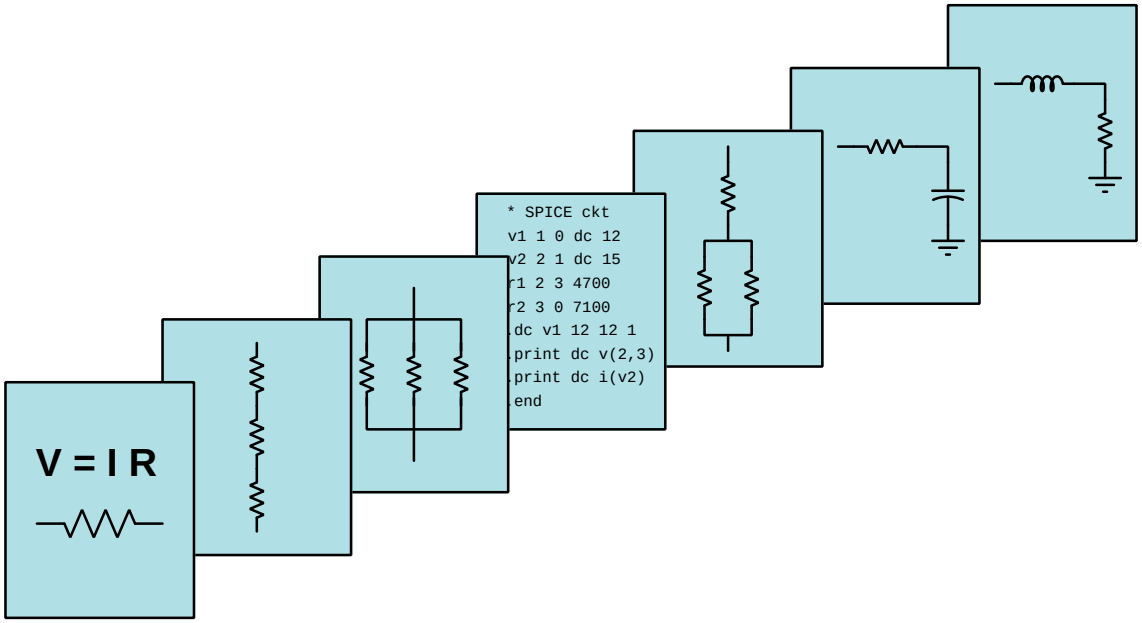
Modular Electronics Learning (ModEL) project



Digital Counters

Last update = 9 January 2025

Contents

Chapter 1

Introduction

1.1 Recommendations for students

Counters are one of the first types of practical digital circuits students learn to build using flip-flops. In particular, the JK flip-flop finds widespread use as a fundamental building-block of counters due to its ability to *toggle* between one state and the other at every clock pulse. When in toggle mode, a JK flip-flop essentially acts as a 2:1 frequency divider, which is precisely the characteristic needed to generate binary count sequences.

Counter circuits see widespread application in digital electronics, especially in digital computing circuits, for their ability to generate binary sequences.

Important concepts related to counters include **binary** numeration, **latches**, **flip-flops**, **edge-triggering**, **significant bits**, **frequency division**, **timing diagrams**, **synchronous** versus **asynchronous** circuits, **propagation delay**, **set-up time**, **ripple**, **strobing**, **modulus**, **octave**, **decade**, and **duty cycle**.

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

- How might an experiment be designed and conducted to determine whether a counter IC of unknown model number has synchronous or asynchronous outputs? 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 determine whether a counter IC of unknown model number has a synchronous or asynchronous clear (reset) input? What hypothesis (i.e. prediction) might you pose for that experiment, and what result(s) would either support or disprove that hypothesis?
- What distinguishes a flip-flop from a latch?
- What distinguishes a JK flip-flop from an SR flip-flop?
- How do multiple flip-flops work together to form a counter?

- What is *ripple* and why does it arise in certain counter circuits?
- What do the Preset and Clear inputs do on a flip-flop or latch?
- Why are adequate *set-up* and *hold* times important for digital circuits?
- What is the "toggle" mode for a JK flip-flop, and how is this exploited within counter circuits?
- What causes propagation delay in digital circuits?
- What does it mean to "strobe" a digital circuit?
- Why is set-up time an important concept for synchronous counter circuit design?
- What is the difference between the terms "synchronous" versus "asynchronous" as they are applied to counter circuits, compared to how these same terms are applied to Preset and Clear inputs on individual flip-flops?
- How do logic gates work to give a counter circuit up/down counting ability?
- What must we do to a counter circuit to *truncate* its maximum count value?
- What frequency ratio is represented by an octave?
- What frequency ratio is represented by a decade?
- How may we combine counters to achieve larger frequency division ratios?
- Why does the synchronous or asynchronous nature of a flip-flop's clear input matter when using a number of them to make a reduced-modulus counter?

1.2 Challenging concepts related to digital counters

The following list cites concepts related to this module's topic that are easily misunderstood, along with suggestions for properly understanding them:

- **Synchronous versus Asynchronous inputs** – counter circuits rely on a "clock" signal to increment or decrement their count values, with some inputs having effect only when the clock pulse permits (i.e. "synchronous") and other inputs having immediate effect regardless of clock state (i.e. "asynchronous"). Clear and preset inputs may fall into either of these categories depending on the internal design of the flip-flops comprising the counter. This challenging concept is made even more confusing by the fact that counters may be broadly categorized as either "synchronous" or "asynchronous" in terms of their output states updating together (synchronous) versus in a rippling fashion (asynchronous), and this has absolutely nothing to do with the counter's preset or clear inputs being synchronous or asynchronous!

- **Counter modulus** – this is nothing more than a measure of how many unique counter states a counter offers in total. Modulus may be reduced by adding external logic to force the counter to either clear or preset to a specified count before it normally would.

1.3 Recommendations for instructors

This section lists realistic student learning outcomes supported by the content of the module as well as suggested means of assessing (measuring) student learning. The outcomes state what learners should be able to do, and the assessments are specific challenges to prove students have learned.

- **Outcome** – Demonstrate effective technical reading and writing

  Assessment – Students present their outlines of this module's instructional chapters (e.g. Case Tutorial, Tutorial, Historical References, etc.) ideally as an entry to a larger Journal document chronicling their learning. These outlines should exhibit good-faith effort at summarizing major concepts explained in the text.

  Assessment – Students show how timing diagrams were obtained by the author in the Tutorial chapter's examples.

- **Outcome** – Apply foundational circuit concepts to the analysis of counter circuits made of individual flip-flops

  Assessment – Identify counter characteristics such as count direction, synchronous/asynchronous output, etc. based on an analysis of a given schematic diagram; e.g. pose problems in the form of the "Up-counter or down-counter?" and "Synchronous counter direction" and "Counter circuit identification" Conceptual Reasoning questions.

- **Outcome** – Design a frequency divider circuit

  Assessment – Identify how counter(s) and logic gates may be combined to form frequency dividers with given division ratios; e.g. pose problems in the form of the "Frequency division" and "Frequency division using clear versus using preset" Quantitative Reasoning questions.

- **Outcome** – Independent research

  Assessment – Locate counter IC datasheets and properly interpret some of the information contained in those documents including proper logic levels, maximum clock frequency, set-up and hold time requirements, synchronous versus asynchronous inputs, etc.

Chapter 2

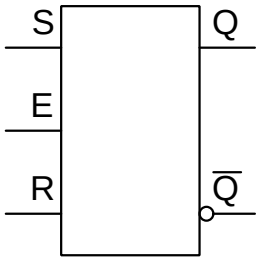
Case Tutorial

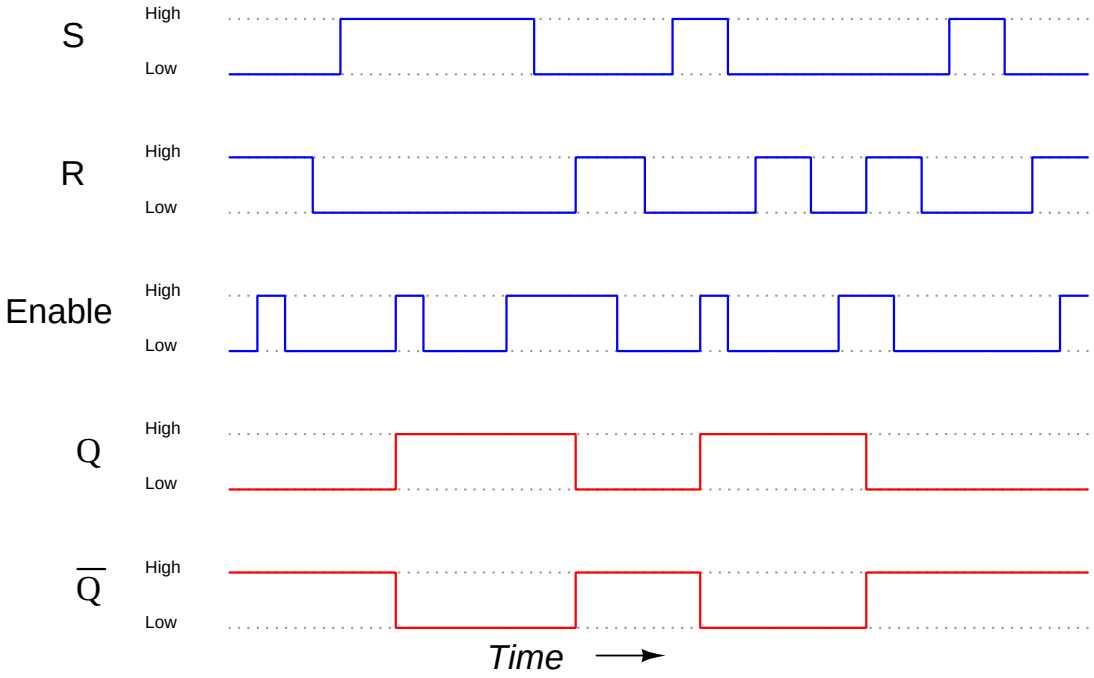
The idea behind a *Case Tutorial* is to explore new concepts by way of example. In this chapter you will read less presentation of theory compared to other Tutorial chapters, but by close observation and comparison of the given examples be able to discern patterns and principles much the same way as a scientific experimenter. Hopefully you will find these cases illuminating, and a good supplement to text-based tutorials.

These examples also serve well as challenges following your reading of the other Tutorial(s) in this module – can you explain *why* the circuits behave as they do?

2.1 Example: timing diagrams for latches and flip-flops

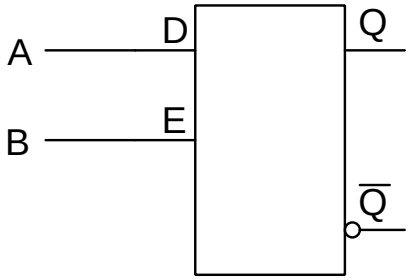
2.1.1 Example: enabled SR latch timing diagram

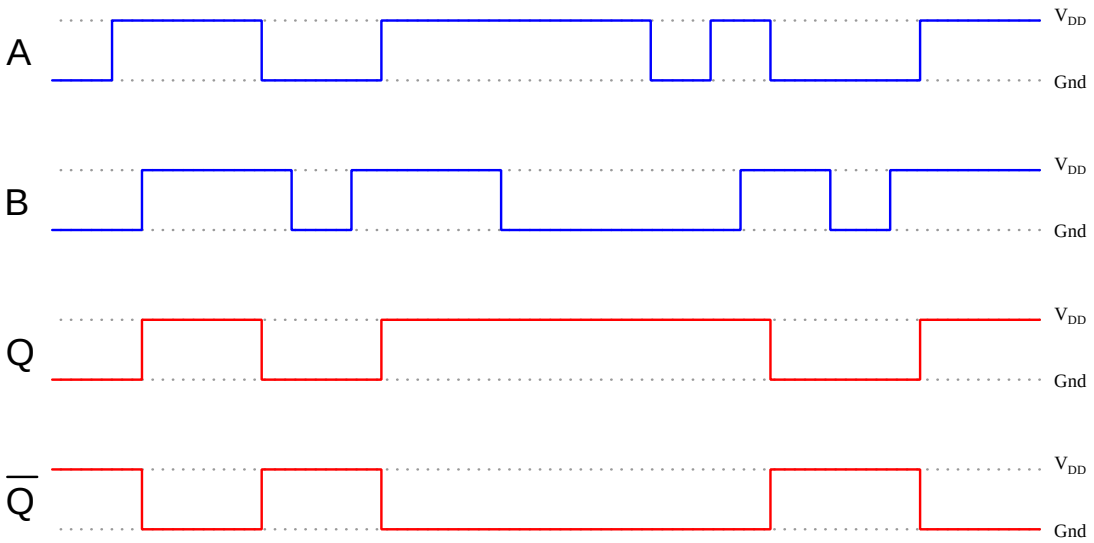




The red-colored output signals assume Q began in a low state and $\overline{Q}$ in a high state.

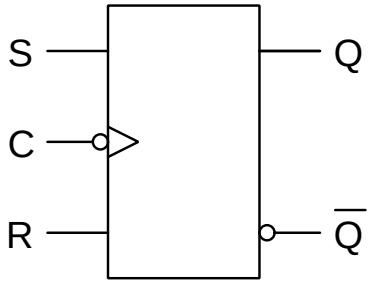
2.1.2 Example: enabled D latch timing diagram

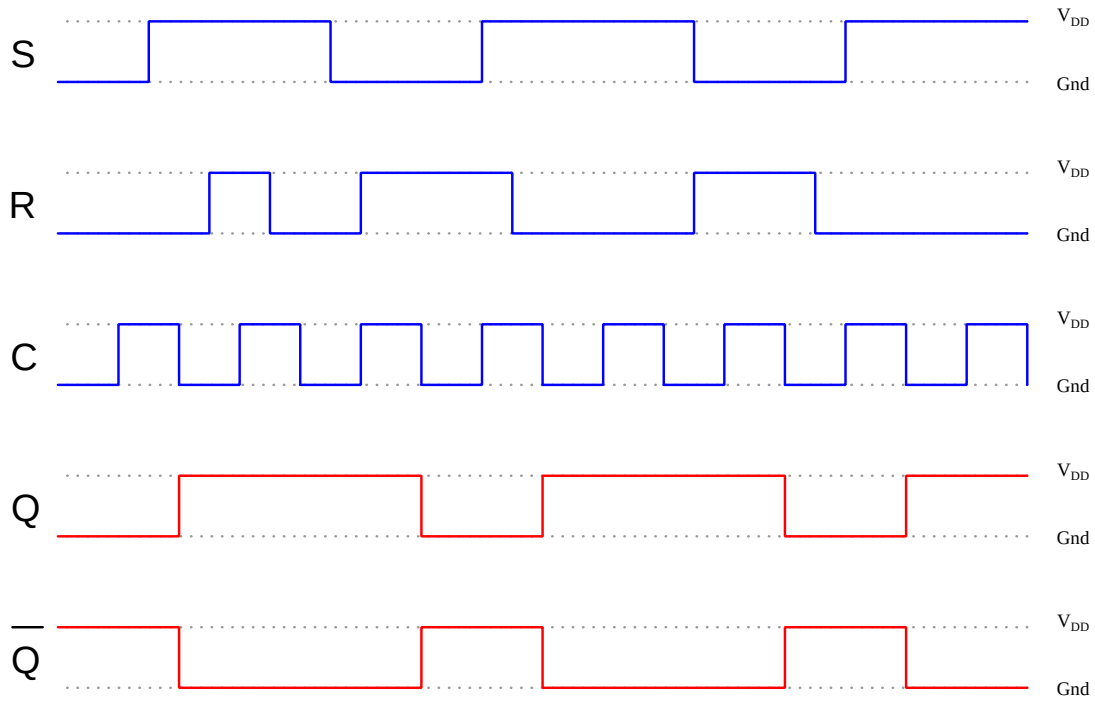




The red-colored output signals assume Q began in a low state and $\overline{Q}$ in a high state.

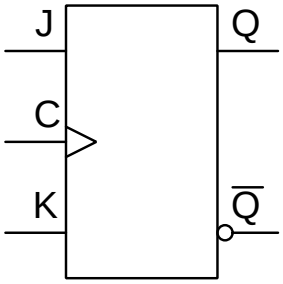
2.1.3 Example: SR flip-flop timing diagram

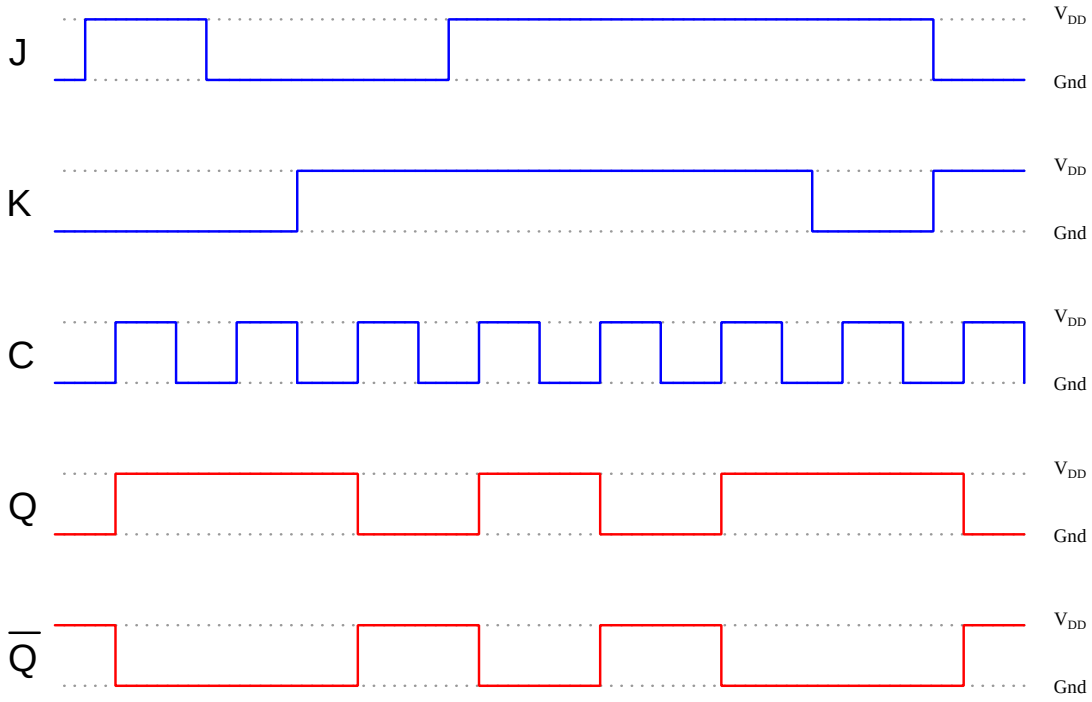




The red-colored output signals assume Q began in a low state and $\overline{Q}$ in a high state.

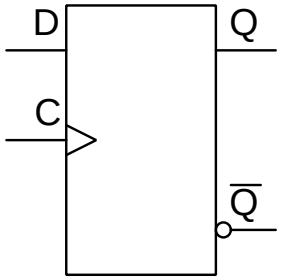
2.1.4 Example: JK flip-flop timing diagram

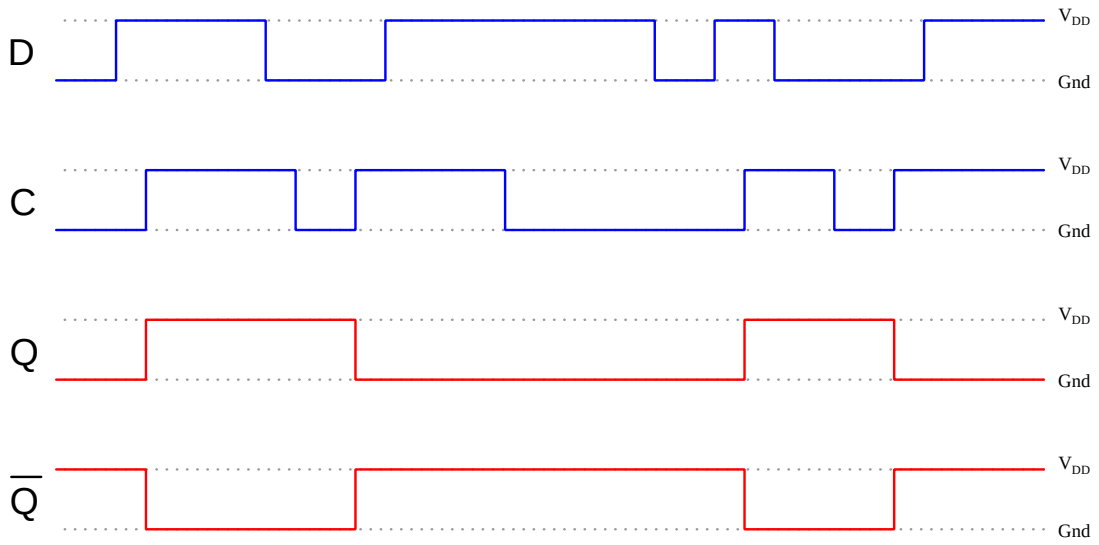




The red-colored output signals assume Q began in a low state and $\overline{Q}$ in a high state.

