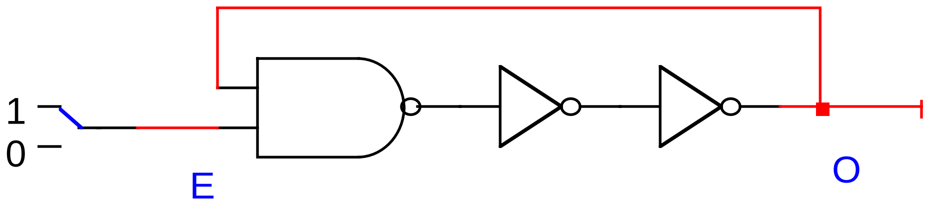
Module 3

Logic Circuits With Memory

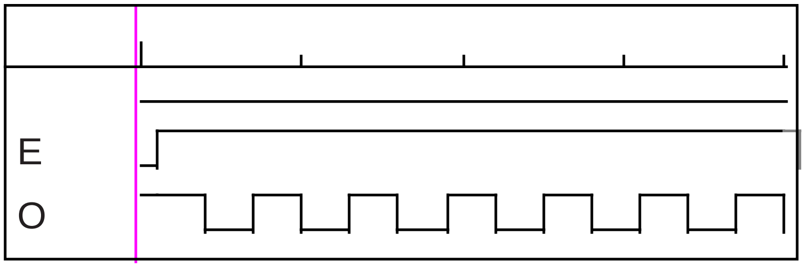
ELEMENTARY FEEDBACK

Consider the following circuit:



Unlike combinational logic circuits that we've seen thus far, notice that this circuit has a FEEDBACK line connected from the output back to the input.

This gives the circuit an interesting behaviour as can be seen from the timing diagram below:



When signal E = 0, the NAND gate is forced to 1, so the output O is held constant at 1. However when E = 1, the NAND acts as an inverter.

We observe from the simulation that the circuit oscillates with a fixed period. The output is no longer strictly a function of the input.

THE STATE TRANSITION TABLE

In describing circuits such as the simple oscillator we need to describe the output as a time-varying function, e.g. O(t).

We call O(t) the PRESENT STATE and O(t+Δt) the NEXT STATE.

We can summarize the behaviour of this simple circuit by noting that

$$O(t+\Delta t) = f(E,O(t)).$$

This behaviour can be described using a function similar in appearance to a truth table, the STATE TRANSITION TABLE.

E	O(t)	O(t+Δt)
0	0	1
0	1	1
1	0	1
1	1	0

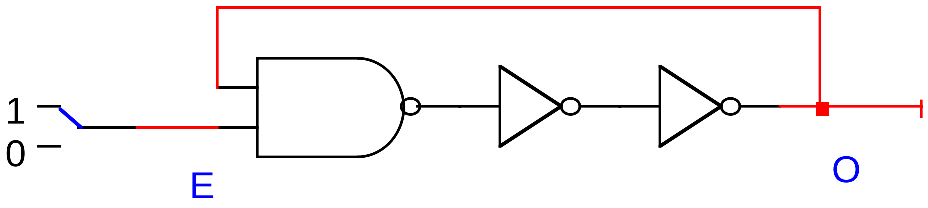
The state transition can describe the behaviour of a logic circuit with feedback much in the same way that a truth table describes the behaviour of combinational logic.

The feedback line serves as a MEMORY. O(t) is referred to as a STATE VARIABLE.

The NEXT STATE, O(t+Δt), is a function of the PRESENT STATE, O(t), and the INPUT E. This is exactly the relation-ship described by the state transition table.

THE STATE TRANSITION TABLE cont.

Let’s take a closer look at the oscillator we built earlier. How do we go about analyzing it?



Assume for the moment that O(t) = 0 and E = 1, and that all gates have a delay of 1 Tg (gate delay).

If O(t=0) = 1, then the initial state of the NAND gate is 1.

At t = 1 Tg, the output of the NAND changes from 1 to 0.

At t = 2 Tg, the output of the first inverter changes from 0 to 1.

At t = 3 Tg, the output O changes from 1 to 0.

At t = 4 Tg, the NAND responds by changing from 0 to 1.

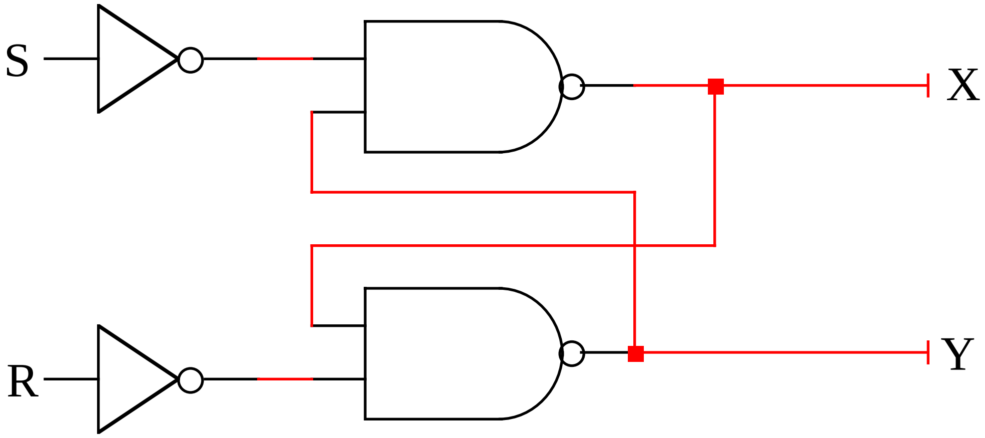
At t = 6 Tg, the O changes from 0 to 1.

Δt for this circuit = 3 tg. The period of oscillation is 2 x Δt.

Hence the oscillator frequency is $\dfrac{1}{2\,\Delta\,t} = \dfrac{1}{6\text{ Tg}}$

THE S-R LATCH

Consider the following circuit:



This circuit is considerably more complex than the simple oscillator considered earlier.

S	R	X(t)	Y(t)	X(t+Δt)	Y(t+Δt)
0	**0**	**0**	**0**	**1**	**1**
0	**0**	**0**	**1**	**0**	**1**
0	**0**	**1**	**0**	**1**	**0**
0	**0**	**1**	**1**	**0**	**0**
0	**1**	**0**	**0**	**0**	**1**
0	**1**	**0**	**1**	**0**	**1**
0	**1**	**1**	**0**	**0**	**1**
0	**1**	**1**	**1**	**0**	**1**
1	**0**	**0**	**0**	**1**	**0**
1	**0**	**0**	**1**	**1**	**0**
1	**0**	**1**	**0**	**1**	**0**
1	**0**	**1**	**1**	**1**	**0**
1	**1**	**0**	**0**	**1**	**1**
1	**1**	**0**	**1**	**1**	**1**
1	**1**	**1**	**0**	**1**	**1**
1	**1**	**1**	**1**	**1**	**1**

THE S-R LATCH cont.

Several observations can be made from a direct observation of the state transition table:

- If $S=1$ and $R=0$, then $X=1$ and $Y=0$ unconditionally. In other words, the output depends only on the input (as in the case of combinational logic).

- If $S=0$ and $R=1$, then $X=0$ and $Y=1$ unconditionally.

- If $S=0$ and $R=0$, then $X(t+\Delta t) = X(t)$ and $Y(t+\Delta t)=Y(t)$, provided that $X(t) \neq Y(t)$.

- If $S=0$ and $R=0$ and $X(t) = Y(t)$, the circuit acts as an oscillator, i.e. $X(t+\Delta t)$ and $Y(t+\Delta t)$ are UNSTABLE states.

If the circuit could be operated such that $X \neq Y$, then one can observe 3 distinct states:

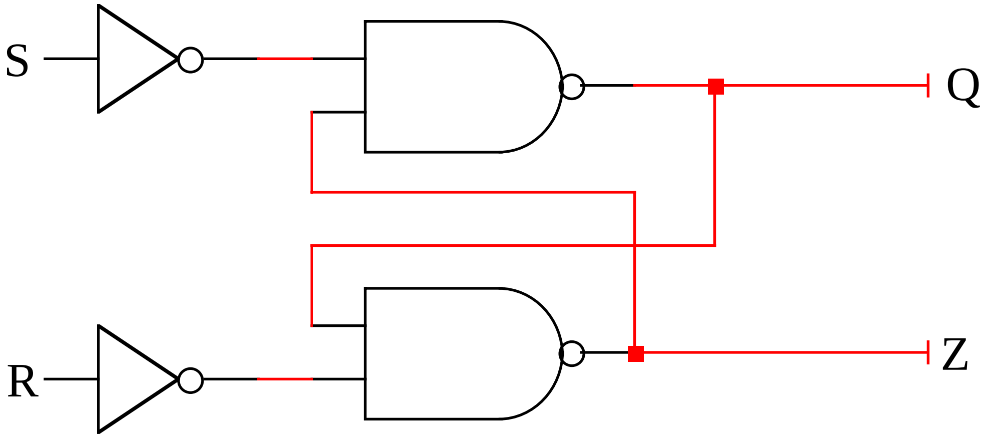
$S=1$, $R=0$	SET mode, $X(t+\Delta t) = 1$ unconditionally
$S=0$, $R=1$	RESET mode, $X(t+\Delta t) = 0$ unconditionally
$S=0$, $R=0$	MEMORY mode $X(t+\Delta t) = X(t)$

where $Y = \overline{X}$

This circuit is called a LATCH and forms a basic memory element (the building block of static rams).

We next have to consider the conditions under which we can guarantee that $Y = \overline{X}$.

ANALYSIS OF THE S-R LATCH



We will use the variable Q to denote the primary output of the latch. The second output, labelled Z in the diagram, is usually understood to correspond to $\overline{Q}$ when the circuit is operating as a latch.

The equations for Q and z are respectively:

$$Q = \overline{(\,\overline{S}\,Z)}$$

$$Z = \overline{(\,\overline{R}\,Q)}$$

Applying de Morgan’s law we derive an alternate form

$$\overline{Q} = \overline{S + \overline{Z}}$$

$$\overline{Z} = \overline{R + \overline{Q}}$$

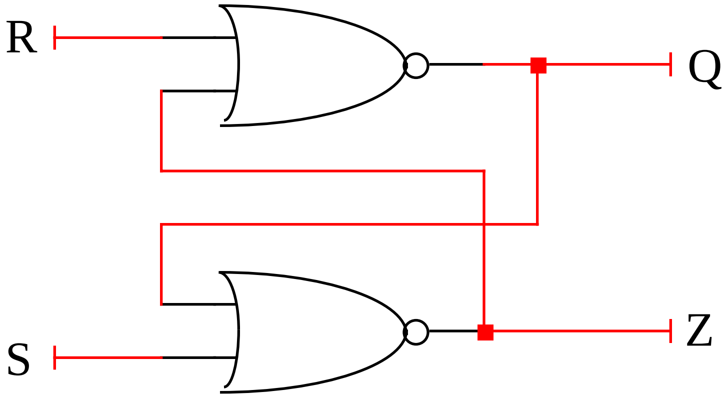
ANALYSIS OF THE S-R LATCH cont.

Now if we substitute Z for $\overline{Q}$ and Q for $\overline{Z}$ we obtain an alternate realization for the S-R latch:

$$Z = \overline{S + Q}$$

$$Q = \overline{R + Z}$$

This should not be surprising given the Duality principle.



In the analysis that follows, we will use the NOR version of the latch to derive a set of operating conditions that guarantees $Z = \overline{Q}$.

We begin by writing the circuit equations:

$$Q(t + Tg) = \overline{[R(t) + Z(t)]}$$

$$Z(t + Tg) = \overline{[S(t) + Q(t)]}$$

Notice that this takes circuit propagation delay into account.

ANALYSIS OF THE S-R LATCH cont.

Applying De Morgan:

$$Q(t+Tg) = \overline{R(t)} \bullet \overline{Z(t)}$$

Tg units later:

$$Q(t+2Tg) = \overline{R(t+Tg)} \bullet \overline{Z(t+Tg)}$$

Substituting:

$$Q(t+2Tg) = \overline{R(t+Tg)} \bullet \left[S(t)+Q(t)\right]$$

$$Z(t+2Tg) = \overline{S(t+Tg)} \bullet \left[R(t)+Z(t)\right]$$

When is Z = $\overline{Q}$?

Negate:

$$\overline{Q(t+2Tg)} = \overline{\left\{\overline{R(t+Tg)} \bullet \left[S(t)+Q(t)\right]\right\}}$$

De Morgan:

$$\begin{aligned}
\overline{Q(t+2Tg)} &= R(t+Tg) + \overline{\left[S(t)+Q(t)\right]} \\
&= R(t+Tg) + \overline{S(t)} \bullet \overline{Q(t)} \\
&= R(t+Tg)\left[S(t) + \overline{S(t)}\right] + \overline{S(t)} \bullet \overline{Q(t)} \\
&= \overline{S(t)}\left[R(t+Tg) + \overline{Q(t)}\right] + S(t) \bullet R(t+Tg)
\end{aligned}$$

If Z = $\overline{Q}$ Then

$$\overline{S(t+Tg)} \bullet \left[R(t)+Z(t)\right] = \overline{S(t)}\left[R(t+Tg) + \overline{Q(t)}\right] + S(t) \bullet R(t+Tg)$$

ANALYSIS OF THE S-R LATCH cont.

If $Z = \overline{Q}$ Then

$$\overline{S(t+Tg)} \bullet \left[R(t) + Z(t) \right] \;=\; \overline{S(t)} \left[R(t+Tg) + \overline{Q(t)} \right] + S(t) \bullet R(t+Tg)$$

Which means that:

- $S(t+Tg) = S(t)$ S held stable for at least Tg units
- $R(t+Tg) = R(t)$ R held stable for at least Tg units
- $S(t) \bullet R(t) = 0$ $S \neq 1$ and $R \neq 1$

Furthermore, in order to correctly latch the data in the first place, R (or S) must be toggled high for at least 2Tg units to be sure that the latch is correctly reset (or set).

