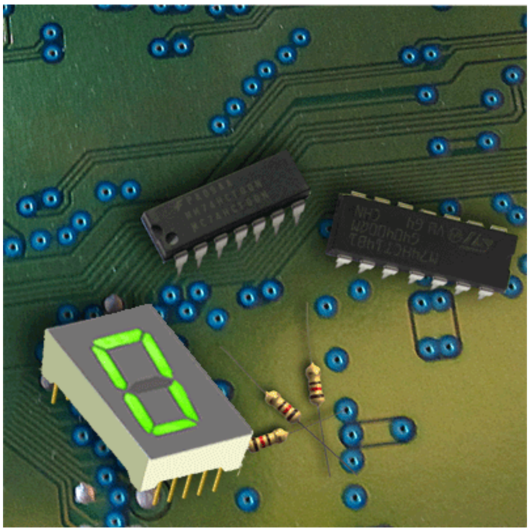
Digital Electronics

5.0 Sequential Logic

What you'll learn in Module 5

Introduction

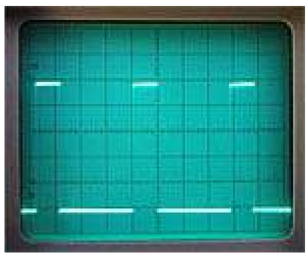


The logic circuits discussed in Digital Electronics Module 4 had output states that depended on the particular combination of logic states at the input connections to the circuit. For this reason these circuits are called combinational logic circuits.

Module 5 looks at digital circuits that use SEQUENTIAL LOGIC.

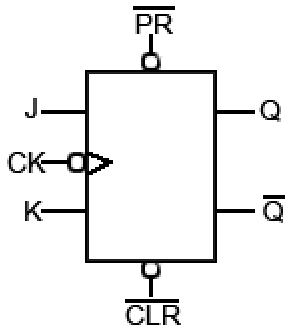
In these circuits the output depends, not only on the combination of logic states at its inputs, but also on the logic states that existed previously. In other words the output depends on a SEQUENCE of events occurring at the circuit inputs. Examples of such circuits include clocks, flip-flops, bi-stables, counters, memories, and registers. The actions of these circuits depend on a range of basic sub-circuits.

Clock Circuits



Module 5.1 deals with clock oscillators, which are basically types of square wave generators or oscillators that produce a continuous stream of square waves or a continuous train of pulses (a "square" wave whose mark to space ratio is NOT 1:1). These pulses are used to sequence the actions of other devices in the sequential logic circuit so that all the actions taking place in the circuit are properly synchronised.

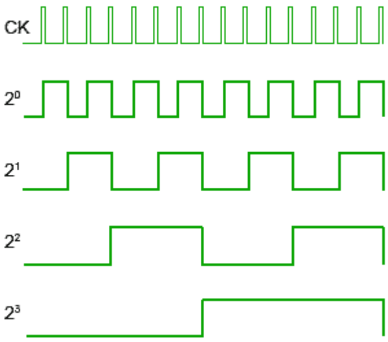
Bi-Stable Logic Devices



Bi-stable devices (popularly called Flip-flops) described in Modules 5.2 to 5.4, are sub-circuits, usually contained within ICs, and are the most basic type of 1-bit memory. They have outputs that can take up one of two stable states, Logic 1 or logic 0 or off. Once the device is triggered into one of these two states by an external input pulse, the output remains in that state until another pulse is used to reverse that state, so that a logic 1 output becomes logic 0 or vice versa. Again the circuit remains stable in this state until an input signal is used to reverse the output state. Hence the circuit is said to have Bi (two) stable output states.

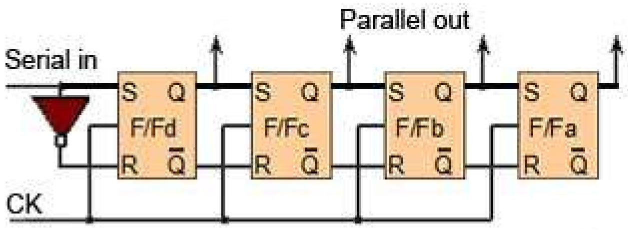
Counters

Various types of digital counters are described in Module 5.6. Consisting of arrangements of bi-stables, they are very widely used in many types of digital systems from computer arithmetic to TV screens and digital clocks.



