10 REGISTER CIRCUITS

10.1 INTRODUCTION

What to Expect

Flip-flops are the smallest possible memory available for a digital device. A flip-flop can store a single bit of data and "remembers" if that bit is on or off. Flip-flops are combined in groups of eight or more to create a register, which is a memory device for a byte or word of data in a system. The following topics are included in this chapter.

- Analyzing a sequential circuit using a timing diagram
- Developing and using an SR latch
- Developing and using D, JK, Toggle, and Master-Slave flip-flops
- Designing memory components with registers
- Converting data between serial and parallel formats using registers

10.2 TIMING DIAGRAMS

Unlike combinational logic circuits, timing is essential in sequential circuits. Normally, a device will include a *clock* IC that generates a square wave that is used to control the sequencing of various activities throughout the device. In order to design and troubleshoot these types of circuits, a timing diagram is developed that shows the relationship between the various input, intermediate, and output signals. Figure 10.1 is an example timing diagram.

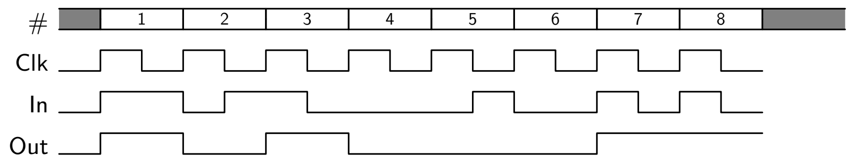

Figure 10.1: Example Timing Diagram

Figure 10.1 is a timing diagram for a device that has three signals, a clock, an Input, and an Output.

Signals in a timing diagram can be described as low//high, 0//1, on//off, or any other number of ways.

- #: The first line of the timing diagram is only a counter that indicates the number of times the system clock has cycled from Low to High and back again. For example, the first time the clock changes from Low to High is the beginning of the first cycle. This counter is only used to facilitate a discussion about the circuit timing.

- Clk: A clock signal regularly varies between Low and High in a predictable pattern, normally such that the amount of Low time is the same as the High time. In a circuit, a clock pulse is frequently generated by applying voltage to a crystal since the oscillation cycle of a crystal is well known and extremely stable. In Figure 10.1 the clock cycle is said have a 50% duty cycle, that is, half-low and half-high. The exact length of a single cycle would be measured in micro- or nano-seconds but for the purposes of this book all that matters is the relationship between the various signals, not the specific length of a clock cycle.

- In: The input is Low until Cycle #1, then goes High for one cycle. It then toggles between High and Low at irregular intervals.

- Out: The output goes High at the beginning of cycle #1. It then follows the input, but only at the start of a clock cycle. Notice that the input goes high halfway through cycle #2 but the output does not go high until the start of cycle #3. Most devices are manufactured to be *edge triggered* and will only change their output at either the positive or negative edge of the clock cycle (this example is positive edge triggered). Thus, notice that no matter when the input toggles the output only changes when the clock transitions from Low to High.

Given the timing diagram in Figure 10.1 it would be relatively easy to build a circuit to match the timing requirements. It would have a single input and output port and the output would match the input on the positive edge of a clock cycle.

Whenever the input for any device changes it takes a tiny, but measurable, amount of time for the output to change since the various transistors, capacitors, and other electronic elements must reach saturation and begin to conduct current. This lag in response is known as *propagation delay* and that is important for an engineer to consider when building a circuit. It is possible to have a circuit that is so complex that the output has still not changed from a previous input signal before a new input signal arrives and starts the process over. In Figure 10.2 notice that the output goes High when the input goes Low, but only after a tiny propagation delay.

Ideal circuits with no propagation delay are assumed in this book in order to focus on digital logic rather than engineering.

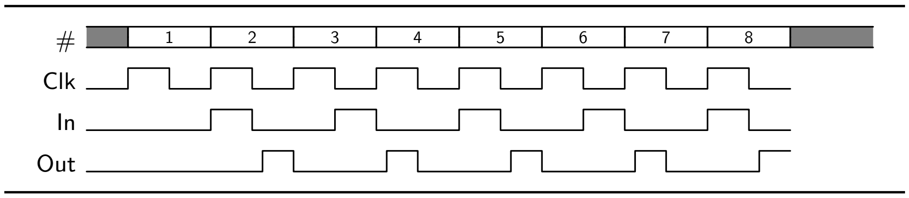

Figure 10.2: Example Propagation Delay

It is also true that a square wave is not exactly square. Due to the presence of capacitors and inductors in circuits, a square wave will actually build up and decay over time rather than instantly change. Figure 10.3 shows a typical charge/discharge cycle for a capacitance circuit and the resulting deformation of a square wave. It should be kept in mind, though, that the times involved for capacitance charge/discharge are quite small (measured in nanoseconds or smaller); so for all but the most sensitive of applications, square waves are assumed to be truly square.