By maintaining the inputs stable for a minimum of 2Tg units and ensuring that $S \neq 1$ and $R \neq 1$, then $Y = \overline{Q}$.

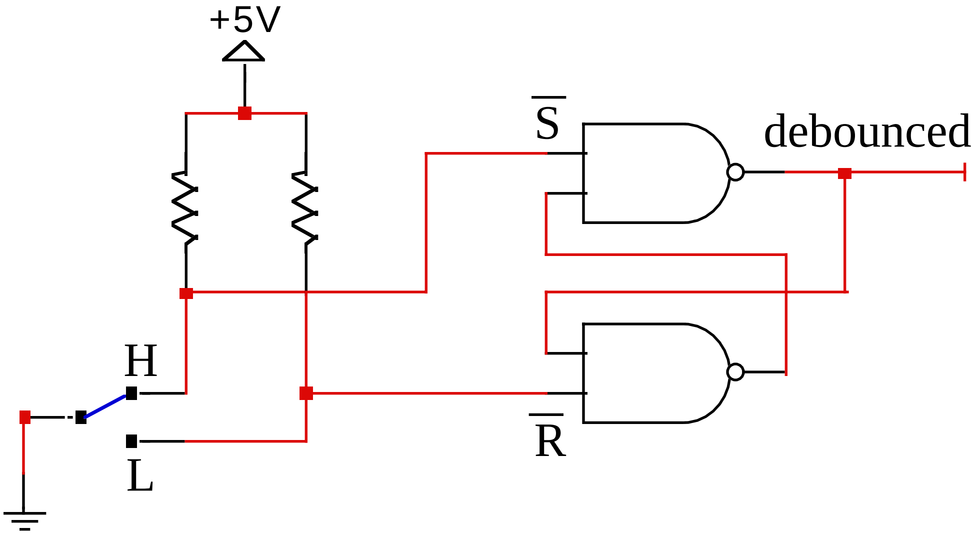
S	R	Q	Q(t+Δt)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	undef
1	1	1	undef

Under these assumptions the state transition table for the S-R latch is given at left.

Since Z = Q’, it does not appear in the table.

APPLICATION - SWITCH DEBOUNCING

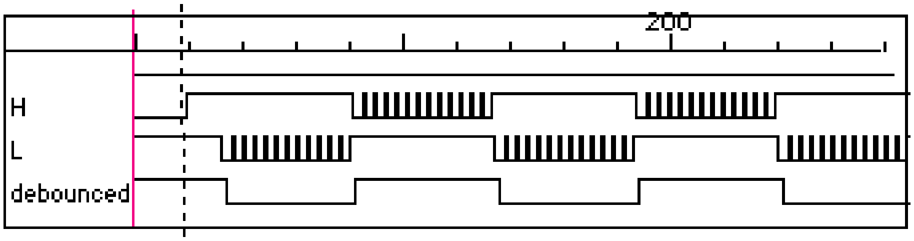
A slight variation of the S-R latch using 2 NAND gates yields an $\overline{S}\,\overline{R}$ latch. Consider the circuit shown below.



In the state shown, the input to the latch corresponds to $\overline{S} = 0$ and $\overline{R} = 1$, which means the output is 1.

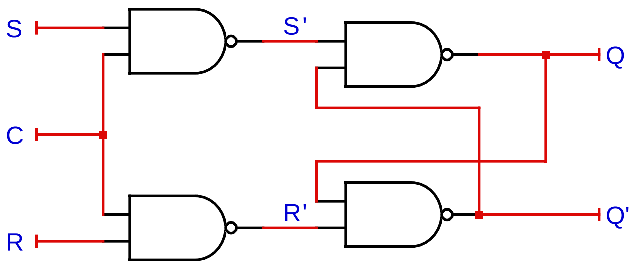
When the switch moves off the H contact, $\overline{S} = \overline{R} = 1$, so the latch maintains its state. On contact with L, $\overline{S} = 1$, and $\overline{R} = 0$, changing the output of the latch from 1 to 0.

Successive bounces off the L contact cannot change the state of the latch since this corresponds to $\overline{S} = \overline{R} = 1$.



THE CLOCKED S-R LATCH

Consider the modified version of the S-R latch shown below.



It looks like the NAND version of the latch, but has an additional control line labeled C, called the CLOCK. Using standard methods the state transition table is determined as follows:

C	S	R	Q(t)	Q(t+Δt)
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	undef
1	1	1	1	undef

Notice that when C = 1, the state transition table is identical to that of the non-clocked latch.

When C=0, both input NANDs are forced to 1 which forces the latch into the MEMORY state.

The Clock line thus determines when the latch is allowed to change state.

THE CLOCKED S-R LATCH cont.

From now on we will refer to the clocked latch as a FLIP-FLOP provided that we operate it according to the following rules:

- Inputs (S & R) can only change when C is low.
- Inputs must be held stable for the entire interval when the clock is high.
- The clock must be held high for at least 2 Tg (to ensure that the data gets latched properly).

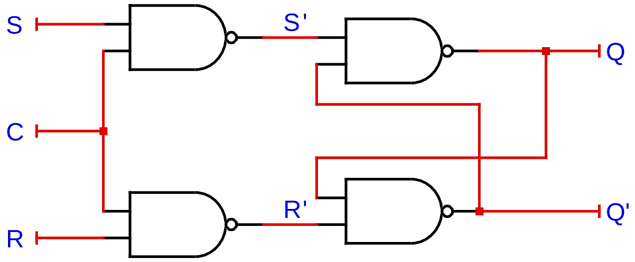
In this mode of operation the role of the clock is to synchronize state changes. Rather than defining the next state of Q(t) as Q(t+Δt), we instead use the notation Q(t+Tc), or $\hat{Q}$ to indicate that the state change is determined by external synchronization (i.e. a clock).

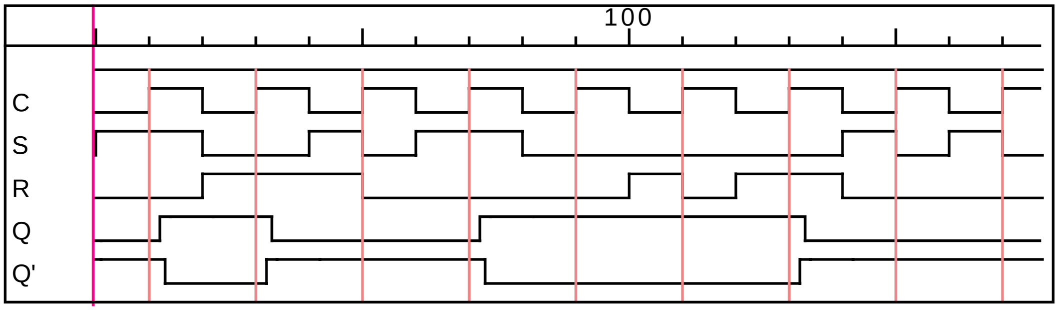
This also simplifies the state transition table:

S	R	Q	$\hat{Q}$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	undef
1	1	1	undef

THE CLOCKED S-R LATCH cont.

The timing behaviour of the clocked S-R latch is shown in the following LogicWorks simulation.





Unfortunately the clocked latch has a major limitation which makes it very difficult to use (i.e. as a flip-flop) in practice.

As long as the clock is high, the output will always follow the input (i.e. state changes are not completely governed by the clock).

Some control is afforded by creating asymmetric clock pulses (i.e. short HIGH intervals), but this can make design using these devices VERY complicated as we shall see shortly.

THE J-K FLIP-FLOP

The J-K flip-flop is a slight variation on the S-R where the S=1 R=1 state is given a distinct behaviour.

J	K	Q	$\hat{Q}$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Notice that we've renamed S and R with J and K to indicate this new bahaviour.

When J=K=1, the next state of Q is its complement.

We call this behaviour TOGGLING.

In fact, the J-K flip-flop can be derived from the S-R as follows.

All we need to do is to figure out
what to feed the S and R inputs of the S-R flip-flop when we encounter the case of J = K = 1 with Q = 1 and Q = 0 respectively.

J	K	Q	S	R
0	0	0	0	d
0	0	1	d	0
0	1	0	0	d
0	1	1	0	1
1	0	0	1	0
1	0	1	d	0
1	1	0	1	0
1	1	1	0	1

THE J-K FLIP-FLOP cont.

From the truth table for S and R vs. J, K, and Q, we obtain the following Karnaugh maps:

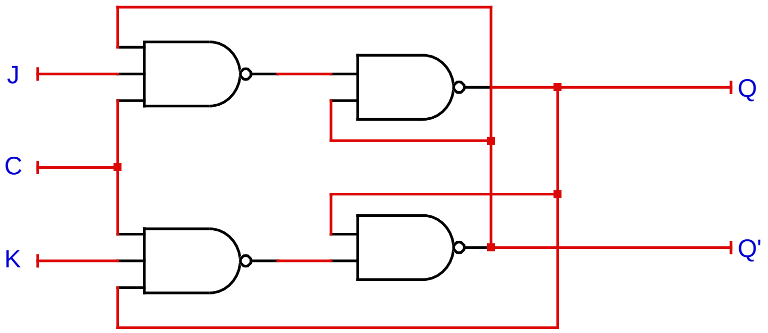
S	KQ			
J	00	01	11	10
0	0	d	0	0
1	1	d	0	1

R	KQ			
J	00	01	11	10
0	d	0	1	d
1	0	0	1	0

From the maps we obtain the following functions for S and R as functions of J, K, and Q:

$$S = J\overline{Q} \qquad R = KQ$$

Which results in the following circuit:



Note that the ANDs associated with the expressions for S and R fold into the NAND gates as shown above.

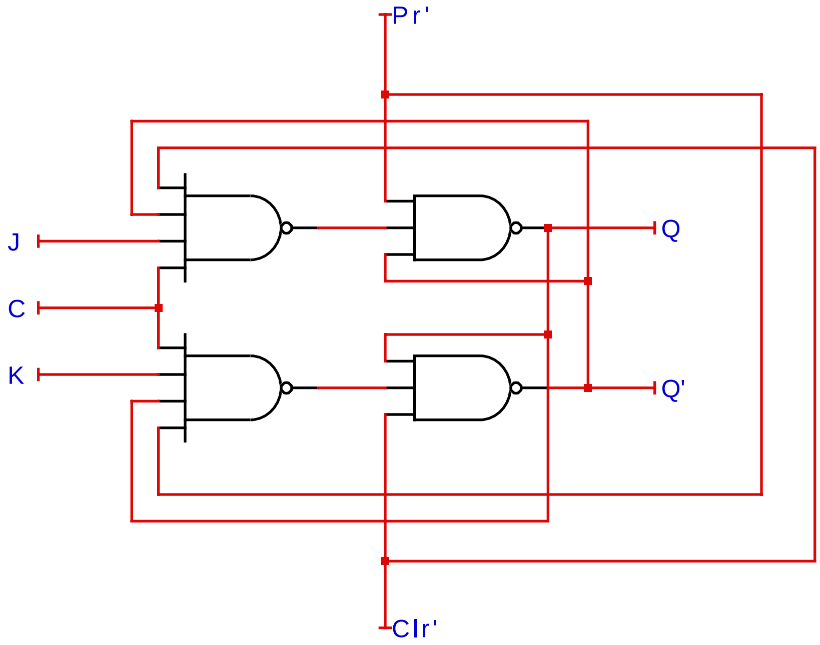
According to the theory, this should implement the state transition table for the J-K flip-flop. But does it?

THE J-K FLIP-FLOP cont.

Some modification of the circuit will be required before proceeding with the simulation.

Because of the additional feedback lines, it is impossible to force the flip-flop into an initial state, e.g. by setting J = 1, K = 0.

Practical flip-flops have PRESET and CLEAR lines which allow the flip-flop to be forced into an initial state, regardless of the clock.



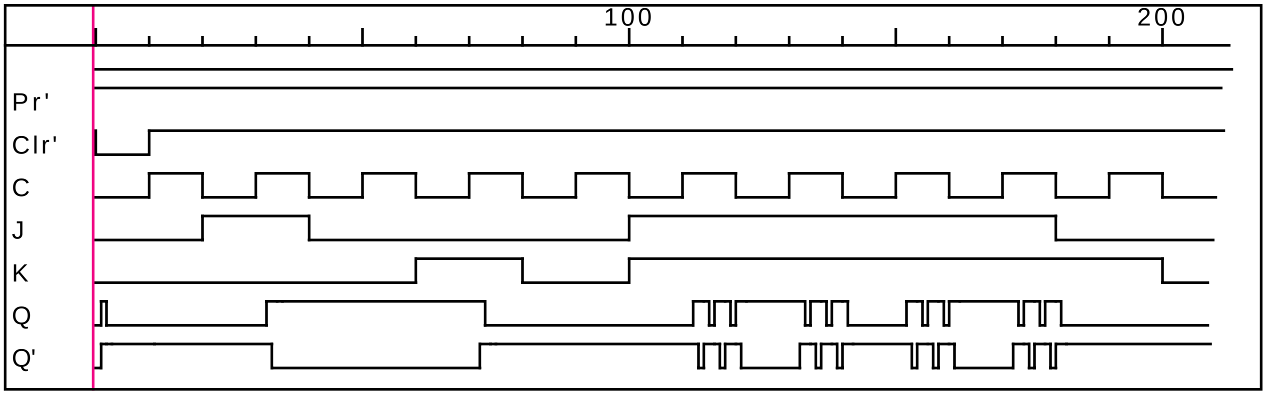
When $\overline{\text{Pr}}$ = 0 and $\overline{\text{Clr}}$ = 1, the top right and bottom left NANDs are forced to 1, which forces the bottom right NAND to 0. We say that this PRESETS the flip-flop to 1.

THE J-K FLIP-FLOP cont.