Shift Registers

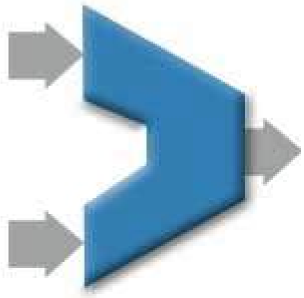
Also consisting of arrays of bi-stable elements, the shift registers described in Module 5.7 are temporary storage devices (memories) for multi-bit digital data. The data can be stored in the register either one bit at a time (serial input) or as one or more bytes at a time (parallel input).



The register can then output the data in either serial or parallel form. Shift registers are vital to receiving or transmitting data in digital communications systems. They can also be used in digital arithmetic for operations such as multiplication and division.

A Simple ALU

A simple arithmetic and logic unit (ALU) is described in Module 5.8 and combines many of the combinational and sequential logic circuits described in modules 4 and 5 to demonstrate how a very complex application is built by combining a number of much simpler digital sub circuits.



5.1 Clock Circuits

What you'll learn in Module 5.1

After studying this section, you should be able to:

Understand the need for clock generators.

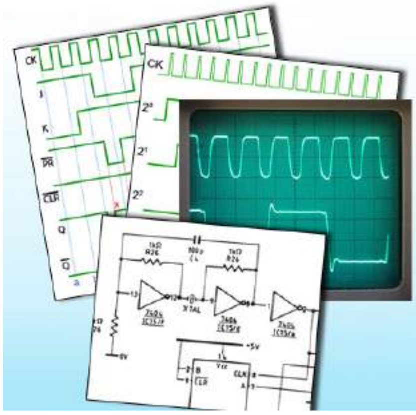
Recognise clock generator circuits

- RC clocks
- Crystal controlled clocks.

Understand the operation of common clock generators.

Clocks and Timing Signals

Most sequential logic circuits are driven by a clock oscillator. This usually consists of an astable circuit producing regular pulses that should ideally:



1. Be constant in frequency.

Many clock oscillators use a crystal to control the frequency. Because crystal oscillators generate normally high frequencies, where lower frequencies are required the original oscillator frequency is divided down from a very high frequency to a lower one using counter circuits.

2. Have fast rising and falling edges to its pulses.

It is the edges of the pulses that are important in timing the operation of many sequential circuits, the rise and fall times are usually be less than 100ns. The outputs of clock circuits will typically have to drive more gates than any other output in a given system. To prevent this load distorting the clock signal, it is usual for clock oscillator outputs to be fed via a buffer amplifier.

3. Have the correct logic levels.

The signals produced by the clock circuits must have appropriate the logic levels for the circuits being supplied.

Some examples of clock oscillator circuits are given below.

Simple Clock Oscillator

Fig 5.1.1 is probably the simplest oscillator possible, having only three components. Notice that the gate is a Schmitt inverter. This device has an extremely fast change over between logic states. Also the level at which it responds to an input change from 0 to 1 (Vt+) is higher than the level at which it changes from 1 to 0 (Vt-). The operation of the circuit is as follows.

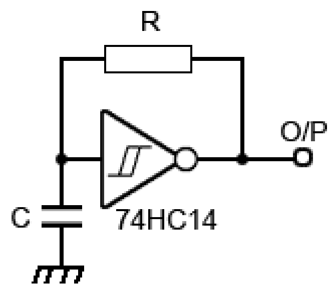


Fig. 5.1.1 Basic Schmitt Trigger Oscillator

Suppose the gate input is at logic 0, because the gate is an inverter, the output must be at logic 1, and C will therefore charge up via R from the output. This will happen with the normal CR charging curve. Once Vt+ is reached at the gate input, the gate output will rapidly switch to 0. The resistor is now connected effectively between the positive plate of C and zero volts. Thus the capacitor now discharges via R until the gate input voltage reduces to Vt- when the output will change to logic 1 once more, starting the charging and discharging cycle over again.

This Schmitt RC oscillator can produce a pulse waveform with an excellent wave shape and very fast rise and fall times. The mark to space ratio, as shown in Fig 5.1.2 is approximately 1:3.