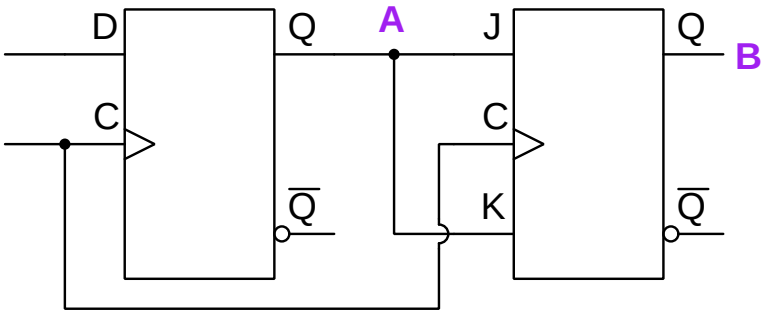
2.1.5 Example: D flip-flop timing diagram

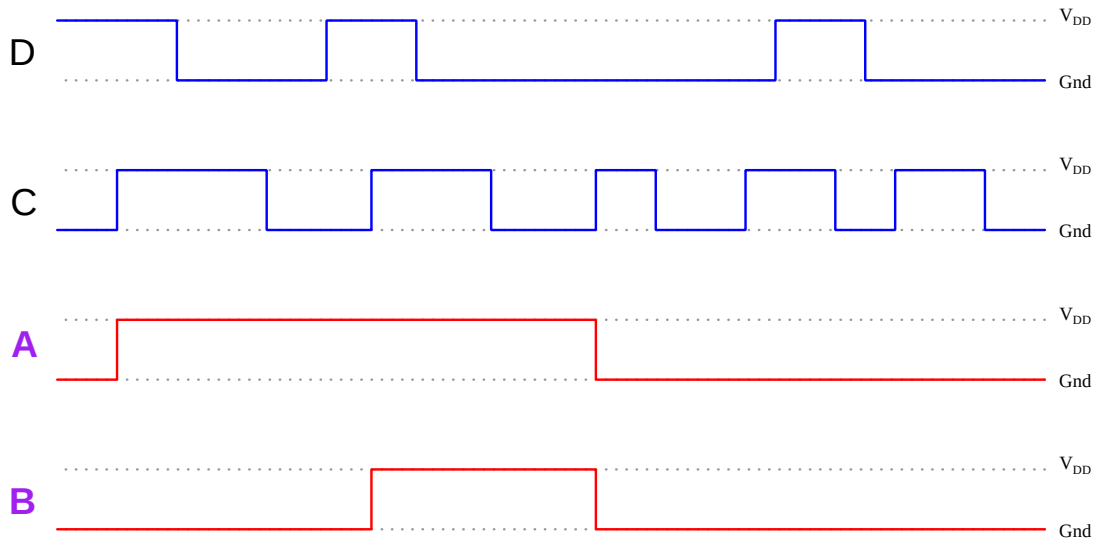




The red-colored output signals assume Q began in a low state and $\overline{Q}$ in a high state.

2.1.6 Example: cascaded flip-flops timing diagram

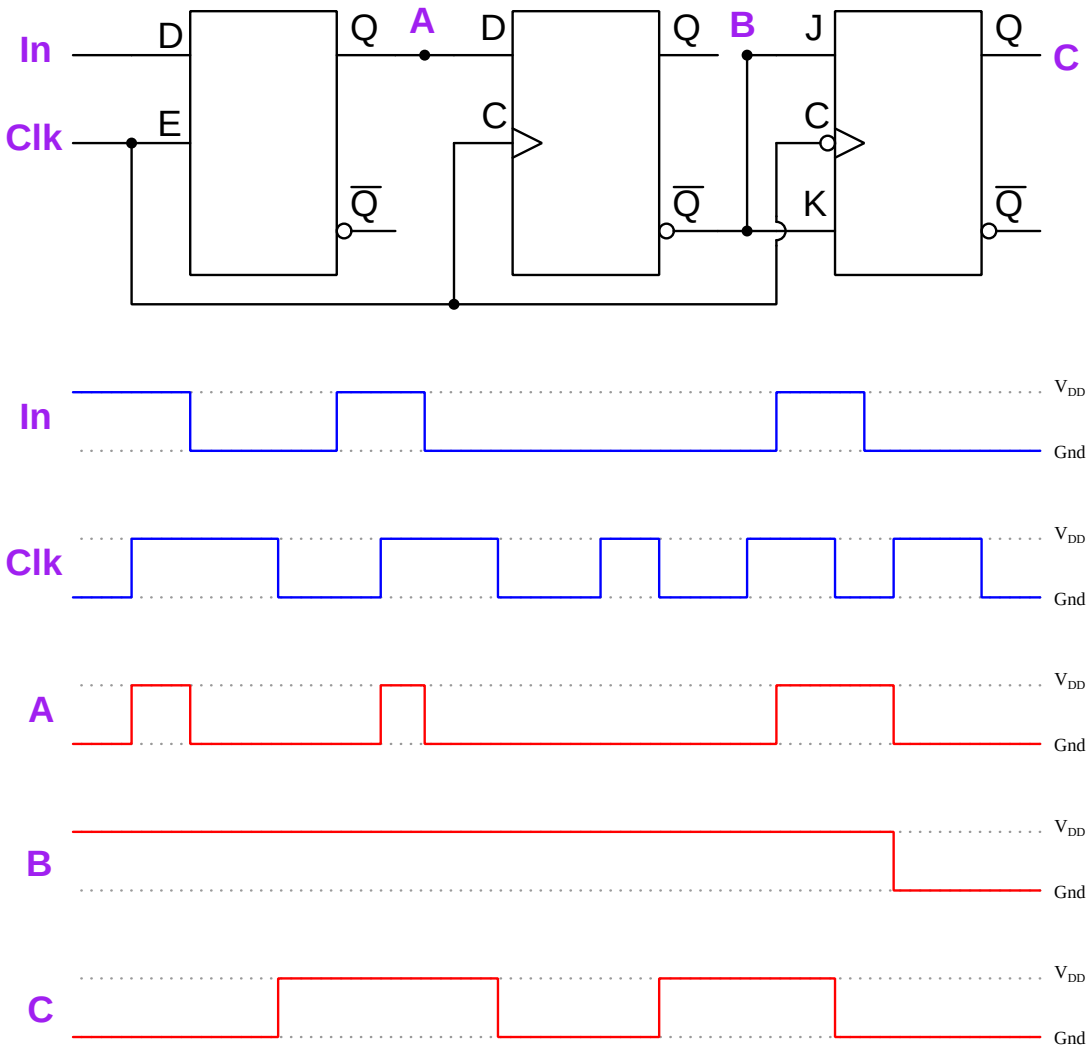




The red-colored output signals assume both Q outputs began in a low state and both $\overline{Q}$ outputs in a high state.

2.1.7 Example: cascaded latch/flip-flops timing diagram

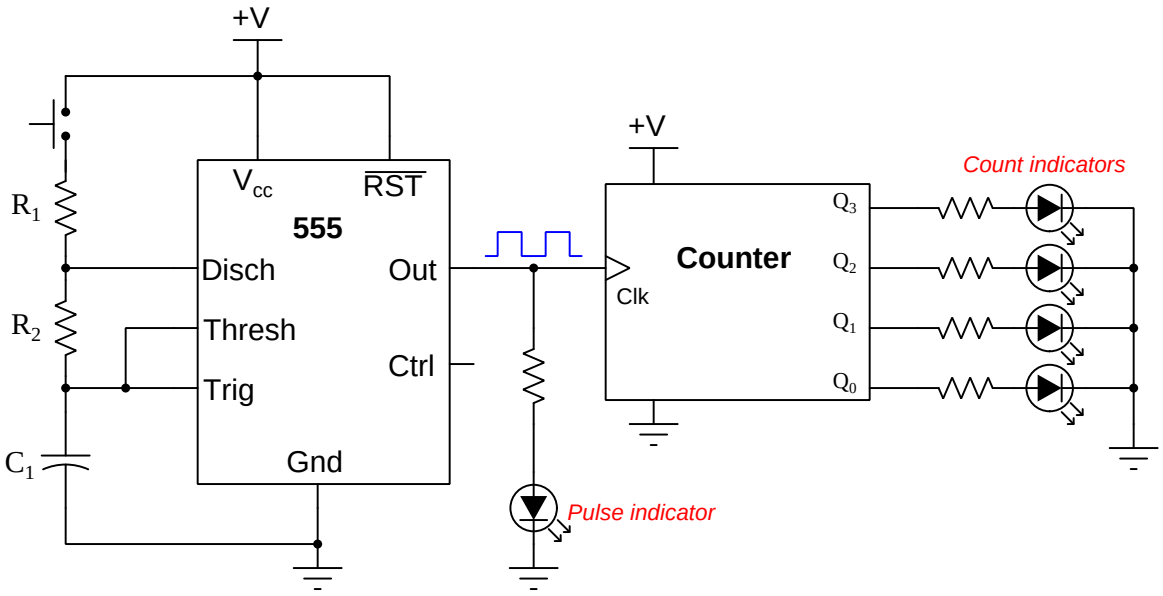
Note that the first element is a D-type *latch* while the second is a D-type *flip-flop*. Also, note that the JK flip-flop is *negative* edge-triggered rather than positive:



The red-colored output signals assume all Q outputs began in a low state and all $\overline{Q}$ outputs in a high state.

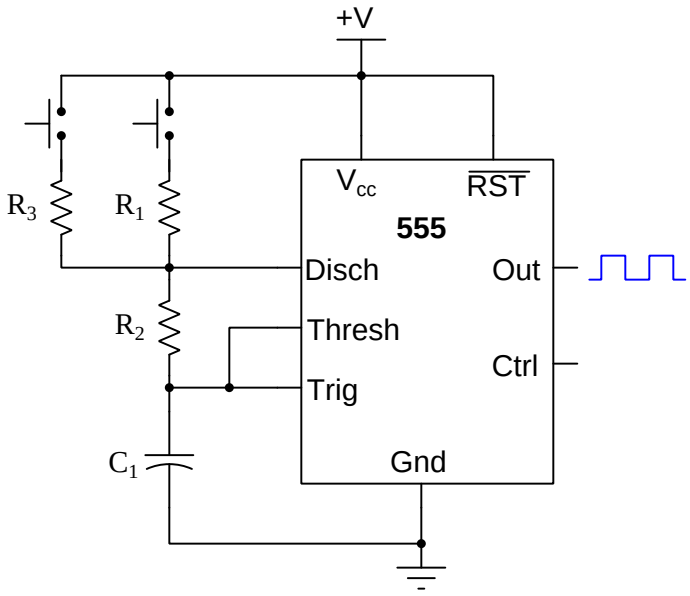
2.2 Example: clock pulse generator

When testing prototype counter circuits, we need a reliable source of clock pulses to drive the counter(s). One readily available IC to do this is the model 555 timer circuit which uses combinations of resistance and capacitance to create time delays and/or oscillations. Here we see a 555 timer wired to be an *astable* circuit which means its output terminal will endlessly toggle between "high" and "low" states, thus constituting a clock pulse suitable for driving a counter circuit:



Pressing the pushbutton switch causes this circuit to begin oscillating. Releasing the pushbutton causes it to cease oscillations, leaving the Out terminal waiting in its "high" state.

If you would like to have two different speeds of clock pulse (e.g. a fast versus a slow), you may add another pushbutton switch and resistor:



The larger the resistor placed in series with the pushbutton switch, the longer the "high" time of the clock pulse, which in turn decreases its frequency. You may calculate the duration of each portion of the clock pulse using the following formulae:

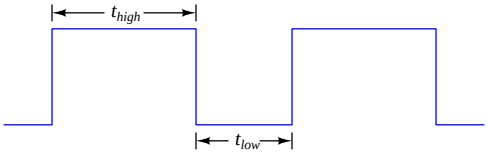
$$t_{high} = 0.693(R_1 + R_2)C \qquad t_{low} = 0.693R_2C$$

Where,

t = Time, in seconds
R = Resistance, in Ohms
C = Capacitance, in Farads

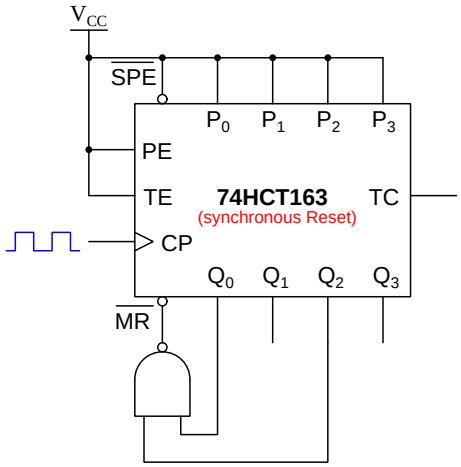
Clock pulse frequency is given by this formula, with frequency (f) in Hertz:

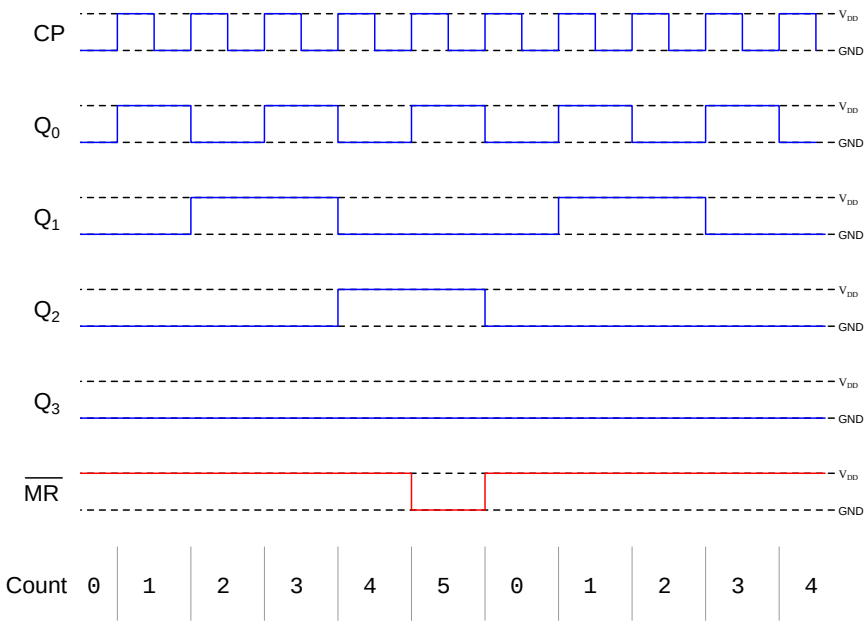
$$f = \frac{1.44}{(R_1 + 2R_2)C}$$



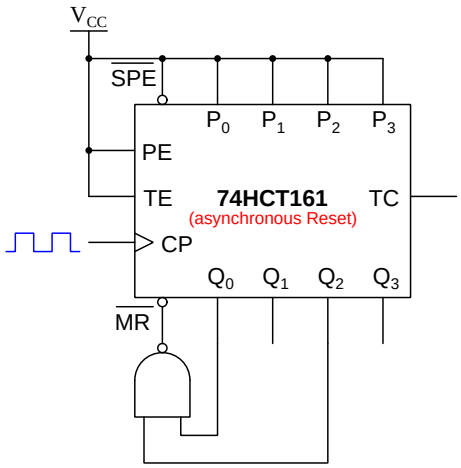
2.3 Example: reduced-modulus counters

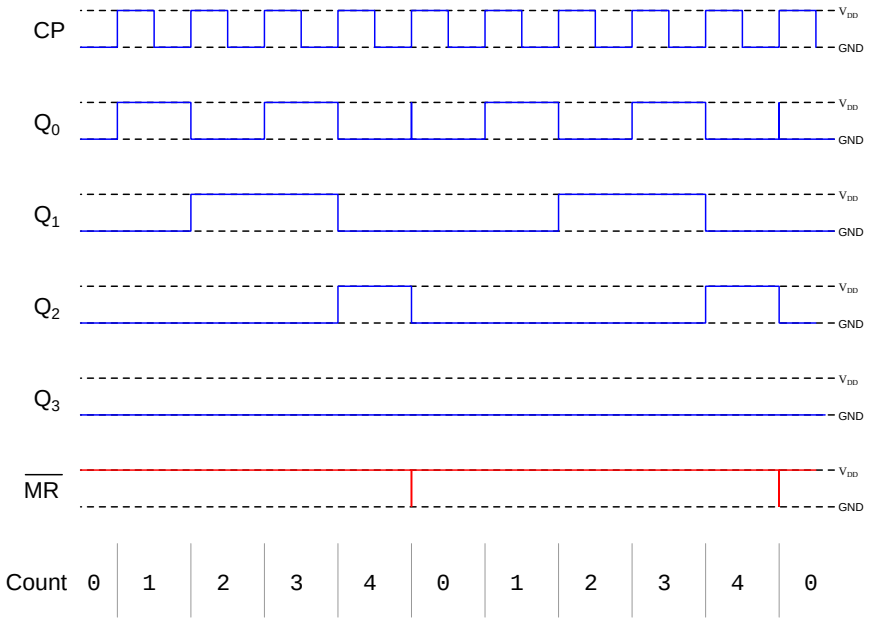
The timing diagram for this counter circuit assumes all Q output lines begin in the low state (i.e. that the count starts at zero).



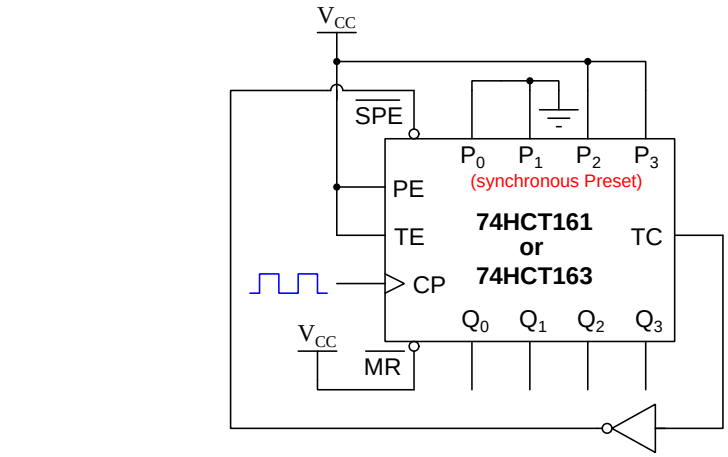


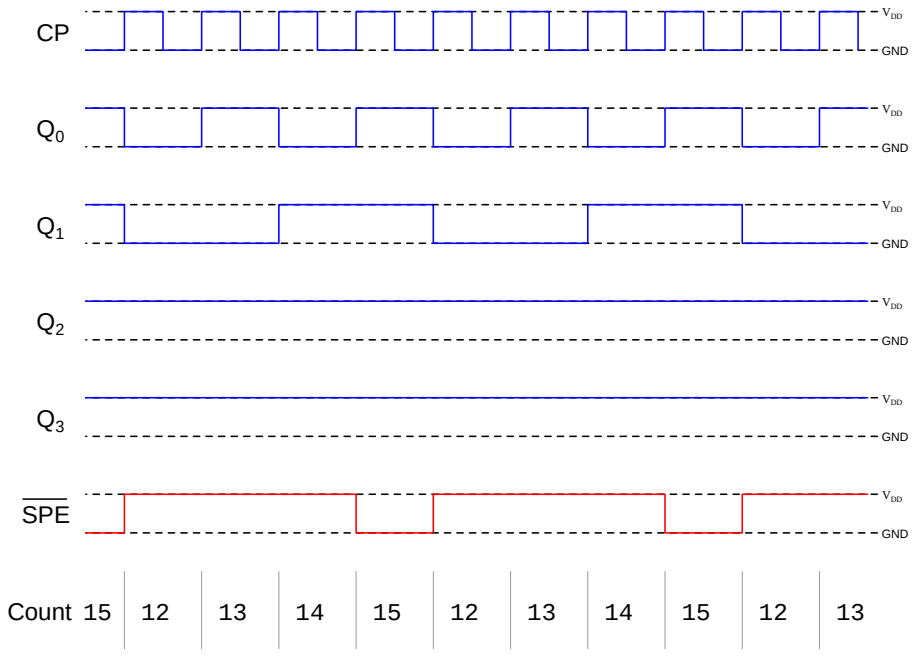
The timing diagram for this counter circuit assumes all Q output lines begin in the low state (i.e. that the count starts at zero).





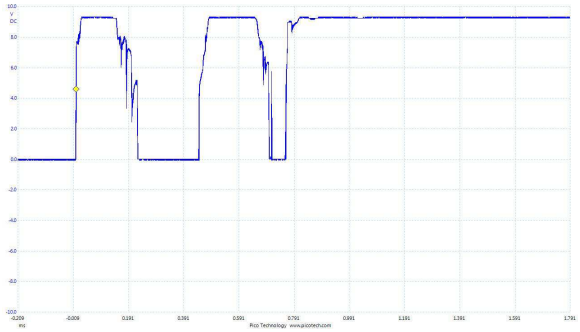
The timing diagram for this counter circuit assumes all Q output lines begin in the high state (i.e. that the count starts at fifteen).

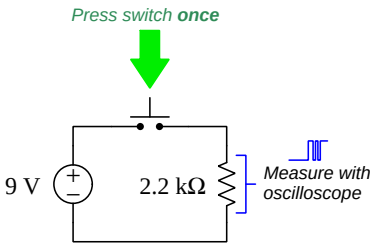




2.4 Switch contact bounce

When mechanical switch contacts open and close, they tend to do so in a "noisy" fashion, making intermittent contact when opening and when closing. We may capture these intermittent contact events using an *oscilloscope* to graphically plot voltage over time:





This particular oscillograph was captured during the time when the pushbutton switch was pressed a single time. What we see the oscilloscope detect during this time is actually *three* distinct closures of the switch's contacts, with jagged rising and falling edges. The reason we see three closures of the switch over a span of less than 1 millisecond is because the metal contact surfaces are literally *bouncing* of of one another as the pushbutton force acts to press them together. This phenonenon is not unlike dropping a ball on a hard floor surface, the ball bouncing several times before coming to rest on that floor.

Switch bounce is a common problem when any mechanical switch contact generates a signal for the input of a digital counter circuit, the purpose being for that counter to increment or decrement *once* for each switch actuation. "Bouncing" switch contacts will "fool" the counter into counting multiple times per switch actuation instead of just once.

Various techniques exist to mitigate switch bounce:

- **Use mercury-wetted switch contacts** – these are special switch contacts housed in a hermetically-sealed glass tube with a small amount of liquid mercury present. This liquid mercury adheres to the metal switch contact faces, providing a mercury "bridge" maintaining continuity between the contact faces when they are separated by very small distances, thereby maintaining contact during the "bouncing" period.

