CHAPTER TEN

Memory Cells

The previous chapters presented the concepts and tools behind processing binary data. This is only half of the battle though. For example, a logic circuit uses inputs to calculate an output, but where do these inputs come from? Some of them come from switches and other hardwired inputs, but many times a processor uses signals it has stored from previous operations. This might be as simple as adding a sequence of values one at a time to a running total: the running total must be stored somewhere so that it can be sent back to the inputs of the adder to be combined with the next value.

This chapter introduces us to memory by presenting the operation of a single memory cell, a device that is capable of storing a single bit.

10.1 New Truth Table Symbols

The truth tables that represent the operation of memory devices have to include a few new symbols in order to represent the functionality of the devices. For example, a memory cell is capable of storing a binary value, either a one or a zero. A new symbol, however, is needed to represent the stored value since its specific value is known only at the instant of operation.

10.1.1 Edges/Transitions

Many devices use as their input a change in a signal rather than a level of a signal. For example, when you press the "on" button to a computer, it isn't a binary one or zero from the switch that turns on the computer. If this was the case, as soon as you removed your finger, the machine would power off. Instead, the computer begins its power up sequence the instant your finger presses the button, i.e., the button transitions from an off state to an on state.

There are two truth table symbols that represent transitions from one logic value to another. The first represents a change in a binary signal from a zero to a one, i.e., a transition from a low to a high. This transition is called a *rising edge* and it is represented by the symbol ↑. The second symbol represents a transition from a one to a zero. This transition is called a *falling edge* and it is represented by the symbol ↓.

Figure 10-1 presents a binary signal with the points where transitions occur identified with these two new symbols.

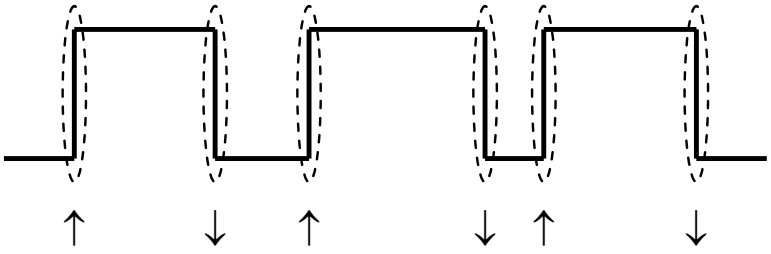


Figure 10-1 Symbols for Rising Edge and Falling Edge Transitions

10.1.2 Previously Stored Values

If a memory cell is powered, it contains a stored value. We don't know whether that value is a one or a zero, but there is something stored in that cell. If a logic circuit uses the stored value of that cell as an input, we need to have a way of labeling it so that it can be used within a boolean expression.

Just as A, B, C, and D are used to represent inputs and X is used to represent an output, a standard letter is used to represent stored values. That letter is ***Q***. To indicate that we are referring to the value of Q that was most recently stored, we use $\boldsymbol{Q_0}$.

For example, if a boolean expression is being used to represent what we are about to store in a memory cell, we would write the expression:

Q = value to be stored in memory cell

If an expression used the value previously stored in the memory cell as an input, our expression might look like:

X = expression using Q_0

10.1.3 Undefined Values

Some circuits have conditions that either are impossible to reach or should be avoided because they might produce unpredictable or even damaging results. In these cases, we wish to indicate that the signal is undefined. We do this with the letter ***U***.

For example, consider the binary circuit that operates the light inside a microwave oven. The inputs to this circuit are a switch to monitor whether the door has been opened and a signal to indicate whether the

magnetron is on or off. Note that the magnetron never turns on when the door is opened, so this circuit has an undefined condition.

Door	Magnetron	Light		D	M	L
Closed	Off	Off		0	0	0
Closed	On	On	⇒	0	1	1
Open	Off	On		1	0	1
Open	On	Shouldn't happen		1	1	U

Figure 10-2 Sample Truth Table Using Undefined Output

10.2 The S-R Latch

Computer memory is made from arrays of cells, each of which is capable of storing a single bit, a one or a zero. The digital circuitry that is used to route bits to and from memory is similar to much of that already presented here. Nothing has been said, however, about how the bit is stored.

The goal is to send a logic one or a logic zero to a device, then after leaving it unattended for a period of time, come back and find the value still there. A simple wire alone cannot do this. A digital value placed on a wire will, after the value is removed, quickly loose the charge and become unreadable or erroneous.