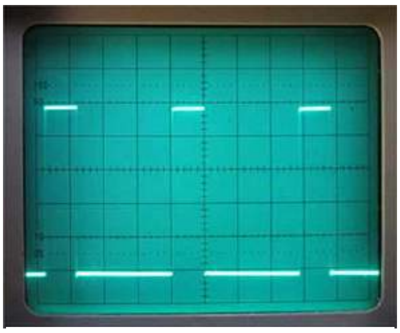


Fig. 5.1.2 Typical Basic Schmitt Oscillator Output

The frequency of oscillation depends on the time constant of R and C, but is also affected by the characteristics of the logic family used. For the 74HC14 the frequency (f)is calculated by:

$$f = \frac{1}{0.8RC}$$

When using the 74HCT14 the 0.8 correction factor is replaced by 0.67, however either of these formulae will give an approximate frequency. Whichever logic family is used, the frequency will vary with changes in supply voltage. Although this basic oscillator gives an excellent performance in many simple applications, if a stable frequency is an important factor in the choice of clock oscillator, there are of course better options.

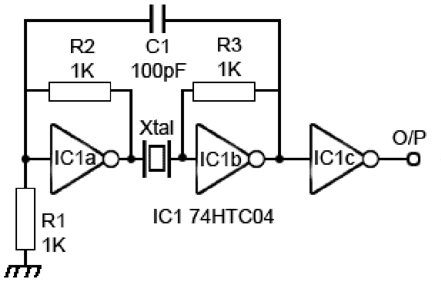


Fig. 5.1.3 Crystal Controlled Clock Oscillator

Crystal Controlled Clock Oscillator

Fig. 5.1.3 uses three gates from a 74HCT04 IC, and a crystal to provide an accurate frequency of oscillation. Here, the oscillator is running at 3.276MHz but this can be reduced by dividing the output frequency down to a lower value by dividing it by 2 a number of times using a series of flip-flops.

The top waveform in Fig 5.1.4 shows the clock signal generated by Fig 5.1.3, and beneath it is the clock signal frequency divided by 4 after passing it through two flip-flops. Notice that after passing the signal through flip-flops, as well as being reduced in frequency, the wave shape is considerably squarer and now has a 1:1 mark to space ratio.

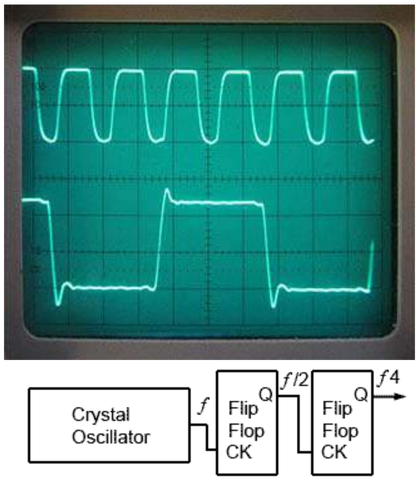


Fig. 5.1.4 Clock Frequency Divided by 4

The 555 Timer as a Clock Generator

Another option in circuits not requiring very high frequency clock signals is to use the 555 Timer in astable mode as a clock generator. This IC is able to produce good quality pulse or square wave signals over a wide range of frequencies, lower than those possible with crystal oscillators, also the frequency stability will not be as good as with crystal controlled oscillators.

Several oscillator design options are discussed in Oscillators Module 4.4

Two Phase Clock Signals

Some older microprocessor systems required two-phase clock signals which, provided that the source clock signal operated at twice the frequency required by the microprocessor, saved processing time as the microprocessor was able to perform two actions per clock cycle instead of one.

Producing a Two-Phase Clock Signal

If a clock signal with a 1:1 mark space ratio is used, two non-overlapping clock pulses can be created, as shown in Fig. 5.1.5. These signals are usually called □01 and □02 (□, the Greek letter phi is used to indicate phase).

In Fig 5.1.5 a single clock signal having a 1:1 mark to space ratio is fed into a JK flip-flop working in toggle mode. This is achieved by making both J and K logic 1.