When $\overline{\text{Pr}}$ = 1 and $\overline{\text{Clr}}$ = 0, the top left and bottom right NANDs are forced to 1, which forces the top right NAND to 0. We say that this CLEARS the flip-flop to 0.

With this modification, we can guarantee the initial state of the flip-flop (especially important in a simulation!).

We’re now ready to run the simulation:



The result? Almost, but not quite. The toggle (J = K = 1) mode does not work properly. The flip-flop oscillates while the clock is high. Why?

The problem is referred to as a RACE condition. Because of the feedback lines from $\overline{\text{Q}}$ to J and Q to K, the signal has the opportunity to work its way back to the input BEFORE the clock is pulled low.

To operate properly as a J-K flip-flop, the clock must be adjusted so that only ONE state change occurs.

THE J-K FLIP-FLOP cont.

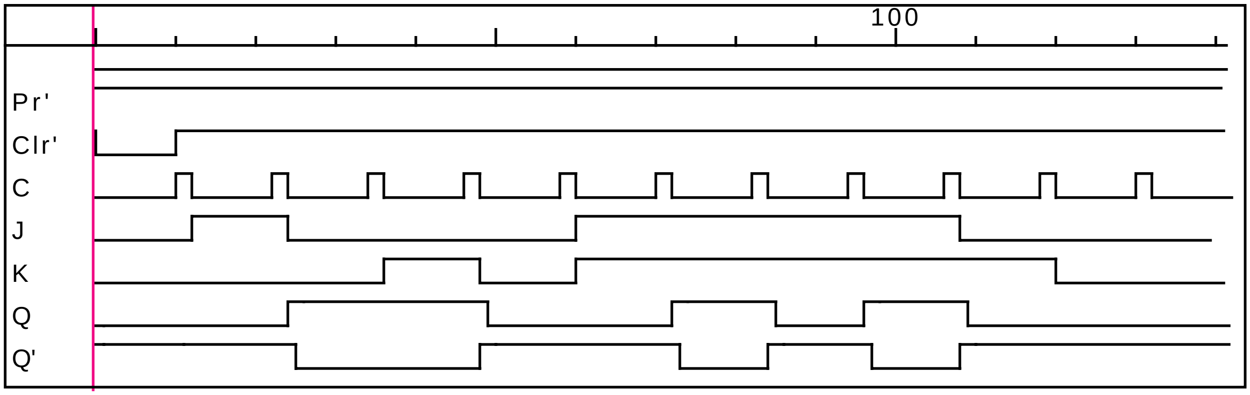
From examination of the circuit, it takes 2 Tg for the latch to change state + an additional 1 Tg for the signal to propagate through the input NAND gates.

If the clock is high for ≥ 3 Tg, a race condition will occur.

If the clock is high for < 2 Tg, the latch will not operate correctly.

So, for this circuit to operate properly, the clock high period must be exactly 2 Tg.

Let’s test the theory through simulation:

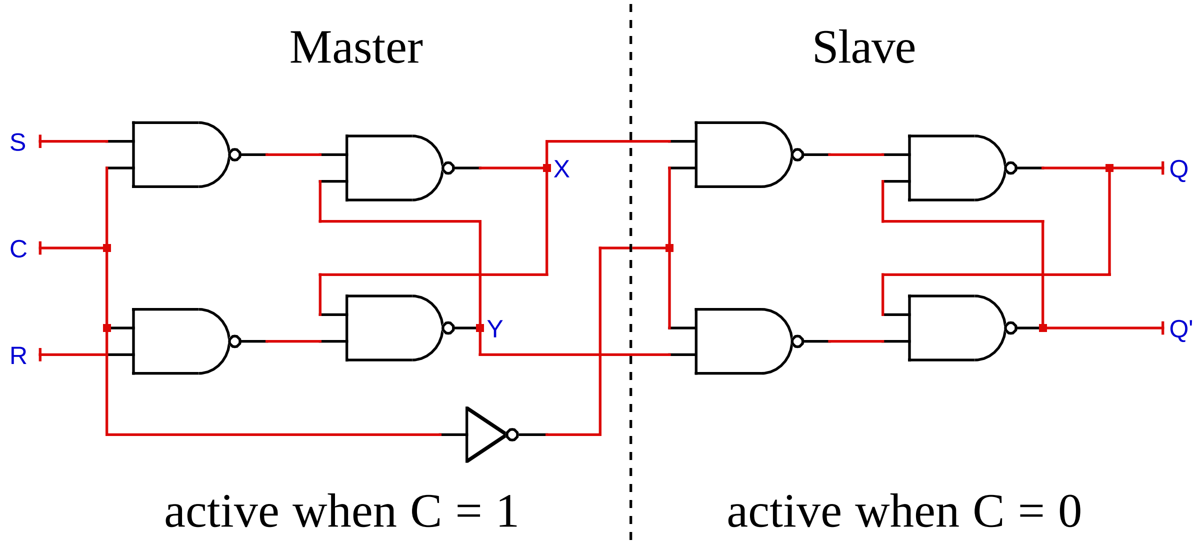


This time everything works, but the circuit is clearly impractical because of its dependence on precise clock timing.

We need a different flip-flop design that is less dependent on the clock signal.

THE MASTER-SLAVE FLIP-FLOP

Consider the following circuit:



It is called a MASTER-SLAVE flip-flop and is designed to make timing less dependent on the clock.

Notice that it is formed from two clocked latches, the first (master) operates when C = 1, and the second (slave) operates when C = 0.

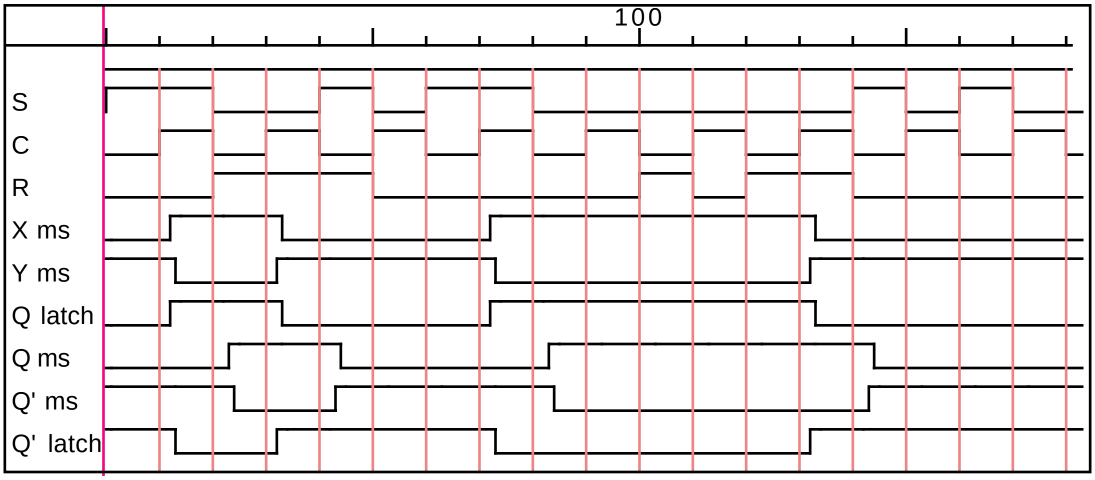
When C = 1, the master changes state according to S and R; the slave is forced to memory mode.

When C = 0, the master is forced into memory mode, and the slave is activated, changing state according to the master’s outputs X and Y.

As long as C = $\overline{C}$ there cannot be more than a single state change for a given clock pulse, i.e. races are eliminated.

THE MASTER-SLAVE FLIP-FLOP cont.

We can compare the behaviour of the clocked latch and master-slave flip flops in the following simulation.



Notice that the master-slave and clocked latch flip-flops have identical responses (as one would expect) except for when the outputs change state.

- The clocked latch changes state when C goes from 0 -> 1

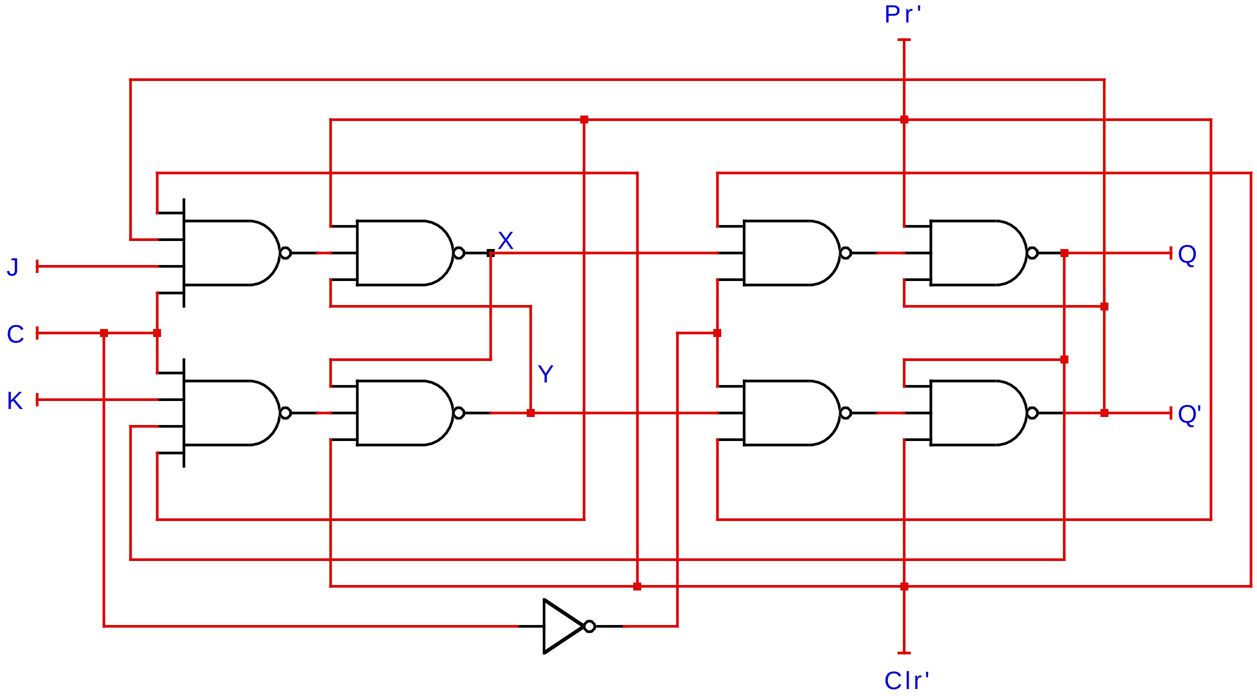
- The master slave changes when C goes from 1 -> 0

This simulation only confirms that the master slave and clocked latch have the same state transition table.

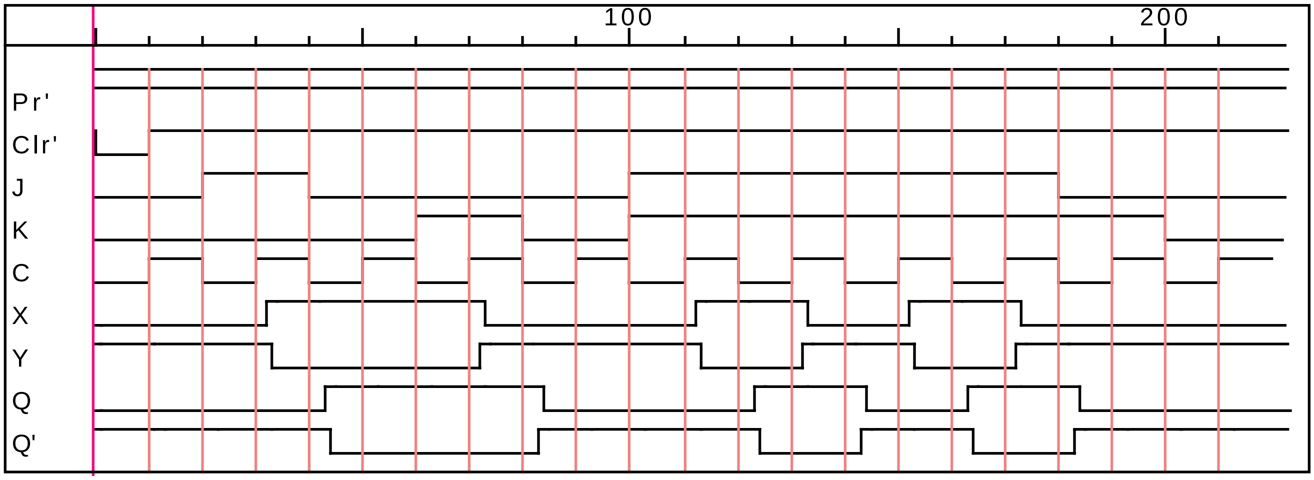
In order to verify that race conditions are prevented, we need to consider a master-slave implementation of the J-K flip-flop.

THE J-K MASTER-SLAVE FLIP-FLOP

We can construct the master-slave version of the J-K from the clocked latch version shown earlier by cascading the additional slave stage as shown below.



Let's repeat the earlier simulation where Tc = 10 Tg.