Early memory stored data in small doughnut shaped rings of iron. Wires that were woven through the centers of the iron rings were capable of magnetizing the rings in one of two directions. If each ring was allowed to represent a bit, then one direction of magnetization would represent a one while the other represented a zero. As long as nothing disturbed the magnetization, the value of the stored bit could be detected later using the same wires that stored the original value.

With the advent of digital circuitry, the use of iron (magnetic material) was no longer necessary. A circuit could be developed where the output could be routed back around to the circuit's inputs. This "ring" provides feedback which allows the circuit's current data to affect the circuit's future data and thus maintain its condition. The circuit in Figure 10-3 is a simple example of this.

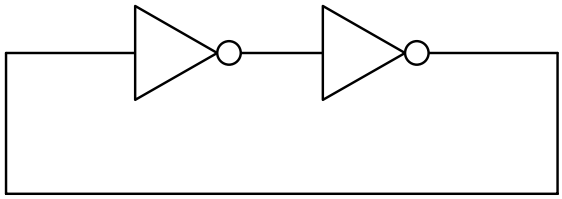


Figure 10-3 Primitive Feedback Circuit using Inverters

The output of the first inverter in the circuit of Figure 10-3 is fed into the input of a second inverter. Since the inverse of an inverse is the original value, the input to the first inverter is equal to the output of the second inverter. If we connect the output of the second inverter to the input of the first inverter, then the logic value will be maintained until power to the circuit is removed.

In order to take the next step, sometimes it is helpful to redraw the circuit of Figure 10-3 so that the inverters are side-by-side.

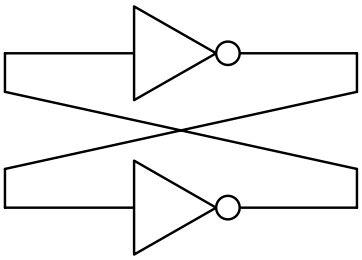
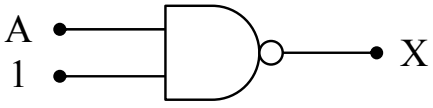


Figure 10-4 Primitive Feedback Circuit Redrawn

The problem with the circuit of Figure 10-4 is that there is no way to modify the value that is stored. We need to replace either one or both of the inverters with a device that has more than one input, but one that can also operate the same way as the inverter during periods when we want the data to be stable. It turns out that the NAND gate can do this. Figure 10-5 presents the truth table for the NAND gate where one of the inputs is always connected to a one.



A	1	X
0	1	1
1	1	0

Figure 10-5 Operation of a NAND Gate with One Input Tied High

Notice that the output X is always the inverse of the input A. The NAND gate operates just like an inverter with a second input. Figure 10-6 replaces the inverters of Figure 10-4 with NAND gates.

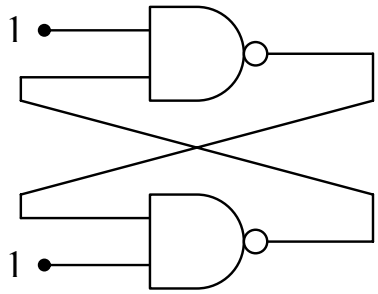


Figure 10-6 Primitive Feedback Circuit Redrawn with NAND Gates

As long as the free inputs to the two NAND gates remain equal to one, the circuit will remain stable since it is acting as a pair of inverters connected together in series. It is also important to note that if the top inverter outputs a zero, the bottom inverter outputs a one. Likewise, if a one is output from the top inverter, then a zero is output from the bottom one. These two possible states are shown in Figure 10-7.

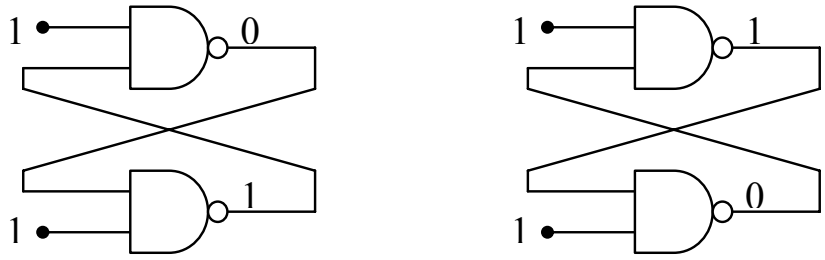
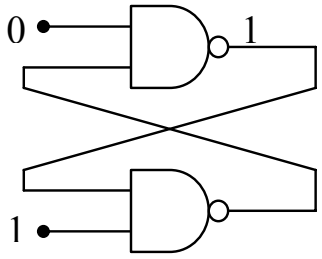


