

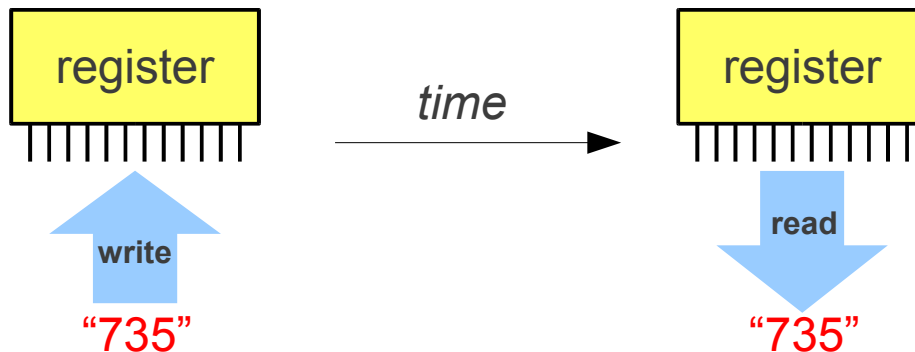
Designing Computer Systems

Counters

Designing Computer Systems

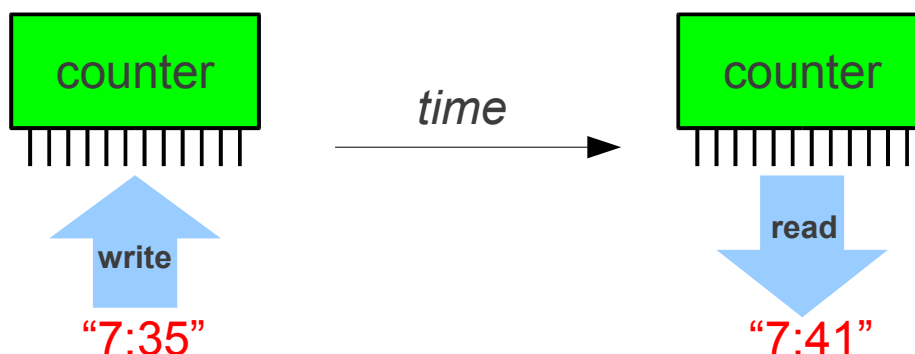
Counters

A register has a critical role to play in digital computing systems. It stores data so it can be used in the future. Its singular purpose is to accurately preserve a value for an indeterminate period.



A value **735** is stored (written) in a register. Later, the register is read and the value **735** is obtained. The register's value is changed only when a new value is written or if the power source is removed.

Another form of storage is widely used in digital systems. It is constructed using similar components, but it has a different objective. Ironically, if it behaves like a register, *it is deemed faulty*.

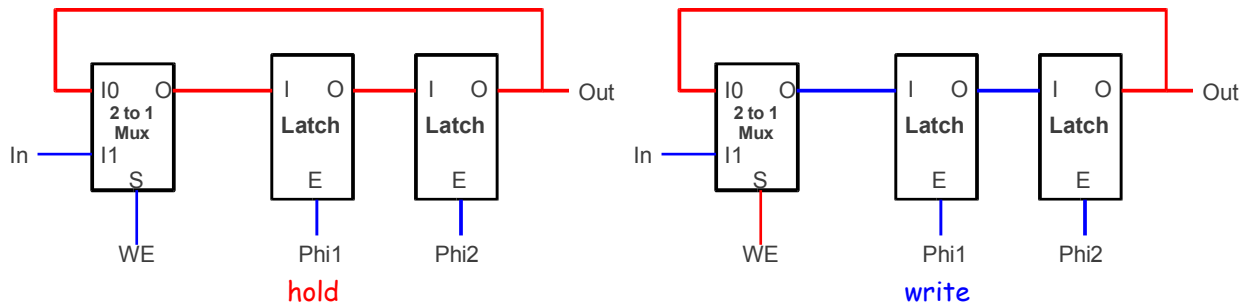


This storage is, of course, a *counter* and it is used for many purposes including time keeping. Here the clock is more than just a periodic sequencer of storage access. It also provides the time base that defines the counting interval. A counter is a multiple-bit storage element that follows a binary sequence in sync with a clock. It can also be described as counting the intervals of time defined by the clock's period. With a one Hertz clock, a counter *counts seconds*.

Stopwatch 101: Despite clocks being a prevalent example of counters in our world, its fraught with the peculiarities of time conventions. Time begins at 12:00. Then at 12:59, it *advances* to 1:00. Ante Meridian (AM) ends at 11:59 followed by 12:00 Post Meridian (PM). Who made this system up?