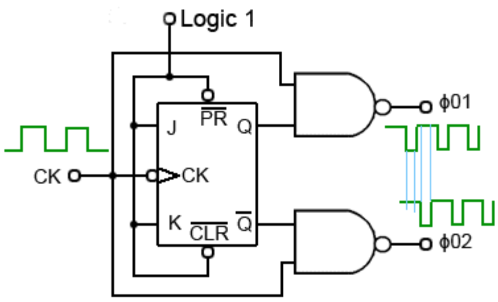


Fig. 5.1.5 Two-Phase Clock Circuit

The active low $\overline{PR}$ and $\overline{CLR}$ inputs take no part in the operation of this circuit so are also tied to logic 1. In toggle mode the Q output of the JK flip-flop inverts the logic levels at Q and $\overline{Q}$ at every falling edge of the clock (CK) input, also Q and $\overline{Q}$ always remain at opposite logic states.

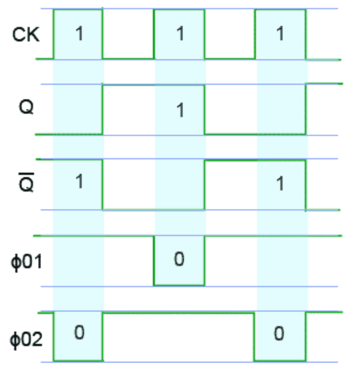


Fig. 5.1.6 Producing Non-Overlapping Clock Pulses

Fig. 5.1.6 illustrates the operation of Fig 5.1.5. Each of the NAND gates will produce a logic 0 output whenever both its inputs are at logic 1. The NAND gate producing □01 therefore creates a logic 0 pulse whenever CK and Q are at logic 1, and the NAND gate producing □02 creates a logic 0 pulse whenever CK and $\overline{Q}$ are at logic 1. Typical output waveforms are illustrated in Fig. 5.1.7.

If positive going clock pulses are required, the outputs from the NAND gates may be inverted using Schmitt inverters, which will also help to sharpen the rise and fall times of the clock waveforms.

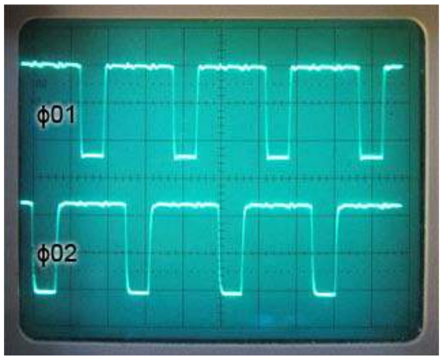


Fig. 5.1.7 Two Phase Clock Waveforms

Distributing Clock Signals

For more demanding applications there are very many specialised clock oscillator ICs available that are typically optimised for a particular range of applications, such as computer hardware, wireless communications, automotive or medical applications etc.

Clock Fan-out

Whatever circuit is used to generate a clock signal, it is important that its output has sufficient fan-out capability to drive the necessary number of ICs requiring a clock input, and that the clock signal is not degraded in amplitude, speed of its rise and fall times or accuracy of its frequency. Also, by maintaining fast rise and fall times, ringing on the waveform can become a problem. The waveform should be kept as close as possible to a perfect square wave shape.

Circuit Capacitance

Because the clock must feed many gates, the small capacitance of each of these gates will add, to become an appreciable capacitance, which loads the clock output tending to slow the rise and fall time of the clock signal. To avoid this, the clock output must have a low enough impedance to rapidly charge and discharge any natural capacitance in the circuit. The usual way to achieve this is to feed the clock signal via a special clock buffer gate, which will have the necessary low output impedance and a large fan out factor. Schmitt trigger gates may also be used to restore the shape and integrity of clock signals before they are applied to gates in different parts of the circuit.

Cross-talk

Where the clock signal has to be distributed around large circuits, there is a greater chance of introducing noise, and possible 'cross-talk' where data in one conductor is radiated into another nearby conductor. Problems such as this will increase the likelihood of 'skew' errors, i.e. clock signals arriving at different parts of the circuit at slightly different times, due to small changes in the phase of some of the distributed clock signals. Miniaturisation brought about by surface mount technology can help minimise these problems. Also when clock signals need to be sent from one system to another over an external wired or wireless link it is common to use one of the several ECL or LVDS logic families with their differential outputs to minimise interference, and there are many application specific ICs (ASICS) using these technologies for high frequency clock distribution.