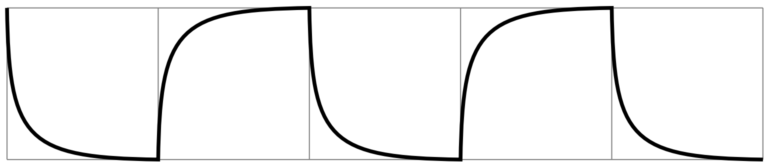

Figure 10.3: Capacitor Charge and Discharge

All devices are manufactured with certain tolerance levels built-in such that when the voltage gets "close" to the required level then the device will react, thus mitigating the effect of the deformed square wave. Also, for applications where speed is critical, such as space exploration or medical, high-speed devices are available that both sharpen the edges of the square wave and reduce propagation delay significantly.

10.3 flip-flops

10.3.1 *Introduction*

Flip-flops are digital circuits that can maintain an electronic state even after the initial signal is removed. They are, then, the simplest of memory devices. Flip-flops are often called *latches* since they are able to "latch" and hold some electronic state. In general, devices are called flip-flops when sequential logic is used and the output only changes on a clock pulse and they are called latches when combinational logic is used and the output constantly reacts to the input. Commonly, flip-flops are used for clocked devices, like counters, while latches are used for storage. However, these terms are often considered synonymous and are used interchangeably.

10.3.2 *SR Latch*

This latch is often also called an RS Latch.

One of the simplest of the flip-flops is the SR (for *Set-Reset*) latch. Figure 10.4 illustrates a logic diagram for an *SR latch* built with NAND gates.

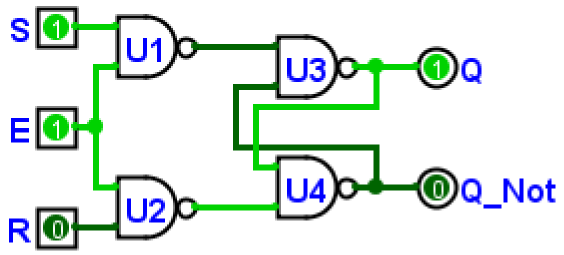

Figure 10.4: SR Latch Using NAND Gates

Table 10.2 is the truth table for an *SR Latch*. Notice that unlike truth tables used earlier in this book, some of the outputs are listed as "Last Q" since they do not change from the previous state.

Inputs			Outputs	
E	S	R	Q	Q'
0	0	0	Last Q	Last Q'
0	0	1	Last Q	Last Q'
0	1	0	Last Q	Last Q'
0	1	1	Last Q	Last Q'
1	0	0	Last Q	Last Q'
1	0	1	0	1
1	1	0	1	0
1	1	1	Not Allowed	Not Allowed

Table 10.1: Truth Table for SR Latch

In Figure 10.4, input E is an enable and must be high for the latch to work; when it is low then the output state remains constant regardless of how inputs S or R change. When the latch is enabled, if $S = R = 0$ then the values of Q and Q' remain fixed at their last value; or, the circuit is "remembering" how they were previously set. When input S goes high, then Q goes high (the latch's output, Q, is "set"). When input R goes high, then Q' goes high (the latch's output, Q, is "reset"). Finally, it is important that in this latch inputs R and S cannot both be high when the latch is enabled or the circuit becomes unstable and output Q will oscillate between high and low as fast as the NAND gates can change; thus, input 111_2 must be avoided. Normally, there is an additional bit of circuit prior to the inputs to ensure *S* and *R* will always be different (an XOR gate could do that job).

An SR Latch may or may not include a clock input; if not, then the outputs change immediately when any of the inputs change, like in a combinational circuit. If the designer needs a clocked type of memory device, which is routine, then the typical choice would be a *JK Flip-Flop*, covered later. Figure 10.5 is a timing diagram for the SR Latch in Listing 10.4.

