shown in the text on your own, referring back to the text to see that your results agree. This may seem trivial and unnecessary, but it is critically important to ensuring you actually understand what is presented, especially when the concepts at hand are complicated and easy to misunderstand. Apply this same strategy to become proficient in the use of *circuit simulation software*, checking to see if your simulated results agree with the results shown in the text.
- Above all, recognize that learning is hard work, and that a certain level of frustration is unavoidable. There are times when you will struggle to grasp some of these concepts, and that struggle is a natural thing. Take heart that it will yield with persistent and varied[4] effort, and never give up!

Students interested in using these modules for self-study will also find them beneficial, although the onus of responsibility for thoroughly reading and answering questions will of course lie with that individual alone. If a qualified instructor is not available to challenge students, a workable alternative is for students to form study groups where they challenge[5] one another.

To high standards of education,

Tony R. Kuphaldt

---

[4] As the old saying goes, "Insanity is trying the same thing over and over again, expecting different results." If you find yourself stumped by something in the text, you should attempt a different approach. Alter the thought experiment, change the mathematical parameters, do whatever you can to see the problem in a slightly different light, and then the solution will often present itself more readily.

[5] Avoid the temptation to simply share answers with study partners, as this is really counter-productive to learning. Always bear in mind that the answer to any question is far less important in the long run than the method(s) used to obtain that answer. The goal of education is to empower one's life through the improvement of clear and independent thought, literacy, expression, and various practical skills.

Appendix C

Tools used

I am indebted to the developers of many open-source software applications in the creation of these learning modules. The following is a list of these applications with some commentary on each.

You will notice a theme common to many of these applications: a bias toward *code*. Although I am by no means an expert programmer in any computer language, I understand and appreciate the flexibility offered by code-based applications where the user (you) enters commands into a plain ASCII text file, which the software then reads and processes to create the final output. Code-based computer applications are by their very nature *extensible*, while WYSIWYG (What You See Is What You Get) applications are generally limited to whatever user interface the developer makes for you.

The `GNU/Linux` computer operating system

> There is so much to be said about Linus Torvalds' `Linux` and Richard Stallman's `GNU` project. First, to credit just these two individuals is to fail to do justice to the *mob* of passionate volunteers who contributed to make this amazing software a reality. I first learned of `Linux` back in 1996, and have been using this operating system on my personal computers almost exclusively since then. It is *free*, it is completely *configurable*, and it permits the continued use of highly efficient `Unix` applications and scripting languages (e.g. shell scripts, Makefiles, `sed`, `awk`) developed over many decades. `Linux` not only provided me with a powerful computing platform, but its open design served to inspire my life's work of creating open-source educational resources.

Bram Moolenaar's `Vim` text editor

> Writing code for any code-based computer application requires a *text editor*, which may be thought of as a word processor strictly limited to outputting plain-ASCII text files. Many good text editors exist, and one's choice of text editor seems to be a deeply personal matter within the programming world. I prefer `Vim` because it operates very similarly to `vi` which is ubiquitous on `Unix`/`Linux` operating systems, and because it may be entirely operated via keyboard (i.e. no mouse required) which makes it fast to use.

Donald Knuth's TEX typesetting system

Developed in the late 1970's and early 1980's by computer scientist extraordinaire Donald Knuth to typeset his multi-volume magnum opus *The Art of Computer Programming*, this software allows the production of formatted text for screen-viewing or paper printing, all by writing plain-text code to describe how the formatted text is supposed to appear. TEX is not just a markup language for documents, but it is also a Turing-complete programming language in and of itself, allowing useful algorithms to be created to control the production of documents. Simply put, TEX *is a programmer's approach to word processing*. Since TEX is controlled by code written in a plain-text file, this means anyone may read that plain-text file to see exactly how the document was created. This openness afforded by the code-based nature of TEX makes it relatively easy to learn how other people have created their own TEX documents. By contrast, examining a beautiful document created in a conventional WYSIWYG word processor such as Microsoft `Word` suggests nothing to the reader about *how* that document was created, or what the user might do to create something similar. As Mr. Knuth himself once quipped, conventional word processing applications should be called WYSIAYG (What You See Is *All* You Get).