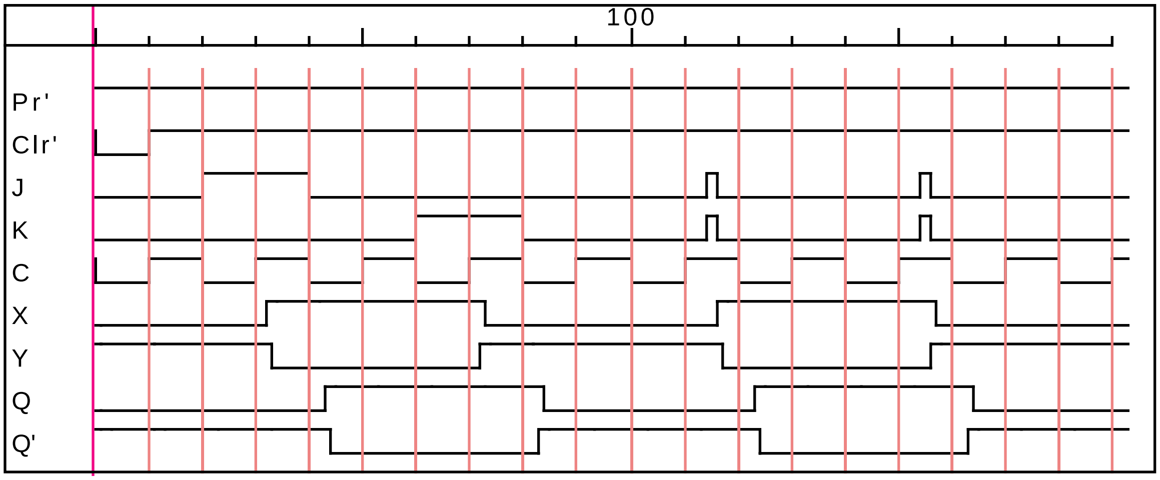


THE J-K MASTER-SLAVE FLIP-FLOP

The J-K master-slave flip-flop gets around the earlier dependency on the clock pulse width (to avoid race conditions).

As long as Tc ≥ 2 Tg, the circuit will function correctly.

Unfortunately, the master-slave still has one liability which is illustrated in the following simulation.



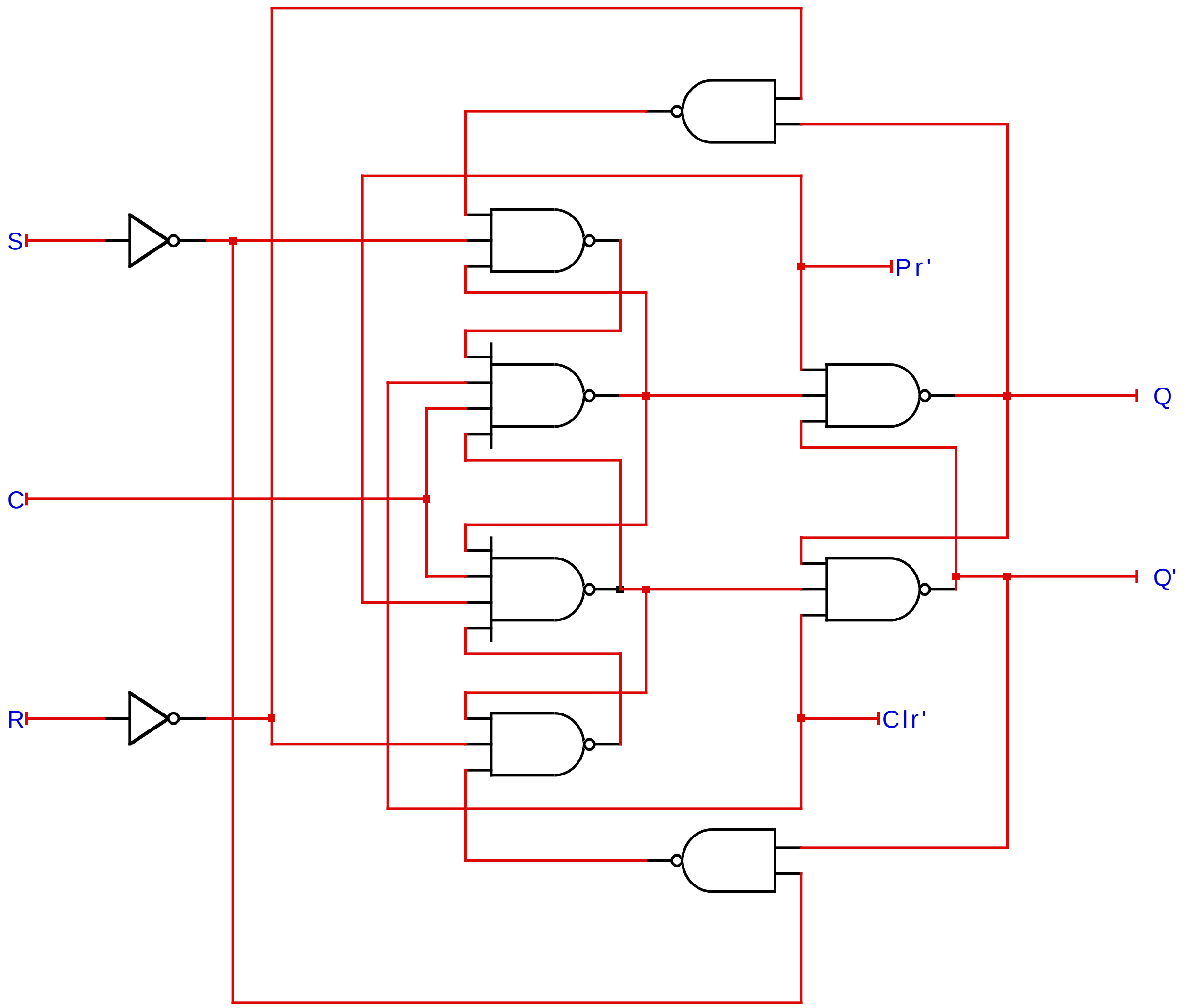
Normally the flip-flop "remembers" the state of J-K just prior to the 1 -> 0 clock transition.

The J = 1, K = 1 state poses problems because the master will latch J = K = 1, regardless if either changes to 0 before the clock transition.

This is called 1's and 0's catching where the 1 refers to the flip-flop changing from 0 to 1, and the 0 to the flip-flop changing from 1 to 0.

EDGE-TRIGGERED FLIP-FLOP

Consider the circuit shown below.



As you might guess from the labels, this is another S-R flip-flop (which can be converted to a J-K in the usual way).

In terms of complexity, it is about the same order as the master-slave version we considered earlier. This architecture is refer-red to as EDGE-TRIGGERED for reasons that will hopefully become clear shortly.

EDGE-TRIGGERED FLIP-FLOP cont.

An analysis of the edge-triggered flip-flop (which is quite complex) leads to the following state transition table.

C(t)	C(t+T)	S(t)	R(t)	Q(t)	Q(t+T)	
0	0	don’t care		0	0	no clock transition
0	0	don’t care		1	1	
0	1	0	0	0	0	rising
0	1	0	0	1	1	rising
0	1	0	1	0	0	rising
0	1	0	1	1	0	rising
0	1	1	0	0	1	rising
0	1	1	0	1	1	rising
0	1	1	1	undef	undef	rising
0	1	1	1	undef	under	rising
1	0	don’t care		0	0	falling edge
1	0	don’t care		1	1	
1	1	don’t care		0	0	no clock transition
1	1	don’t care		1	1	

Notice that the table explicitly represents transitions of the clock, i.e. C(t) and C(t+T).

The essential property of the edge-triggered flip-flop is that it samples its inputs ON THE RISING EDGE of the clock.

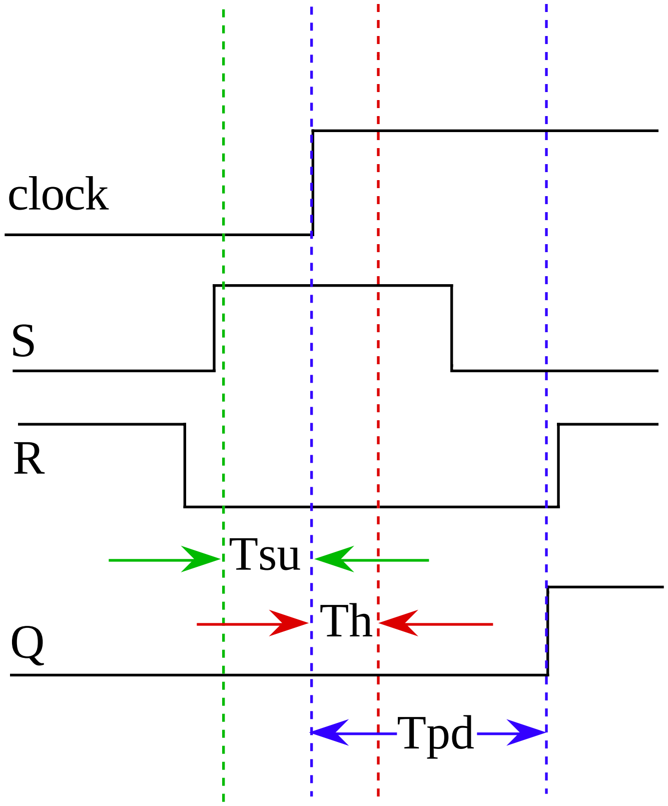
Input changes at any other time are ignored. The transition table above corresponds to a RISING edge-triggered flip-flop.

EDGE-TRIGGERED FLIP-FLOP cont.

It is also possible to design FALLING edge-triggered flip-flops which have a similar state transition table except that inputs are sampled on the falling edge of the clock.

In both cases the flip-flops change after an interval marked by the active edge of the clock, the propagation delay, Tpd.

Circuit properties impose certain constraints on how these flip-flops are operated. These are summarized below.



There are 3 important parameters, all measured with respect to the active edge of the clock:

- Tsu: Set-up time
- Th: Hold time
- Tpd: Propagation delay

Tsu defines the interval prior to the active edge when INPUTS MUST BE HELD STABLE.

Th defines the interval following the active edge of the clock during which inputs must be held stable.

Tpd defines the maximum interval following the active edge when the flip-flop outputs change in response to the input.

EDGE-TRIGGERED FLIP-FLOP cont.

Tsu + Th defines a WINDOW during which inputs must be held stable.

In fact, we can use this terminology to define the operating constraints for all the flip-flops we have encountered thus far.

Clocked Latch:

- Tsu = 0 and Th is the interval during which the clock is high. Tpd is 3 Tg for the circuit shown earlier.

Master-Slave:

- For J-K or S-R, Tsu = interval during which clock is high. For D, Tsu = 3 Tg, the time required for the master to latch prior to the 1 -> 0 clock transition. Th = 0, and Tpd = 3 Tg, the same for the clocked latch.

Edge-Triggered:

- These parameters have no dependency on the clock, i.e., they are CIRCUIT DEPENDENT. This is why edge-triggered flip-flops have such widespread use in design.

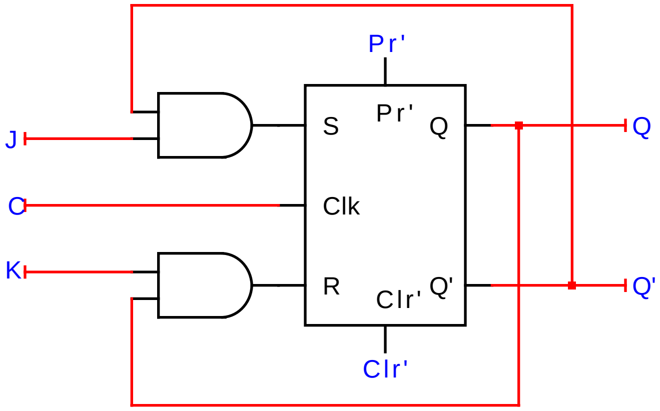
For D implementations, it’s virtually impossible to tell master-slave and edge-triggered flip-flops apart. They both can be made to sample and change states on either the rising or falling edges of the clock.

EDGE-TRIGGERED FLIP-FLOP cont.

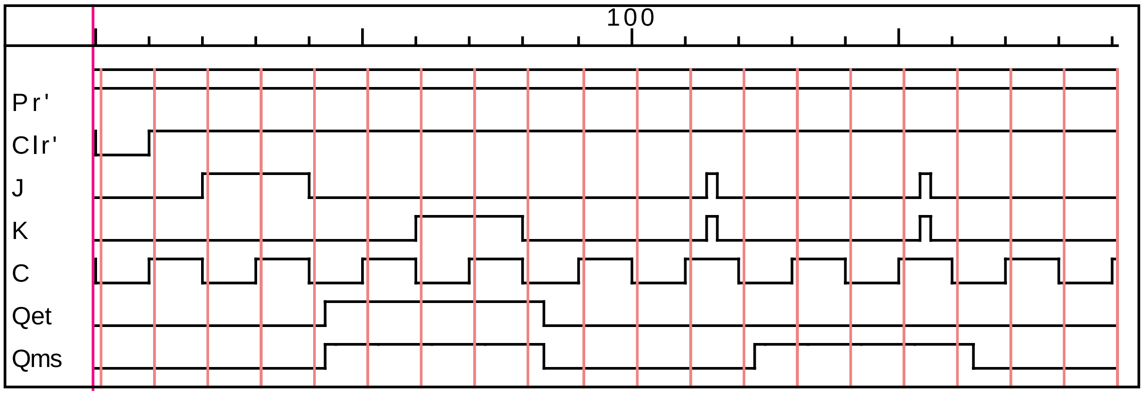
Recall that the edge-triggered flip-flop grew out of a limitation of the J-K master-slave, the problem with 1's and 0's catching.

Let's repeat the earlier simulation, but this time we'll compare the outputs of the two different flip-flops.

A J-K edge-triggered flip-flop is fabricated in the usual way:
(we'll use a falling-edge triggered version in the experiment)



And here's the simulation comparing the 2 flip-flop types:

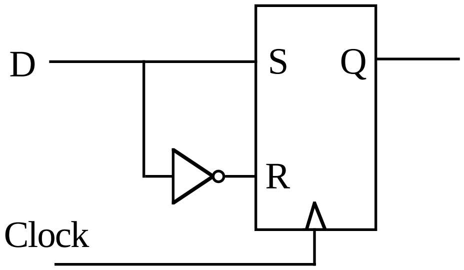


The ET version is not susceptible to 1's and 0's catching.

OTHER FLIP-FLOP TYPES

D FLIP-FLOP

A subset of the S-R (J-K) flip-flop where R = S’. It produces a delay of exactly 1 clock period.