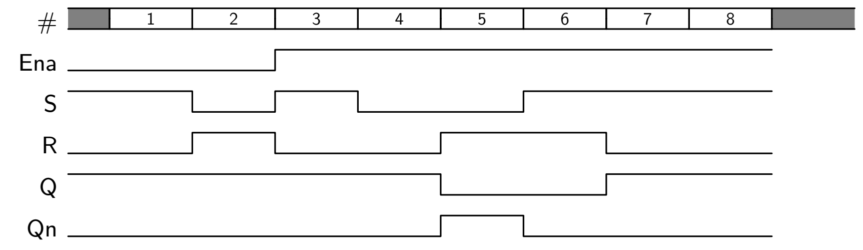

Figure 10.5: SR Latch Timing Diagram

At the start of Figure 10.5 *Enable* is low and there is no change in Q and Q′ until frame 3, when *Enable* goes high. At that point, S is high and R is low so Q is high and Q′ is low. At frame 4 both S and R are low so there is no change in Q or Q′. At frame 5 R goes high so Q goes low and Q′ goes high. In frame 6 both S and R are high so both Q and Q′ go low. Finally, in frame 7 S stays high and R goes low so Q goes high and Q′ stays low.

Logisim-evolution includes an *S-R Flip-Flop* device, as illustrated in Figure 10.6.

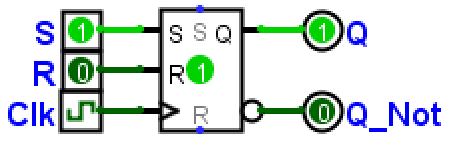

Figure 10.6: SR Latch

The *S-R Flip-Flop* has *S* and *R* input ports and *Q* and *Q′* output ports. Also notice that there is no *Enable* input but there is a *Clock* input. Because this is a clocked device the output would only change on the edge of a clock pulse and that makes the device a flip-flop rather than a latch. As shown, *S* is high and the clock has pulsed so *Q* is high, or the flip-flop is "set." Also notice that on the top and bottom of the device there is an *S* and *R* input port that are not connected. These are "preset" inputs that let the designer hard-set output *Q* at either one or zero, which is useful during a power-up routine. Since this device has no *Enable* input it is possible to use the *R* preset port as a type of

On a logic diagram a clock input is indicated with a triangle symbol.

enable. If a high is present on the *R* preset then output *Q* will go low and stay there until the *R* preset returns to a low state.

10.3.3 *Data (D) Flip-Flop*

A Data Flip-Flop (or *D Flip-Flop*) is formed when the inputs for an *SR Flip-Flop* are tied together through an inverter (which also means that *S* and *R* cannot be high at the same time, which corrects the potential problem with two high inputs in an *SR Flip-Flop*). Figure 10.7 illustrates an *SR Flip-Flop* being used as a *D Flip-Flop* in *Logisim-evolution* .

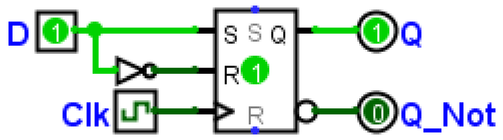

Figure 10.7: D Flip-Flop Using SR Flip-Flop

In Figure 10.7, the *D* input (for "Data") is latched and held on each clock cycle. Even though Figure 10.7 shows the inverter external to the latch circuit, in reality, a *D Flip-Flop* device bundles everything into a single package with only *D* and clock inputs and *Q* and *Q′* outputs, as in Figure 10.8. Like the *RS Flip-Flop*, *D Flip-Flops* also have "preset" inputs on the top and bottom that lets the designer hard-set output *Q* at either one or zero, which is useful during a power-up routine.

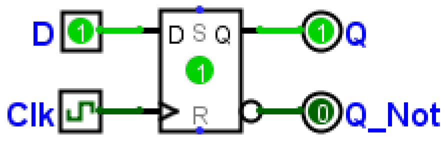

Figure 10.8: D Flip-Flop

Figure 10.9 is the timing diagram for a Data Flip-Flop.

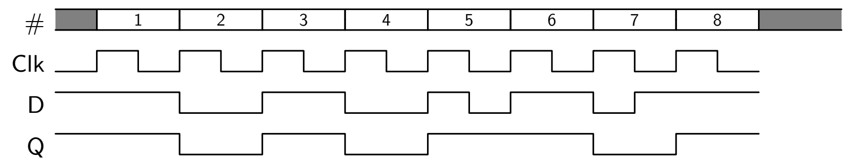

Figure 10.9: D Latch Timing Diagram

In Figure 10.9 it is evident that output *Q* follows input *D* but only on a positive clock edge. The latch "remembers" the value of *D* until the next clock pulse no matter how it changes between pulses.

10.3.4 *JK Flip-Flop*

The *JK flip-flop* is the "workhorse" of the flip-flop family. Figure 10.10 is the logic diagram for a *JK flip-flop*.