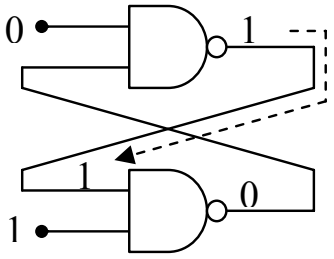
Figure 10-7 Only Two Possible States of Circuit in Figure 10-6

What happens if we change the free input of either NAND gate? Remember that if either input to a NAND gate is a zero, then the output is forced to be a 1 regardless of the other input. That means that if a zero is placed on the free input of the top NAND gate, then the output of that NAND gate is forced to one. That one is routed back to the input of the lower NAND gate where the other input is one. A NAND gate with all of its inputs set to one has an output of zero. That zero is routed back to the input of the top NAND gate whose other input is a zero. A NAND gate with all of its inputs set to zero has an output of one. This means that the system has achieved a stable state.

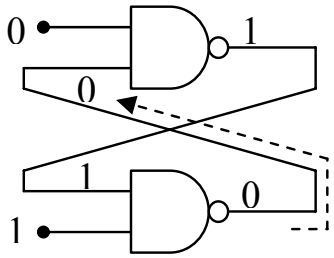
If the free input of the top NAND gate returns to a one, the zero input to it from the lower NAND gate makes it so that there is no change to the one currently being output from it. In other words, returning the free input of the top NAND gate back to a one does not change the output of the circuit. These steps are represented graphically in the circuit diagrams of Figure 10-8.



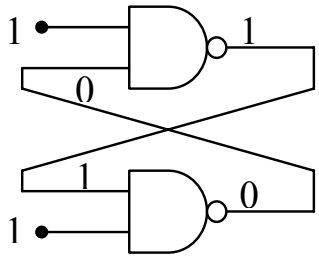
a.) A zero to the free input of the top NAND gate forces a one to its output



b.) That one passes to the bottom NAND which in turn outputs a zero



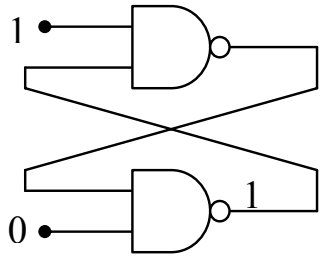
c.) A zero from the bottom NAND returns to the lower input of the top NAND



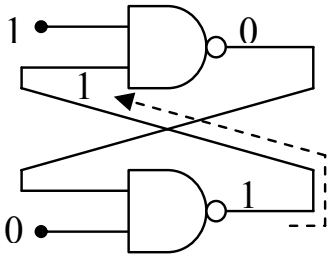
d.) The second zero at the top NAND holds its output even if the free input returns to 1

Figure 10-8 Operation of a Simple Memory Cell

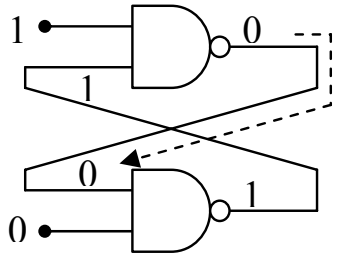
This means that the circuit can be used to store a one in the top NAND gate and a zero in the bottom NAND gate by toggling the free input on the top NAND gate from a one to a zero and back to a one. Figure 10-9 shows what happens when we toggle the free input on the bottom NAND gate from a one to a zero and back to a one.



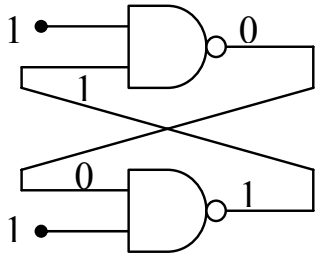
a.) A zero to the free input of the bottom NAND gate forces a one to its output



b.) That one passes to the top NAND which in turn outputs a zero



c.) A zero from the top NAND returns to the lower input of the bottom NAND



d.) The second zero at the bottom NAND holds its output even if the free input returns to 1

Figure 10-9 Operation of a Simple Memory Cell (continued)

This configuration of two NAND gates represents the basic circuit used to store a single bit of data using logic gates. Notice that in step d of both Figure 10-8 and Figure 10-9, the circuit is stable with the opposing NAND gates outputting values that are inverses of each other. In addition, notice that the circuit's output is changed by placing a zero on the free input of one or the other NAND gates.