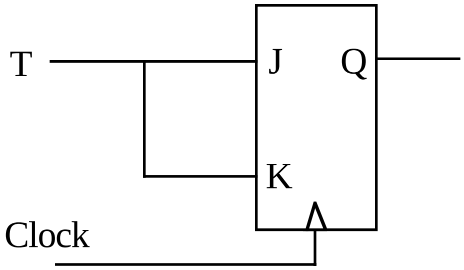


D(t)	Q(t)	Q(t+T)
0	0	0
0	1	0
1	0	1
1	1	1

D flip-flops are used to build registers. The edge-triggered variety has a particularly simple circuit (as compared to the S-R implementation shown earlier).

T FLIP-FLOP

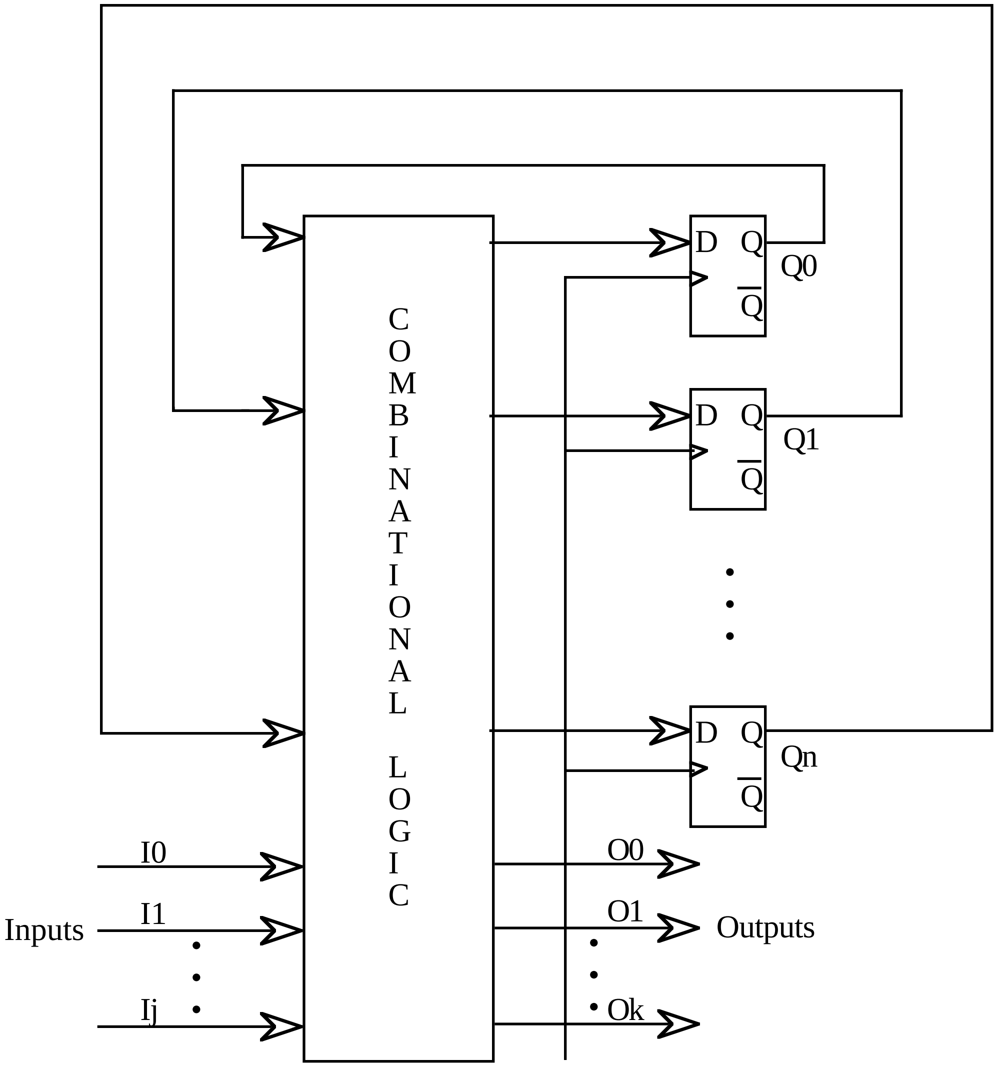
This is another subset of the J-K flip-flop in which J = K. It is the basic building block used to fab-ricate COUNTERS.



T(t)	Q(t)	Q(t+T)
0	0	0
0	1	1
1	0	1
1	1	0

ANALYSIS OF SEQUENTIAL CIRCUITS

Flip-flops are the simplest forms of the more general class of SEQUENTIAL CIRCUITS. The general model for such a circuit is shown below.



The logical description of such a circuit is given by its corresponding state transition table. We will now consider is how to determine the state transition table given the circuit diagram.

ANALYSIS OF SEQUENTIAL CIRCUITS cont.

The state transition table will have the following form:

I1	I2	Ij	Qn	Q1	Q0	$\tilde{Q}n$	$\tilde{Q}1$	$\tilde{Q}0$	O0	O1	Ok
0	0	0	0	0	0						
0	0	0	0	0	1						
...	...	...	...	...	...						
1	1	1	1	1	1						

The left side is composed of the INPUTS, I0, I1, ..., Ij and STATE VARIABLES Q0, Q1, ..., Qn. Each flip-flop (memory element) is associated with a unique state variable.

The right side is composed of the NEXT STATES of each state variable, i.e., $\tilde{Q}0$, $\tilde{Q}1$, ..., $\tilde{Q}n$, and the OUTPUTS, O0, O1, ..., Qk.

The outputs are easy to determine as they are just combinational logic functions, i.e., Oi = f(I1, I2, ..., Ij; Q0, Q1, ..., Qn).

The next states of the Qi are determined in similar fashion, but an additional step is required.

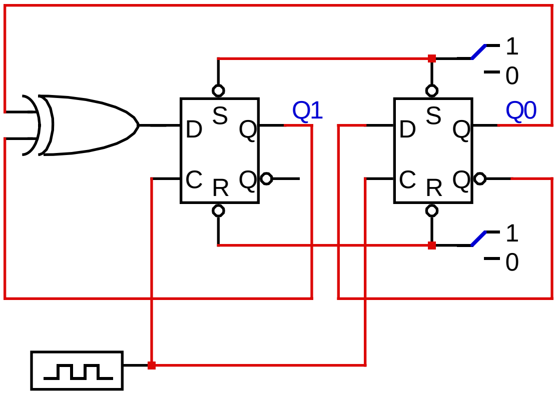
- First, determine the functions associated with each flip-flop control input. In the case of D flip-flops, one would have a single expression for each D input where Di = f(I1, I2, ..., Ij; Q0, Q1, ..., Qn). For J-K flip-flops, one would have two equations, i.e., for Ji and Ki.

ANALYSIS OF SEQUENTIAL CIRCUITS cont.

- Once the flip-flop equations are determined, the state transition table for the particular flip-flop is used to determine the next state.

This is quite straightforward for the case of D flip-flops, as Q(t+T) = D(t).

Example 1:



1. Outputs

In this case the circuit has no formal outputs. Sometimes the "output" consists of the state variables themselves.

2. Flip-Flop Equations

D1 = XOR(Q1, Q0)

D0 = (Q0)'

We now compute Q1(t+T) and Q0(t+T) by substituting the expressions for D1 and D0 respectively, i.e.,

Q1(t+T) = XOR(Q1(t), Q0(t))

$Q0(t+T) = \overline{Q0(t)}$

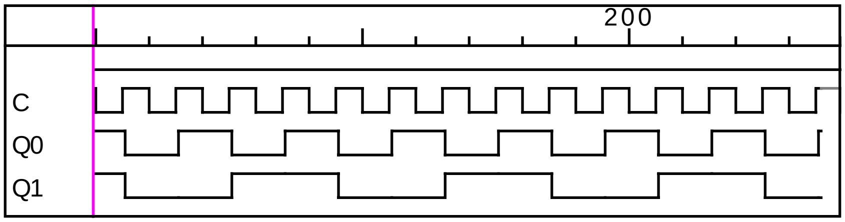
It is now easy to fill in the right hand side of the state transition table and predict the behaviour of the circuit.

ANALYSIS OF SEQUENTIAL CIRCUITS cont.

State Transition Table:

Q1(t)	Q0(t)	Q1(t+T)	Q1(t+T)
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

According to the table, this circuit corresponds to a Modulo-2 counter. Let’s check this prediciton against the state transition table.



Which is exactly the case. (By the way, the notations Q(t+T) and $\tilde{Q}$ are used interchangably in these notes, so don’t be confused).

Sequential circuits with D flip-flops are particularly easy to analyze, because the D flip-flop equation (which summarizes its state transition table) amounts to an identity operator.

Analyzing circuits with J-K flip-flops is a bit more complicated.

ANALYSIS OF SEQUENTIAL CIRCUITS cont.

Let’s start off by deriving the equation for Q(t+T) as a function of J(t), K(t), and Q(t).

Q(t+T)

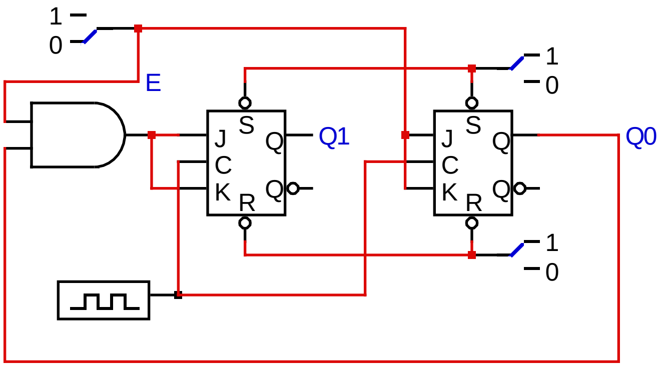
Q(t) \ JK	00	01	11	10
0	0	0	1	1
1	1	0	0	1

The state transition table for the J-K flip-flop is represented at left in Karnaugh map form.

From the map we determine that $Q(t+T) = J\,\overline{Q(t)} + \overline{K}\,Q(t)$.

For analyzing sequential circuits, it’s much easier to work with so-called FLIP-FLOP equations because they allow us to derive total expressions for Q(t+T) via substitution.

Another Example:



This circuit has a single input E.

From the schematic at left we find:

$$J0 = K0 = E$$

$$J1 = K1 = E \bullet Q0$$

Thus:

$$Q1(t+T) = E\,Q0(t)\,\overline{Q1(t)} + \overline{E\,Q0(t)}\,Q1(t)$$

$$Q0(t+T) = E\,\overline{Q0(t)} + \overline{E}\,Q0(t)$$

ANALYSIS OF SEQUENTIAL CIRCUITS cont.

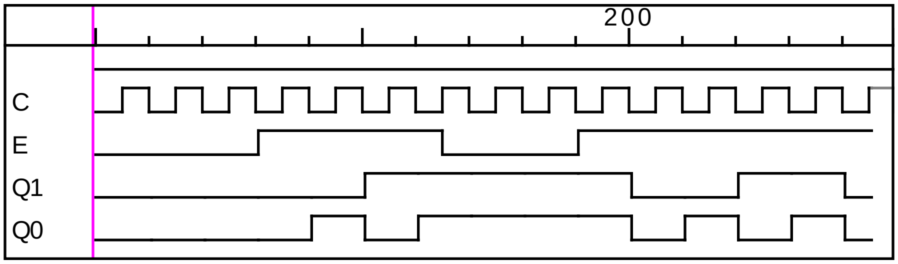
From these equations we can fill in the state transition table as follows:

E	Q1(t)	Q0(t)	Q1(t+T)	Q0(t+T)
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
0	1	1	1	1
1	0	0	0	1
1	0	1	1	0
1	1	0	1	1
1	1	1	0	0

This behaviour is similar to the circuit we encountered earlier as long as the E = 1.

However, when E = 0, the counter doesn't count - it simply holds its current state. The E stands for ENABLE.

Again, let's verify the prediction of the state transition table against the circuit simulation. It works!



TIMING ANALYSIS OF SEQUENTIAL CIRCUITS

The subject of timing analysis comprises an entire field of study within Digital Systems Design.

For now we'll consider one particular model of timing behaviour based on edge-triggered flip-flops. The reasons for this model will become apparent in more advanced courses.

Edge-Triggered Timing Model:

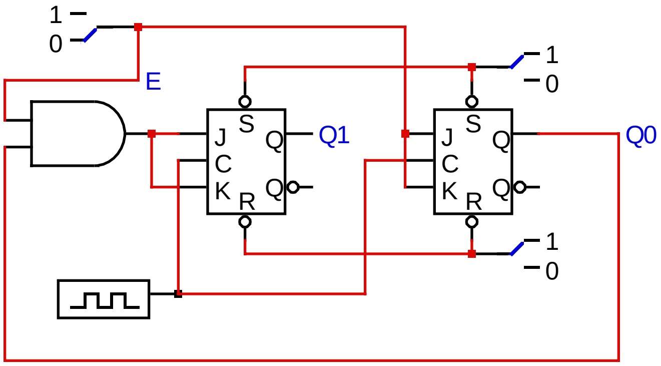
- All registers are composed of edge-triggered flip-flops.
- All registers are synchronized by a single clock on the same clock edge.
- The clock is connected to all registers at all times (i.e. the clock may NOT be gated using combinational logic).
- All inputs must respect Tsu and Th requirements imposed by the maximum values over all registers in the circuit.
- The clock must respect any timing constraints imposed by registers. For edge-triggered flip-flops this usually corresponds to a minimum clock pulse width, Tw.

Both of the circuits that we have encountered thus far meet these requirements.

Timing analysis (for the purposes of this course) will involve determining the maximum operating frequency of the circuit.

TIMING ANALYSIS cont.

Consider the counter circuit analyzed earlier:



Assume that register Q1 has parameters

Tsu = 5nS
Th = 5nS
Tpd = 20nS
Tw = 20nS

and that Q0 has parameters Tsu = 0nS, Th=10nS, Tpd=25nS and Tw=25nS.