5.2 SR Flip-flops

What you'll learn in Module 5.2

After studying this section, you should be able to:

Describe SR flip-flop circuits and can:

- Describe typical applications for SR flip-flops.
- Recognize standard circuit symbols for SR flip-flops.
- Recognize SR flip-flop integrated circuits.

Recognise alternative forms of SR flip-flops.

- Clocked SR flip-flop.
- High Activated SR flip-flops (The RS Latch).
- Compile truth tables for SR flip-flops.
- Construct timing diagrams to explain the operation of SR flip-flops.

Use circuit simulation software to construct SR flip-flops.

Typical applications for SR Flip-flops.

The basic building bock that makes computer memories possible, and is also used in many sequential logic circuits is the flip-flop or bi-stable circuit. Just two inter-connected logic gates make up the basic form of this circuit whose output has two stable output states. When the circuit is triggered into either one of these states by a suitable input pulse, it will 'remember' that state until it is changed by a further input pulse, or until power is removed. For this reason the circuit may also be called a Bi-stable Latch.

The SR flip-flop can be considered as a 1-bit memory, since it stores the input pulse even after it has passed. Flip-flops (or bi-stables) of different types can be made from logic gates and, as with other combinations of logic gates, the NAND and NOR gates are the most versatile, the NAND being most widely used. This is because, as well as being universal, i.e. it can be made to mimic any of the other standard logic functions, it is also cheaper to construct.

Other, more widely used types of flip-flop are the JK, the D type and T type, which are developments of the SR flip-flop and will be studied in Modules 5.3 and 5.4.

The SR Flip-flop.

The SR (Set-Reset) flip-flop is one of the simplest sequential circuits and consists of two gates connected as shown in Fig. 5.2.1. Notice that the output of each gate is connected to one of the inputs of the other gate, giving a form of positive feedback or 'cross-coupling'.

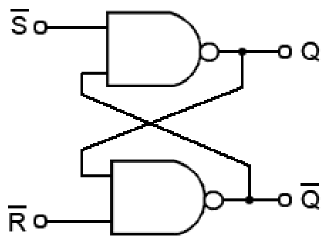


Fig 5.2.1 SR Flip-flop (low activated)

The circuit has two active low inputs marked $\overline{S}$ and $\overline{R}$, 'NOT' being indicated by the bar above the letter, as well as two outputs, Q and $\overline{Q}$. Table 5.2.1 shows what happens to the Q and $\overline{Q}$ outputs when a logic 0 is applied to either the $\overline{S}$ or $\overline{R}$ inputs.

1. Q output is set to logic 1 by applying logic 0 to the $\overline{S}$ input.
2. Returning the $\overline{S}$ input to logic 1 has no effect. The 0 pulse (high-low-high) has been 'remembered' by the Q.
3. Q is reset to 0 by logic 0 applied to the $\overline{R}$ input.
4. As $\overline{R}$ returns to logic 1 the 0 on Q is 'remembered' by Q.

Table 5.2.1

	$\overline{S}$	$\overline{R}$	Q	$\overline{Q}$	Comments
1.	0	1	1	0	Q is set to 1 by 0 on $\overline{S}$
2.	1	1	1	0	No change, (1 on Q is remembered)
3.	1	0	0	1	Q is reset to 0 by 0 on $\overline{R}$
4.	1	1	0	1	No change, (0 on Q is remembered)
5.	0	0	1	1	Both inputs at 0 – both outputs are at 1 (Non-allowed state)
6.	1	1	?	?	Inputs change from 0,0 to 1,1 together - outputs will be INDETERMINATE

Problems with the SR Flip-flop

There are however, some problems with the operation of this most basic of flip-flop circuits. For conditions 1 to 4 in Table 5.2.1, $\overline{Q}$ is the inverse of Q. However, in row 5 both inputs are 0, which makes both Q and $\overline{Q}$=1, and as they are no longer opposite logic states, although this state is possible, in practical circuits it is 'not allowed'.