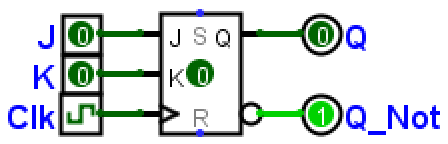

Figure 10.10: JK Flip-Flop

Internally, a *JK flip-flop* is similar to an *RS Latch*. However, the outputs to the circuit (Q and Q') are connected to the inputs (J and K) in such a way that the unstable input condition (both R and S high) of the *RS Latch* is corrected. If both inputs, J and K, are high then the outputs are toggled (they are "flip-flopped") on the next clock pulse. This toggle feature makes the *JK flip-flop* extremely useful in many logic circuits.

Figure 10.11 is the timing diagram for a *JK flip-flop*.

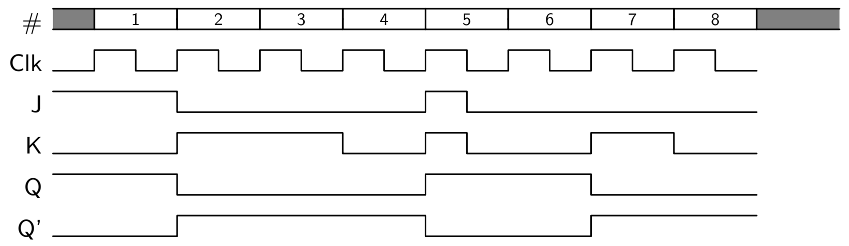

Figure 10.11: JK Flip-Flop Timing Diagram

Table 10.2 summarizes timing diagram 10.11. Notice that on clock pulse five both J and K are high so Q toggles. Also notice that Q' is not indicated in the table since it is always just the complement of Q.

Clock Pulse	J	K	Q
1	H	L	H
2	L	H	L
3	L	H	L
4	L	L	L
5	H	H	H
6	L	L	H
7	L	H	L
8	L	L	L

Table 10.2: JK Flip-Flop Timing Table

10.3.5 *Toggle (T) Flip-Flop*

If the J and K inputs to a *JK Flip-Flop* are tied together, then when the input is high the output will toggle on every clock pulse but when the input is Low then the output remains in the previous state. This is often referred to as a *Toggle Flip-Flop* (or *T Flip-Flop*). *T Flip-Flops* are not usually found in circuits as separate ICs since they are so easily created by soldering together the inputs of a standard *JK Flip-Flop*. Figure 10.12 is a *T Flip-Flop* in *Logisim-evolution* .

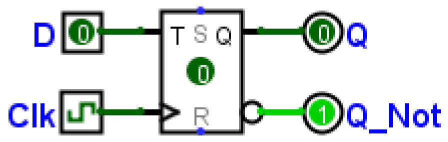

Figure 10.12: T Flip-Flop

Figure 10.13 is the timing diagram for a *T flip-flop*.

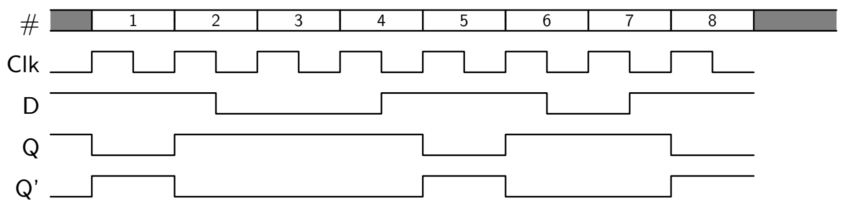

Figure 10.13: Toggle Flip-Flop Timing Diagram

In Figure 10.13 when *D*, the data input, is high then *Q* toggles on the positive edge of every clock cycle.

10.3.6 *Master-Slave Flip-Flops*

Master-Slave Flip-Flops are two flip-flops connected in a cascade and operating from the same clock pulse. These flip-flops tend to stabilize an input circuit and are used where the inputs may have voltage glitches (such as from a push button). Figure 10.14 is the logic diagram for two JK *flip-flops* set up as a Master-Slave Flip-Flop.

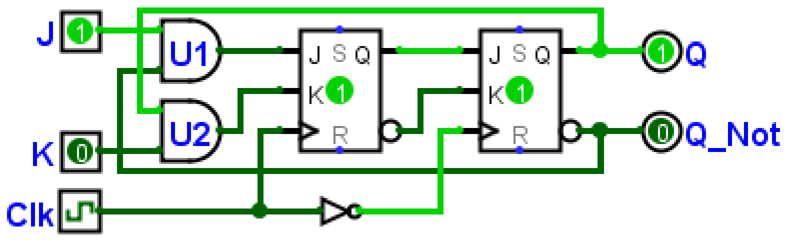

Figure 10.14: Master-Slave Flip-Flop

Because of the NOT gate on the clock signal these two flip-flops will activate on opposite clock pulses, which means the flip-flop one will enable first and read the JK inputs, then flip-flop two will enable and read the output of flip-flop one. The result is that any glitches on the input signals will tend to be eliminated.