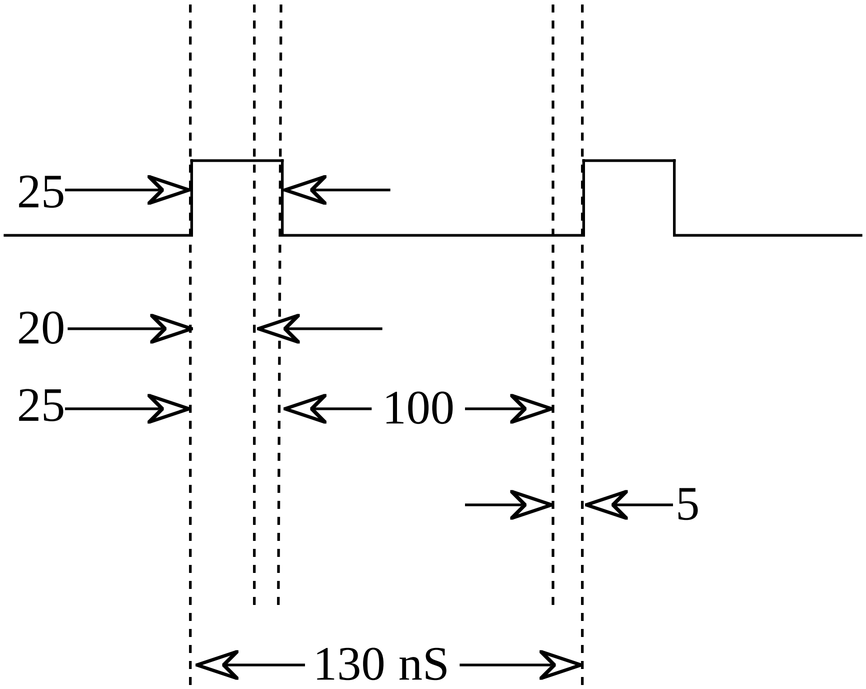
Assume (for the sake of argument) that the single gate in the circuit has a propagation delay of 100nS. In general we're interested in the LONGEST combinal logic delay from output back to input over the entire circuit. We call this parameter Tcl. In this case Tcl is simply the propagation delay of the gate, i.e. Tcl = 100nS.

Determine the maximum operating frequency for this circuit.

The maximum (worst case) delays for each parameter are Tsu = 5nS, Th = 10nS, Tpd = 25nS, Tw = 25nS, Tcl = 100nS.

To determine maximum operating frequency, we need to trace through one cycle of the clock:

TIMING ANALYSIS cont.



We begin at the rising edge of the clock (assuming flip-flops are rising edge triggered).

- 20nS after the clock edge, flip-flop Q1 changes in response.
- 5nS later, at Tc + 25, flip-flop Q2 changes in response.
- The inputs to the gate are now stable. 100nS later the output changes in response to these inputs.
- Thus after a total of 125nS, the inputs to the flip-flops are stable. However, before clocking again, an additional 5nS (set-up time for Q0) must be waited.

TIMING ANALYSIS cont.

The minimum time between clock edges thus totals 130nS, which means the maximum frequency is

$$\frac{1}{130\text{nS}} = 7.69\ \text{Mhz}.$$

From this analysis we may infer the following formula:

$$f_{max} = \frac{1}{Tsu + Tcl + Tpd},$$

where Tsu, Tcl, and Tpd correspond to the maximum delays over the entire circuit.

What happened to Th?

Nothing. Because Tpd is generally greater than Th, the inputs to each flip-flop are guaranteed to be stable provided that any external inputs are held stable as well.

One of the biggest problems in designing these systems is controlling when external inputs change relative to the clock.

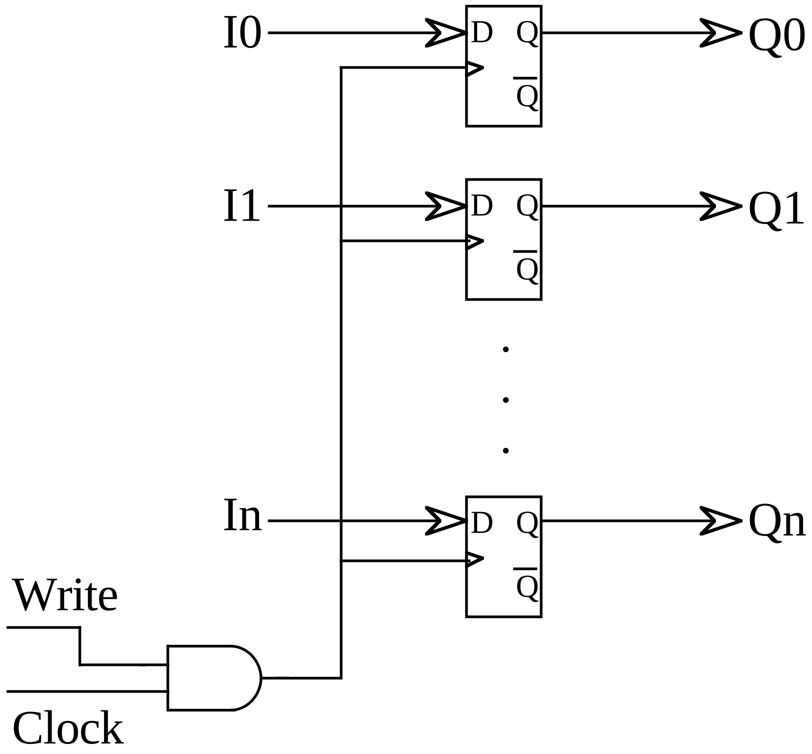
The reason why this clocking strategy is used is because it greatly simplifies the circuit timing requirements (and the subsequent analysis).

SEQUENTIAL CIRCUIT BUILDING BLOCKS

As was the case with combinational logic, system designers often specify systems in terms of common building blocks such as counters and registers.

REGISTERS

The function of a register is to hold data. Making one is easy - an n-bit register is composed of n D flip-flops.



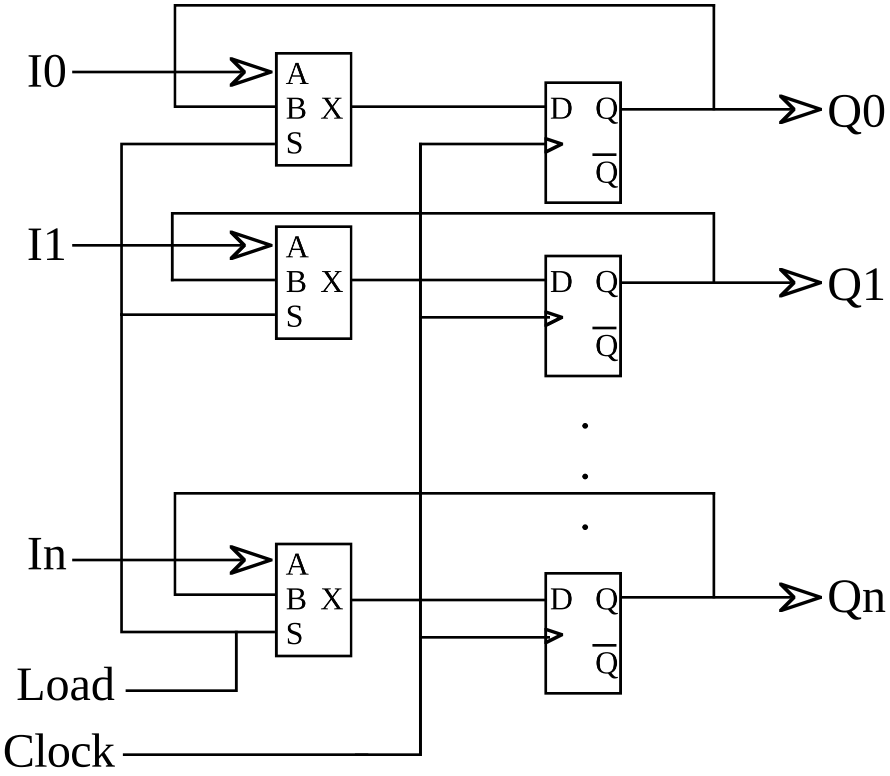
Unfortunately the simple scheme at left violates our timing rules!

The only way in which data can be selectively held or written is to gate the clock as shown.

We could use a more complex flip-flop that incorporates a memory state, but using an edge-triggered flip-flip is attractive because it can be made from only 6 NAND gates.

SEQUENTIAL CIRCUIT BUILDING BLOCKS cont.

An alternative strategy, is to fabricate registers using D flip-flops and multiplexers as shown below.



By using multiplexers as shown, the flip-flop inputs can select from either the current output, in which case the register will HOLD its current contents, or from the external inputs, in which case the register will LOAD new values.

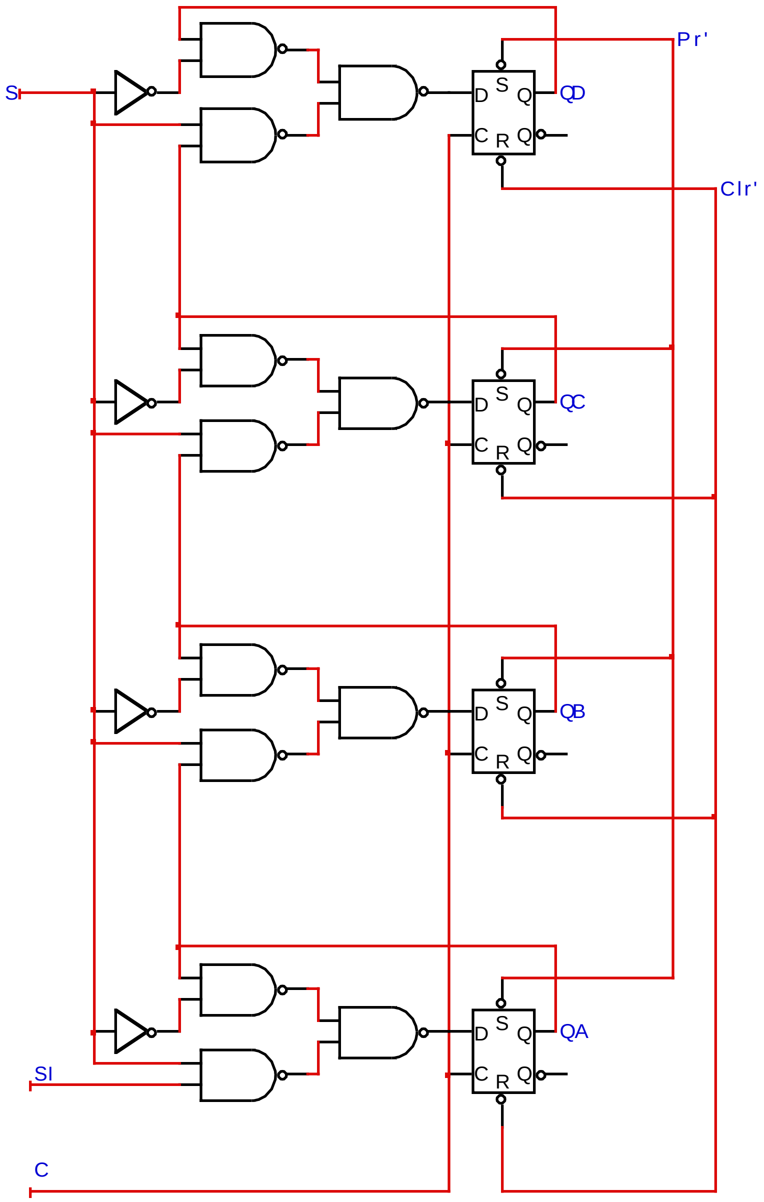
This scheme is sometimes referred to as a SYNCHRONOUS register.

In fact, one can generalize the scheme by using larger multiplexers to select from a number of different inputs.

SEQUENTIAL CIRCUIT BUILDING BLOCKS cont.

We can take the synchronous register idea further to construct SHIFT REGISTERS.

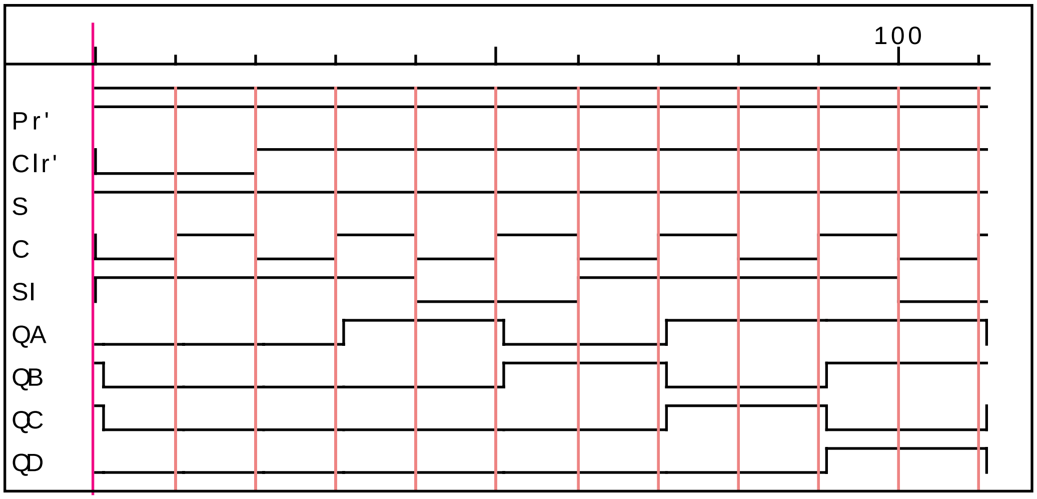
Consider the circuit below.



From the circuit diagram one can identify two modes of behaviour:

- S = 0: the contents of the register do not change since each output is re-circulated via the multiplexer.
- S = 1: Each flip flop gets its input from the one preceeding. QA gets its input from the SI input.

In other words, each bit is shifted to the RIGHT from QA to QD. The SI input feeds into the leftmost bit.



In the simulation, the pattern 1011 is SERIALLYshifted into the register.

After 4 clock pulses the pattern appears in PARALLEL at outputs QD to QA respectively.