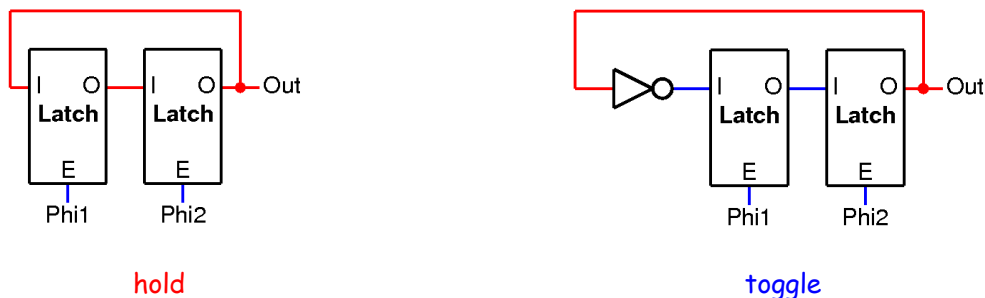
Stopwatches employ similar counter principles, but with more clarity. It begins at 0:00 when the clear button is pressed. When the start/stop button is pressed, the counter starts (or stops) measuring the passage of time. The lap button displays an interval while the counter continues.

Remember the Register: Before exploring the construction of a counter, let's review the register. Because the counter must retain its current value while it is updated with its new value, it is based on a two latch cell that is similar to the register.

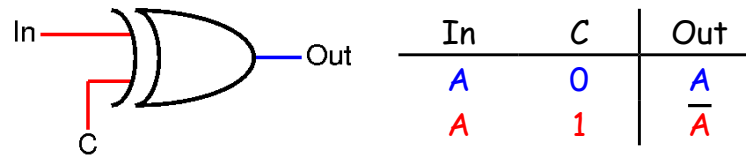


The 2-to-1 mux selects one of two values for the first latch's input: **copy the current value** from the second latch (right), or **a new value from In** (left). Because a register's roll is to preserve a value until it is overwritten, this is a good selection.

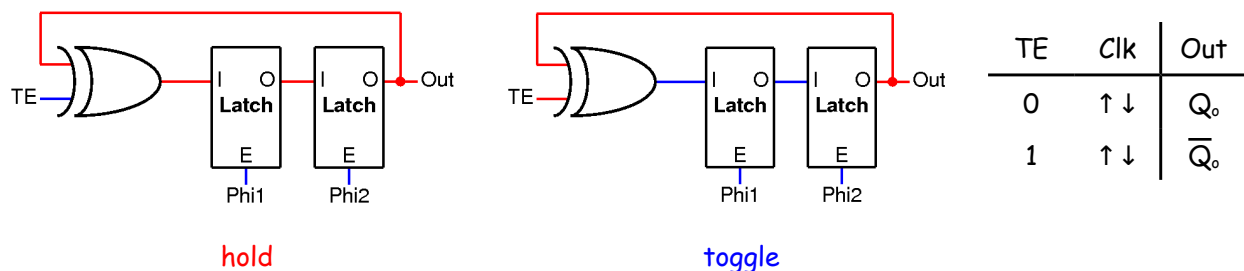
Counters need to count. In a binary world, a one-bit counter is rather dull: 0, 1, 0, 1, 0, 1, ... this is well-described as toggling. But sometimes a one-bit counter needs to hold a value (1 or 0) for multiple cycles 0, 0, 0, ... or 1, 1, 1, ... The hold mode already exists in the register design. In this mode, on the left, the output value loops back to the input to **hold** the state. The toggling mode, on the right, is almost the same, except the complement of the output loops back to the input to **toggle** the state.



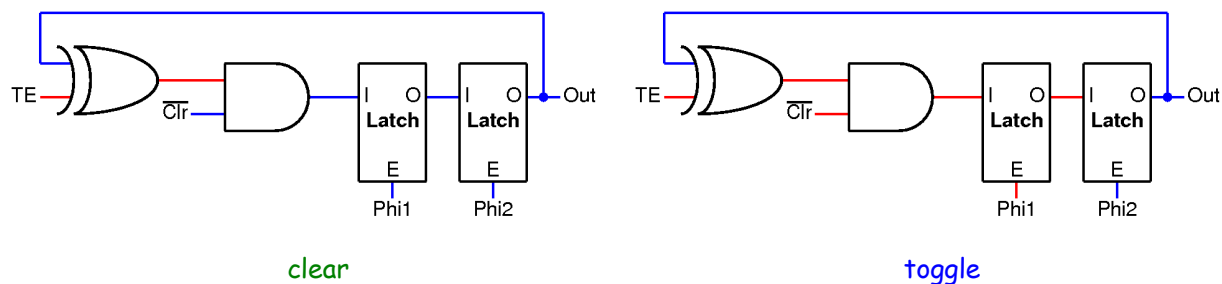
Since a cell will sometimes hold its value and sometime toggle its value, it must be able to do both, controlled by an input *toggle enable* (TE). This new input will control a selective inverter, otherwise known as a XOR gate (odd parity).