Leslie Lamport's LATEX extensions to TEX

Like all true programming languages, TEX is inherently extensible. So, years after the release of TEX to the public, Leslie Lamport decided to create a massive extension allowing easier compilation of book-length documents. The result was LATEX, which is the markup language used to create all ModEL module documents. You could say that TEX is to LATEX as `C` is to `C++`. This means it is permissible to use any and all TEX commands within LATEX source code, and it all still works. Some of the features offered by LATEX that would be challenging to implement in TEX include automatic index and table-of-content creation.

Tim Edwards' `Xcircuit` drafting program

This wonderful program is what I use to create all the schematic diagrams and illustrations (but not photographic images or mathematical plots) throughout the ModEL project. It natively outputs PostScript format which is a true vector graphic format (this is why the images do not pixellate when you zoom in for a closer view), and it is so simple to use that I have never had to read the manual! Object libraries are easy to create for `Xcircuit`, being plain-text files using PostScript programming conventions. Over the years I have collected a large set of object libraries useful for drawing electrical and electronic schematics, pictorial diagrams, and other technical illustrations.

`Gimp` graphic image manipulation program

Essentially an open-source clone of Adobe's `PhotoShop`, I use `Gimp` to resize, crop, and convert file formats for all of the photographic images appearing in the ModEL modules. Although `Gimp` does offer its own scripting language (called `Script-Fu`), I have never had occasion to use it. Thus, my utilization of `Gimp` to merely crop, resize, and convert graphic images is akin to using a sword to slice bread.

`SPICE` circuit simulation program

`SPICE` is to circuit analysis as TEX is to document creation: it is a form of markup language designed to describe a certain object to be processed in plain-ASCII text. When the plain-text "source file" is compiled by the software, it outputs the final result. More modern circuit analysis tools certainly exist, but I prefer `SPICE` for the following reasons: it is *free*, it is *fast*, it is *reliable*, and it is a fantastic tool for *teaching* students of electricity and electronics how to write simple code. I happen to use rather old versions of `SPICE`, version 2g6 being my "go to" application when I only require text-based output. `NGSPICE` (version 26), which is based on Berkeley `SPICE` version 3f5, is used when I require graphical output for such things as time-domain waveforms and Bode plots. In all `SPICE` example netlists I strive to use coding conventions compatible with all `SPICE` versions.

Andrew D. Hwang's `ePiX` mathematical visualization programming library

This amazing project is a `C++` library you may link to any `C/C++` code for the purpose of generating PostScript graphic images of mathematical functions. As a completely free and open-source project, it does all the plotting I would otherwise use a Computer Algebra System (CAS) such as `Mathematica` or `Maple` to do. It should be said that `ePiX` is *not* a Computer Algebra System like `Mathematica` or `Maple`, but merely a mathematical *visualization* tool. In other words, it won't determine integrals for you (you'll have to implement that in your own `C/C++` code!), but it can graph the results, and it does so beautifully. What I really admire about `ePiX` is that it is a `C++` programming library, which means it builds on the existing power and toolset available with that programming language. Mr. Hwang could have probably developed his own stand-alone application for mathematical plotting, but by creating a `C++` library to do the same thing he accomplished something much greater.

`gnuplot` mathematical visualization software

Another open-source tool for mathematical visualization is `gnuplot`. Interestingly, this tool is *not* part of Richard Stallman's GNU project, its name being a coincidence. For this reason the authors prefer "gnu" *not* be capitalized at all to avoid confusion. This is a much "lighter-weight" alternative to a spreadsheet for plotting tabular data, and the fact that it easily outputs directly to an X11 console or a file in a number of different graphical formats (including PostScript) is very helpful. I typically set my `gnuplot` output format to default (X11 on my Linux PC) for quick viewing while I'm developing a visualization, then switch to PostScript file export once the visual is ready to include in the document(s) I'm writing. As with my use of `Gimp` to do rudimentary image editing, my use of `gnuplot` only scratches the surface of its capabilities, but the important points are that it's *free* and that it *works well*.

`Python` programming language

Both Python and C++ find extensive use in these modules as instructional aids and exercises, but I'm listing Python here as a *tool* for myself because I use it almost daily as a *calculator*. If you open a Python interpreter console and type `from math import *` you can type mathematical expressions and have it return results just as you would on a hand calculator. Complex-number (i.e. *phasor*) arithmetic is similarly supported if you include the complex-math library (`from cmath import *`). Examples of this are shown in the Programming References chapter (if included) in each module. Of course, being a fully-featured programming language, Python also supports conditionals, loops, and other structures useful for calculation of quantities. Also, running in a console environment where all entries and returned values show as text in a chronologically-ordered list makes it easy to copy-and-paste those calculations to document exactly how they were performed.