Figure 10-10 presents the standard form of this circuit with the inputs labeled $\overline{S}$ and $\overline{R}$ and the outputs labeled Q and $\overline{Q}$. The bars placed above the S and R inputs indicate that they are active low inputs while the bar above the lower output of Q indicates that it is an inverted value of the Q output.

This circuit is referred to as the *S-R latch*. The output Q is *set* to a one if the $\overline{S}$ input goes low while $\overline{R}$ stays high. The output Q is *reset* to a zero if the R input goes low while S stays high. If both of these inputs are high, i.e., logic one, then the circuit maintains the current value of Q. The truth table for the S-R latch is shown in Figure 10-11.

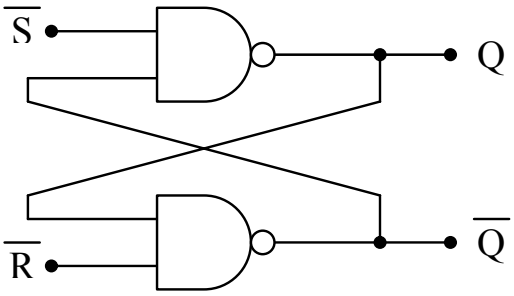


Figure 10-10 S-R Latch Capable of Storing a Single Bit

$\overline{S}$	$\overline{R}$	Q	$\overline{Q}$
0	0	U	U
0	1	1	0
1	0	0	1
1	1	Q_0	$\overline{Q_0}$

Figure 10-11 S-R Latch Truth Table

Notice that the row of the truth table where both inputs equal zero produces an undefined output. Actually, the output is defined: both Q and its inverse are equal to one. What makes this case undefined is that when both of the inputs return to one, the output of the system becomes unpredictable, and possibly unstable. It is for this reason that the top row of this truth table is considered illegal and is to be avoided for any implementation of the S-R latch circuit.

10.3 The D Latch

The S-R latch is presented in this chapter to show how latches store data. In general, every memory device that employs logic gates has embedded in it an S-R latch. For the rest of this book, we will be treating latches as "black boxes" addressing only the controls and how they affect Q and its inverse.

The typical data storage latch is referred to as a data latch or a ***D latch***. There are slight variations between different implementations of the D latch, but in general, every D latch uses the same basic inputs and outputs. Figure 10-12 presents the block diagram for the fully implemented D-latch.

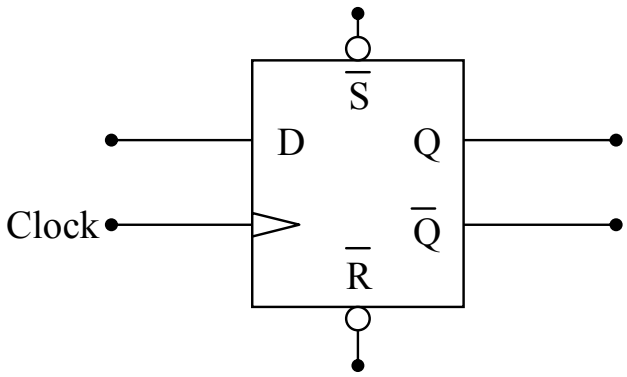


Figure 10-12 Block Diagram of the D Latch

The outputs, Q and $\overline{Q}$, operate just as they did for the S-R latch outputting the stored data and its inverse. The active low inputs, $\overline{S}$ and $\overline{R}$, also operate the same as they did for the S-R latch. If $\overline{S}$ goes low while $\overline{R}$ is held high, the output Q is set to a one. If $\overline{R}$ goes low while $\overline{S}$ is held high, the output Q is reset to zero. If both S and R are high, then Q maintains the data bit it was last storing. The output of the circuit is undefined for both S and R low.

The two new inputs, D and Clock, allow the circuit to specify the data being stored in Q. D, sometimes called the data input, is the binary value that we wish to store in Q. Clock tells the circuit when the data is to be stored.