When the control input is low, A is passed through **unchanged**. When the control input is high, A is **inverted** \overline{A} . This gate can be used to create a cell that can toggle its output on each cycle of the clock (right). Or it can hold a value unchanged through a clock cycle (left).

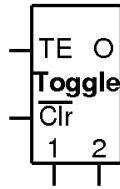


What State Am I: This cell has an essential feature, to selectively toggle. But it lacks definition. How can the state be set to a known value? The stopwatch must be able to be **cleared**, forcing all bits of the counter to zero. The current value must be **masked**. So an AND gate is used.



The \overline{Clr} signal is active low, indicated by the bar over the signal name. When the clear signal is asserted (low), the current value, toggled or held, is **masked to zero**. When the \overline{Clr} signal is deasserted (high), the AND gate passes the toggled or held state from the second latch. In this example, it is **toggled**.

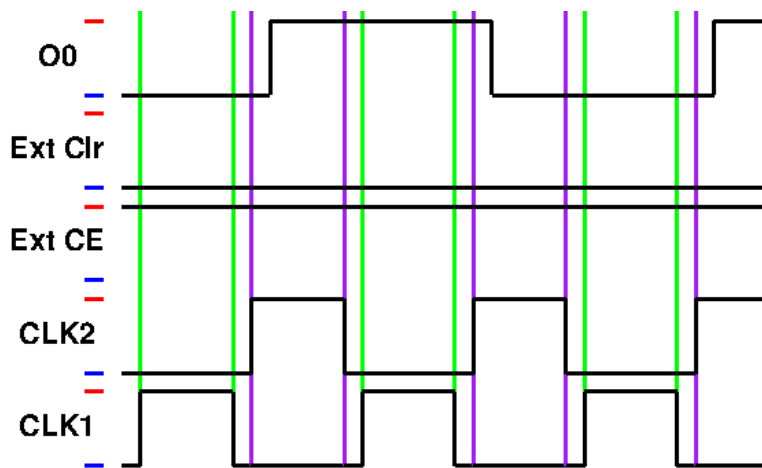
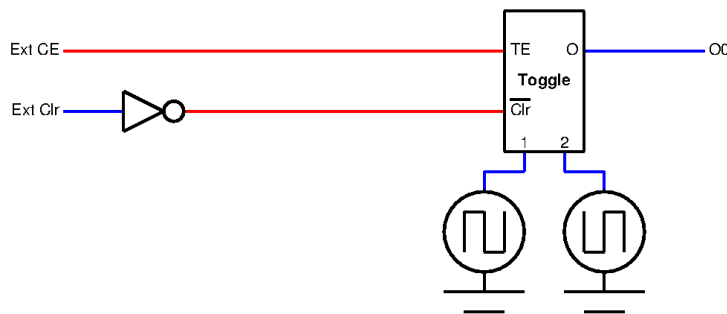
This design represents a one bit *toggle cell*. It can operate in three modes, **clear**, **hold**, and **toggle**. It is the foundation of multiple-bit counters. A simple icon is used to capture this implementation.



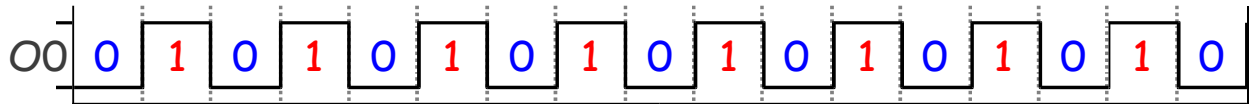
TE	$\overline{\text{Clr}}$	Clk	Out	
X	0	$\uparrow\downarrow$	0	clear
0	1	$\uparrow\downarrow$	Q_o	hold
1	1	$\uparrow\downarrow$	$\overline{Q_o}$	toggle

A stopwatch has two external controls. Count Enable (CE) allows the counter to advance with time. When it is not asserted, the counter is frozen at the current value. Clear (Clr) resets the counter value to zero. In both cases, the assertion of these signals is not immediate. The effect of these signals is seen at the counter's outputs at the end of each cycle. This is called *synchronous* operation since behavior is synchronized with the system clock.