Often, only one physical IC is needed to provide the two flip-flops for a master-slave circuit because dual *JK flip-flops* are frequently found on a single IC. Thus, the output pins for one flip-flop could be connected directly to the input pins for the second flip-flop on the same IC. By combining two flip-flops into a single IC package, circuit design can be simplified and fewer components need to be purchased and mounted on a circuit board.

10.4 REGISTERS

10.4.1 *Introduction*

A register is a simple memory device that is composed of a series of flip-flops wired together such that they share a common clock pulse. Registers come in various sizes and types and are often used as "scratch pads" for devices. For example, a register can hold a number entered on a keypad until the calculation circuit is ready do something with that number or a register can hold a byte of data coming from a hard drive until the CPU is ready to move that data someplace else.

10.4.2 *Registers As Memory*

Internally, a register is constructed from *D Flip-Flops* as illustrated in Figure 10.15.

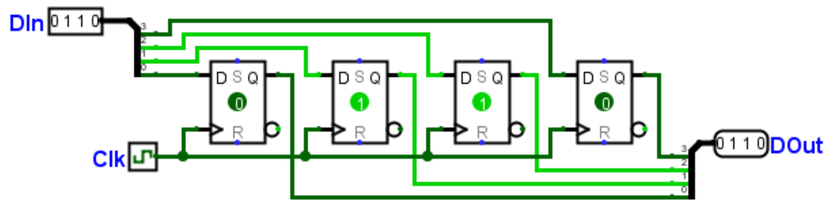

Figure 10.15: 4-Bit Register

Data are moved from the input port, in the upper left corner, into the register; one bit goes into each of the four *D Flip-Flops*. Because each latch constantly outputs whatever it contains, the output port, in the lower right corner, combines the four data bits and displays the contents of the register. Thus, a four-bit register can "remember" some number until it is needed, it is a four-bit memory device.

10.4.2.1 *74x273 Eight-Bit Register*

In reality, designers do not build memory from independent flip-flops, as shown in Figure 10.15. Instead, they use a memory IC that contains the amount of memory needed. In general, purchasing a memory IC is cheaper and more reliable than attempting to build a memory device.

One such memory device is a 74x273 eight-bit register. This device is designed to load and store a single eight-bit number, but other memory devices are much larger, including Random Access Memory (RAM) that can store and retrieve millions of eight-bit bytes.

10.4.3 *Shift Registers*

Registers have an important function in changing a stream of data from serial to parallel or parallel to serial; a function that is called "shifting" data. For example, data entering a computer from a network or USB port is in serial form; that is, one bit at a time is streamed into or out of the computer. However, data inside the computer are always moved in parallel; that is, all of the bits in a word are placed on a bus and moved through the system simultaneously. Changing data from serial to parallel or vice-verse is an essential function and enables a computer to communicate over a serial device.

Figure 10.16 illustrates a four-bit parallel-in/serial-out shift register. The four-bit data into the register is placed on *D0-D3*, the *Shift_Write* bit is set to 1 and the clock is pulsed to write the data into the register.

Then the *Shift_Write* bit is set to 0 and on each clock pulse the four bits are shifted right to the *SOUT* (for "serial out") port.

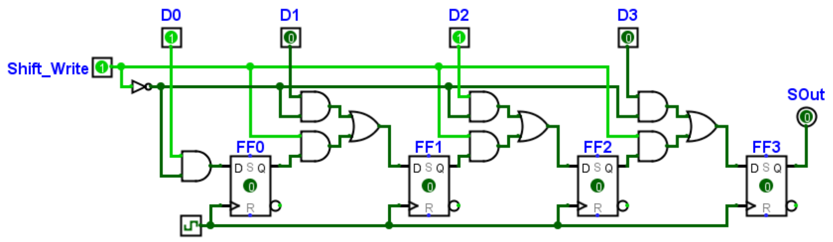

Figure 10.16: Shift Register

It is possible to also build other shift register configurations: serial-in/serial-out, parallel-in/parallel-out, and serial-in/parallel-out. However, universal shift registers are commonly used since they can be easily configured to work with data in either serial or parallel formats.