In row 6 both inputs are at logic 1 and the outputs are shown as 'indeterminate', this means that although Q and $\overline{Q}$ will be at opposite logic states it is not certain whether Q will be 1 or 0, Notice however that in the absence of any input pulses, both inputs are normally at logic 1. This is normally OK, as the outputs will be at the state remembered from the last input pulse. The indeterminate or uncertain logic state only occurs if the inputs change from 0,0 to 1,1 together. This should be avoided in normal operation, but is likely to happen when power is first applied. This could lead to uncertain results, but the flip-flop will work normally once an input pulse is applied to either input.

The SR Flip-flop is therefore, a simple 1-bit memory. If the $\overline{S}$ input is taken to logic 0 then back to logic 1, any further logic 0 pulses at $\overline{S}$ will have no effect on the output.

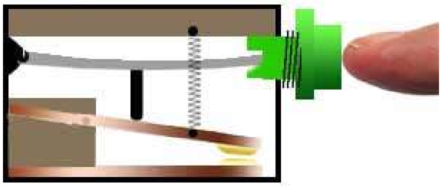


Fig. 5.2.2 Switch Bounce
(Single frame from animation)

Switch De-Bouncing

The fact that repeated pulses at the $\overline{S}$ (or the $\overline{R}$) inputs are ignored after the initial pulse has set or reset the Q output, makes the SR Flip-flop useful for switch de-bouncing.

When any moving object collides with a stationary object it tends to bounce; the contacts in switches are no exception to this rule. Although the contacts may be tiny and the movement small, as the contacts close they will tend to bounce rather than close and stay closed. This causes a number of very fast on and off states for a short time, until the contacts stop bouncing in the closed position. The length of time of the bouncing may be very short, as shown in Fig. 5.2.3 where a number of fast pulses occur for about 2ms after the switch is initially closed (red arrow). For many applications this switch bounce may be ignored, but in digital circuits the repeated ones and zeros occurring after a switch is closed, will be recognised as additional switching actions.

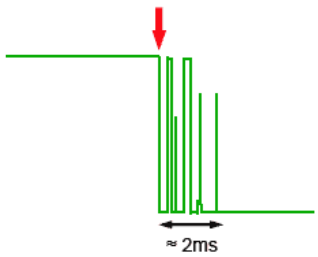


Fig. 5.2.3 Typical Switch Bounce Spikes

Switch De-Bounce Circuit

The SR flip-flop is very effective in removing the effects of switch bounce and Fig 5.2.4 illustrates how a SR flip-flop can be used to produce clean pulses using SWI, which is a 'break before make' changeover switch. When SW1 connects the upper contact to 0V, the $\overline{S}$ input changes from logic 1 to logic 0 and $\overline{R}$ is 'pulled up' to logic 1 by R1.

As soon as $\overline{S}$ is at logic 0, (at time 'a' in Fig. 5.2.4) output Q will be at logic 1 and any further pulses due to switch bounce will be ignored.

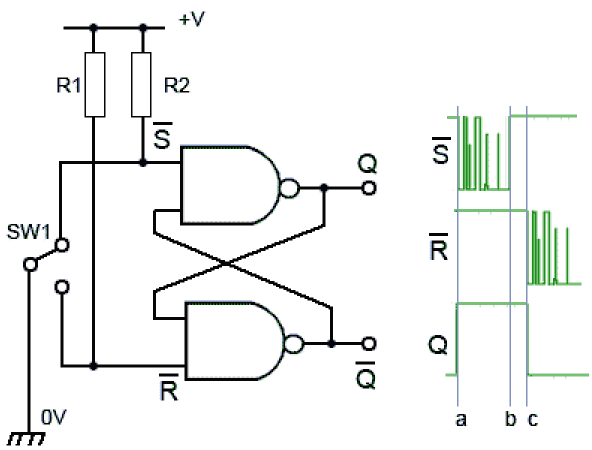


Fig. 5.2.4 SR Flip-flop Switch De-Bouncing Circuit

When SW1 is switched to the lower contact, there will be a short time (between times 'b' and 'c' in Fig. 5.2.4) when neither $\overline{S}$ or $\overline{R}$ is connected to 0V. During this time $\overline{S}$ returns to logic 1, therefore both inputs will be at logic 1 until time 'c', when SW1 connects $\overline{R}$ to 0V and Q is reset to logic 0 completing the output pulse. The use of a 'break before make' rather than a 'make before break' switch is important, as it ensures that during the changeover period (time 'b' to time 'c' in Fig. 5.2.4) both inputs are at logic 1 rather than the non-allowed state where both inputs would be logic 0. This ensures that outputs Q and $\overline{Q}$ are never at the same logic state.