N-bit Counter: A multi-bit counter begins with a single toggle cell connected to the external signals. The external count enable controls the toggle cell's toggle enable. The external clear controls the counter's clear; but it must be inverted to be active low. A **one-bit counter** is nothing more than a toggle cell.

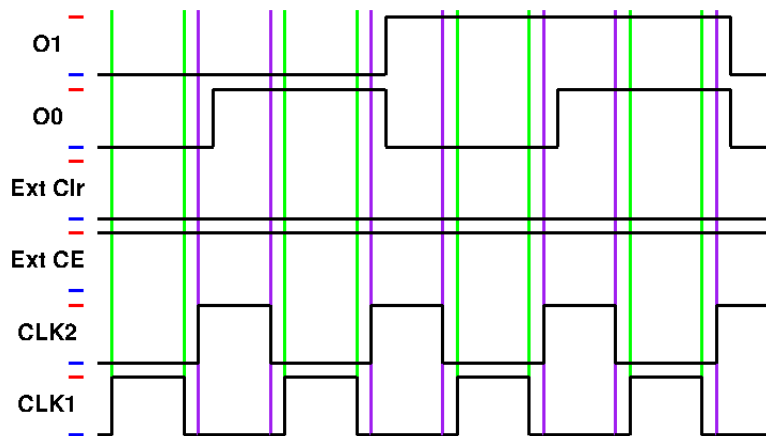
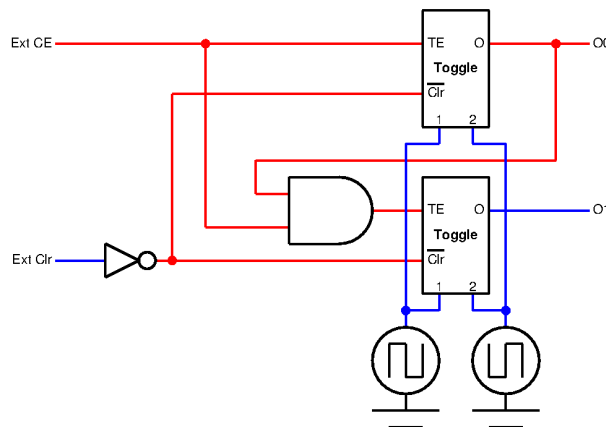


But when counting is enabled and the device is not being cleared, it counts with the system clock, its output toggling on each cycle. The timing diagram shows the one bit counter's output. The vertical dashed lines represent the system clock.

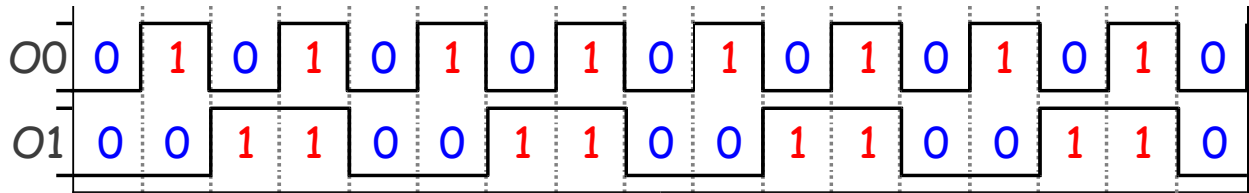


Because it *toggles* once per clock cycle, its cycle period is twice the system clock period, and its frequency is half the system clock. For this reason, a one bit counter is sometimes called a *divide by two* counter since it outputs a clock that is half the system clock frequency.