SEQUENTIAL CIRCUIT BUILDING BLOCKS cont.

Shift registers are commonly used for parallel to serial and serial to parallel data conversion.

One can use a multiplexer with additional inputs to fabricate a UNIVERSAL shift register with 4 modes:

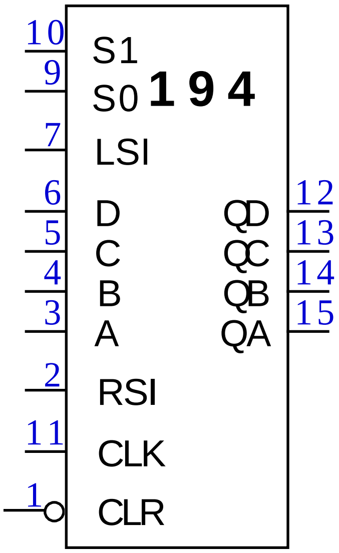
HOLD register contents preserved.

LOAD register contents replaced by data at external inputs.

LEFT data shifted from QA to QD with new data entering
SHIFT on the left at QA.

RIGHT data shifted from QD to QA with new data entering
SHIFT on the right at QD.

The resulting module might look something like the following:



	Inputs		Next State			
Function	S1	S0	QA~	QB~	QC~	QD~
HOLD	0	0	QA	QB	QC	QD
S. LEFT	0	1	RSI	QA	QB	QC
S. RIGHT	1	0	QB	QC	QD	LSI
LOAD	1	1	A	B	C	D

SEQUENTIAL CIRCUIT BUILDING BLOCKS cont.

COUNTERS

Consider the following binary count sequence:

Q3	Q2	Q1	Q0
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

Is there any pattern to when the outputs change state?

Q0: Always toggles

Q1: Toggles whenever Q0 = 1

Q2: Toggles whenever Q0 • Q1 = 1

Q3: Toggles whenever Q0 • Q1 • Q0 = 1

Recall the state transition table for a T (toggle) flip-flop:

T	Q(t)	Q(t+T)
0	0	0
0	1	1
1	0	1
1	1	0

The Q’s in our counter correspond to T flip-flops.

SEQUENTIAL CIRCUIT BUILDING BLOCKS cont.

Assume that Ti corresponds to the i'th bit of an n-bit counter.

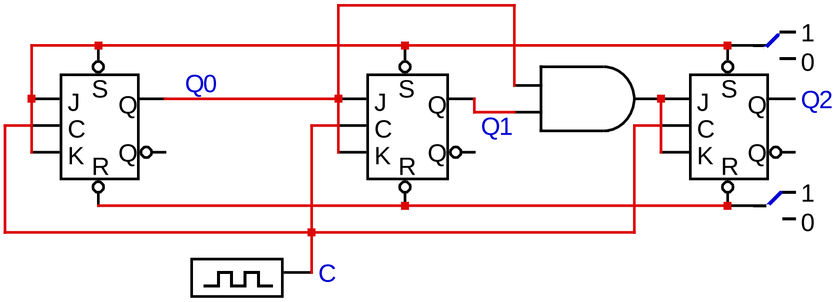
From observation of the counting sequence, we want Qi to toggle whenever Q0 • Q1 • ... • Q(i-1) = 1.

Thus

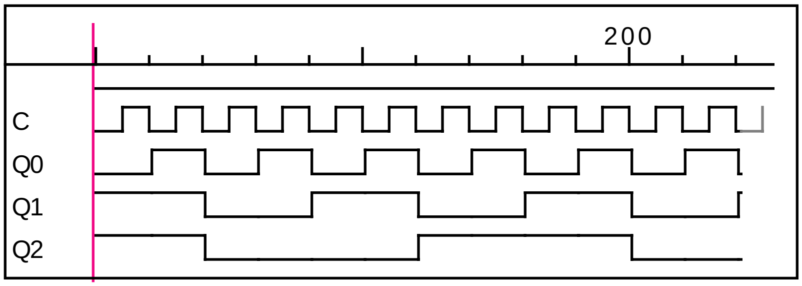
$$Ti = Q0 \bullet Q1 \bullet \ldots \bullet Q(i-1)$$

corresponds to the input equation for each of the n flip-flops.

Let's test the theory by constructing a circuit



and performing a simulation.



This is the expected result (notice that the J-K flip-flop is falling-edge triggered; also recall that T = J = K).

SEQUENTIAL CIRCUIT BUILDING BLOCKS cont.

COUNTING BACKWARDS

Q3	Q2	Q1	Q0
1	1	1	1
1	1	1	0
1	1	0	1
1	1	0	0
1	0	1	1
1	0	1	0
1	0	0	1
1	0	0	0
0	1	1	1
0	1	1	0
0	1	0	1
0	1	0	0
0	0	1	1
0	0	1	0
0	0	0	1
0	0	0	0

At left is the binary sequence corresponding to a 4-bit DOWN counter.

What is the rule for state changes?

Q0: Always toggles.

Q1: Toggles when Q0 = 0.

Q2: Toggles when Q0 = Q1 = 0.

Q3: Toggles when Q0 = Q1 = Q0 = 0.

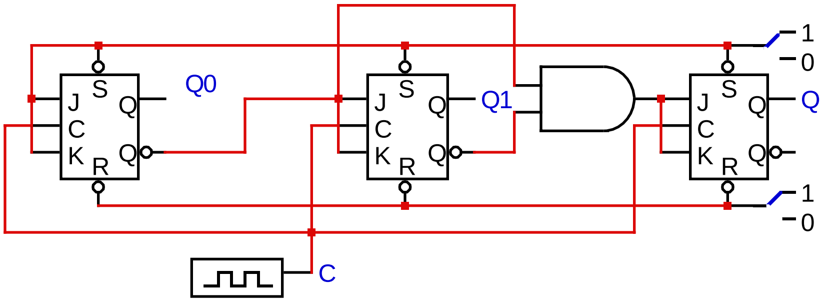
If Ti is the input to the i’th toggle flip-flop, then

$$Ti = Q0' \bullet Q1' \bullet ... \bullet Q(i-1)'$$

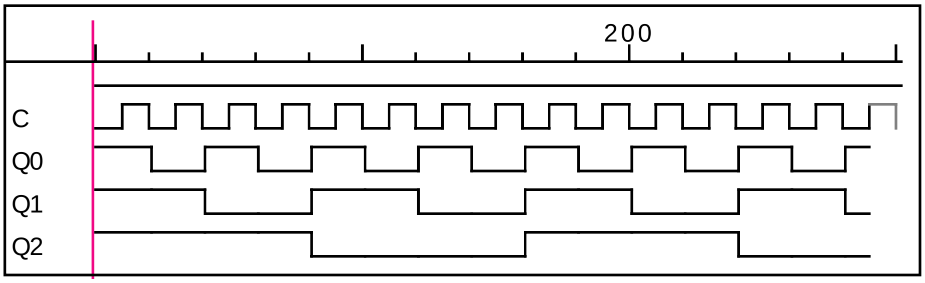
As before we can test the theory by building and simulating the corresponding circuit.

SEQUENTIAL CIRCUIT BUILDING BLOCKS cont.

3-Bit Down Counter



Counter Simulation



The simulation confirms what we expected.

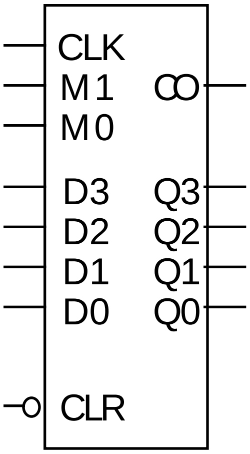
Transformation of the up counter to a down counter involved a relatively minor change.

This suggest the use of a multiplexer to build a counter that can selectively count up or down.

In fact we can push this idea further to construct a UNIVERSAL counter that can count up, count down, hold its current state, or load a new value from external inputs.

SEQUENTIAL CIRCUIT BUILDING BLOCKS cont.

UNIVERSAL COUNTER



We can put together the concepts encountered thus far to construct a universal counter with the following functions.

M1	M0	Function
0	0	HOLD current state
0	1	Count up
1	0	Count down
1	1	Q3 Q2 Q1 Q0 <-- D3 D2 D1 D0

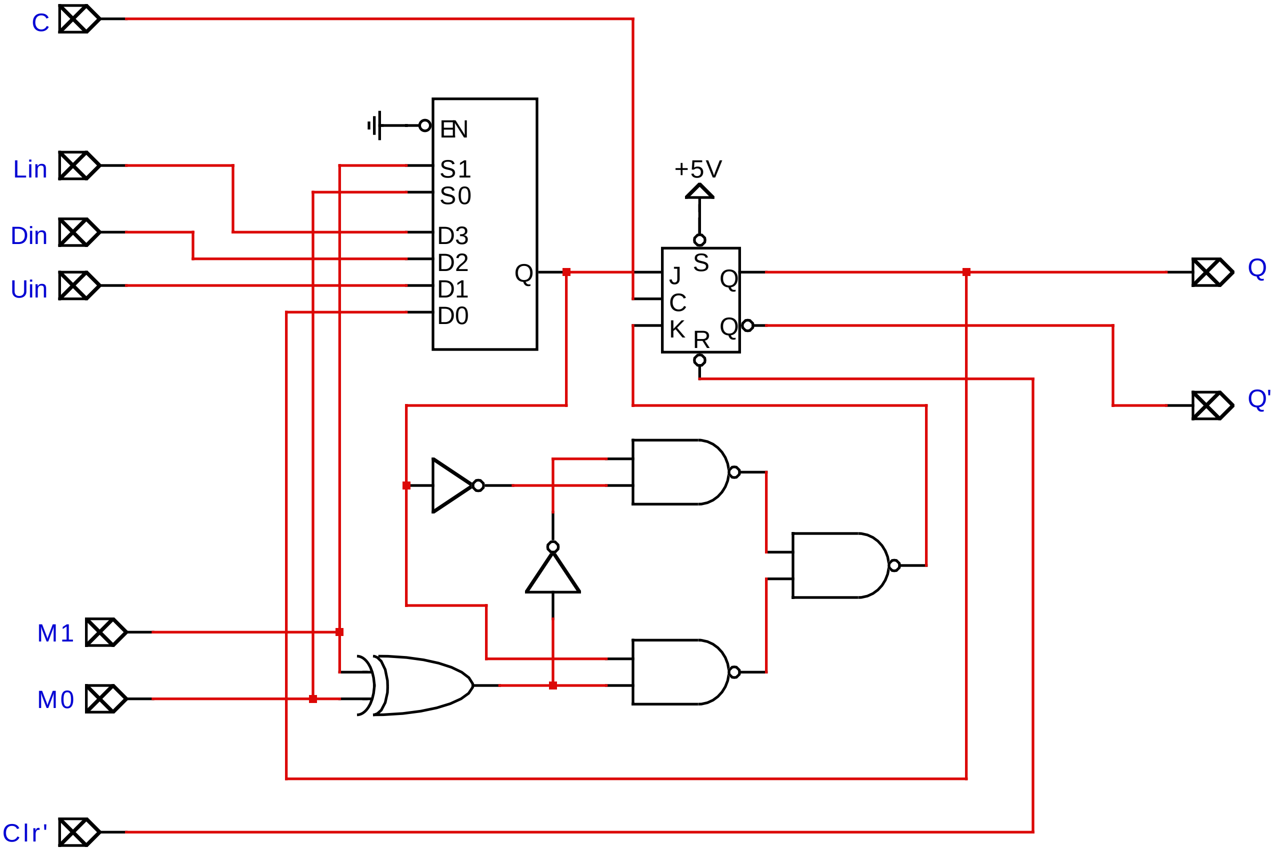
A Carry Out (CO) line signals when the count = 1 1 1 1.

Strategy:

- Design a counter cell that can support each of the required functions.
- Use multiplexers to select between inputs and toggle functions.
- Use multiplexers to convert a J-K alternately to a T and D flip-flop.
- Modularize the design to any counter length.

SEQUENTIAL CIRCUIT BUILDING BLOCKS cont.