Although, during the change over of SW1 both inputs are at logic 1, this does not produce the indeterminate state described in Table 5.2.1, as one or other of the inputs is always at logic 0 before both inputs become logic 1.

The RS Latch

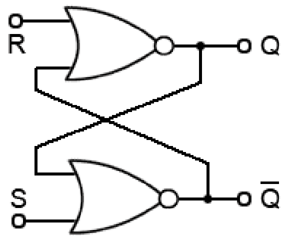


Fig. 5.2.5 High Activated RS Latch

Flip-flops can also be considered as latch circuits due to them remembering or 'latching' a change at their inputs. A common form of RS latch is shown in Fig. 5.2.5. In this circuit the $\overline{S}$ and $\overline{R}$ inputs have now become S and R inputs, meaning that they will now be 'active high'.

They have also changed places, the R input is now on the gate having the Q output and the S input is on the $\overline{Q}$ gate. These changes occur because the circuit is using NOR gates instead of NAND.

The active high operation of the RS Latch is shown in Table 5.2.2.

Table 5.2.2

	R	S	Q	$\overline{Q}$	Comments
1.	0	1	1	0	Q is set to 1 by 1 on S
2.	0	0	1	0	No Change after set
3.	1	0	0	1	Q is reset to 0 by 1 on R
4.	0	0	0	1	No Change after reset
5.	1	1	0	0	Not allowed (both outputs at 0)
6.	0	0	?	?	If both inputs change from 11 to 00 outputs will be INDETERMINATE

1. Q is set to 1 when the S input goes to logic 1.

2. This is remembered on Q after the S input returns to logic 0.

3. Q is reset set to 0 when the R input goes to logic 1.

4. This is remembered on Q after the R input returns to logic 0.

5. If both inputs are at logic 1, Q is the same as $\overline{Q}$ (the non-allowed state).

6. The state of the outputs cannot be guaranteed if the inputs change from 1,1 to 0, 0 at the same time.

Timing Diagrams

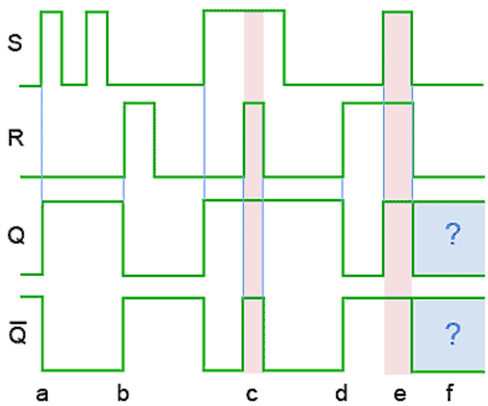


Fig. 5.2.6 Timing Diagram for a High Activated RS Latch

Truth tables are not always the best method for describing the action of a sequential circuit such as the SR flip-flop. Timing diagrams, which show how the logic states at various points in a circuit vary with time, are often preferred.

Fig. 5.2.6 shows a timing diagram describing the action of the basic RS Latch for logic changes at R and S. At time (a) S goes high and sets Q, which remains high until time (b) when S is low and R goes high, resetting Q. During period (c) both S and R are high causing the non-allowed state where both outputs are high. After period (c) Q remains high until time (d) when R goes high, resetting Q. Period (e) is another non-allowed period, at the end of which both inputs go low causing an indeterminate output condition in period (f).