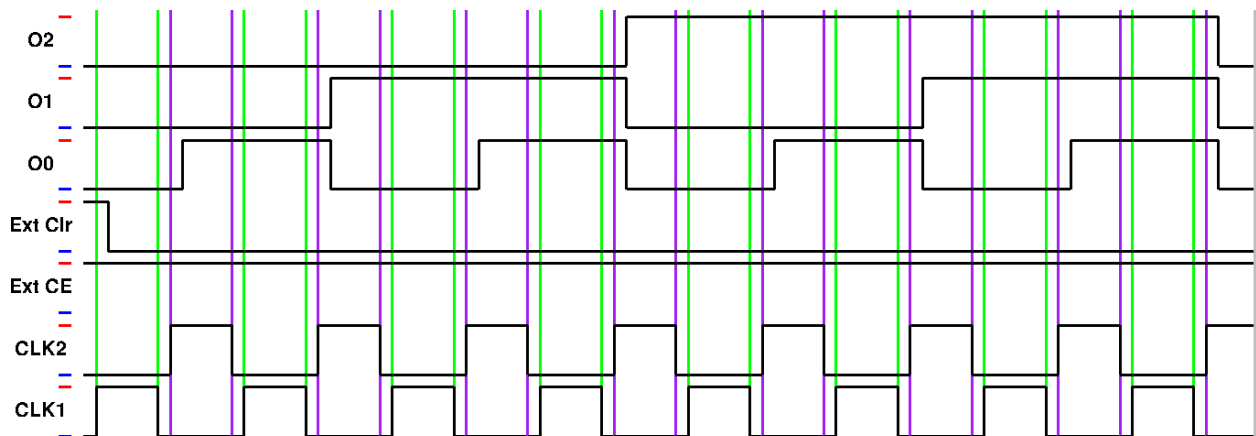
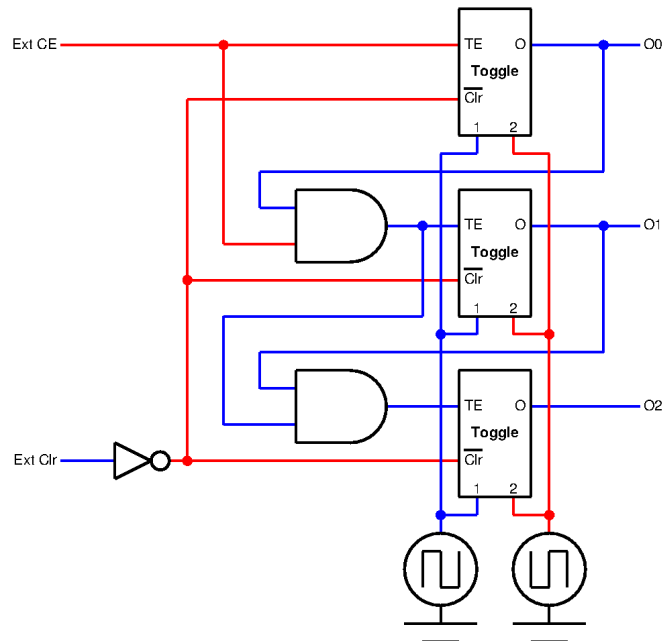
A **two bits counter** begins with two toggle cells. The first cell is connected directly to the external count enable and clear (as in the one bit counter). But the second toggle cell must be connected differently or it will toggle (count) identically to the first cell. For the second cell, it should toggle when the external count enable is asserted and the output of the first toggle cell is high.



Since this is only true on half the cycles, the second cell will toggle *every other cycle*. Since the first output is already half the system clock, the second output is a divide by 4 (one fourth the system clock frequency).

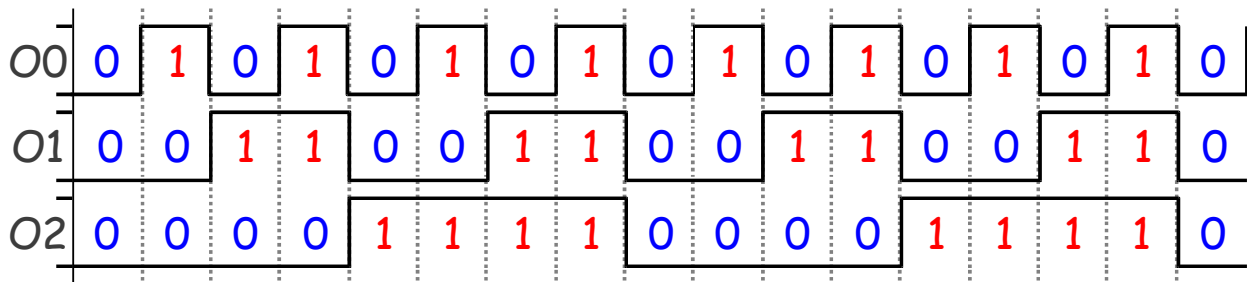


A **three bit counter** adds a third toggle cell.