- **Use an electronic switch** – eliminating mechanical switch contacts altogether is a direct way of solving this problem. "Switches" using magnetic-field sensors and transistors as the switching elements may be used to sense the operation of a pushbutton actuator, and in the case of limit switches used to detect machine motion there exist both inductive-style and capacitive-style proximity switches that will do the same task in a bounce-less manner.

- **Connect a capacitor in the circuit** – inserting a small capacitor into the circuit to stabilize the voltage signal is another way to "de-bounce" mechanical switch contacts, preventing those

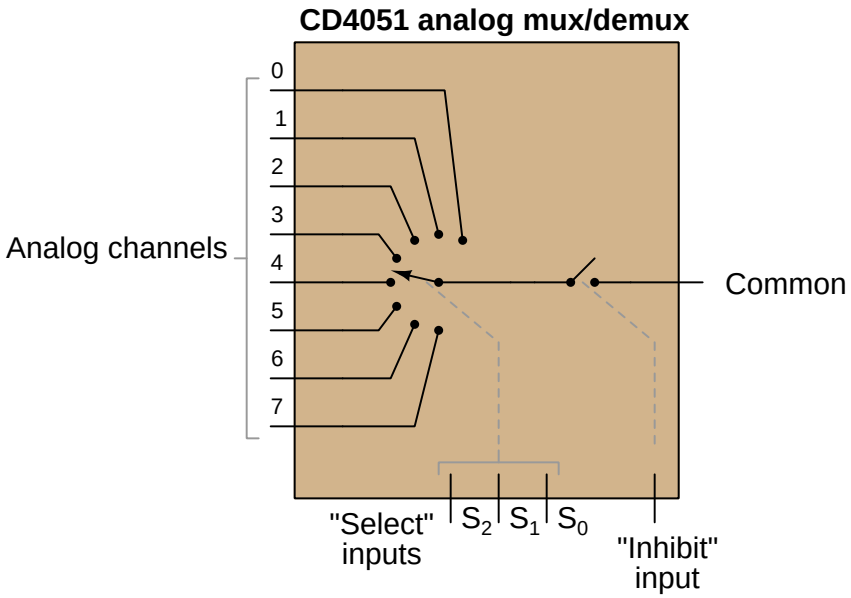
contacts from generating transient voltage pulses of too-short duration. This technique works especially well in digital electronic circuits if the capacitor-stabilized voltage signal is sampled by the input of a *Schmitt-trigger* style of logic gate, this type of gate circuit designed to tolerate voltage levels between valid "high" and valid "low" states.

- **Use a shift register** – a digital *shift register* circuit sampling the switch's signal in its serial input terminal, driven by a clock pulse signal of suitable frequency, will populate that register's bits with successive states of the switch. An AND logic function reading all parallel bits from the register then provides a "de-bounced" signal that will be "high" (1) only if *all* previous states of the switch were also "high" (i.e. only if the switch contacts have remained closed for a certain duration of time established by the clock pulse frequency and the number of parallel bits offered by the shift register).

- **Use software sampling** – if the switch's signal goes to the input of a microcontroller or other similarly programmable digital device, that device may be programmed to perform the same repeated sampling and testing of the switch signal described in the shift register technique.

2.5 Example: arbitrary waveform generator using an analog multiplexer

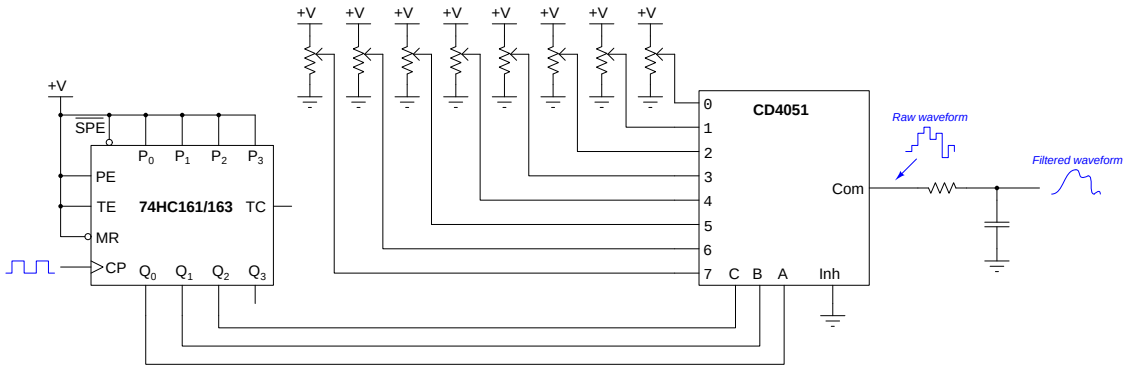
Multiplexer (mux) ICs are made for both digital (high/low) and analog (variable-voltage) signals. The model CD4051 and 74HC4051 integrated circuits are examples of analog multiplexers, which are also capable of functioning as demultiplexers since the inputs and output may both sink and source current. This particular analog mux/demux model also features an "inhibit" input which controls an internal MOSFET to either connect or disconnect the common terminal from the selected I/O pin. A functional diagram is shown below:



All digital inputs (the three select bits plus the inhibit) are active-high. Therefore, making the inhibit input pin high forces the internal MOSFET on the common terminal to turn off and places the device into an analog "high-impedance" mode where none of the eight analog channels connects to the common pin, while making the inhibit pin low connects the common pin to whichever analog channel is selected by the three select bits.

One useful circuit you can build with such an IC is a simple *arbitrary waveform generator* where a digital counter cycles through all the channel values for the mux, and the mux in turn sequentially connects eight adjustable DC voltage signals one at a time with the common signal to create a stepped waveform having whatever shape you desire using eight steps.

A schematic diagram of such a circuit appears below, DC power supply terminals omitted from the schematic for simplicity:

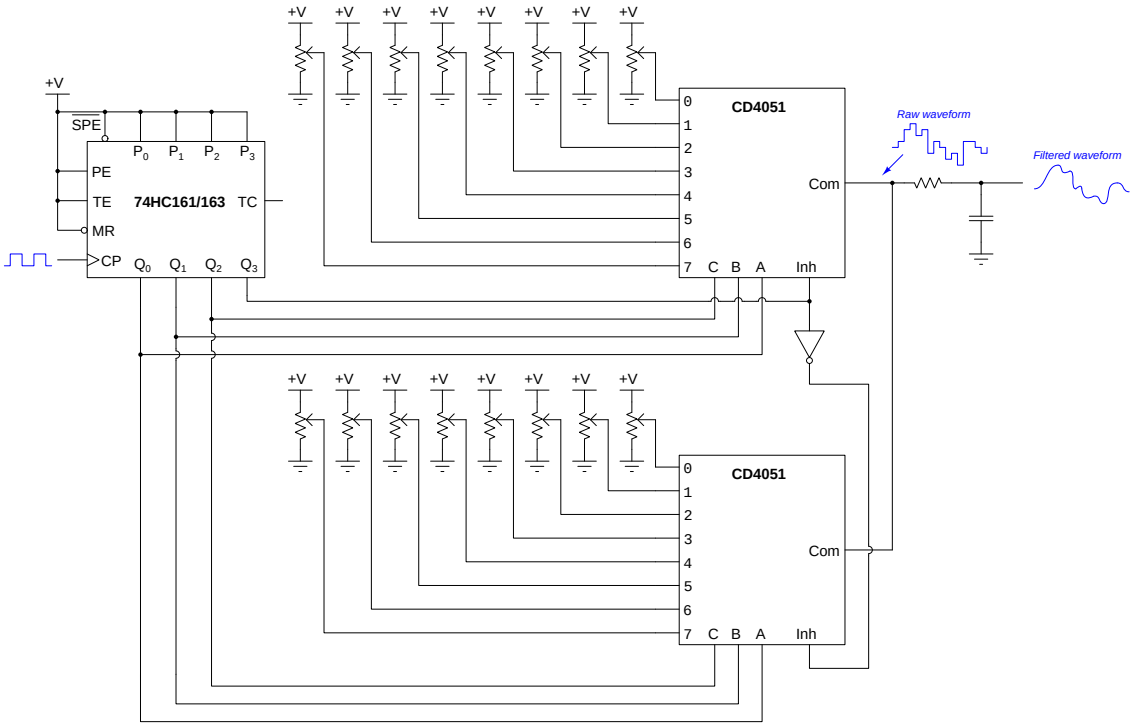


The clock signal may be supplied by a benchtop signal generator, by the output terminal of a 555 timer IC connected for astable operation, or any other suitable digital pulse signal source. Since the mux cycles through all its channels every *eight* counts of the counter IC, the final output signal will have a frequency one-eighth that of the digital clock pulse driving the counter.

Such a circuit is useful as a crude signal generator in its own right, as well as a source of interesting audio tones (for electronic music synthesis). The R and C values for the low-pass "smoothing" filter should be chosen to avoid loading down the eight signal-programming potentiometers; i.e. select a value for R that is at least ten times larger than the whole-resistance value of any single potentiometer, and select a suitable C value to provide a cutoff frequency that works well to soften the stair-step edges of the raw signal. This filter network's cutoff value, of course, depends on your intended analog signal frequency: the higher the signal frequency, the higher the cutoff needs to be in order to properly "smooth" the signal's wave-shape without attenuating it too much.

Note how the CD4051 IC's inhibit input is tied to ground, forcing it to be "low" all the time. This enables, rather than inhibits, the mux and allows the three select bits to choose which potentiometer signal gets passed along to the common output terminal and into the RC filter network. An alternative way to use this inhibit input is to connect it to a digital pulse waveform with a variable duty cycle (i.e. a pulse-width modulated or PWM signal). If this PWM pulse signal's frequency is substantially greater than the counter's clock frequency, the inhibit pin will be activated and de-activated multiple times during each of the eight "steps" of the arbitrary waveform, serving as an analog amplitude control for the arbitrary waveform. When the PWM duty cycle is low (i.e. "low" for more time each period than "high") the arbitrary waveform signal will largely pass through the 4051 mux unimpeded, but when the PWM signal is adjusted to a greater duty cycle value the mux will spend more and more time in its high-impedance mode which means the RC filter's capacitor won't be charged or discharged to as great a degree with each step of the eight-step cycle, resulting in a filtered output waveform with the same basic wave-shape but being smaller in peak-to-peak amplitude.

A more conventional use of the multiplexer's inhibit input is to permit multiple muxes to work together, allowing for more than eight analog signals to be sampled. Here we see two CD4051 analog mux ICs connected to the four-bit 74HC161/163[1] counter IC to allow a sixteen-step analog waveform to be synthesized:



In this circuit, the counter's most-significant bit (MSB) output Q_3 will be "low" during the first eight counts (0 through 7) and then "high" during the last eight counts (8 through 15), enabling the upper CD4051 mux for the first eight and the lower CD4051 mux for the latter eight count states. When each of these mux ICs are disabled, only the *other* mux will be able to send a potentiometer voltage value through to the RC filter.

Having a sixteen-step arbitrary waveform gives one the ability to generate a wave-shape with a "smoother" profile, having sixteen steps to work with rather than just eight. Of course, this means the synthesized analog signal will have a frequency that is now sixteen times slower than that of the digital clock.

[1] The only functional difference between the model 74HC161 and 74HC163 counters is the behavior of their reset inputs, the '161 being asynchronous reset and the '163 being synchronous. Since we're not using the reset input at all, this distinction is irrelevant and therefore either model of counter will suffice. Both of these counter ICs happen to be *synchronous* in their counting, though, which is good for this application because otherwise a counter with asynchronous ("ripple") count bits might cause the mux to falsely select the wrong channel as it transitions from one count state to the next.

Chapter 3

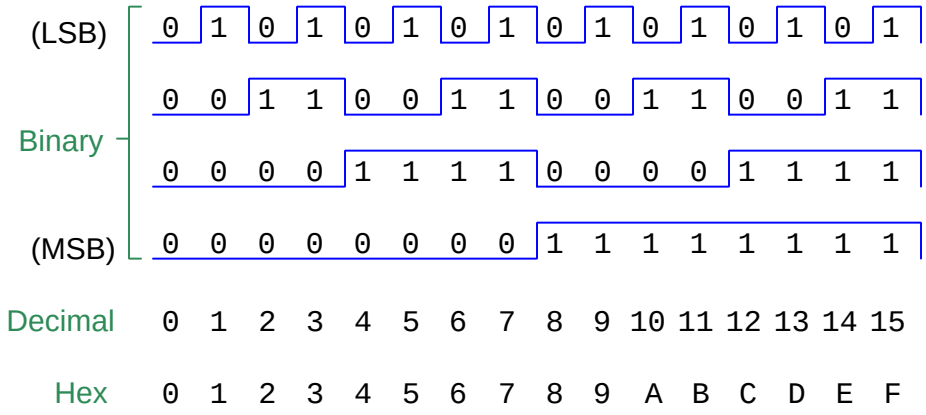
Simplified Tutorial

3.1 Binary count sequences

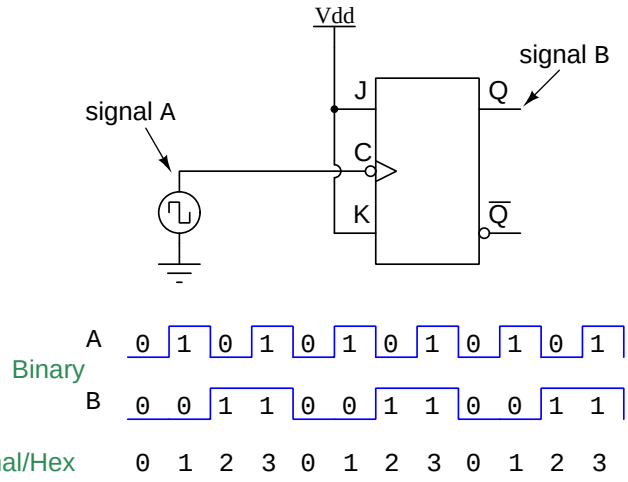
If we closely examine the bit states of a binary count sequence (as shown below), it becomes evident that the respective bits of the binary word are *toggling* (oscillating) between 0 and 1 on a periodic basis. Beginning with the least-significant bit (LSB), scan the sequence of numbers vertically to see how that bit alternates between 0 and 1 with every step of the count sequence. Then, examine the next-most significant bit to identify the alternating pattern there:

Binary	Decimal	Hex
0 0 0 0	0	0
0 0 0 1	1	1
0 0 1 0	2	2
0 0 1 1	3	3
0 1 0 0	4	4
0 1 0 1	5	5
0 1 1 0	6	6
0 1 1 1	7	7
1 0 0 0	8	8
1 0 0 1	9	9
1 0 1 0	1 0	A
1 0 1 1	1 1	B
1 1 0 0	1 2	C
1 1 0 1	1 3	D
1 1 1 0	1 4	E
1 1 1 1	1 5	F

This pattern of decreasing frequency becomes more apparent when we represent each bit as a digital pulse waveform and plot them adjacent to one another for comparison:



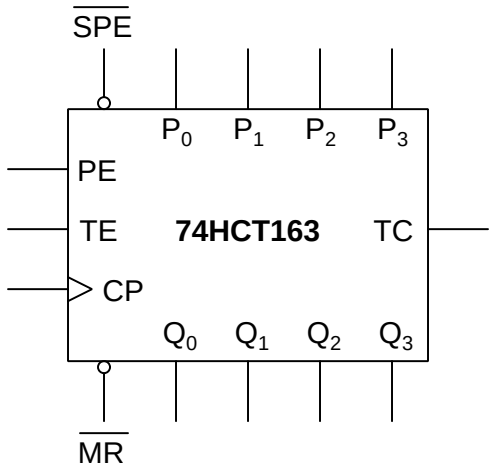
Note how each successive bit – from the LSB to the MSB – toggles at *half* the frequency as the bit before it. If we wish to build a digital circuit that counts in binary, then, all we need is a collection of elements capable of dividing the frequency of a pulse signal by a factor of two. Fortunately, there is such a digital "building block" element well-suited for this purpose, and it is called a *JK flip-flop*. Below is a schematic diagram of this device performing the 2:1 frequency division necessary to create a two-bit binary "up" counting sequence from a single square-wave "clock" signal:



Also note how the count sequence naturally "rolls over" back to zero after reaching the maximum count value, as a function of the simple frequency-division method of counting. If we cascade multiple JK flip-flops, each one triggered by the output of the previous flip-flop, we may construct binary counter circuits with as many bits as there are flip-flops. For a deeper exploration of counter circuit design, consult the Full Tutorial chapter of this module.

3.2 Counter ICs

Complete counters are manufactured as integrated circuit (IC) assemblies, one such counter shown below:



While counter models differ in features and not all match those of the 74HCT163, it is nevertheless instructive to identify the inputs and outputs of this counter to get a general idea of what IC counters can do:

- Outputs Q_0 through Q_3 are the bits of the count, from LSB (Q_0) to MSB (Q_3)
- Output TC is the Terminal Count which goes "high" when the count value reaches its maximum ("terminus") before resetting back to zero, which is useful when cascading multiple counter ICs
- Input $\overline{MR}$ is the Master Reset, in this case being active-low which means a "low" state resets the count value to zero and a "high" state permits normal counting
- Input CP is the Clock Pulse, which for this particular counter IC increments the count value at every rising edge
- Inputs TE and PE are both active-high enables but function in slightly different ways, the TE forcing the TC output in addition to "freezing" the count value when disabled
- Inputs $\overline{SPE}$ and P_0 through P_3 work together to "preset" or "force" the count to a specified value; the bit states driven to P_0, P_1, P_2, and P_3 taking effect at the next clock pulse when the $\overline{SPE}$ input is enabled (active-low)

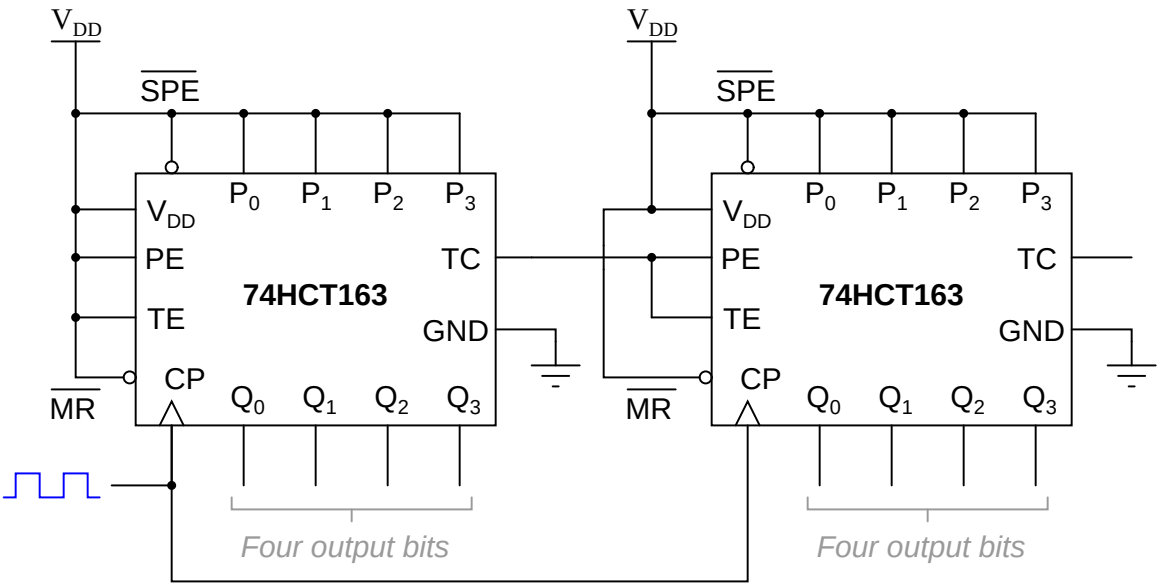
Some common features of IC counters not present in the model 74HCT163 include up/down count direction control and BCD versus binary count range. It is also important to realize the distinction between *synchronous* versus *asynchronous* inputs: a "synchronous" input is one where an active input state does not have an effect on the count value until the next clock pulse arrives, whereas an "asynchronous" input is one whose active status immediately affects the count. For example,

some counters' reset inputs are synchronous while other models have asynchronous reset inputs – the model 74HCT163 happens to sport a synchronous $\overline{MR}$ input while the model 74HCT161's Master Reset input is asynchronous.

3.3 Extending count range

All counter ICs are inherently limited in the number of bits they offer, for the simple reason that every counter has a limited number of internal flip-flop circuits and a limited number of pins. If we have an application where more bits are needed than what our available counter ICs offer, we may either build our own counter from individual flip-flops or, more practically, *cascade* multiple counter ICs together. This is what the "TC" (Terminal Count) and enable inputs are for on a counter IC: a way for the preceding counter to control when the next counter increments or decrements.

An example of counter cascading is shown below, again using the model 74HCT163:



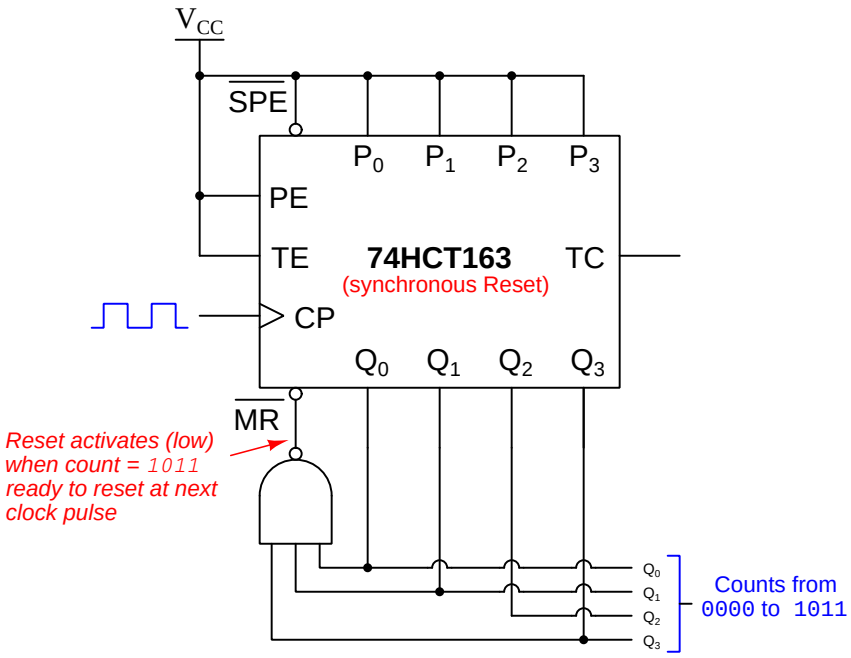
The left-hand counter increments with every clock pulse, but the right-hand counter waits until the left-hand counter has reached its terminal count value (i.e. `1111`) before recognizing the next clock pulse. This way, the right-hand counter will increment when the left-hand counter "rolls over" from `1111` back to `0000`. Together, these cascaded four-bit counter ICs comprise an eight-bit counter. Not surprisingly, we may cascade more than two counters to achieve a counter array with as many bits as we might require.