Appendix D

Creative Commons License

Creative Commons Attribution 4.0 International Public License

By exercising the Licensed Rights (defined below), You accept and agree to be bound by the terms and conditions of this Creative Commons Attribution 4.0 International Public License ("Public License"). To the extent this Public License may be interpreted as a contract, You are granted the Licensed Rights in consideration of Your acceptance of these terms and conditions, and the Licensor grants You such rights in consideration of benefits the Licensor receives from making the Licensed Material available under these terms and conditions.

Section 1 – Definitions.

a. **Adapted Material** means material subject to Copyright and Similar Rights that is derived from or based upon the Licensed Material and in which the Licensed Material is translated, altered, arranged, transformed, or otherwise modified in a manner requiring permission under the Copyright and Similar Rights held by the Licensor. For purposes of this Public License, where the Licensed Material is a musical work, performance, or sound recording, Adapted Material is always produced where the Licensed Material is synched in timed relation with a moving image.

b. **Adapter's License** means the license You apply to Your Copyright and Similar Rights in Your contributions to Adapted Material in accordance with the terms and conditions of this Public License.

c. **Copyright and Similar Rights** means copyright and/or similar rights closely related to copyright including, without limitation, performance, broadcast, sound recording, and Sui Generis Database Rights, without regard to how the rights are labeled or categorized. For purposes of this Public License, the rights specified in Section 2(b)(1)-(2) are not Copyright and Similar Rights.

d. **Effective Technological Measures** means those measures that, in the absence of proper authority, may not be circumvented under laws fulfilling obligations under Article 11 of the WIPO Copyright Treaty adopted on December 20, 1996, and/or similar international agreements.

e. **Exceptions and Limitations** means fair use, fair dealing, and/or any other exception or

limitation to Copyright and Similar Rights that applies to Your use of the Licensed Material.

f. **Licensed Material** means the artistic or literary work, database, or other material to which the Licensor applied this Public License.

g. **Licensed Rights** means the rights granted to You subject to the terms and conditions of this Public License, which are limited to all Copyright and Similar Rights that apply to Your use of the Licensed Material and that the Licensor has authority to license.

h. **Licensor** means the individual(s) or entity(ies) granting rights under this Public License.

i. **Share** means to provide material to the public by any means or process that requires permission under the Licensed Rights, such as reproduction, public display, public performance, distribution, dissemination, communication, or importation, and to make material available to the public including in ways that members of the public may access the material from a place and at a time individually chosen by them.

j. **Sui Generis Database Rights** means rights other than copyright resulting from Directive 96/9/EC of the European Parliament and of the Council of 11 March 1996 on the legal protection of databases, as amended and/or succeeded, as well as other essentially equivalent rights anywhere in the world.

k. **You** means the individual or entity exercising the Licensed Rights under this Public License. **Your** has a corresponding meaning.

Section 2 – Scope.

a. License grant.

1. Subject to the terms and conditions of this Public License, the Licensor hereby grants You a worldwide, royalty-free, non-sublicensable, non-exclusive, irrevocable license to exercise the Licensed Rights in the Licensed Material to:

A. reproduce and Share the Licensed Material, in whole or in part; and

B. produce, reproduce, and Share Adapted Material.

2. Exceptions and Limitations. For the avoidance of doubt, where Exceptions and Limitations apply to Your use, this Public License does not apply, and You do not need to comply with its terms and conditions.

3. Term. The term of this Public License is specified in Section 6(a).

4. Media and formats; technical modifications allowed. The Licensor authorizes You to exercise the Licensed Rights in all media and formats whether now known or hereafter created, and to make technical modifications necessary to do so. The Licensor waives and/or agrees not to assert any right or authority to forbid You from making technical modifications necessary to exercise the Licensed Rights, including technical modifications necessary to circumvent Effective Technological Measures.

For purposes of this Public License, simply making modifications authorized by this Section 2(a)(4) never produces Adapted Material.

5. Downstream recipients.

A. Offer from the Licensor – Licensed Material. Every recipient of the Licensed Material automatically receives an offer from the Licensor to exercise the Licensed Rights under the terms and conditions of this Public License.