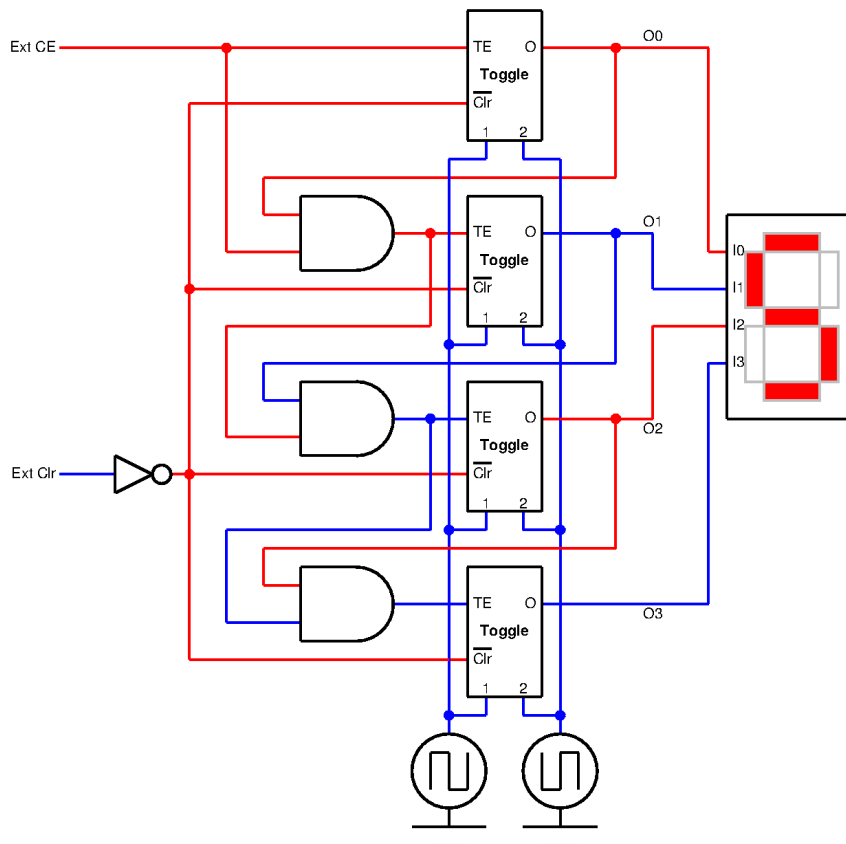


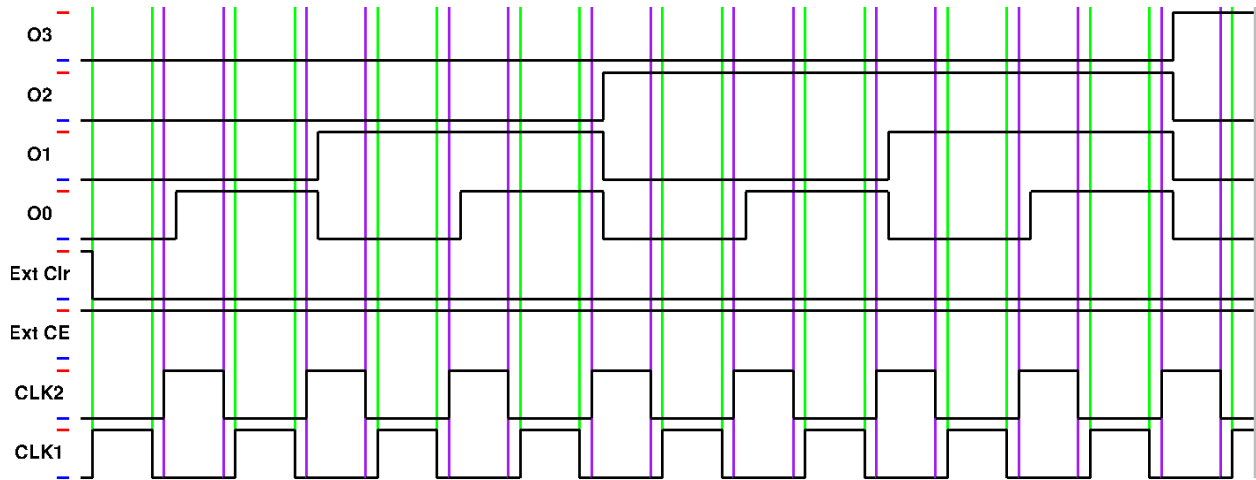
Whereas the first cell toggles every cycle, and the second toggle cell toggles every other cycle, the third toggle cell toggles *every fourth cycle*, when all lesser

significant bits are high. Its toggle enable is only high when O0 and O1 are high, along with the external count enable. Since two of these signals are already combined with an AND gate, its output can be ANDed with O1. The third output (O2) is a divide by 8 (one eighth the system clock frequency).