This circuit acts much like a camera. Just because a camera is pointing at something does not mean that it is storing that image. The only way that the image is going to be stored is if someone presses the button activating the shutter. Once the shutter is opened, the image is captured to film or to digital media.

The Clock input acts like the button on the camera. A specific transition or level of the binary value at the Clock input captures the binary value present at the D input and stores it to Q.

There is another characteristic of taking a picture with a camera that has an analogy with the storage of data in a D latch. Assume for a moment that we are talking about cameras that use film instead of digital media to store an image. If the camera's shutter is opened thus exposing the film to light for the entire time the user's finger was pressing the button, then every picture would be over exposed. The shutter should just open for an instant, typically, the instant that the user's finger is coming down on the button.

Alternatively, if the shutter was activated when the user's finger came up instead of down on the button, a number of good shots would be missed and the user might become quite frustrated. It is important to define specifically when the image is captured with regards to the button.

Different implementations of the D latch use different definitions of when the data is captured with respect to Clock. Some operate like cameras do only capturing data when the Clock signal transitions, either rising edge or falling edge. These D latches are referred to as ***edge-triggered latches***. The instant the D latch detects the appropriate transition, the binary value that is present at the input D is copied to Q. The data will remain at Q until the next appropriate transition. The truth tables in Figure 10-13 show how the inputs Clock and D of both the rising and falling edge-triggered latches affect the data stored at Q.

D	Clock	Q	$\overline{Q}$
X	0	Q_0	$\overline{Q_0}$
X	1	Q_0	$\overline{Q_0}$
X	↓	Q_0	$\overline{Q_0}$
0	↑	0	1
1	↑	1	0

a.) Rising Edge

D	Clock	Q	$\overline{Q}$
X	0	Q_0	$\overline{Q_0}$
X	1	Q_0	$\overline{Q_0}$
X	↑	Q_0	$\overline{Q_0}$
0	↓	0	1
1	↓	1	0

b.) Falling Edge

Figure 10-13 Edge-Triggered D Latch Truth Tables

Notice that the value on D does not affect the output if the Clock input is stable, nor does it have an effect during the clock transition other than the one for which it was defined. During these periods, the values stored at the latch's outputs remain set to the values stored there from a previous data capture.

D latches can also be designed to capture data during a specified level on the Clock signal rather than a transition. These are called ***transparent latches***. They latch data much like an edge triggered latch, but while Clock is at the logic level previous to the transition, they pass all data directly from the D input to the Q output. For example, when a zero is input to the Clock input of a D latch designed to capture data when Clock equals zero, the latch appears to vanish, passing the signal

D straight to Q. The last value present on D when the Clock switches from zero to one is stored on the output until Clock goes back to zero. Figure 10-14 presents this behavior using truth tables for both the active low and active high transparent D latches.

D	Clock	Q	$\overline{Q}$
X	1	Q_0	$\overline{Q_0}$
0	0	0	1
1	0	1	0

a.) Active Low

D	Clock	Q	$\overline{Q}$
X	0	Q_0	$\overline{Q_0}$
0	1	0	1
1	1	1	0

b.) Active High

Figure 10-14 Transparent D Latch Truth Tables

A transparent D latch acts like a door. If Clock is at the level that captures data to the output, i.e., the door is open, any signal changes on the D input pass through to the output. Once Clock goes to the opposite level, the last value in Q is maintained in Q.

The rest of this chapter presents some applications of latches including processor support circuitry, I/O circuits, and memory.

10.4 Divide-By-Two Circuit

In some cases, the frequency of a clock input on a circuit board is too fast for some of the computer's peripherals. An edge-triggered D latch can be used to divide a clock frequency in half. The circuit presented in Figure 10-15 does this.

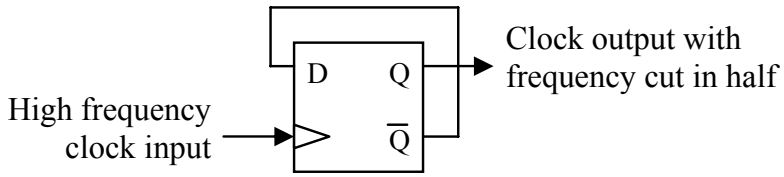


Figure 10-15 Divide-By-Two Circuit

Assume, for example, that we are using a rising edge-triggered latch for this circuit. By connecting the inverse of the Q output to the D input, the output Q is inverted or toggled every time the clock goes

from a zero to a one. Since there is only one rising edge for a full cycle of a periodic signal, it takes two cycles to make the output Q go through a full cycle. This means that the frequency of the output Q is one half that of the input frequency at the clock input. Figure 10-16 presents a timing diagram of this behavior.