B. No downstream restrictions. You may not offer or impose any additional or different terms or conditions on, or apply any Effective Technological Measures to, the Licensed Material if doing so restricts exercise of the Licensed Rights by any recipient of the Licensed Material.

6. No endorsement. Nothing in this Public License constitutes or may be construed as permission to assert or imply that You are, or that Your use of the Licensed Material is, connected with, or sponsored, endorsed, or granted official status by, the Licensor or others designated to receive attribution as provided in Section 3(a)(1)(A)(i).

b. Other rights.

1. Moral rights, such as the right of integrity, are not licensed under this Public License, nor are publicity, privacy, and/or other similar personality rights; however, to the extent possible, the Licensor waives and/or agrees not to assert any such rights held by the Licensor to the limited extent necessary to allow You to exercise the Licensed Rights, but not otherwise.

2. Patent and trademark rights are not licensed under this Public License.

3. To the extent possible, the Licensor waives any right to collect royalties from You for the exercise of the Licensed Rights, whether directly or through a collecting society under any voluntary or waivable statutory or compulsory licensing scheme. In all other cases the Licensor expressly reserves any right to collect such royalties.

Section 3 – License Conditions.

Your exercise of the Licensed Rights is expressly made subject to the following conditions.

a. Attribution.

1. If You Share the Licensed Material (including in modified form), You must:

A. retain the following if it is supplied by the Licensor with the Licensed Material:

i. identification of the creator(s) of the Licensed Material and any others designated to receive attribution, in any reasonable manner requested by the Licensor (including by pseudonym if designated);

ii. a copyright notice;

iii. a notice that refers to this Public License;

iv. a notice that refers to the disclaimer of warranties;

v. a URI or hyperlink to the Licensed Material to the extent reasonably practicable;

B. indicate if You modified the Licensed Material and retain an indication of any previous modifications; and

C. indicate the Licensed Material is licensed under this Public License, and include the text of, or the URI or hyperlink to, this Public License.

2. You may satisfy the conditions in Section 3(a)(1) in any reasonable manner based on the medium, means, and context in which You Share the Licensed Material. For example, it may be reasonable to satisfy the conditions by providing a URI or hyperlink to a resource that includes the required information.

3. If requested by the Licensor, You must remove any of the information required by Section 3(a)(1)(A) to the extent reasonably practicable.

4. If You Share Adapted Material You produce, the Adapter's License You apply must not prevent recipients of the Adapted Material from complying with this Public License.

Section 4 – Sui Generis Database Rights.

Where the Licensed Rights include Sui Generis Database Rights that apply to Your use of the Licensed Material:

a. for the avoidance of doubt, Section 2(a)(1) grants You the right to extract, reuse, reproduce, and Share all or a substantial portion of the contents of the database;

b. if You include all or a substantial portion of the database contents in a database in which You have Sui Generis Database Rights, then the database in which You have Sui Generis Database Rights (but not its individual contents) is Adapted Material; and

c. You must comply with the conditions in Section 3(a) if You Share all or a substantial portion of the contents of the database.

For the avoidance of doubt, this Section 4 supplements and does not replace Your obligations under this Public License where the Licensed Rights include other Copyright and Similar Rights.

Section 5 – Disclaimer of Warranties and Limitation of Liability.

a. Unless otherwise separately undertaken by the Licensor, to the extent possible, the Licensor offers the Licensed Material as-is and as-available, and makes no representations or warranties of any kind concerning the Licensed Material, whether express, implied, statutory, or other. This includes, without limitation, warranties of title, merchantability, fitness for a particular purpose, non-infringement, absence of latent or other defects, accuracy, or the presence or absence of errors,

whether or not known or discoverable. Where disclaimers of warranties are not allowed in full or in part, this disclaimer may not apply to You.

b. To the extent possible, in no event will the Licensor be liable to You on any legal theory (including, without limitation, negligence) or otherwise for any direct, special, indirect, incidental, consequential, punitive, exemplary, or other losses, costs, expenses, or damages arising out of this Public License or use of the Licensed Material, even if the Licensor has been advised of the possibility of such losses, costs, expenses, or damages. Where a limitation of liability is not allowed in full or in part, this limitation may not apply to You.

c. The disclaimer of warranties and limitation of liability provided above shall be interpreted in a manner that, to the extent possible, most closely approximates an absolute disclaimer and waiver of all liability.