3.4 Limiting count range

Another practical concern with using counter circuits is how to limit the terminal count value if necessary. For example, suppose we needed a four-bit counter to count from 0000 (zero) to 1011 (eleven) rather than all the way up to 1111 (fifteen)? How may we reduce the number of total states in the counter's count sequence (also known as the *modulus* of the counter) from sixteen to twelve?

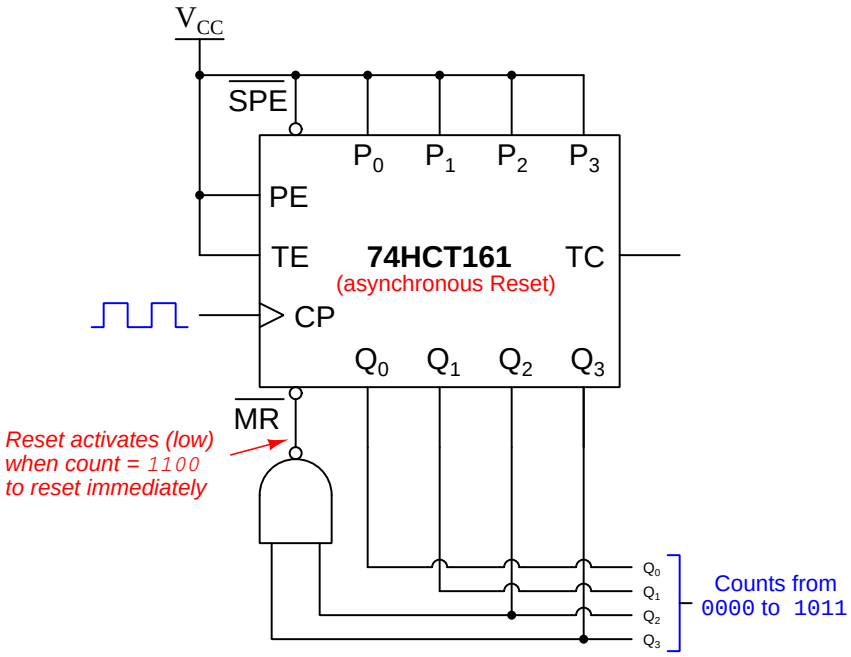
A simple method for reducing the modulus of a counter is to use logic to *decode* for a count value exceeding the desired maximum, and trigger a reset when that happens. For example, if we wish to limit a four-bit counter to a modulus of 12 (i.e. make it count from zero to a maximum of eleven), we could connect an AND gate or a NAND gate (depending on whether the reset input is active-high or active-low) to activate the counter's master reset input when we wish the count value to return to zero.

Here is an example of the 74HCT163 counter configured for a modulus of twelve:



The model 74HCT163 counter happens to have a *synchronous* Master Reset function, which means the reset occurs when the $\overline{MR}$ input is active *and* the next clock pulse arrives. If we want this IC's count to progress from 0000 to 1011, then the NAND gate needs to activate the $\overline{MR}$ input at the count value of 1011 so that the next clock pulse brings the counter to 0000 rather than incrementing to 1100.

If our counter IC happens to have an *asynchronous* reset input (i.e. it resets as soon as the reset input activates, without waiting for the next clock pulse) then we need to have the gate decode for the first illegitimate count value (in this case, `1100` = twelve) so that it counts all the way to the maximum value we want, and then as soon as it tries to increment to the next count value it very quickly resets itself back to zero. The model 74HCT161 differs from the 74HCT163 only in the fact that its $\overline{MR}$ input is asynchronous rather than synchronous:



A practical example of reduced-modulus counting is found in the BCD counters used to drive digits of a time clock display. Each "hour" count needs to increment after 59 seconds, which means the "minutes" count must have its modulus limited to 60. Similarly, the "hour" count must be modulus-limited to either 12 or 24 (depending on whether it is a 12-hour or 24-hour clock), and furthermore the "reset" must force the "hour" counter to one[1] rather than zero, since a clock rolls over from its terminal count value to 1:00 rather than 0:00!

[1] This is a good application for the $\overline{SPE}$ and P_0 through P_3 inputs on the 74HCT163 counter, using the "preset" inputs to force the counter to `0001` rather than using the "reset" input to force the counter to `0000`.

Chapter 4

Full Tutorial

4.1 Latches and flip-flops

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*	

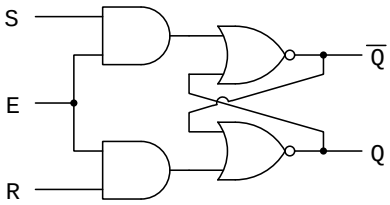
SR latches based on NOR gates latch whenever both inputs are low, whereas SR latches based on NAND gates latch whenever both inputs are high. A latch is considered to be "set" when Q is high and $\overline{Q}$ is low. The "reset" state is just the opposite: Q low and $\overline{Q}$ high. If ever a latch's two outputs are found in the same state, it is considered *invalid*.

[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.

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]:

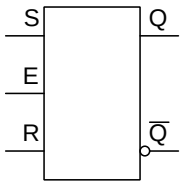
A more useful variation of the SR latch is the *enabled SR latch* with a third input line (E). This additional input line's state either enables or disables the set (S) and reset (R) inputs' functions. When enabled, this latch behaves as any normal SR latch; when disabled, this latch circuit remains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

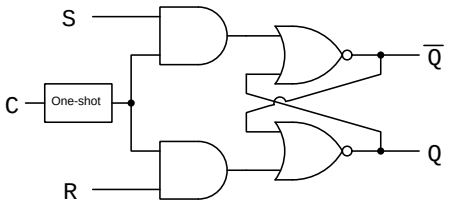
Enabled SR latch symbol



[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).

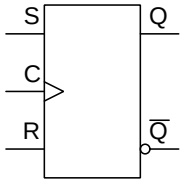
Another variation on this theme is to equip the latch circuit with a "one-shot"[3] circuit designed to detect either the rising or falling edge of a square-wave pulse signal, enabling the latch only during that brief transition from low to high (positive edge) or from high to low (negative edge) depending 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



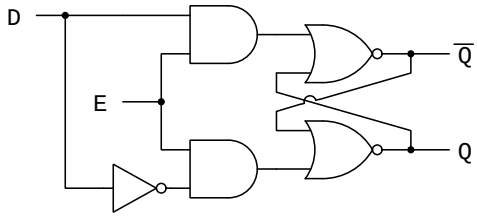
C	S	R	Q	$\overline{Q}$
X	0	0	*Latch*	
X	0	1	*Latch*	
X	1	0	*Latch*	
X	1	1	*Latch*	
⎍	0	0	*Latch*	
⎍	0	1	0	1
⎍	1	0	1	0
⎍	1	1	0	0

SR flip-flop symbol



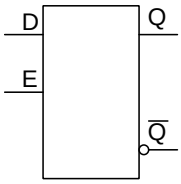
The D-type latch and D-type flip-flop are simplified versions of the SR latch and flip-flop, respectively, having just a single "data" (D) input rather than separate set (S) and reset (R) inputs. With this alteration, the device no longer has an "invalid" state:

D latch internal schematic

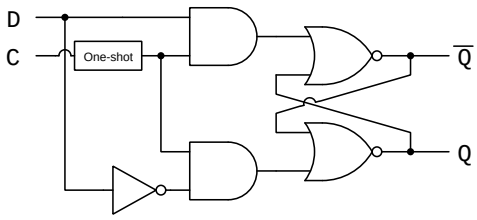


E	D	Q	$\overline{Q}$
0	0	*Latch*	
0	1	*Latch*	
1	0	0	1
1	1	1	0

D latch symbol

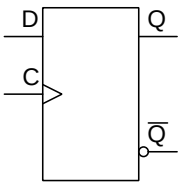


D flip-flop internal schematic



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

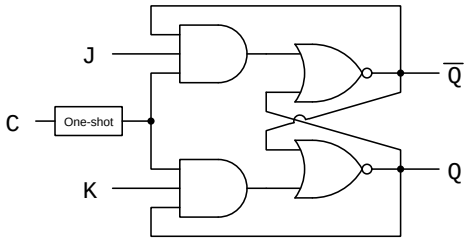
D flip-flop symbol



[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.

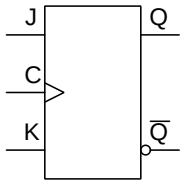
Perhaps the most useful type of flip-flop is the *JK* form. This uses an additional layer of feedback to give it a novel mode called *toggle*. When both J and K inputs are simultaneously active, and the clock pulse signal transitions, the "toggle" mode results in the Q and $\overline{Q}$ output states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

JK flip-flop internal schematic



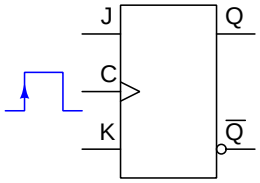
C	J	K	Q	$\overline{Q}$
X	0	0	*Latch*	
X	0	1	*Latch*	
X	1	0	*Latch*	
X	1	1	*Latch*	
⎍	0	0	*Latch*	
⎍	0	1	0	1
⎍	1	0	1	0
⎍	1	1	*Toggle*	

JK flip-flop symbol



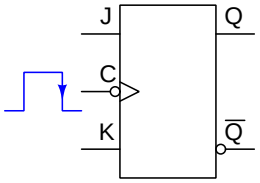
As with other flip-flops, JK flip-flops are designed in both *positive-edge-* and *negative-edge-triggered* versions. If a flip-flop's clock input happens to be a negative-edge style, an inversion "bubble" will be shown at that input terminal of the device. Both versions of a JK flip-flop appear in the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

Positive edge-triggering



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

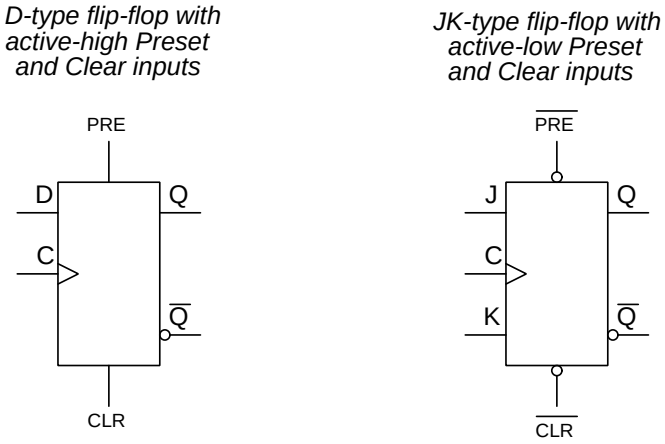
Negative edge-triggering



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!

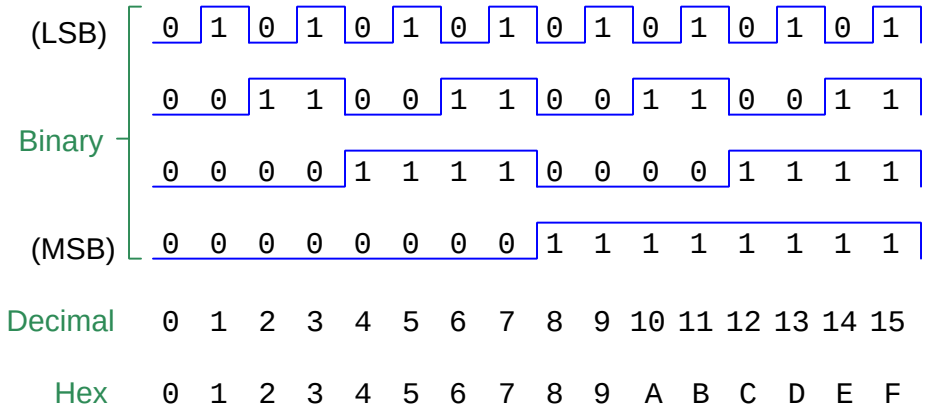
4.2 Counter circuit fundamentals

An important application of flip-flops is in the construction of digital *counter* circuits. A "counter" simply increments or decrements a binary number value with every pulse of a clock signal. We may begin our exploration of counter circuits by studying the count sequence of a four-bit binary number (from 0000 to 1111):

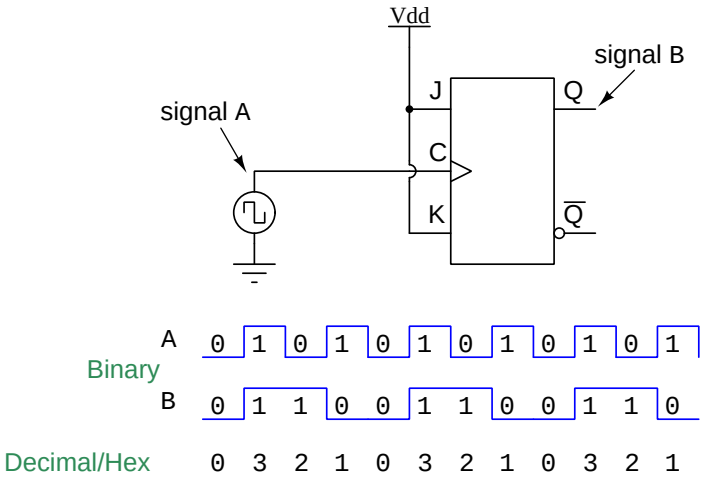
Binary	Decimal	Hex
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

Note how the least significant bit (LSB) toggles (i.e. reverses its logical state) at every step in the count sequence, and each succeeding bit toggles at one-half the frequency of the prior bit. The most significant bit (MSB) only toggles once during the entire sixteen-step count sequence, at the transition between 7 (0111) and 8 (1000).

This pattern of decreasing frequency becomes more apparent when we represent each bit as a digital pulse waveform and plot them adjacent to one another for comparison:

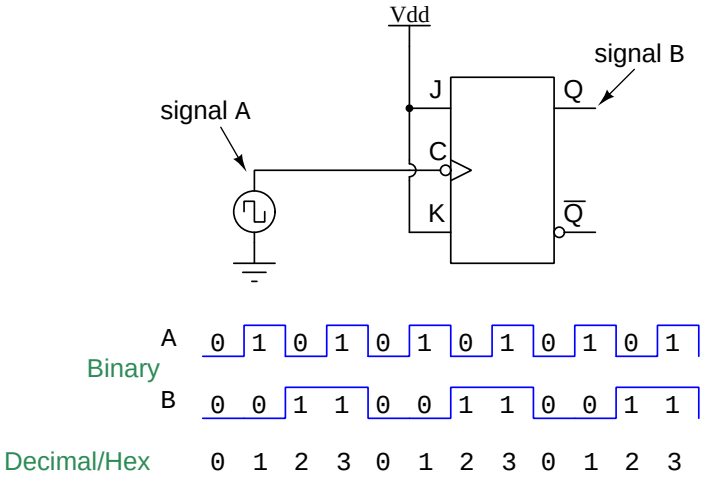


Realizing that the key to a binary count sequence is a 2:1 *frequency division* ratio for each bit from the LSB through the MSB, we may investigate the use of JK flip-flops for this task. Recall that when a JK flip-flop is in its "toggle" mode, its outputs change state once for every two changes of state at the clock input. A single JK flip-flop[4] configured to always be in toggle mode will yield such a 2:1 frequency ratio, and by comparing its clock and Q output logic levels we see a decrementing two-bit binary counting sequence:



[4]Power supply connections have been omitted for the flip-flop circuit for the sake of simplicity. It should be clearly understood, though, that *all* digital integrated circuits require power applied to their supply pins in order to function.

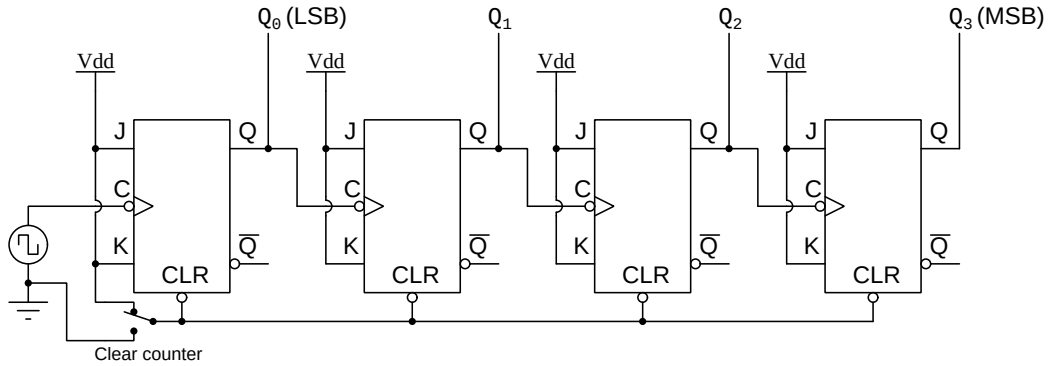
If we desire an incrementing binary count, all we need to do is use a negative-edge triggered flip-flop so that the toggle action occurs every time the clock signal transitions from a 1 to a 0 instead of when it transitions from a 0 to a 1:

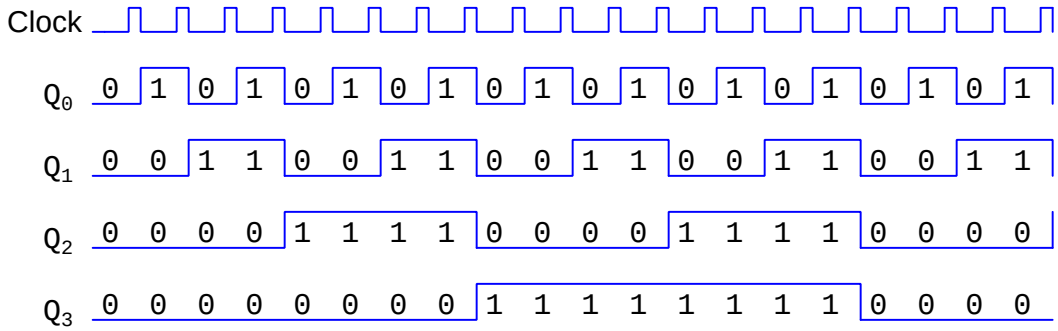


4.3 Asynchronous counters

Building a counter with more bits is as simple as cascading JK flip-flops together so that each successive stage divides the signal frequency by two. Note that we typically do *not* include the clock pulse itself as one of the counter bits, as that was something we did simply for the proof-of-concept example using a single flip-flop. Instead, we let the clock pulse oscillate freely and limit ourselves to the Q outputs of the flip-flops for our counter bits.

The following circuit shows a four-bit binary counter using four JK flip-flops:



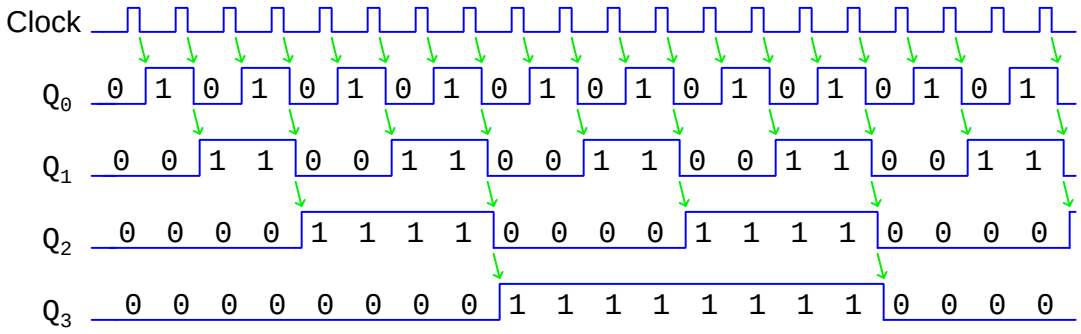


Note how each flip-flop toggles when it receives a downward-transitioning pulse signal from the less-significant flip-flop before it, thanks to the negative-edge triggering feature of these flip-flops. If we wished the counter to decrement instead of increment, we could replace all the flip-flops with positive-edge triggered units[5].

Note also how all the flip-flops' *clear* inputs have been paralleled and fed by a single switch. This gives us the ability to force our four-bit counter circuit to an all-clear (i.e. 0000) state at will. Since the Clear input is *asynchronous*, we may clear the counter independent of the clock pulse signal.

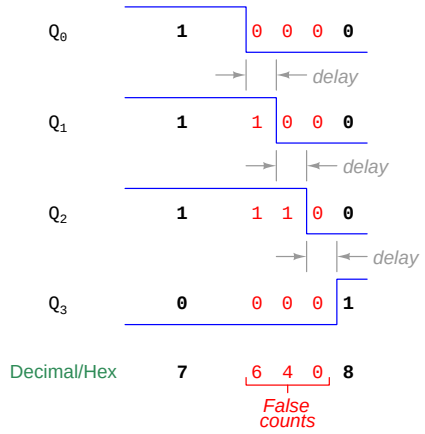
[5] Alternatively, we could use the $\overline{Q}$ outputs instead of the Q outputs, and this would also result in a decrementing count value.