UNIVERSAL COUNTER CELL



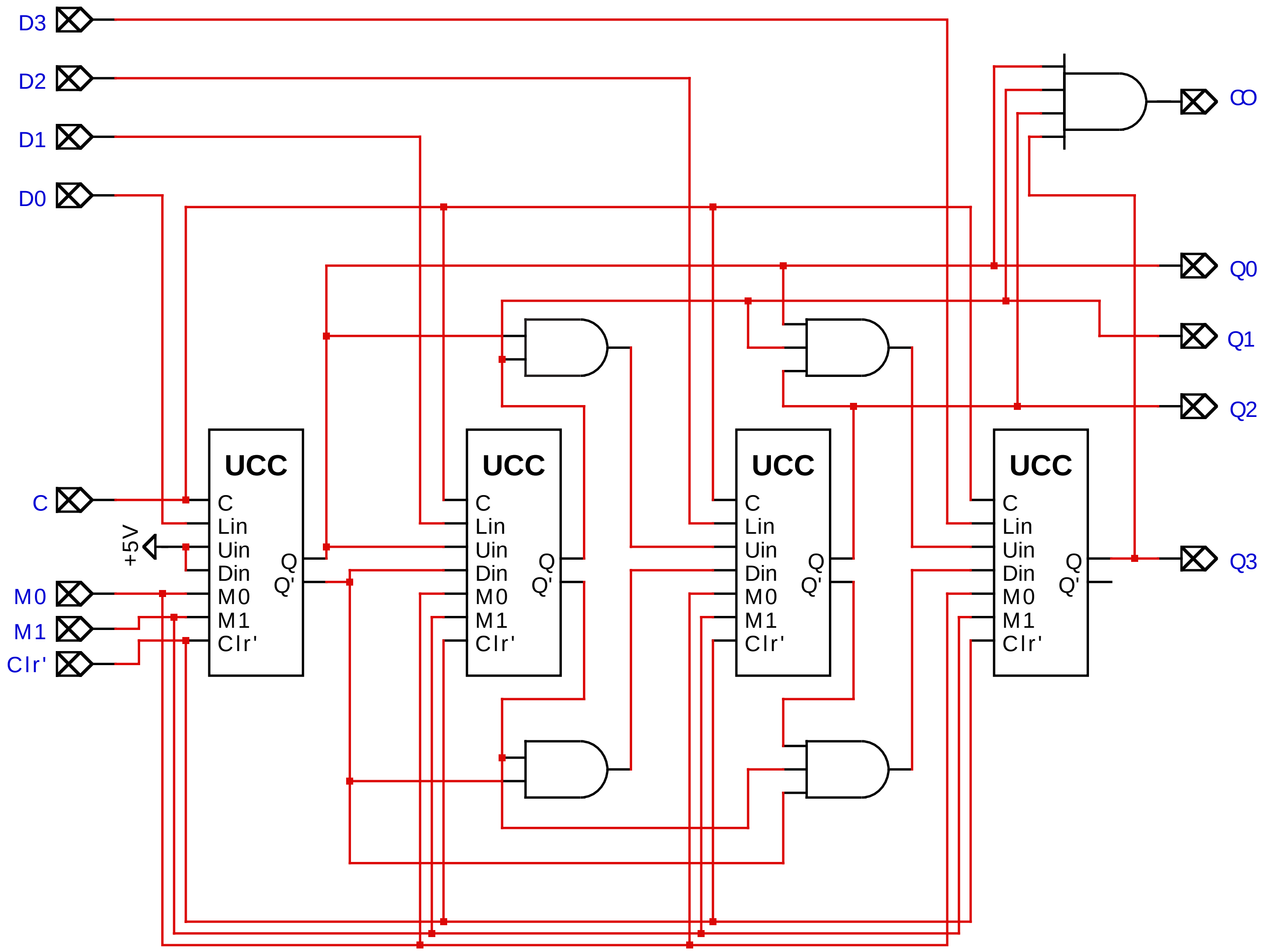
The design uses two multiplexers:

- The 4-input multiplexer at top is used to select among (00) re-circulating, (01) UP toggle function, (10) DOWN toggle function, and (11) LOAD input.

- The 2-input multiplexer at bottom converts the J-K flip-flop to a T for modes (01) and (10) and a D for modes (00) and (11)

SEQUENTIAL CIRCUIT BUILDING BLOCKS cont.

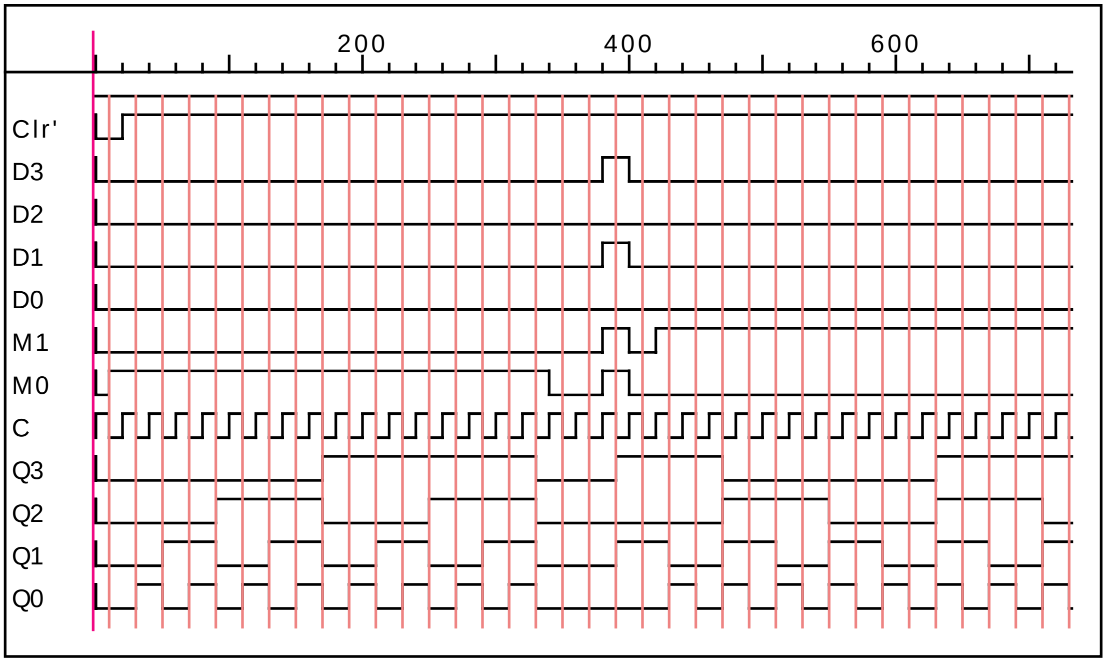
Use the UCC to build the Universal Counter as follows:



SEQUENTIAL CIRCUIT BUILDING BLOCKS cont.

UNIVERSAL COUNTER SIMULATION

The following waveforms illustrate the function of the Universal Counter in each of its 4 operating modes.



Note again that the J-K flip-flop in LogicWorks is falling-edge triggered.

The first part of the trace shows UP counting followed by a HOLD, followed by a LOAD (1010), followed by DOWN counting.

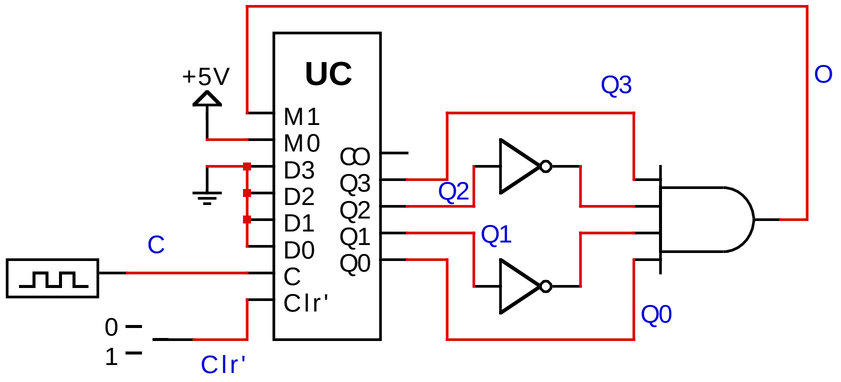
The simulation verifies that the UCC and counter function as expected.

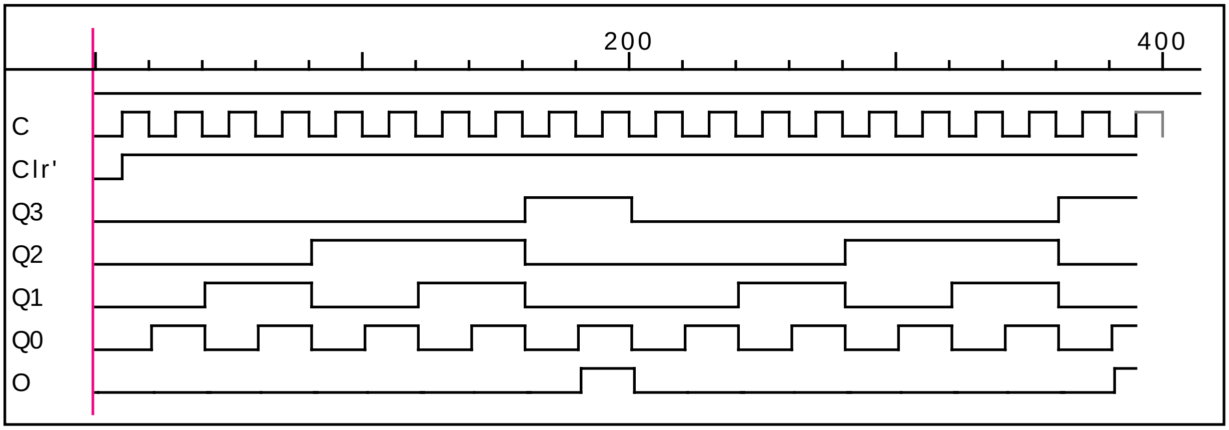
SEQUENTIAL CIRCUIT APPLICATIONS

PROGRAMMABLE COUNTER

The universal counter may be combined with a decoder to provide counters of variable length.

Example: Modulo 10 counter

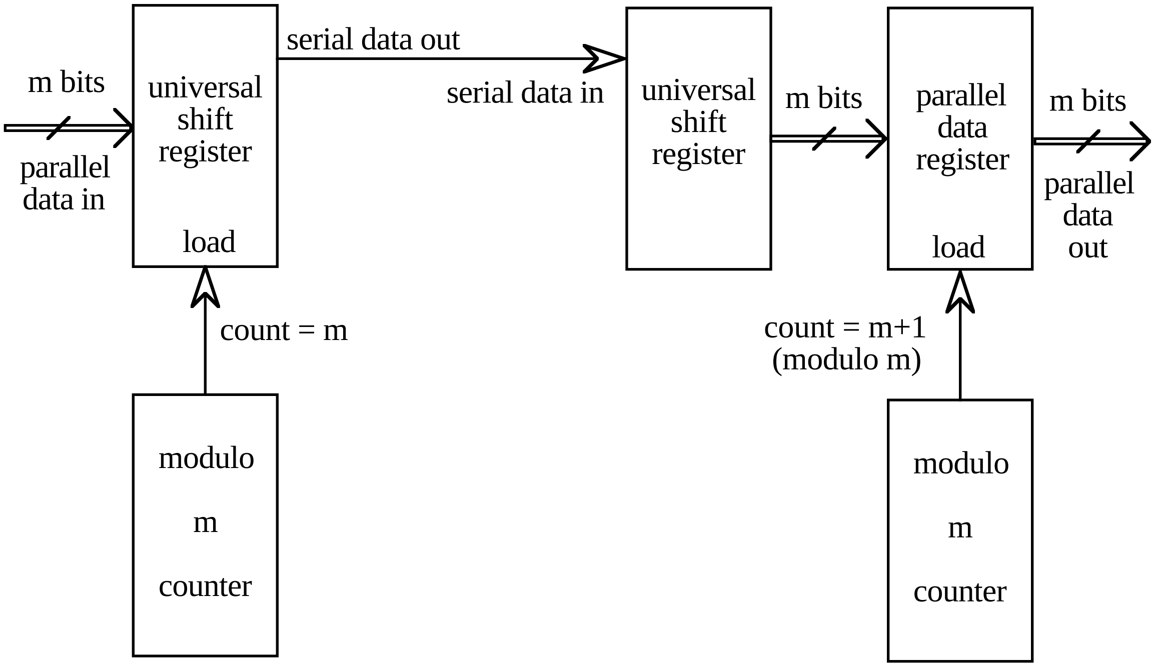




Note: Although it might be tempting to achieve the same effect with the Clr’ input, this input is ASYNCHRONOUS. Using it in this way would violate the timing model.

SEQUENTIAL CIRCUIT APPLICATIONS cont.

PARALLEL-SERIAL <--> SERIAL-PARALLEL



The schematic above shows how counters and shift registers can be combined to build a parallel-to-serial, serial-to-parallel data converter.

- Data enters on the left in parallel the transmit shift register where it is clocked out serially.

- The receive shift register on the right clocks this data in serially and presents it in parallel to a holding register.

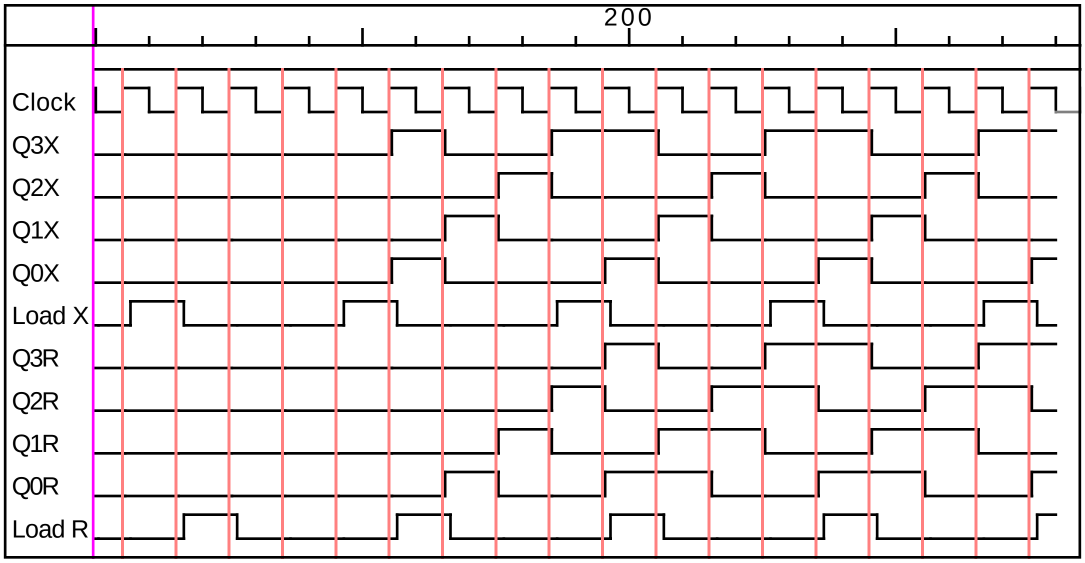
- Data is loaded on both sides every m clock pulses. The data rate from transmitter to receiver is 1/m times the input data rate. Modulo-m counters are used for this purpose.

SEQUENTIAL CIRCUIT APPLICATIONS cont.

PARALLEL-SERIAL <--> SERIAL-PARALLEL cont.

- Note that data entering the receive shift register from the transmit shift register will arrive one clock pulse later. This must be accounted for when decoding the modulo-m counters.

Timing



The above timing diagram corresponds to m = 4. Thus a new "packet" is transmitted (and received) every 4 clock pulses.

Note: Be careful with asynchronous inputs (e.g. Preset and Clear)! These may only be used for initialization purposes. Any other use requires a more sophisticated timing model.