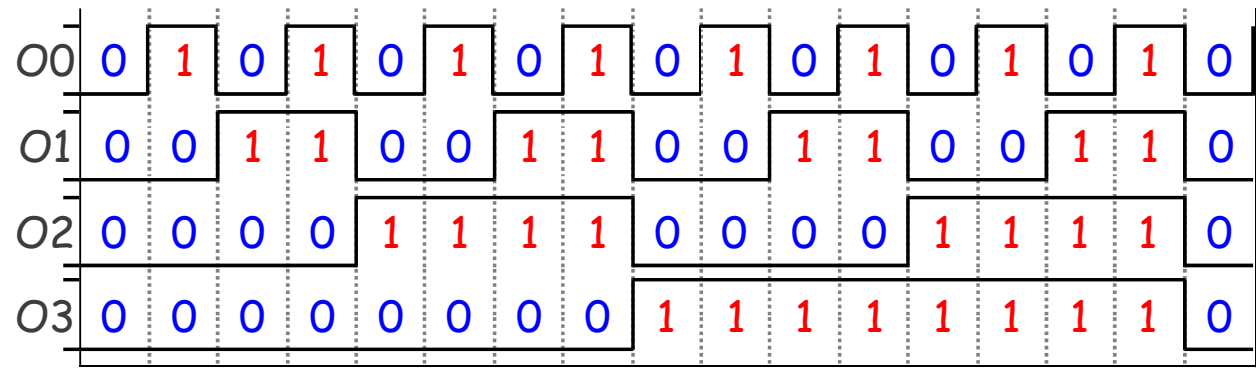


A **four bit counter** follows the same extension process. A fourth toggle cell is added and connected to toggle only when all lesser significant outputs are high, along with the external count enable. The new output, O3, toggles every sixteenth system clock cycle, as a divide by 16 counter. Here a decoder seven segment display has been added to show the numeric counting process.





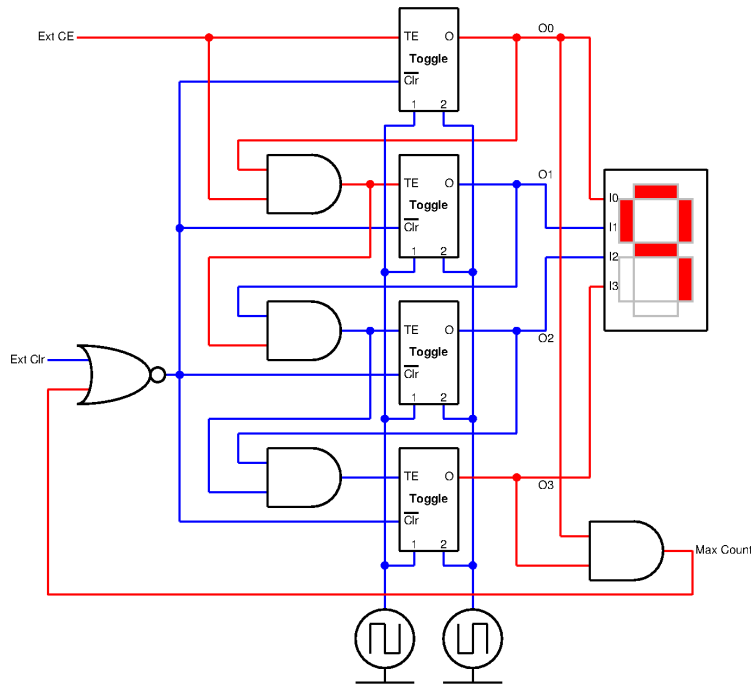
This process can be extended to an i -bit counter that divides the system clock by 2^i .



Divide by N: These counter divide by a power of two (two raised to the number of toggle cells). How would a counter count by an arbitrary value N ? Since we live in decimal world (with ten fingers), a decade counter (divide by ten) would be a handy device. A four bit counter divides by 16, meaning it counts from '0' to '15' ('0' to 'F' in hexadecimal). A decade counter counts from '0' to '9' and then returns to '0'. Its maximum count is nine; when it reaches that count, it should return to zero on the next clock cycle. This can be accomplished using the max count detector and the external clear signal.