An interesting characteristic of this counter circuit is that bit transitions which are supposed to be simultaneous aren't perfectly simultaneous. The reason for this imperfection is the *propagation delay* inherent to digital logic circuits: a small time delay between the flip-flop receiving the clock pulse and the flip-flop's toggle response at its Q output terminal. The following timing diagram will show the same four-bit incrementing count sequence, but with exaggerated propagation delays and angled indicator arrows to better illustrate the effect:



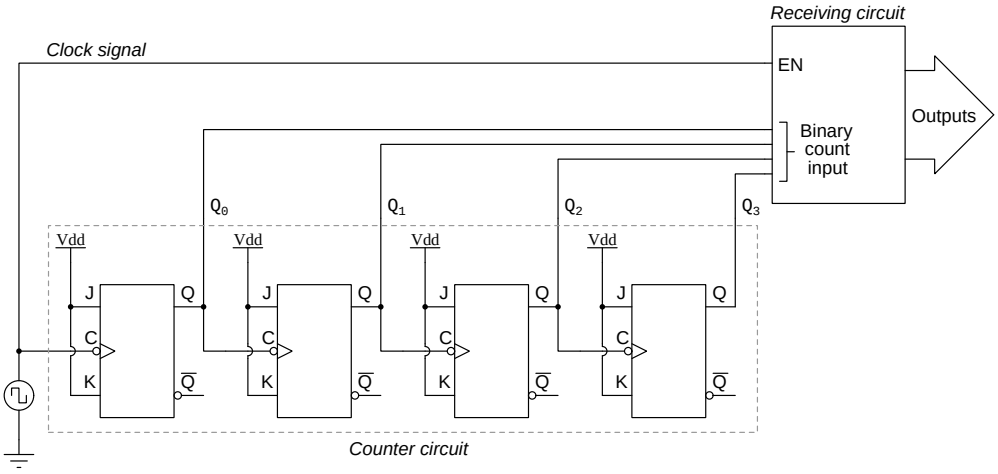
For this reason, this design of digital counter is called an *asynchronous* or *ripple* counter. The more flip-flop stages inside an asynchronous counter, the greater the total time delay between bit transitions from LSB to MSB.

The time delay inherent to asynchronous counters may be serious or negligible depending on the application. If, for example, the counter is being used to drive a digital display and nothing more, these tiny delay times will go unnoticed by human vision. However, if the counter's binary output feeds some high-speed digital device capable of reading the counter's bit states before all the bits have "settled" to their correct values, errors will result. An example is shown here with exaggerated propagation delay times, focusing on the transition between the binary count `0111` and `1000`, which is a worst-case example because every one of the JK flip-flops must toggle:



Instead of cleanly transitioning from a "count" value of seven to eight, the value jumps from seven to six, four, and zero (in rapid succession) and then finally settles at eight.

A clever solution to the problem of a "rippling" count generating false result for a receiving device is to *strobe*[6] that device. Many types of digital circuits receiving multi-bit binary inputs (e.g. decoders and multiplexers) are equipped with at least one *enable* input similar in function to the E input on an enabled SR or D latch. Such a circuit will respond to input states only when all of its enable inputs are active. The technique of "strobing" connects one of the receiving circuit's "enable" inputs to a digital signal timed to disable the receiving chip during the period when rippling may occur. Often the strobing signal may be the same clock pulse driving the counter circuit:

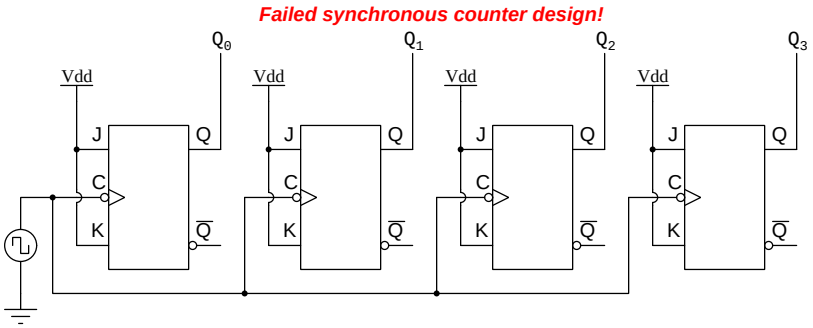


The counter increments on every falling edge of the clock pulse, but the receiving circuit (with its active-high EN enable input) ignores the counter's output as soon as the clock pulse goes low. Only when the clock pulse goes high again will the receiver read the four bit states of the binary count, which should be plenty of time for the counter's outputs to "ripple" and settle to their proper states.

[6] The concept of "strobing" a digital circuit is really the same as using a flashing strobe light to "freeze" the motion of a rotating object: if the strobe light brightly illuminates the object in the same position every time, the object will appear to be still rather than moving. It's the precise timing of the strobe light's pulse that "hides" the rest of the object's motion from our vision. For the digital counter circuit the clock signal's strobe effect on the latch hides the ripple phenomenon so that all we "see" at the latch's output is a clean count.

4.4 Synchronous counters

In order to create a truly *synchronous* counter circuit devoid of ripple, we must somehow ensure all of its flip-flops clock simultaneously instead of each flip-flop waiting on the previous flip-flop to toggle. Yet, if we simply connect the clock pulse signal in common to all the flip-flop clock inputs and keep the J and K inputs tied high, the circuit will *not* count in a binary sequence. Instead, it will alternate between `0000` and `1111`:



Returning to our four-bit binary count sequence for inspiration[7], our task is to find some pattern within the bit states we may use to selectively toggle each stage of the counter, instead of having each flip-flop toggle with every clock pulse:

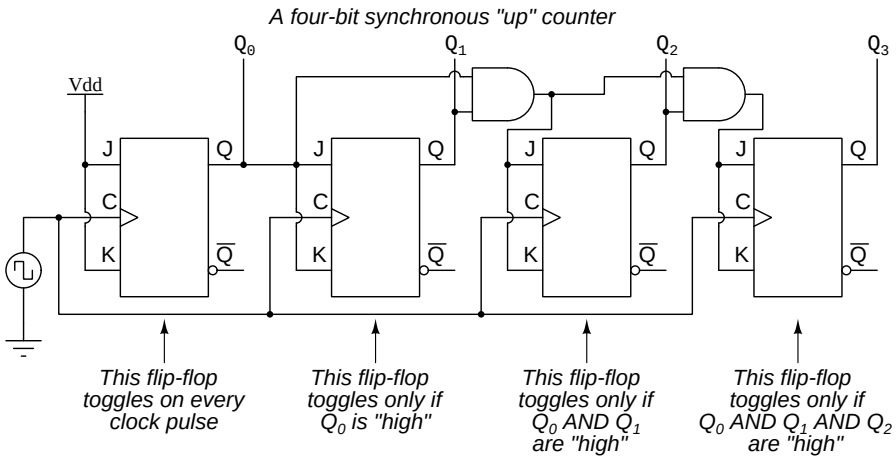
Binary	Decimal	Hex
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

It should be apparent that as the count value increments, each higher-order bit toggles only when

[7] This is not meant to be humorous. When faced with a design problem it is *usually* necessary to "go back" and review basic concepts, in this case the sequence of a four-bit binary count, and try as hard as we can to look at those familiar concepts afresh.

all lower-order bits are high. Thus, we may redesign our synchronous counter so that each flip-flop's J and K inputs go high only when all lower Q bits are high.

The following diagram shows the necessary connections to make a four-bit *synchronous* "up" counter:

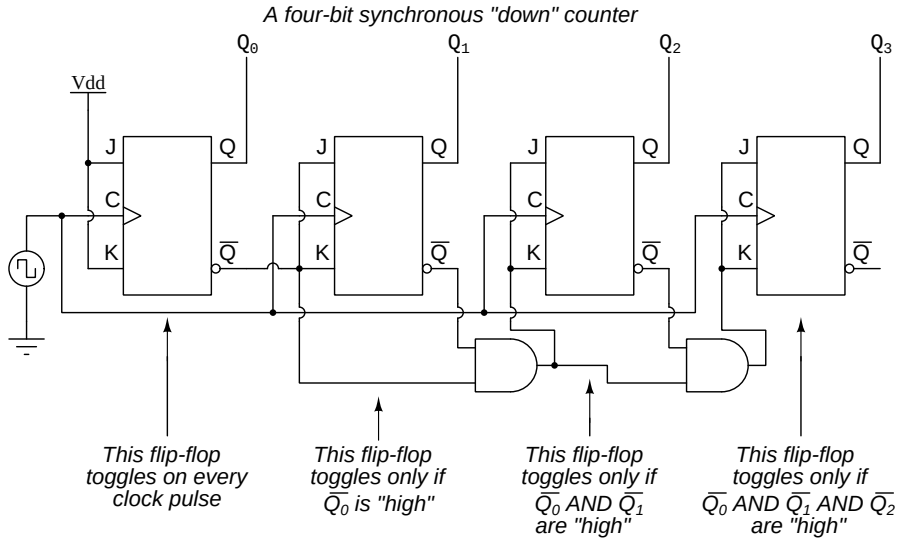


In the circuit shown, we happen to be using positive-edge-triggered JK flip-flops rather than negative-edge, but this is no longer of of any importance. In the asynchronous counter design this distinction determined the counting direction, but here in the synchronous design the count direction is a function of the AND gate logic rather than the clock pulse polarity. We could replace all four flip-flops with units having the opposite clock input type and its count value would still increment.

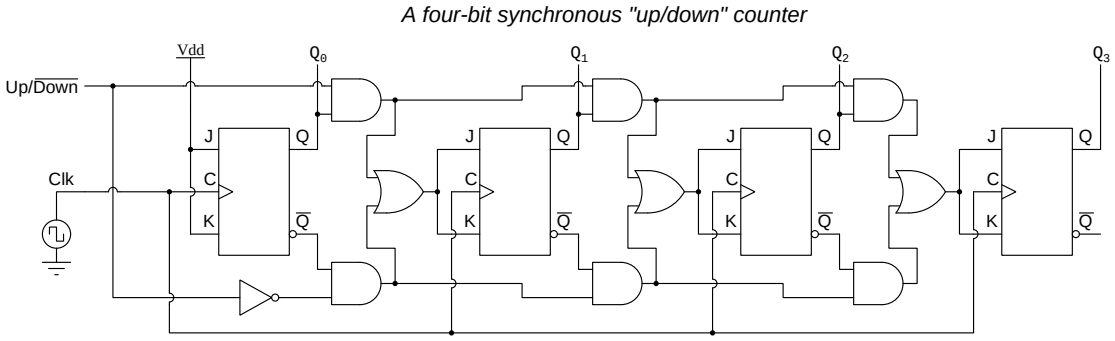
If we analyze this synchronous counter closely, a design flaw seems to be evident. Suppose a low-order flip-flop happens to have its Q output in the low state, ready to toggle to high with the next clock pulse. What is there to prevent that flip-flop's Q output from switching to the high state *and* the next higher-order flip-flop from also toggling because it will see the first flip-flop's Q output go high simultaneous with the clock pulse? Stating the problem differently, we may ask the question "What does a JK flip-flop do when both J and K inputs go from low to high at the same moment in time it receives a clock pulse?"

The answer to this apparent problem is two-fold. First, the next flip-flop actually will *not* see its J and K inputs go high at the same time the clock pulse arrives, thanks to the inherent propagation delay of the previous flip-flop; instead, its J and K inputs will remain low for a brief moment in time after the clock pulse arrives because the previous flip-flop cannot *instantaneously* toggle. Second, even if the J and K high states arrived perfectly simultaneous with the clock pulse, the flip-flop would still not be able to toggle because those high states at the J and K inputs would not have been present for the minimum required *set-up time*. The phenomenon of "set-up time" is also a consequence of gate propagation delay, but internal to the flip-flop's own gates rather than in the gates of the previous flip-flop.

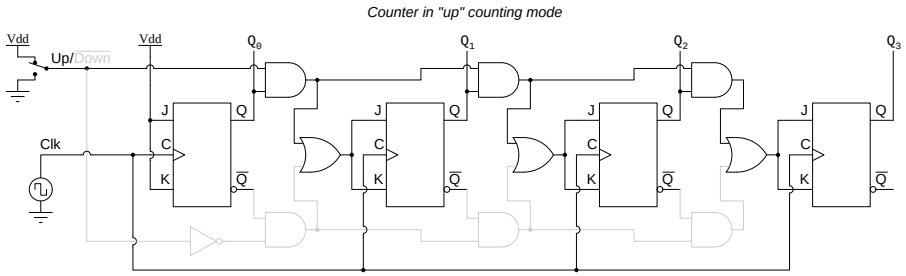
To make a synchronous *down* counter, we need to enable each higher-order flip-flop to prepare to toggle when all lower-order bits are *low*. Fortunately, the $\overline{Q}$ outputs on each flip-flop provide a convenient source of inverted logic signals to trigger AND gates at the necessary times:



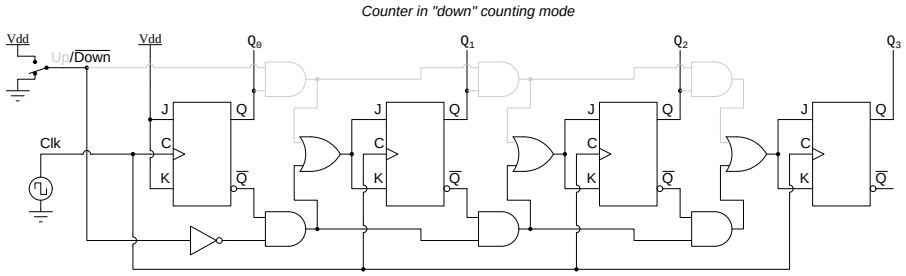
Taking this idea one step further, we may build a synchronous counter circuit with selectable between "up" and "down" count modes by having dual lines of AND gates detecting the appropriate bit conditions for incrementing and decrementing count sequences, respectively, then use OR gates to combine the AND gate outputs to the J and K inputs of each succeeding flip-flop:



Analyzing this up/down counter in its "up" count mode, showing the disabled gates in grey:



Next, showing the counter in its "down" count mode, with disabled gates colored grey:



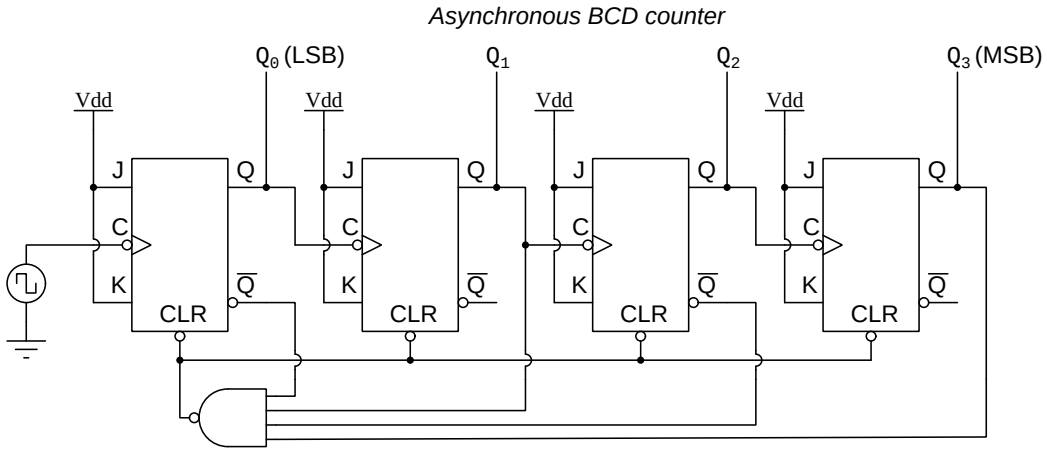
Comparing asynchronous (ripple) and synchronous counter circuits, we see the asynchronous type is simpler to build but suffers from the ripple effect on its output bits. The synchronous counter type lacks that weakness but is a more complex circuit.

4.5 Modulus

The *modulus* of a counter is its maximum number of unique and stable output states. For a plain binary counter, modulus is simply two raised to the power of the number of output bits. Our four-bit counter examples would therefore have moduli of $2^4 = 16$. In some applications, though, we may desire to truncate the counter's range so that it cannot count to its full maximum.

One such application is a *BCD counter*, where the count range of the four-bit counter circuit must be limited from `0000` (zero) to `1001` (nine) in order to represent a single digit of a decimal number. A normal 4-bit counter has a modulus of sixteen (i.e. a count sequence from zero to fifteen, inclusive), but to serve as a BCD counter the modulus must be *ten*.

A simple method to limit modulus is to add logic gates wired to detect the first invalid count value, and immediately trigger the counter to reset back to zero when that occurs. The following example of an asynchronous BCD counter shows how a four-input NAND gate may be wired to sense a count value of `1010` and immediately reset so that the counter seems to only cycle from zero through nine:



Here we can see how useful it is for each flip-flop to have complementary outputs (i.e. both Q and $\overline{Q}$): we exploit this fact in connecting the four inputs of the NAND gate to the respective flip-flops, connecting to Q if we wish to detect a high output state and connecting to $\overline{Q}$ if we wish to detect a low output state[8]. Here, the $\overline{CLR}$ line goes low to reset all flip-flops when Q_3 is high and Q_2 is low and Q_1 is high and Q_0 is low: the combination of states representing "ten" which is our first invalid state for an incrementing BCD counter.

Forcing a reduced modulus for a *down* counter is more complicated, because now the task is to *pre-set*[9] the flip flops to the highest valid count value whenever the first invalid state (all 1's for a down-counter) is detected. Again, a multiple-input NAND gate may be used to sense the invalid

[8] This works because for any JK flip-flop any time Q is low, $\overline{Q}$ is guaranteed to be high.

[9] A simple "thought experiment" proves why this is necessary. Imagine a four-bit BCD "down" counter whose count sequence should be 9-8-7-6-5-4-3-2-1-0. The next state after zero should be nine, but a regular (non-truncated) four-bit "down" counter will naturally cycle back to sixteen (binary `1111`). Therefore, the logic gate must be wired to detect sixteen, and then force the flip-flops to a state of `1001` which is nine.

state, but instead of activating all the flip-flops' "clear" inputs it now must selectively *clear* some flip-flops and *set* others.

Decoding a value of ten (binary `1010`) and clearing all flip-flops when that unwanted count is detected is a method that works to create a BCD counter only if the flip-flops' clear inputs are *asynchronous* in nature. Here we use the term "asynchronous" to refer to the immediacy of those inputs' effect, rather than describing the simultaneity of the four output bit transitions. In other words, resetting all the flip-flops to zero as soon as we detect a count value of ten results in a zero-through-nine count range only if those clear inputs don't have to wait for the next clock pulse to act. If we were to build the exact same counter using JK flip-flops that had *synchronous* clear inputs instead, what would happen is a count sequence of 0-1-2-3-4-5-6-7-8-9-10 and then back to 0: the NAND gate would still properly decode the `1010` output bit states, but the activated clear inputs would not clear the flip-flops until the *next* clock pulse which would mean the "ten" count value would persist just as long as the others. In order to build a BCD counter using JK flip-flops having synchronous clear inputs, we would have to wire the NAND gate to decode a count value of nine (binary `1001`) rather than ten.

Although we may build reduced-modulus counter circuits using JK flip-flops having either type of clear input (synchronous or asynchronous), if we choose to use asynchronous-clear flip-flops we must know that for a brief moment in time our BCD counter circuit actually will output an invalid (ten) count, for as long as it takes for the combined propagation delay of the NAND gate and the flip-flop clear inputs. However, the solution of *strobing* mentioned previously works just as well to mask this problem as it does to mask the problem of ripple.

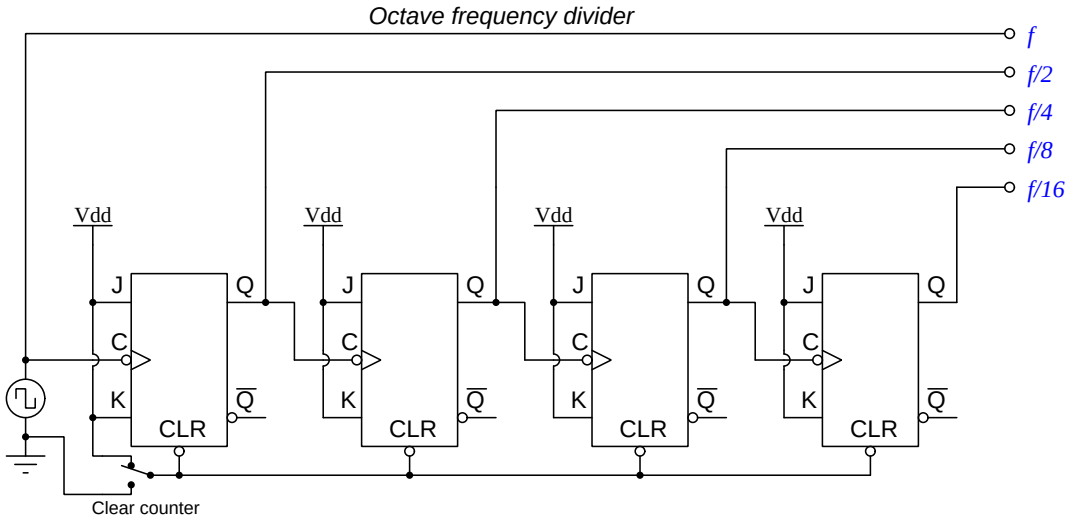
Generalizing to all reduced-modulus up-direction counter applications where we use external logic to "decode" a particular count value (n) which will trigger the reset:

- With synchronous reset the count sequence will proceed from 0 to n and then return to 0, resulting in a modulus equal to $n + 1$
- With asynchronous reset the count sequence will proceed from 0 to $n - 1$ and then return to 0 immediately following an extremely brief count value of n, effectively resulting in a modulus equal to n

4.6 Frequency division

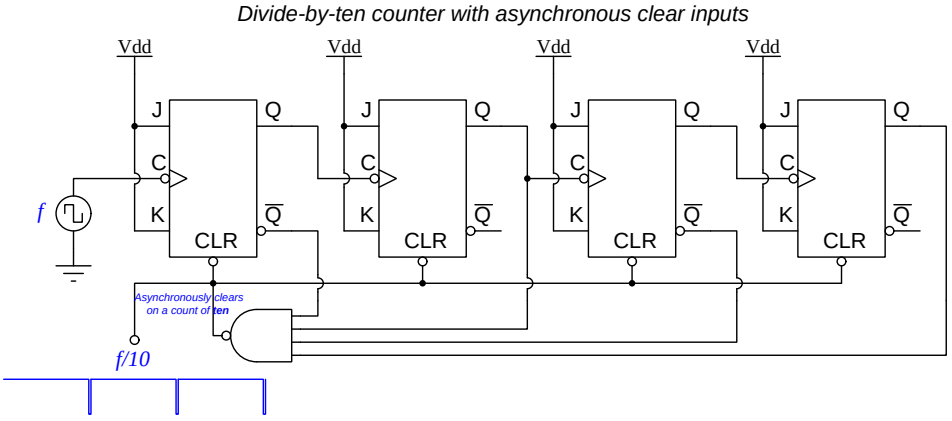
In addition to the obvious application of converting pulse signals into sequential binary count values, counters may also serve as *frequency dividers* for digital pulse signals. This is somewhat ironic, as our introduction to binary counting began with the observation that successive bits of a binary number represent doubled or halved pulse frequencies. After seeing how frequency division could be employed to count in binary, we are now exploring the use of counters to divide frequency.