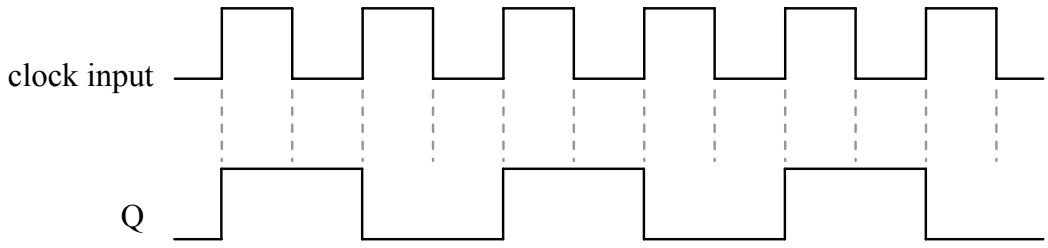


Figure 10-16 Clock and Output Timing in a Divide-By-Two Circuit

By cascading multiple divide-by-two circuits, we get divisions of the original frequency by 2, 4, 8, … , 2^n.

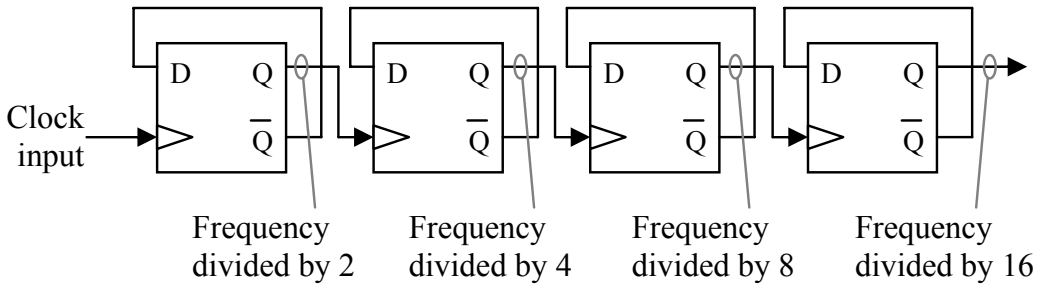


Figure 10-17 Cascading Four Divide-By-Two Circuits

10.5 Counter

By making a slight modification to the cascaded divide-by-two circuits of Figure 10-17, we can create a circuit with a new purpose. Figure 10-18 shows the modified circuit created by using the inverted outputs of the latches to drive the Clock inputs of the subsequent latches instead of using the Q outputs to drive them.

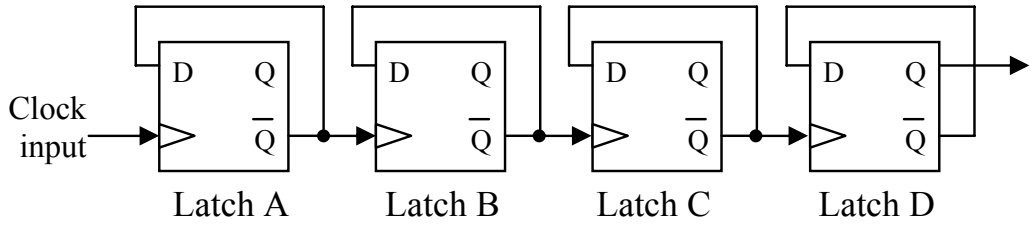


Figure 10-18 Cascading Four Divide-By-Two Circuits

If we draw the outputs of all four latches with respect to each other for this new circuit, we see that the resulting ones and zeros from their outputs have a familiar pattern to them, specifically, they are counting in binary.