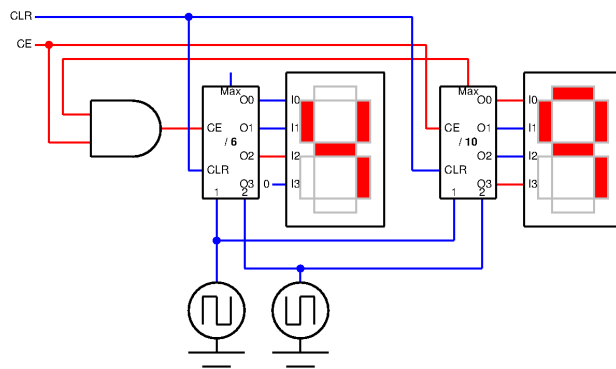
The max count detector monitors the counter outputs, detecting the maximum count. Normally, this would involve testing both high and low values in the maximum count. But since this is an up counter (it starts at zero and counts to all ones), a more compact max count detector only compare *high values* in the max count. Differentiating low bits in the max count is pointless since it will never reach a high value; it will be cleared when it is low. A decade counter (divide by 10) has a

maximum value of nine. So a max count detector will monitor O0 and O3. When they are both high ('9'), the counter is cleared.

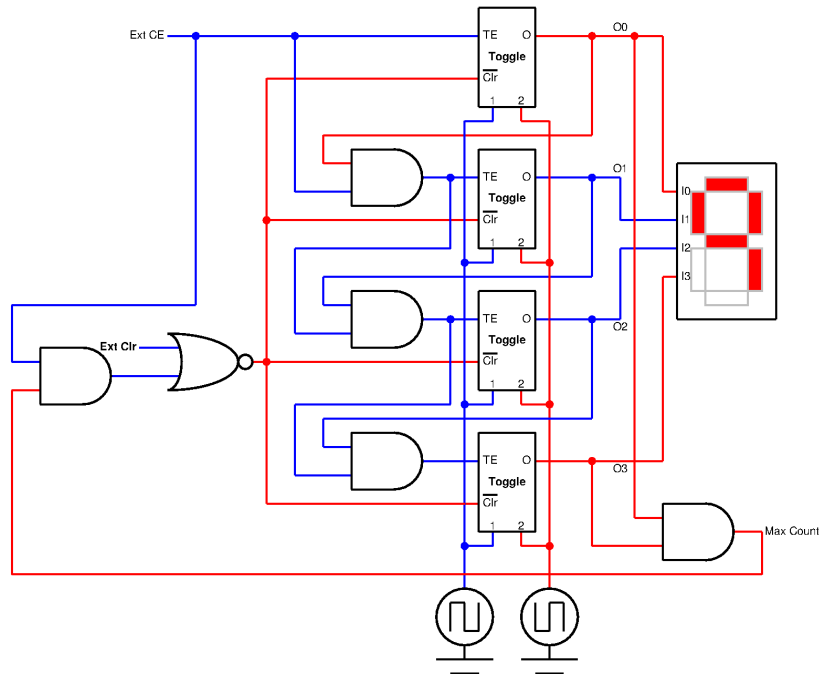


This method applies to an arbitrary value of N. For a divide by N counter, the maximum count is N-1. An N-1 detector will clear the counter on the cycle following the maximum count, restarting the counting sequence.

A Counting Flaw: This scheme works with single digit counters. But often multiple counter is required. For example, a stopwatch uses a decade counter to count seconds, and a divide by six counter to count tens of seconds. Note that the divide by six counter only counts when decade counter has reached its maximum count.



Using this technique will result in a counting flaw when it reaches '50'.



Summary: This chapter has introduced a new type of storage device that changes the stored value (in a predictable way) with time. Here are the main points:

- Counters are built from toggle cells that resemble the register cell used in the last chapter. However they don't have an input. Rather they have a toggle enable (to count) and an active low clear (to reset them to a known value, zero).
- Toggle cells can be cascaded to produce a N bit counter. The counter will begin at zero and count to a maximum count of $2^N - 1$. These are called divide by 2^N counters since they divide the system clock (the two phase non-overlapping clock) by 2^N .
- A *divide by N* counter can be built using a 2^N counter and a **max count** detector. The **Max Count** is $N - 1$. Since the counter starts at zero, only the high values in the maximum count need to be tested. This can be done with an AND gate.
- The **Max Count** signal can assert the Clear to reset the counter state. But only when **Count Enable** is high, when the counter is being instructed to counter *higher* than the maximum count.