A single JK flip-flop configured for "toggle" mode divides frequency by two. Two cascaded JK flip-flops divide frequency by four. We may revisit our plan for an asynchronous (ripple) counter and see how this might be used for power-of-two frequency division:



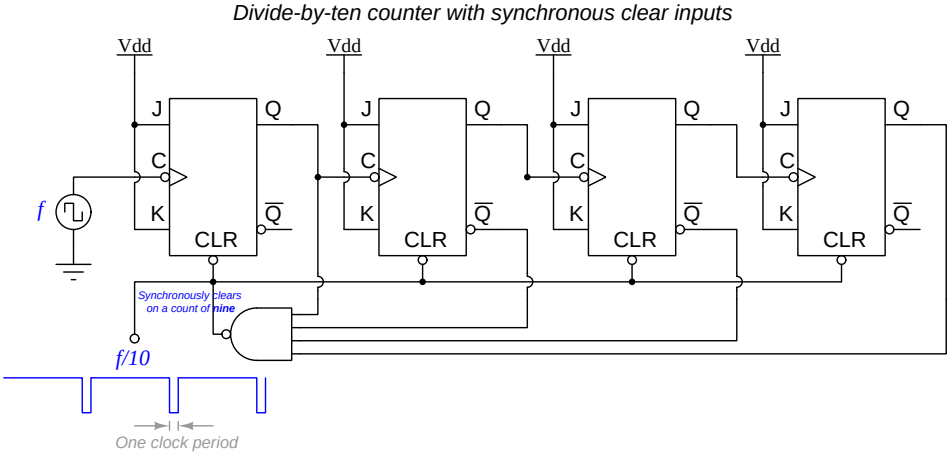
Interestingly, dividing or multiplying frequency by two (or any power of two) results in a musical *octave*. For example, if the clock pulse signal for the above divider oscillated at 440 Hz (used as the "concert pitch" frequency standard by many orchestras for the *A4* pitch in scientific pitch notation), then the $f/2$ output would produce 220 Hz (*A3*) and the $f/4$ output would produce 110 Hz (*A2*), etc. All these tones would correspond to the musical pitch "A" but each within a different octave.

If we desire a frequency division ratio other than some power-of-two, we may use a truncated (i.e. reduced-modulus) counter for the purpose. For example, the BCD counter shown earlier (with JK flip-flops having asynchronous clear inputs) would generate a $f/10$ (divide-by-ten) signal:

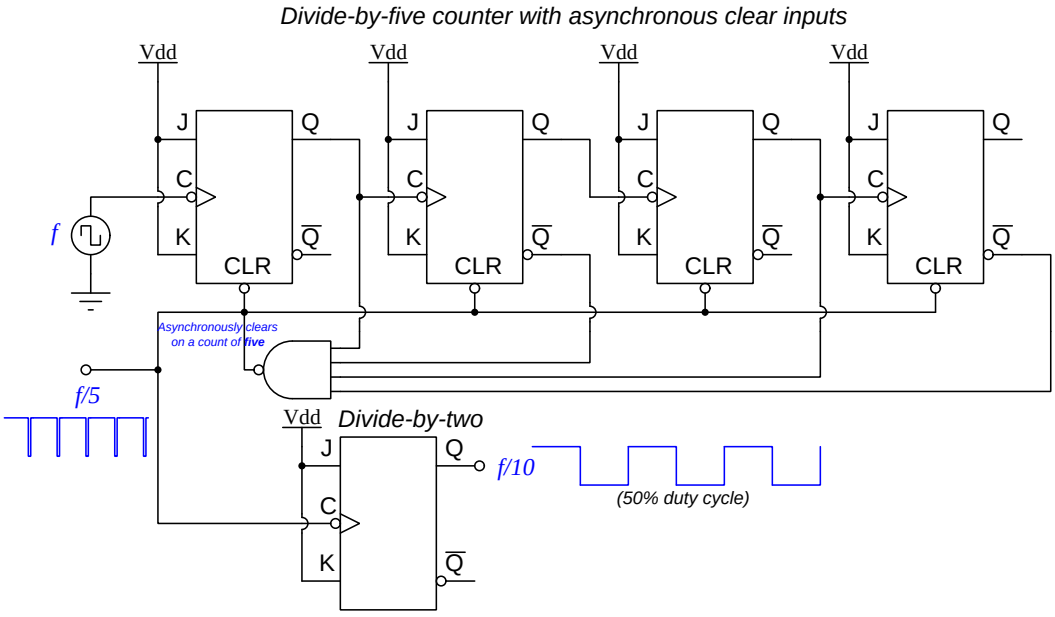


The decade-divided signal is not a symmetrical square wave, because it originates from the NAND gate, the purpose of which is to generate a "resetting" pulse only when the count value reaches ten (1010 binary). This means the pulse width will be very narrow – lasting just long enough for the flip-flops to clear and return the count to zero – but long enough to be useful as a triggering pulse for some other edge-triggered digital circuit.

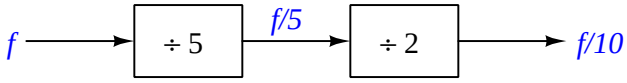
If instead we chose to build a divide-by-ten counter using four JK flip-flops having *synchronous* clear inputs and a NAND gate wired to decode a count value of nine (binary 1001) instead of ten, the NAND gate's output pulse would remain active (low) for one whole clock cycle rather than for the very short time equivalent to the sum of the NAND gate and one flip-flop's propagation delays:



If we need a divide-by-ten pulse with a 50% duty cycle, one solution is to re-wire the NAND gate to make a mod-5 counter, then take the resulting divide-by-five pulse signal and use it to trigger another JK flip-flop to divide the frequency again by two, resulting in a $f/10$ ratio. Here we show a counter made using JK flip-flops with asynchronous clear inputs, the NAND gate wired to decode a count value of five:



Represented in block-diagram form, we can see the $f/10$ division achieved by the compound process of $f/5$ and $f/2$:



Chapter 5

Historical References

This chapter is where you will find references to historical texts and technologies related to the module's topic.

Readers may wonder why historical references might be included in any modern lesson on a subject. Why dwell on old ideas and obsolete technologies? One answer to this question is that the initial discoveries and early applications of scientific principles typically present those principles in forms that are unusually easy to grasp. Anyone who first discovers a new principle must necessarily do so from a perspective of ignorance (i.e. if you truly *discover* something yourself, it means you must have come to that discovery with no prior knowledge of it and no hints from others knowledgeable in it), and in so doing the discoverer lacks any hindsight or advantage that might have otherwise come from a more advanced perspective. Thus, discoverers are forced to think and express themselves in less-advanced terms, and this often makes their explanations more readily accessible to others who, like the discoverer, comes to this idea with no prior knowledge. Furthermore, early discoverers often faced the daunting challenge of explaining their new and complex ideas to a naturally skeptical scientific community, and this pressure incentivized clear and compelling communication. As James Clerk Maxwell eloquently stated in the Preface to his book *A Treatise on Electricity and Magnetism* written in 1873,

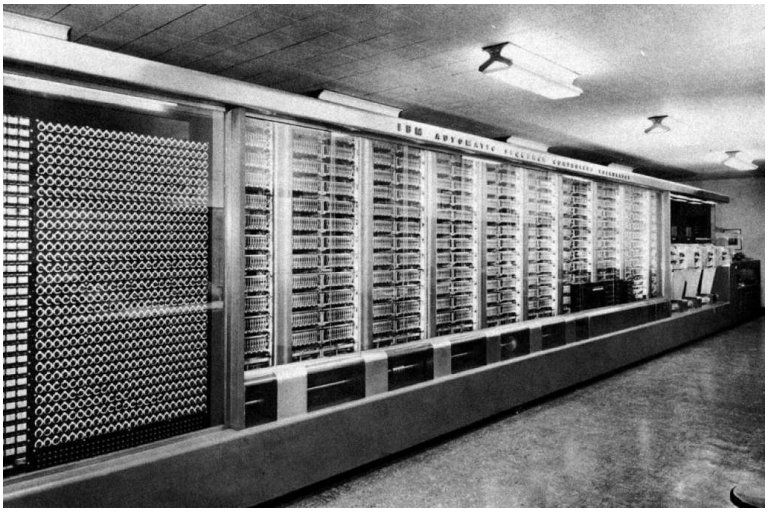
> It is of great advantage to the student of any subject to read the original memoirs on that subject, for science is always most completely assimilated when it is in its nascent state . . . `[page xi]`

Furthermore, grasping the historical context of technological discoveries is important for understanding how science intersects with culture and civilization, which is ever important because new discoveries and new applications of existing discoveries will always continue to impact our lives. One will often find themselves impressed by the ingenuity of previous generations, and by the high degree of refinement to which now-obsolete technologies were once raised. There is much to learn and much inspiration to be drawn from the technological past, and to the inquisitive mind these historical references are treasures waiting to be (re)-discovered.

5.1 Counter circuits in the IBM Automatic Sequence Controlled Calculator

On the 7th of August 1944 the International Business Machine Corporation (IBM) presented an electromechanical computer called the *IBM Automatic Sequence Controlled Calculator* to Harvard University. This room-size calculating machine used thousands of electromechanical relays, switches, paper tape reels, motors, and other devices to perform mathematical calculations. It was used by the United States Navy for classified work over a period of many years.

A photograph showing the front panel of this massive[1] machine appears next, taken from the Historical Introduction chapter of *A Manual of Operation for the Automatic Sequence Controlled Calculator* published by Harvard University Press in 1946:



This early computer was entirely electromechanical, using no semiconductor components. Everything was based on electromechanical devices such as relays and switch contacts. As such, it could be best described as an *electric* computer rather than an *electronic* computer. It was, however, *digital* rather than analog because all numerical quantities were represented by discrete electrical states rather than by continuously-variable voltages. Interestingly, its design was not based on binary ("base-two") numeration, but rather decimal ("base-ten"): counter and register circuits operated on the principle of one out of ten lines (numbered 0 through 9) being energized to represent a single decimal digit.

[1] A quote from page 11 of the *Manual* states that the machine measured 8 feet high, 51 feet long, and weighed approximately 5 tons.

Counters played a large role in the Automatic Sequence Controlled Calculator, and many of these counters relied on an electric drive motor to spin internal wheels. An illustration of the shaft, gear, chain, and sprocket mechanisms used to couple the four horsepower electric motor (B) to the wheels of the sequence and interpolator units is shown here:

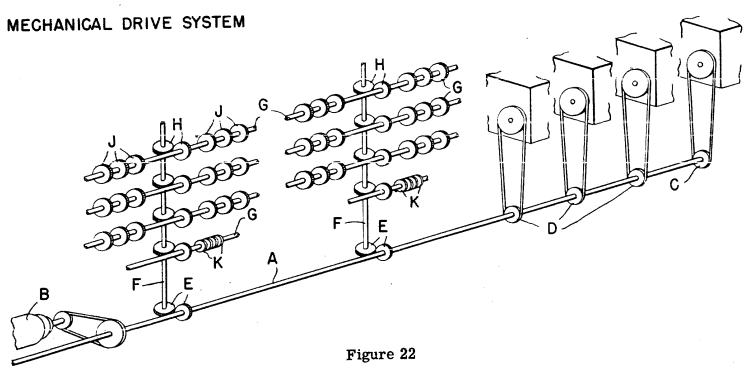
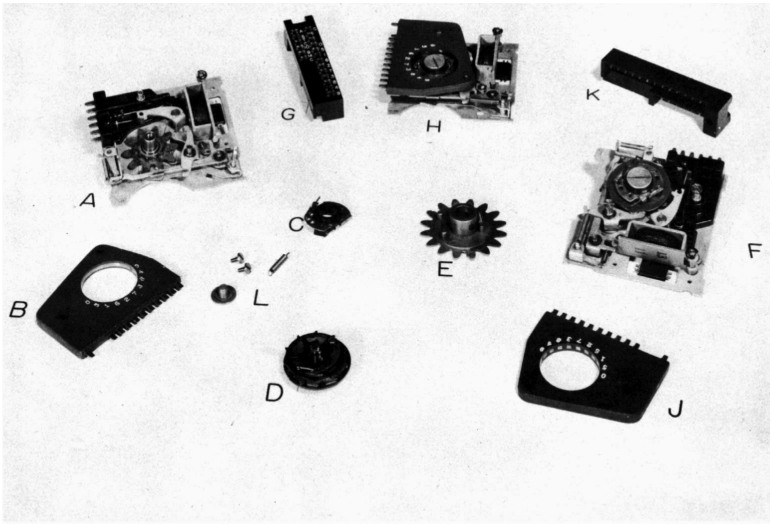


Figure 22

Each of these particular counters consisted of the following components, shown disassembled in the following photograph:



XVII Storage Counter

Two paragraphs found on page 59 of the computer's operating manual describes how these "storage counters" functioned. The first of these paragraphs describes each counter's internal construction:

> Each counter wheel is an electro-mechanical assembly consisting of the following major components shown in plate XVII: (1) a commutator mounted in a molded plastic part, B and J, commonly called a "molding", having a half slip ring and ten segmental contacts numbered 0 through 9; (2) a pair of stranded wire brushes, C and F, which

> rotate to connect one of the contact segments with the commutator half slip ring; (3) a magnetically controlled clutch, D, which engages to connect a continuously rotating gear, A, with the sleeve on which the rotating brushes are mounted; (4) a ten's carry contact which operates in conjunction with an external relay circuit to provide carry to the counter wheel in the next higher columnar position when the counter wheel under consideration passes through ten; (5) a nine's carry contact which also operates in conjunction with an external relay circuit to provide carry to the next higher contact wheel when the wheel under consideration stands on nine and the next lower wheel has passed through ten; (6) and finally, a socket, G and K, by which the counter assembly may be jack-connected to the calculator wiring.

The next of these two paragraphs gives a brief description of how the counters were incremented:

> The ten segments of the commutator are usually called the number "spots". The time interval necessary for the brush to traverse the distance between two successive spots is one-sixteenth of a cycle, the number spots being so spaced in the commutator as to minimize the ratio of the mechanical backlash to the distance traversed between spots. In order to read, say, a seven into a counter, the counter magnet is picked up at "seven time", thus engaging the clutch. The brushes are spun past six spots and the clutch is magnetically disengaged or knocked off at "zero time". Obviously, nine equally timed and spaced impulses must be provided to pick up the counter magnets in order to read in the nine digits and all counters must be knocked off at zero time, (Fig. 23).

A sentence found earlier on that same page of the manual (page 59) makes an interesting observation about the common motor drive for these counters:

> Since the sequence and interpolator mechanisms and counter wheels are all driven by a single gear-connected mechanical system, it is clear that all mechanical parts of the machine revolve in synchronism with each other.

Describing these counters in more modern terms, we could say they are all *synchronous* units with the motor's mechanical drive system being a common *clock* "signal". Each counter is enabled to count by means of a clutch which engages the counting wheel with the continuously-spinning shaft (i.e. clock signal).

These "storage counters" were not the only counting mechanisms in the Automatic Sequence Controlled Calculator, but they do serve to illustrate some of the parallels between 1944-era digital computer technology and modern digital circuitry.

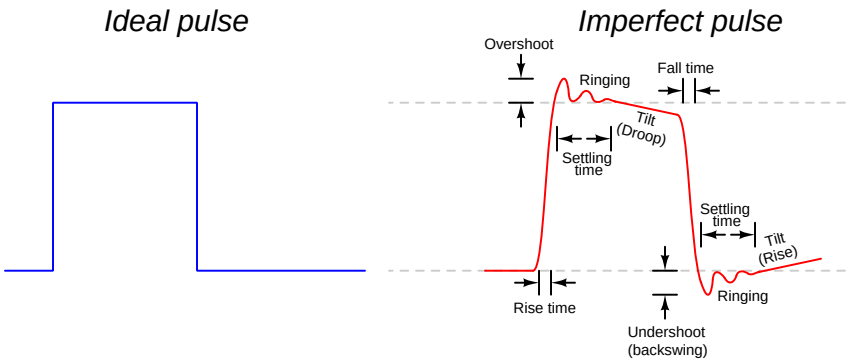
Chapter 6

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.

6.1 Digital pulse criteria

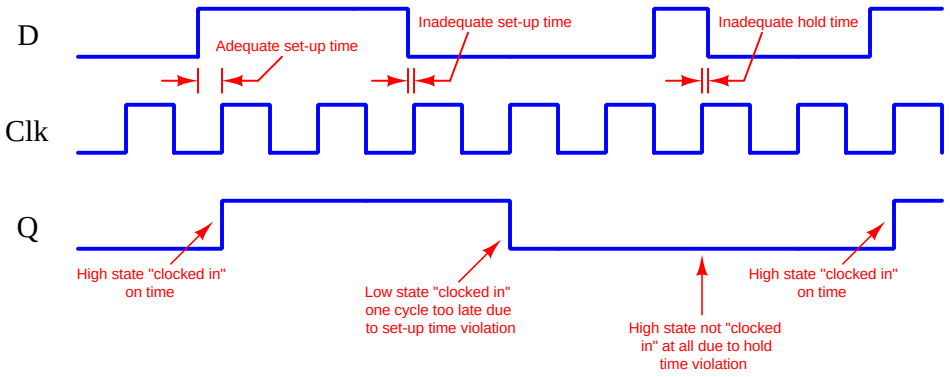
Ideal "pulse" signals have infinitely-steep rise and fall times, level "high" and "low" states well within the specified voltage ranges, and no other imperfections or artifacts. Real pulses always deviate from ideal, often in multiple ways:



Rise and fall times are strongly influenced by parasitic capacitance existing on the signal line with reference to ground, as well as the current-sourcing and current-sinking capability of the logic gate or other device generating the pulse signal. The capacitive "Ohm's Law" formula $I = C\frac{dV}{dt}$ predicts how much current will be necessary to create a linear rate-of-rise or rate-of-fall of voltage for a given capacitance. Note that to achieve infinitely steep rise or fall times an *infinite* amount of current would be necessary to charge/discharge whatever capacitance happens to exist on that signal line!

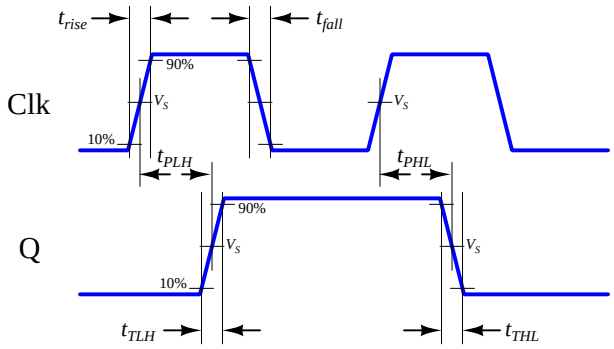
Ringing is caused by *resonance* occurring between parasitic capacitance and parasitic inductance, those two phenomena naturally exchanging energy back and forth with each other to produce the oscillatory waves we call "ringing". Overshoot and undershoot are also the result stored energy within these parasitic L and C circuit properties, and since parasitic capacitance and parasitic inductance can never be fully eliminated from any real circuit it means the effects of over/undershoot and ringing is likewise unavoidable. If you don't see these effects in your pulses, you just aren't viewing them at a fast enough time scale!

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:



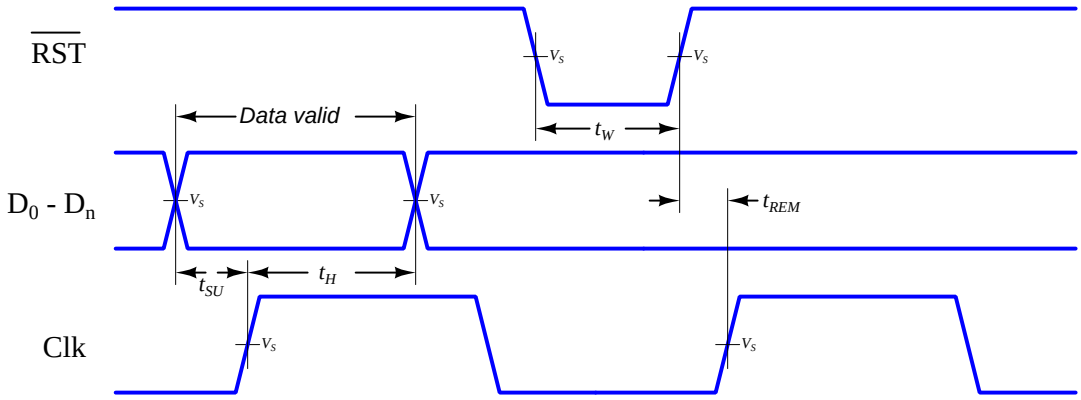
Each of the labels found in this diagram is defined as follows:

- 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 7

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?