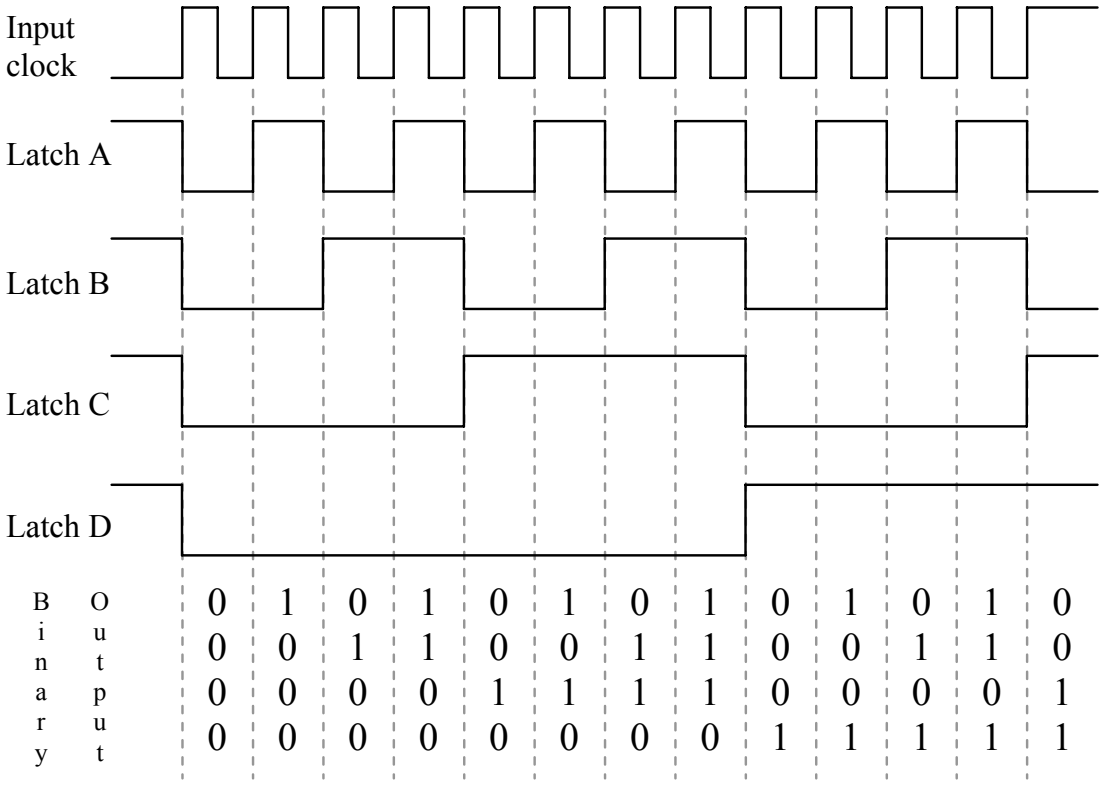


Figure 10-19 Output of Binary Counter Circuit

If the leftmost latch is considered the LSB of a four-bit binary number and the rightmost latch is considered the MSB, then a cycle on the input clock of the leftmost latch will increment the binary number by one. This means that by connecting the inverted output of a divide-by-two circuit to the clock input of a subsequent divide-by-two circuit n times, we can create an n-bit binary counter that counts the pulses on an incoming frequency.

10.6 Parallel Data Output

Not all binary values stay inside the processor. Sometimes, external circuitry needs to have data sent to it. For example, before the advent of the USB port, computers used a 25-pin connector to transmit data. It was called the *parallel port*, and it was used to send and receive eight bits at a time to a device such as a printer or a storage device.

The processor needed to be able to place data on the eight data bits of this port, then latch it so that the data would remain stable while the processor performed another task. The device connected to the other end of the port could then access the data, and when it was done, alert the processor that it needed additional data. The processor would then latch another byte to the data lines for the external device.

The typical circuit used for the data lines of this port was the D latch. By placing an active-low transparent latch at each output bit, the processor could use the Clock to store data in each latch. This arrangement was so common that IC manufacturers made a single chip that contained all of the latches necessary for the circuit. Figure 10-20 presents that circuit.