Section 6 – Term and Termination.

a. This Public License applies for the term of the Copyright and Similar Rights licensed here. However, if You fail to comply with this Public License, then Your rights under this Public License terminate automatically.

b. Where Your right to use the Licensed Material has terminated under Section 6(a), it reinstates:

1. automatically as of the date the violation is cured, provided it is cured within 30 days of Your discovery of the violation; or

2. upon express reinstatement by the Licensor.

For the avoidance of doubt, this Section 6(b) does not affect any right the Licensor may have to seek remedies for Your violations of this Public License.

c. For the avoidance of doubt, the Licensor may also offer the Licensed Material under separate terms or conditions or stop distributing the Licensed Material at any time; however, doing so will not terminate this Public License.

d. Sections 1, 5, 6, 7, and 8 survive termination of this Public License.

Section 7 – Other Terms and Conditions.

a. The Licensor shall not be bound by any additional or different terms or conditions communicated by You unless expressly agreed.

b. Any arrangements, understandings, or agreements regarding the Licensed Material not stated herein are separate from and independent of the terms and conditions of this Public License.

Section 8 – Interpretation.

a. For the avoidance of doubt, this Public License does not, and shall not be interpreted to, reduce, limit, restrict, or impose conditions on any use of the Licensed Material that could lawfully

be made without permission under this Public License.

b. To the extent possible, if any provision of this Public License is deemed unenforceable, it shall be automatically reformed to the minimum extent necessary to make it enforceable. If the provision cannot be reformed, it shall be severed from this Public License without affecting the enforceability of the remaining terms and conditions.

c. No term or condition of this Public License will be waived and no failure to comply consented to unless expressly agreed to by the Licensor.

d. Nothing in this Public License constitutes or may be interpreted as a limitation upon, or waiver of, any privileges and immunities that apply to the Licensor or You, including from the legal processes of any jurisdiction or authority.

Creative Commons is not a party to its public licenses. Notwithstanding, Creative Commons may elect to apply one of its public licenses to material it publishes and in those instances will be considered the "Licensor." Except for the limited purpose of indicating that material is shared under a Creative Commons public license or as otherwise permitted by the Creative Commons policies published at `creativecommons.org/policies`, Creative Commons does not authorize the use of the trademark "Creative Commons" or any other trademark or logo of Creative Commons without its prior written consent including, without limitation, in connection with any unauthorized modifications to any of its public licenses or any other arrangements, understandings, or agreements concerning use of licensed material. For the avoidance of doubt, this paragraph does not form part of the public licenses.

Creative Commons may be contacted at `creativecommons.org`.

Appendix E

References

Bogart, Theodore F. Jr., *Introduction to Digital Circuits*, Glencoe division of Macmillan/McGraw-Hill, 1992.

Appendix F

Version history

This is a list showing all significant additions, corrections, and other edits made to this learning module. Each entry is referenced by calendar date in reverse chronological order (newest version first), which appears on the front cover of every learning module for easy reference. Any contributors to this open-source document are listed here as well.

30 November 2023 – added a comment about all one-hot counters dividing frequency by n (the number of D-type flip-flops) to the Tutorial, and also added a schematic showing two ring counters cascaded to form a larger frequency-division ratio. Also added some instructor notes to one of the Diagnostic Reasoning questions. Also fixed an error in image_2471 courtesy of Galen Bennett, David Mitchell, and Trent Johnson.

28 November 2022 – placed questions at the top of the itemized list in the Introduction chapter prompting students to devise experiments related to the tutorial content.

28 July 2022 – minor edits to the Tutorial and Introduction chapters.

7 December 2021 – divided the Tutorial chapter into sections.

9 May 2021 – commented out or deleted empty chapters.

19 October 2020 – minor edits.

5 October 2020 – significantly edited the Introduction chapter to make it more suitable as a pre-study guide and to provide cues useful to instructors leading "inverted" teaching sessions.

12 March 2020 – minor edits to the Tutorial.

7 March 2020 – added Technical Reference section on digital pulse criteria.

28 January 2020 – added Foundational Concepts to the list in the Conceptual Reasoning section.

23 December 2019 – renamed "Failed LED sequencing circuit" to "Sequential tail-light blinker".

18 November 2019 – added Introduction chapter. Also added ring counters to the Tutorial.

17 November 2019 – document first created.

Index