7.1 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.

7.1.1 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, journal their own 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 SUMMARIZE THE TEXT in the form of a journal entry documenting your learning as you progress through the course of study. Share this summary in dialogue with your classmates and instructor. Journaling is an excellent self-test of thorough reading because you cannot clearly express what you have not read or did 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.

7.1.2 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.

Thought experiments as a problem-solving strategy

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

Binary number

MSB versus LSB

Frequency division

Ripple

Strobing

Asynchronous versus Synchronous

Hexadecimal

Modulus

Octave

Duty cycle

7.1.3 Timing diagram of a binary count sequence

Count from zero to fifteen, in binary, keeping the bits lined up in vertical columns like this:

```
0000
0001
0010
. . .
```

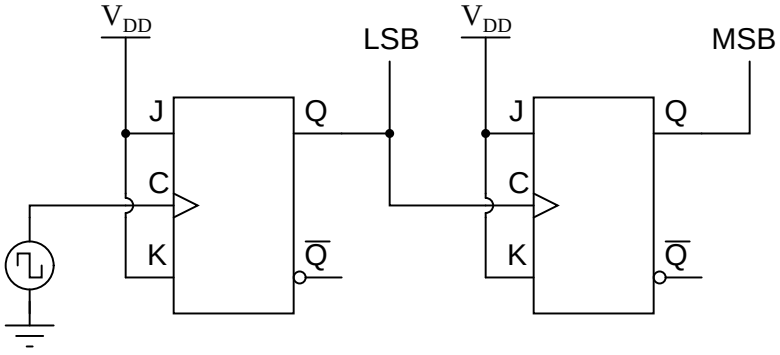
Now, reading from top to bottom, notice the alternating patterns of 0's and 1's in each place (i.e. one's place, two's place, four's place, eight's place) of the four-bit binary numbers. Note how the least significant bit alternates more rapidly than the most significant bit. Draw a timing diagram showing the respective bits as waveforms, alternating between "low" and "high" states, and comment on the *frequency* of each of the bits.

Challenges

- Modify the timing diagram to show a *backwards* counting sequence (i.e. from `1111` to `0000`).

7.1.4 Up-counter or down-counter?

Shown here is a simple two-bit binary counter circuit:



The Q output of the first flip-flop constitutes the least significant bit (LSB), while the second flip-flop's Q output constitutes the most significant bit (MSB).

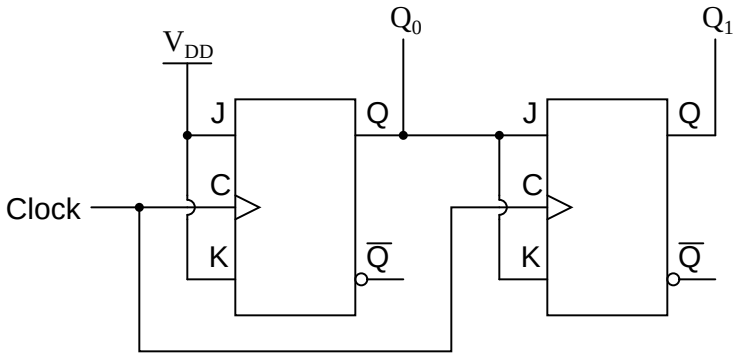
Based on a timing diagram analysis of this circuit, determine whether it counts in an *up* sequence (`00`, `01`, `10`, `11`) or a *down* sequence (`00`, `11`, `10`, `01`). Then, determine what would have to be altered to make it count in the other direction.

Challenges

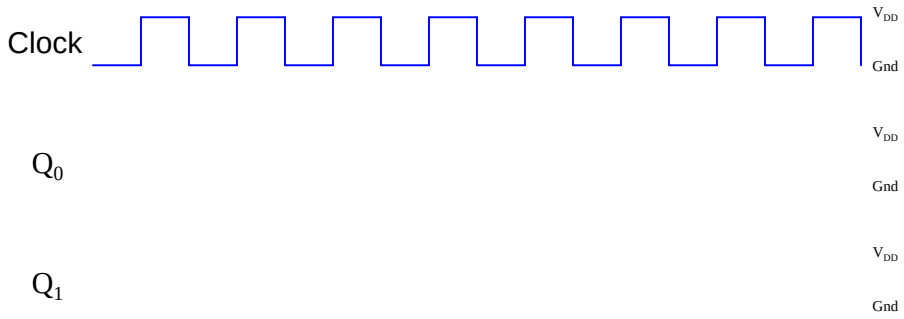
- Is this an example of a synchronous counter or an asynchronous counter?

7.1.5 A counter with no ripple

A style of counter circuit that completely circumvents the "ripple" effect is called the *synchronous* counter:



Complete a timing diagram for this circuit, and explain why this design of counter does not exhibit "ripple" on its output lines:

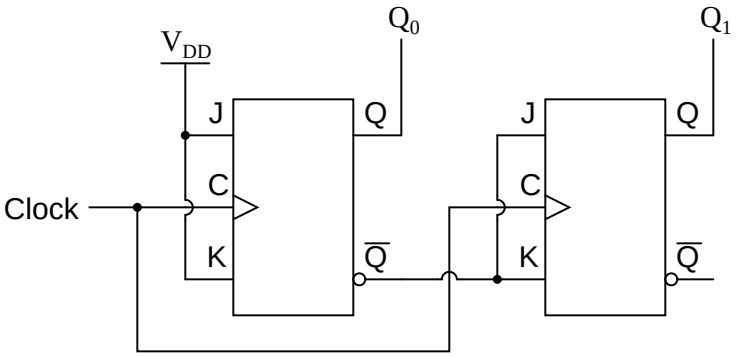


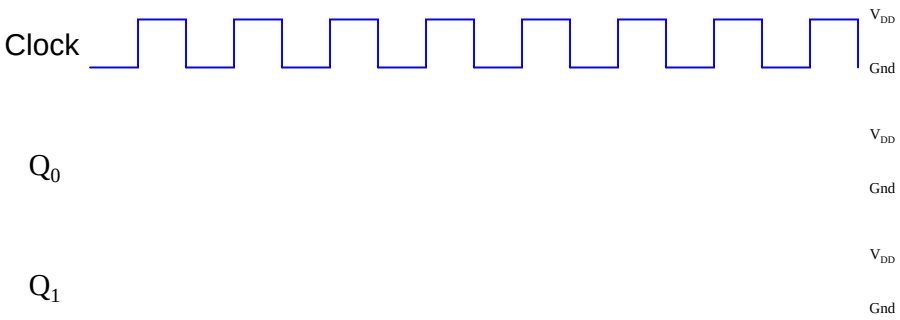
Challenges

- Show the location of and describe the effects of all flip-flop propagation delays in this circuit.

7.1.6 Synchronous counter direction

Complete a timing diagram for this synchronous counter circuit, and identify the direction of its binary count:



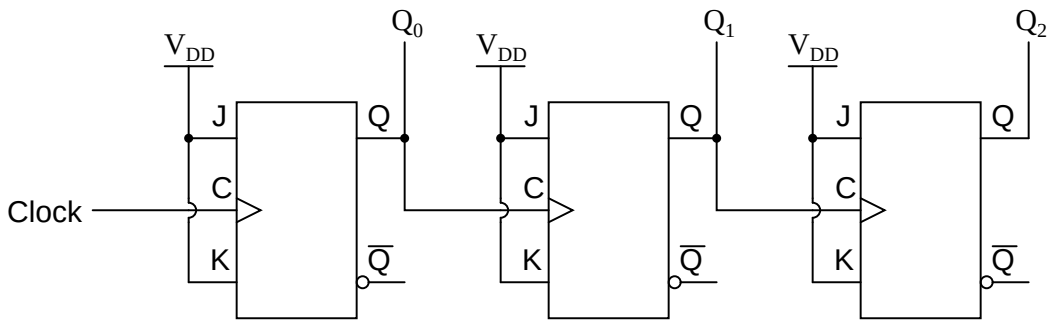


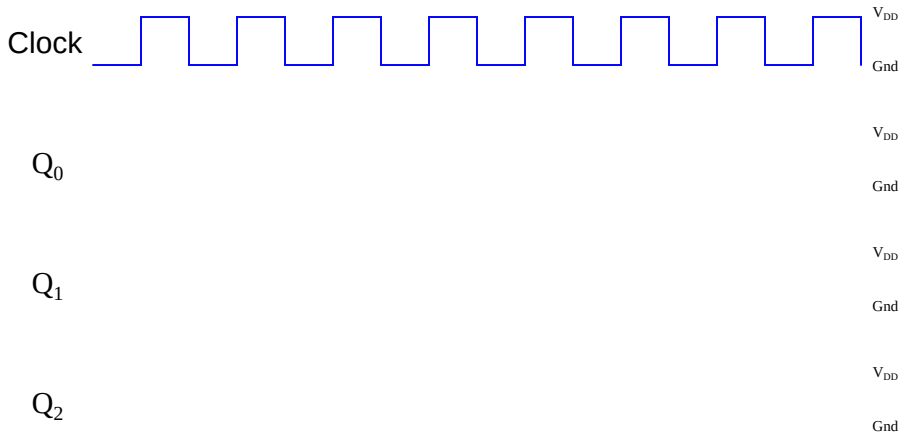
Challenges

- Show the location of and describe the effects of all flip-flop propagation delays in this circuit.

7.1.7 Counter circuit identification

Complete a timing diagram for this circuit, and determine its direction of count, and also whether it is a *synchronous* counter or an *asynchronous* counter:



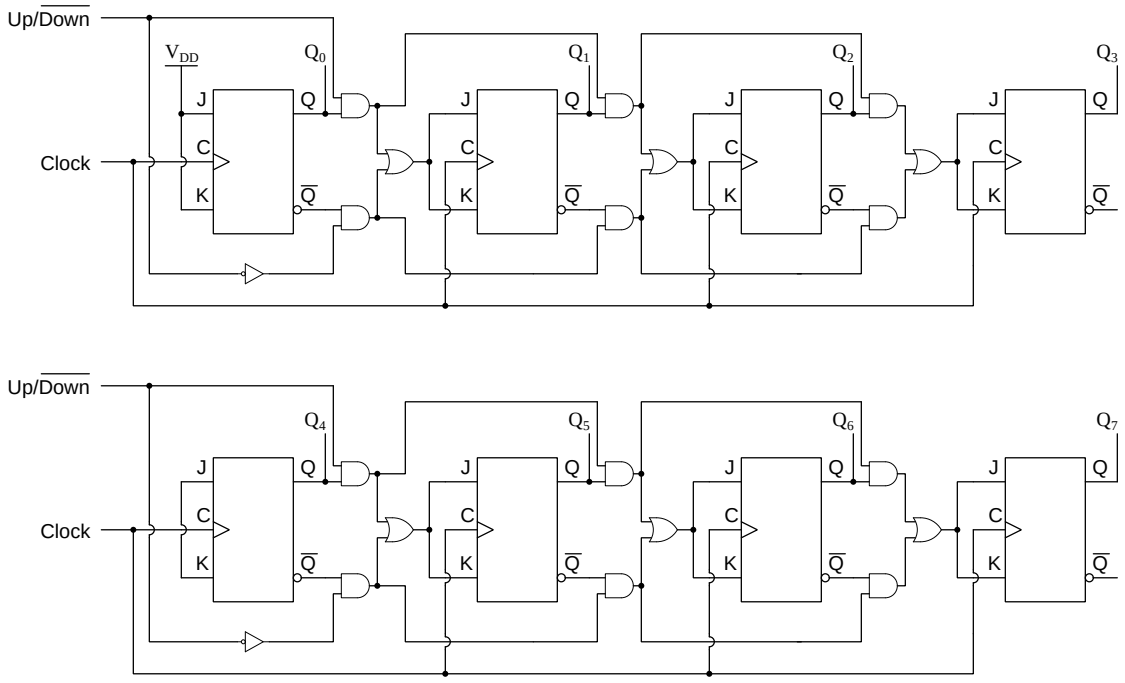


Challenges

- Identify how you could reverse the direction of this circuit's counting sequence.
- Why is it important that all the J and K inputs be tied to V_{DD}?

7.1.8 Cascading counter circuits

Suppose we had two four-bit synchronous up/down counter circuits, which we wished to *cascade* to make one eight-bit counter. Draw the necessary connecting wires (and any extra gates) between the two four-bit counters to make this possible:



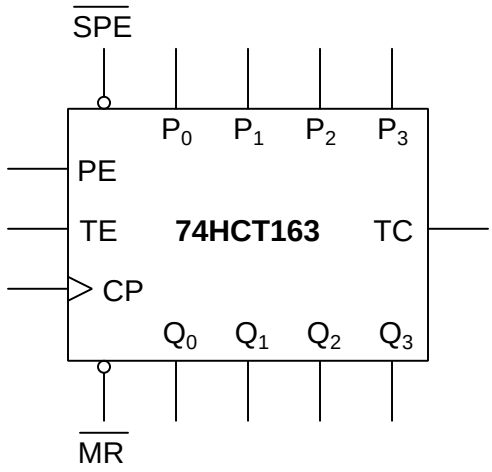
After deciding how to cascade these counters, imagine that you are in charge of building and packaging four-bit counter circuits. The customers who buy your counters might wish to cascade them as you did here, but they won't have the ability to "go inside" the packaging as you did to connect to any of the lines between the various flip-flops. This means you will have to provide any necessary cascading lines as inputs and outputs on your pre-packaged counters. Think carefully about how you would choose to build and package your four-bit "cascadable" counters, and then draw a schematic diagram.

Challenges

- Why go through the trouble of designing cascadable four-bit counters, when you could alternatively manufacture 8-bit, 12-bit, and higher-bit counters as different products to suit customer's needs? In other words, why bother to make your product line *extensible* when you could instead just sell a wider range of products?

7.1.9 74HCT143 counter IC

The model 74HCT163 integrated circuit is a high-speed CMOS, four-bit, synchronous binary counter. It is a pre-packaged unit, will all the necessary flip-flops and selection logic enclosed to make your design work easier than if you had to build a counter circuit from individual flip-flops. Its block diagram looks something like this (power supply terminals omitted, for simplicity):



Research the function of this integrated circuit, from manufacturers' datasheets, and explain the function of each input and output terminal.

- P_0, P_1, P_2, and P_3 =
- Q_0, Q_1, Q_2, and Q_3 =
- CP =
- $\overline{MR}$ =
- $\overline{SPE}$ =
- PE =
- TC =
- TE =

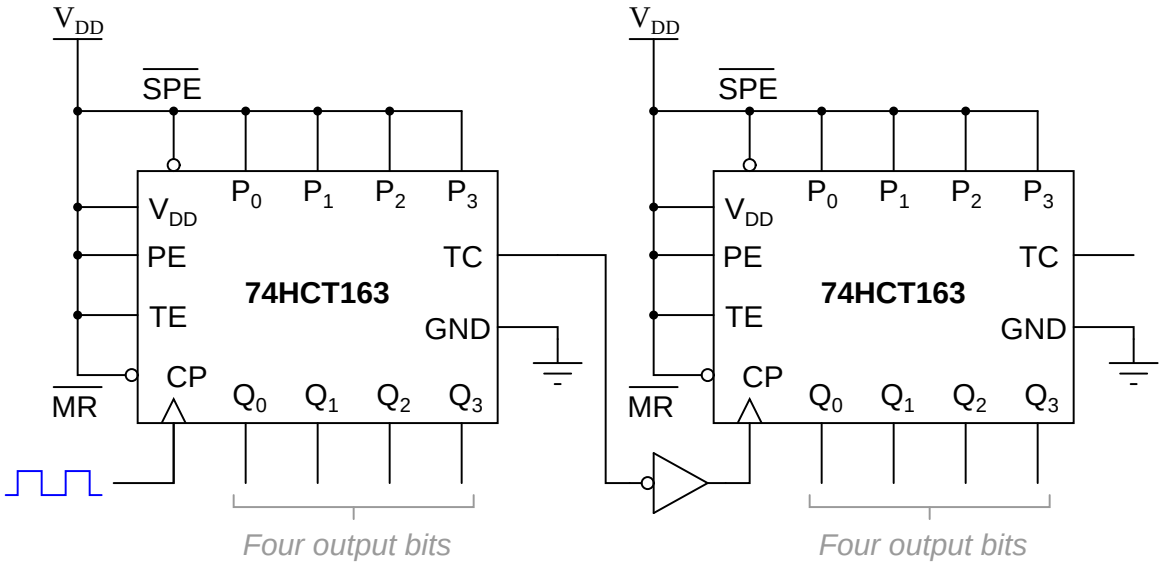
Also, identify whether this is an *up-counter* or a *down-counter*, or capable of counting both directions.

Challenges

- Both the ($\overline{MR}$) and ($\overline{SPE}$) inputs are synchronous for this particular counter circuit. Explain the significance of this fact in regard to how we use this IC.

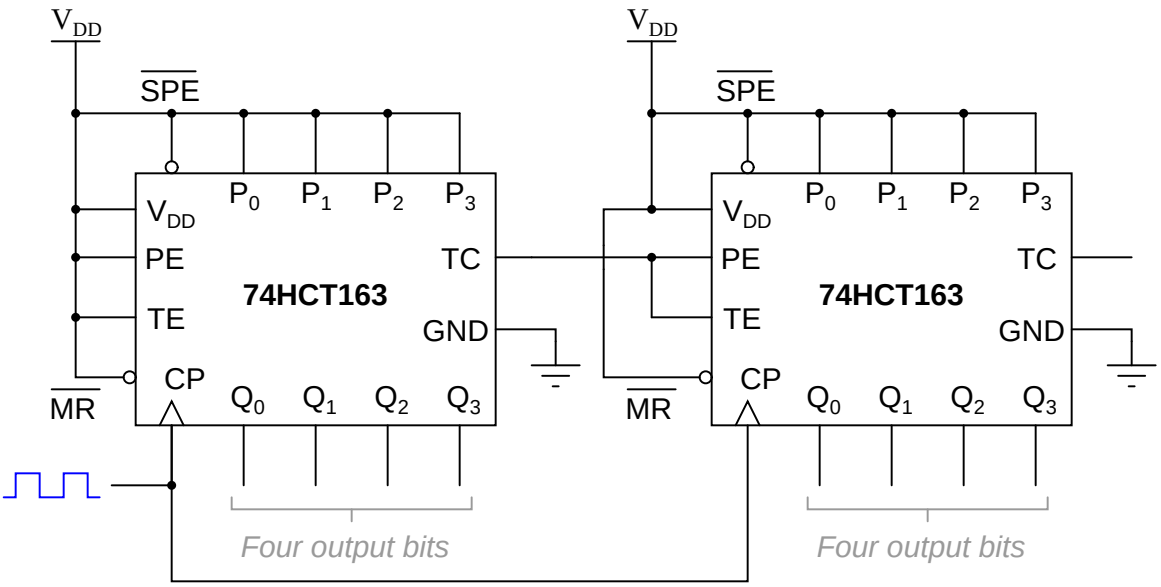
7.1.10 Cascaded 74HCT143 counters

The following eight-bit counter is comprised of two four-bit 74HCT163 synchronous binary counters cascaded together:



Explain how this counter circuit works, and also determine which output bit is the LSB and which is the MSB.

Now, examine this eight-bit counter comprised of the same two ICs:



Explain how this counter circuit works, and how its operation differs from the previous eight-bit counter circuit.

Challenges

- Comment on which method of cascading is preferred for this type of counter IC. Is the functional difference between the two circuits significant enough to warrant concern?

7.2 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.

7.2.1 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.

7.2.2 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	

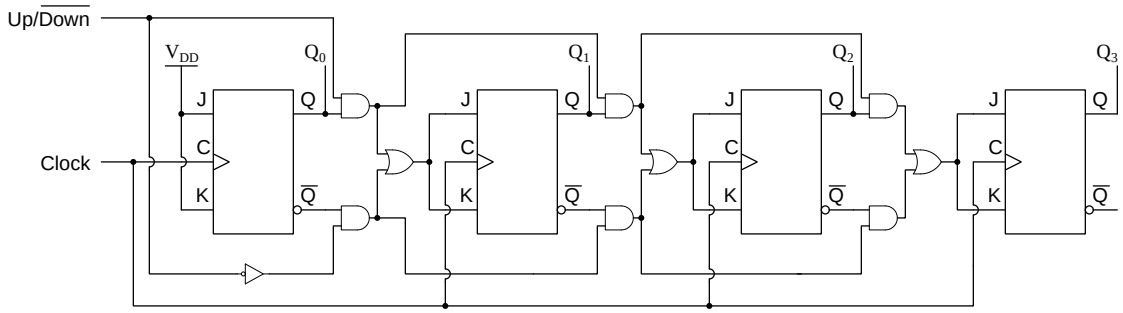
Note how the square-root term (y) is calculated in cell C1, and the denominator term (z) in cell C2. This makes the two final formulae (in cells B1 and B2) simpler to interpret. The positioning of all these cells on the grid is completely arbitrary[10] – all that matters is that they properly reference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

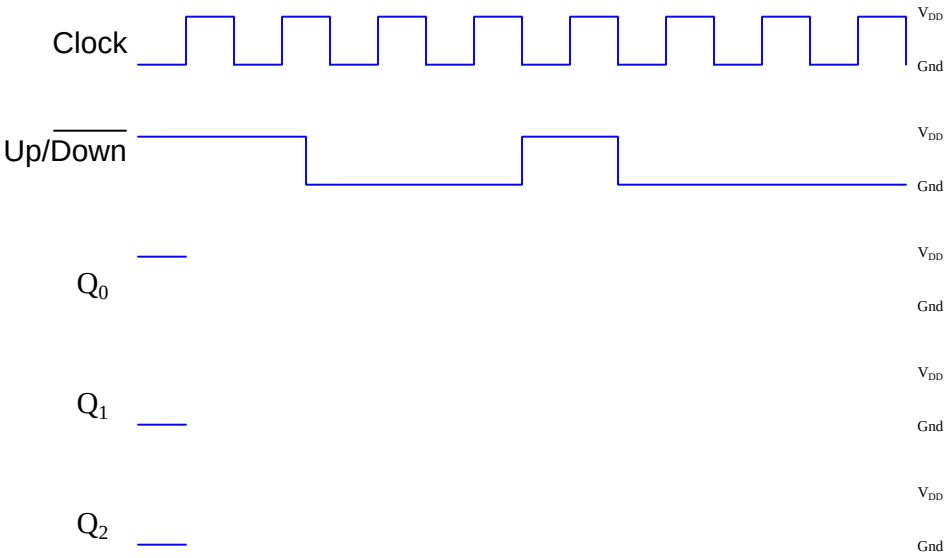
Spreadsheets are particularly useful for situations where the same set of calculations representing a circuit or other system must be repeated for different initial conditions. The power of a spreadsheet is that it automates what would otherwise be a tedious set of calculations. One specific application 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.*

7.2.3 Determining up/down counter state

Determine the count value held by this counter circuit after the final clock pulse shown in the timing diagram:





Challenges

- Suppose the $Up/\overline{Down}$ input remains in its last state shown on the diagram while the clock pulse continues. Identify the next few count values output by the counter.