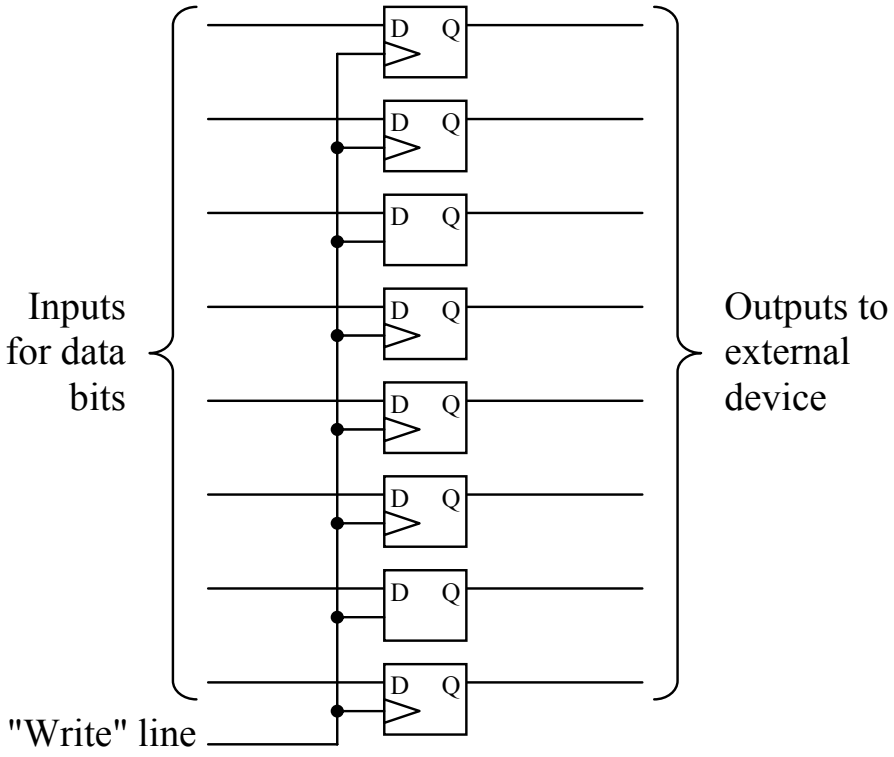


Figure 10-20 Output Port Data Latch Circuitry

By connecting all of the Clocks to a single "write" input, then the processor only needed to place the appropriate data on the data lines and toggle the write line low then high. This would latch the data onto the Q lines of the eight latches where it would remain until the processor placed new data on the data lines and toggled the write line again.

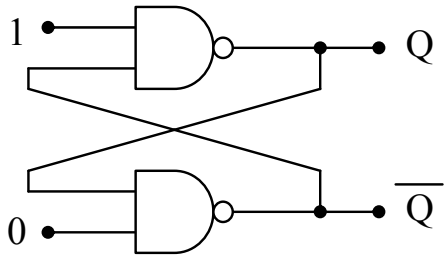
Memory based on logic gates works the same way. To store data, the processor places the data it wants to store onto data lines, then pulses a write signal low then high. This stores the data into latches within the memory circuit.

10.7 What's Next?

The next chapter introduces state machines, the tools that are used to design them, and the circuits that are used to implement them. A state machine is any system that is designed to remember its condition. For example, for a traffic signal to turn yellow, it has to know that it was green. For an elevator to know that it should go down to get to the fifth floor must first know that it is currently on the eighth floor. State machines require memory, and therefore, many of the implementations of state machines use latches.

Problems

1. For the circuit below, what value does Q have?



2. Describe why the S-R latch has an illegal condition.

3. Describe the purpose of each of the following truth table symbols.

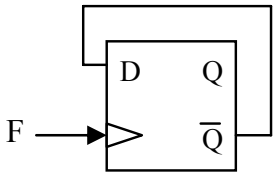
   a.) ↓ b.) ↑ c.) U d.) Q_0

4. If a D latch has the inputs $\overline{S} = 0$, $\overline{R} = 1$, D = 1, and Clock = 0, what is the output Q?

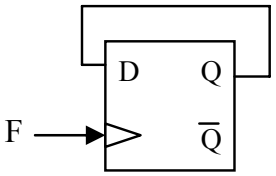
5. True or false: A D latch with only two inputs D and CLK has no illegal states.

6. Which of the following circuits is used to divide the frequency of the signal F in half?

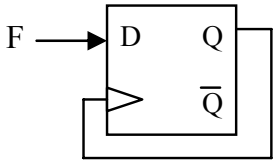
a.)



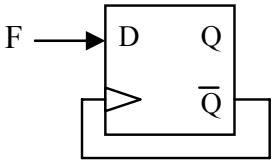
b.)



c.)



d.)



7. Show the D latch output waveform Q based on the inputs R, D, and Clock indicated in the figure below. Assume the latch captures on the rising edge.