7.2.4 Frequency division

Suppose we need to design a set of frequency divider circuits using no digital components except four-bit asynchronous counters and miscellaneous logic gates. Describe how each of the following division ratios could be achieved:

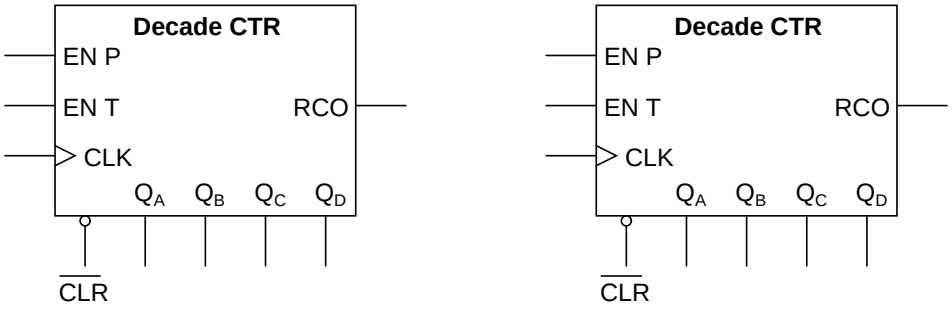
- 2:1
- 13:1
- 5:1
- 8:1
- 24:1
- 19:1

Challenges

- Which of your designs would naturally output a divided pulse signal with a 50% duty cycle, versus a divided pulse signal with a very brief "active" pulse time.

7.2.5 One-minute pulse

Suppose you had an astable multivibrator circuit that output a very precise 1 Hz square-wave signal, but you had an application which requires a pulse once every *minute* rather than once every second. Engineer a solution to this problem using these two counter ICs, added additional components to the circuit as necessary:

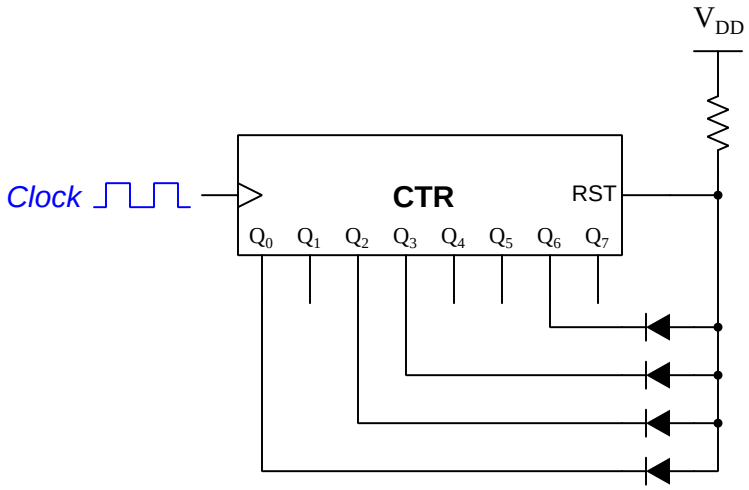


Challenges

- Design a circuit using these same two decade counter ICs so that the output is a square wave with a duty cycle of 50% (i.e. "high" for 30 seconds, then "low" for 30 seconds), rather than a narrow pulse every 60 seconds.

7.2.6 Wired-AND modulus reduction

The following circuit uses four diodes and a resistor to form a *wired-AND* function to limit its modulus:



Explain how this strategy works in contrast to the more conventional method of using AND gates, and also determine the counter's effective modulus with this wired-AND network in place assuming the counter IC's reset line is active-high and synchronous.

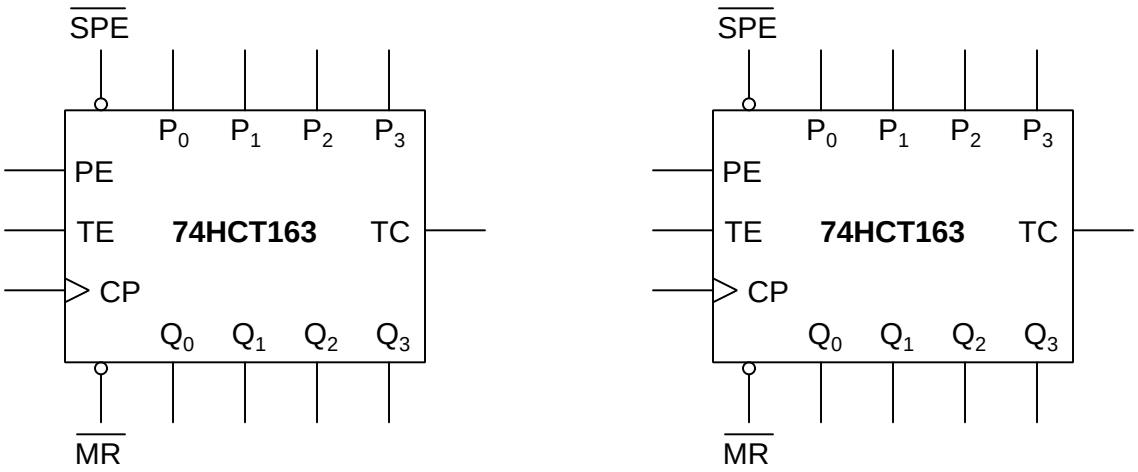
Challenges

- Determine the counter's modulus assuming an active-low reset input.
- Modify this circuit assuming an active-low reset input.

7.2.7 Frequency division using clear versus using preset

When using a counter IC such as the 74HC163 for the purposes of frequency division, there is more than one way to reduce its modulus to achieve the frequency-division ration desired. One way is to wire external circuitry to the counter so that it clears itself back to zero prematurely; another is to wire external circuitry so that it gets *preset* to some non-zero value after reaching its terminal count (binary `1111`).

Sketch all necessary components and wiring to make these counters function as frequency dividers with 6:1 division ratios, using the clear (Master Reset) function in the left-hand example and the preset (SPE) function in the right-hand example. Also identify the terminals for input frequency and output (divided) frequency:



Challenges

- Identify any advantage(s) one strategy has over the other.

7.3 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.

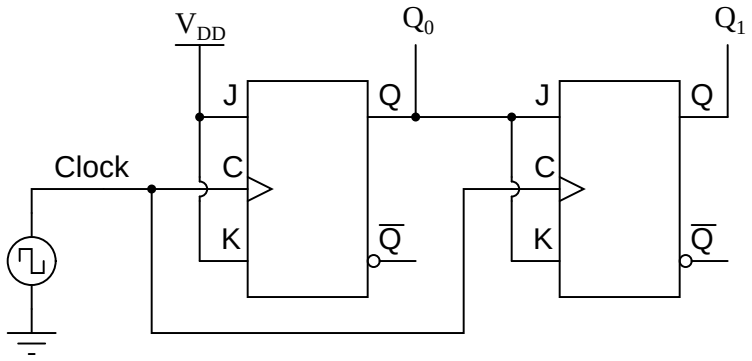
As always, your goal is to fully *explain* your analysis of each problem. Simply obtaining a correct answer is not good enough – you must also demonstrate sound reasoning in order to successfully complete the assignment. Your instructor's responsibility is to probe and challenge your understanding of 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.

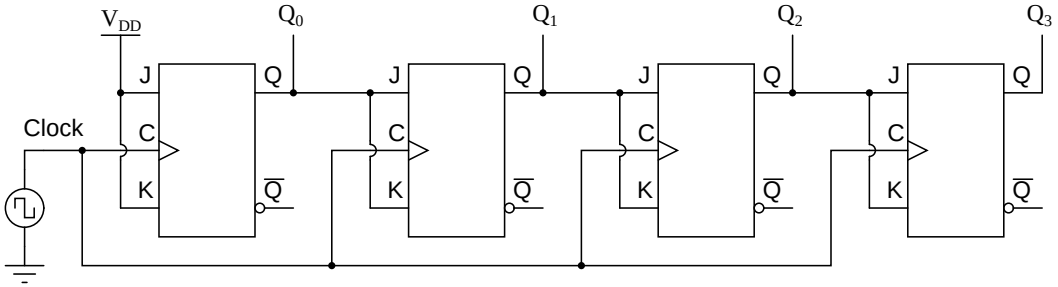
Another means of checking your diagnostic answers, where applicable, is to use circuit simulation software to explore the effects of faults placed in circuits. For example, if one of these diagnostic questions requires that you predict the effect of an open or a short in a circuit, you may check the 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.

7.3.1 Faulty four-bit counter circuit design

A student just learned how a two-bit synchronous binary counter works, and he is excited about building his own. He does so, and the circuit works perfectly:



After that success, student tries to expand on their success by adding more flip-flops, following the same pattern as the two original flip-flops:



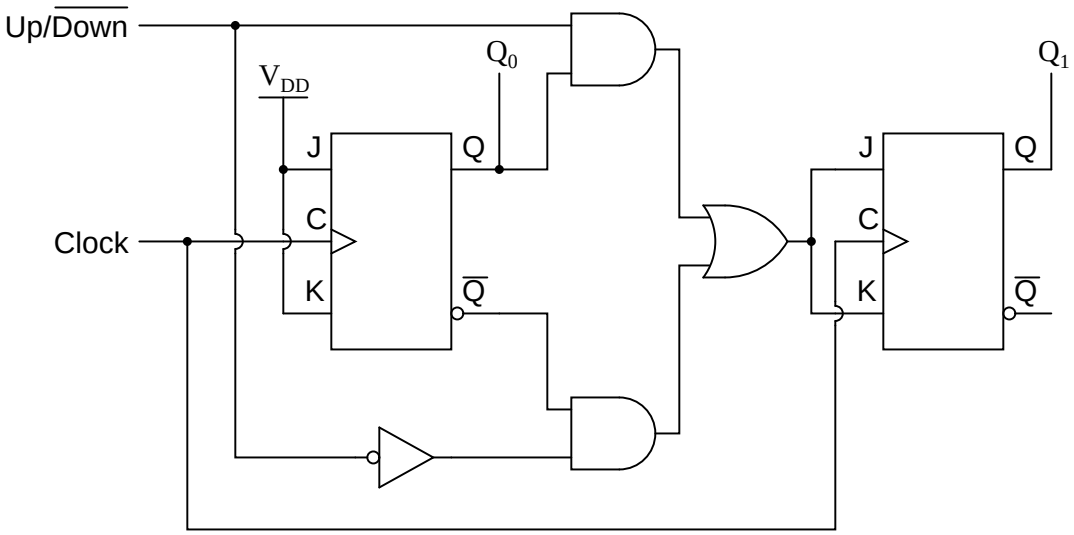
Unfortunately, this circuit does not work as intended – the sequence it generates is *not* a binary count. Determine what the counting sequence of this circuit is, and then try to figure out what modifications would be required to make it count in a proper binary sequence.

Challenges

- What exactly makes this counter circuit *synchronous*?
- How could you construct an *asynchronous* four-bit counter circuit?

7.3.2 Counter with faulted gate

Explain what would happen if the upper AND gate's output were to become "stuck" in the high state regardless of its input conditions. What effect would this kind of failure have on the counter's operation?

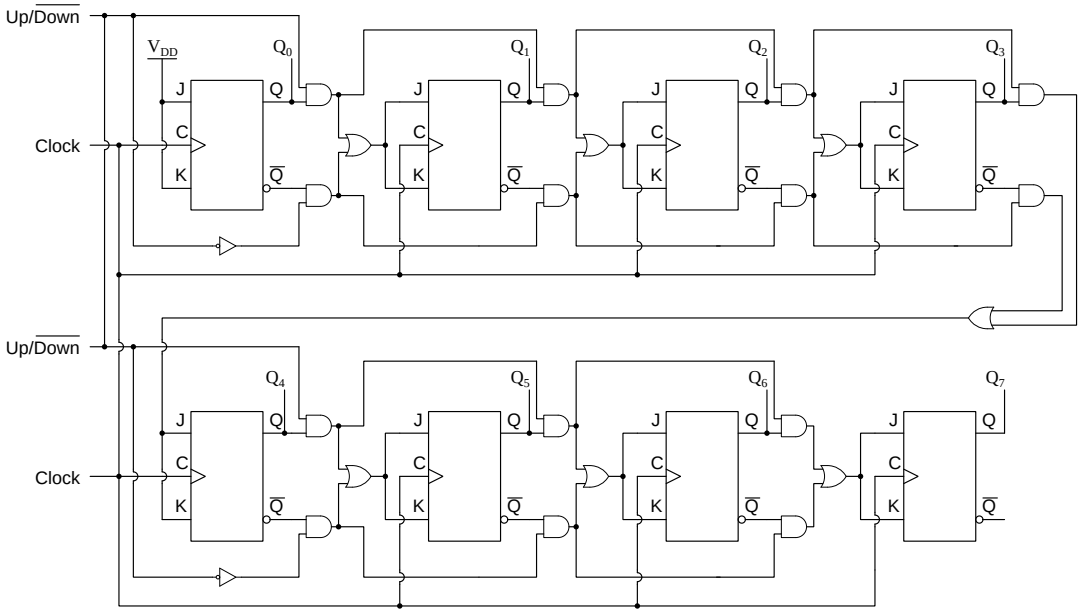


Challenges

- Explain the purpose for the inverter gate in this circuit.

7.3.3 Faulty eight-bit counter design

A student attempts to construct an eight-bit up/down counter, and builds the following circuit:



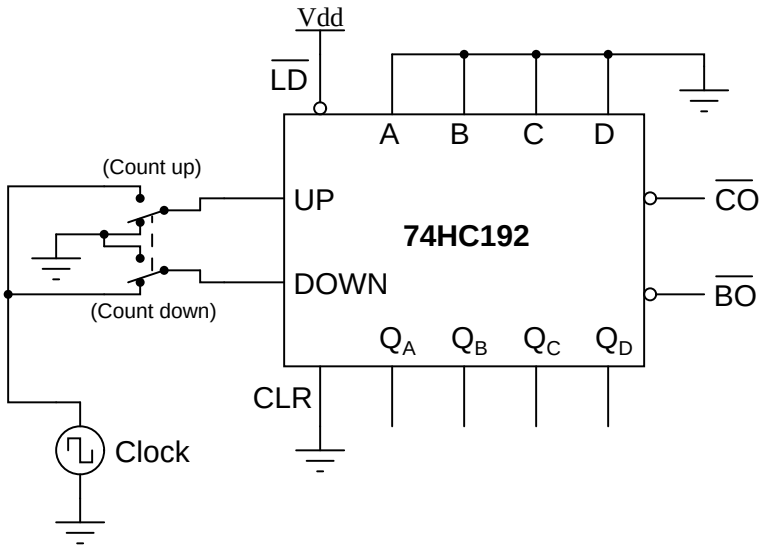
Explain what is wrong with this design.

Challenges

- Identify the incorrect count sequence(s) this circuit would generate as shown.

7.3.4 Incorrect 74HC192 usage

A student is trying to get a 74HC192 up/down counter to function. However, it is simply not cooperating:



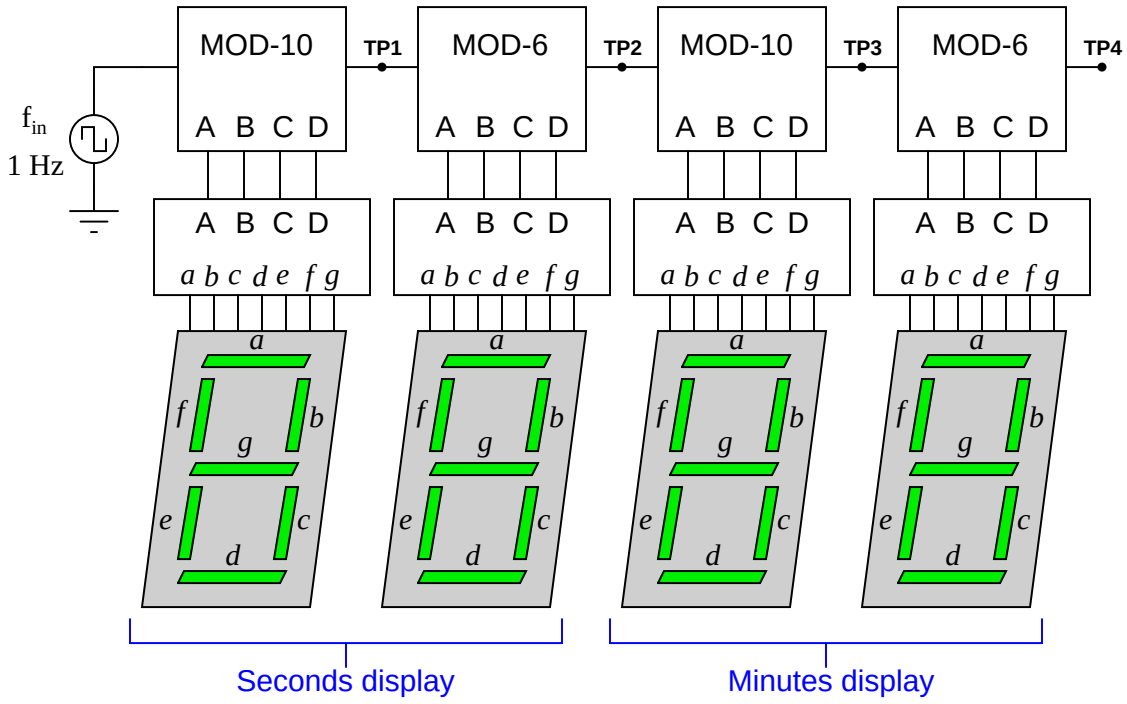
Determine what the student is doing wrong with this 74HC192, and then correct the schematic diagram.

Challenges

- Suppose we wished to pre-load this counter with the value `0b1011`. How exactly would we do this?
- Identify the distinction between the model 74HC192 and the model 74HC193 counter ICs.

7.3.5 Diagnosing a clock circuit

A technician is trying to build a timer project using a set of cascaded counters, each one connected to its own 7-segment decoder and display:



The technician was trying to troubleshoot this circuit, but left without finishing the job. You were sent to finish the work, having only been told that the timer circuit "has some sort of problem". Your first step is to power up the circuit and watch the timing sequence, and after a few minutes of time you fail to notice anything out of the ordinary.

Now, you could sit there for a whole hour and watch the count sequence, but that might take a long time before anything unusual appears for you to see. Devise a test procedure that will allow you to pinpoint problems at a much faster rate.

Challenges

- Suppose your test failed to identify a problem. Identify possible faults that could appear and disappear at random, which could explain the disparity between your assessment of the circuit and the other technician's.

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

A Manual of Operation for the Automatic Sequence Controlled Calculator, Harvard University Press, Cambridge, Massachusetts, 1946.

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.

9 January 2025 – added a new Case Tutorial chapter with a section showing an analog multiplexer used to create an arbitrary waveform generator circuit.

9 November 2024 – divided the Introduction chapter into sections, one with recommendations for students, one with a listing of challenging concepts, and one with recommendations for instructors.

29 April 2024 – added a Case Tutorial section with examples of latching logic circuits and timing diagrams.

20 July 2023 – added a new Case Tutorial section on the phenomenon of switch bounce as well as mitigation techniques.

9-10 May 2023 – minor edits on the definition of "modulus" (number of unique and stable output states of a counter). Also added Case Tutorial sections showing counter circuits with reduced modulus.

27 April 2023 – minor edits to the Full Tutorial chapter.

14 January 2023 – added a Quantitative Reasoning question regarding reducing the modulus of a 74HC163 synchronous counter by using its preset capability rather than its clear capability in order to achieve frequency division.

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.

25 May 2022 – minor additions to the Simplified Tutorial chapter.

4-5 May 2022 – added content to the Full Tutorial and to the Introduction regarding synchronous versus asynchronous clear inputs on JK flip-flops for reduced-count moduli.

17 January 2022 – added a Quantitative Reasoning question regarding wired-AND logic for truncating a counter's modulus.

27-28 July 2021 – changed title of the existing Tutorial to "Full Tutorial" and created both "Simplified Tutorial" and "Case Tutorial" chapters. Also corrected some typographical errors.

10 May 2021 – commented out or deleted empty chapters.

20 October 2020 – corrected error in modulo-5 counter diagram where I had the counter resetting on a count value of six rather than a count value of five.

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.

15 March 2020 – added some instructor notes, and corrected some errors in the instructor notes for the Quantitative Reasoning problem "Frequency division" and Diagnostic Reasoning problem "Counter with faulted gate".

7 June 2020 – added more questions.

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

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

18 November 2019 – minor edits.

17 November 2019 – added more questions.

16 November 2019 – finished introduction and added some questions.

15 November 2019 – finished first draft of tutorial chapter.

14 November 2019 – document first created.

Index