The Clocked SR Flip-flop

Fig. 5.2.7 shows a useful variation on the basic SR flip-flop, the clocked SR flip-flop. By adding two extra NAND gates, the timing of the output changeover after a change of logic states at S and R can be controlled by applying a logic 1 pulse to the clock (CK) input. Note that the inputs are now labelled S and R indicating that the inputs are now 'high activated'. This is because the two extra NAND gates are disabled while the CK input is low, therefore the outputs are completely isolated from the inputs and so retain any previous logic state, but when the CK input is high (during a clock pulse) the input NAND gates act as inverters. Then for example, a logic 1 applied to S becomes a logic 0 applied to the $\overline{S}$ input of the active low SR flip-flop second stage circuit.

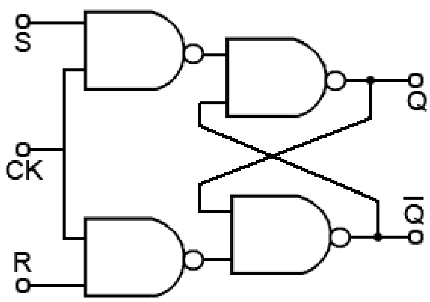


Fig. 5.2.7 High Activated Clocked SR Flip-flop

The main advantage of the CK input is that the output of this flip-flop can now be synchronised with many other circuits or devices that share the same clock. This arrangement could be used for a basic memory location by, for example, applying different logic states to a range of 8 flip-flops, and then applying a clock pulse to CK to cause the circuit to store a byte of data.

The basic form of the clocked SR flip-flop shown in Fig. 5.2.7 is an example of a level triggered flip-flop. This means that outputs can only change to a new state during the time that the clock pulse is at its high level (logic 1). The ability to change the input whilst CK is high can be a problem with this circuit, as any input changes occurring during the high CK period, will also change the outputs. A better method of triggering, which will only allow the outputs to change at one precise instant is provided by edge triggered devices available in D Type and JK flip-flops.

SR Flip-flop ICs

Comprising just two gates, low activated SR flip-flops are simple to implement using standard NAND gates but active low SR flip-flops (called SR flip-flops) are available as Quad packages in the LS TTL family as 74LS279 from Texas Instruments.

Circuit Symbols for Flip-flops

Rather than drawing the schematic circuit for individual gate versions of flip-flops it is common to draw them in block form. Some commonly used block versions of SR and RS flip-flops are shown in Fig. 5.2.8.

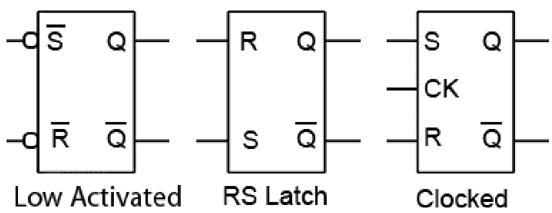


Fig. 5.2.8 SR Flip-flop Circuit Symbols

5.3 D Type Flip-Flops

What you'll learn in Module 5.3

After studying this section, you should be able to:

Understand the operation of D Type flip-flops and can:

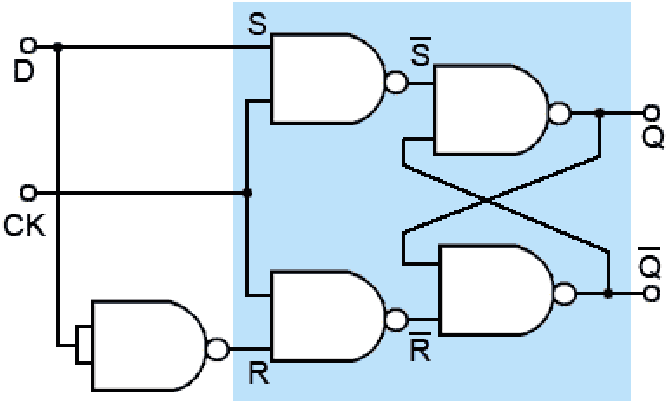
- Describe typical applications for D Type flip-flops.
- Recognize standard circuit symbols for D Type flip-flops.
- Recognize D Type flip-flop integrated circuits.

Recognise alternative forms of D Type flip-flops.

- Edge triggered D Type flip-flops.
- Toggle flip-flops.

Construct timing diagrams to explain the operation of D Type flip-flops.

Use circuit simulation software to construct D Type flip-flops.



Inputs		Outputs	
CK	D	Q	$\overline{Q}$
0	X	No change	
1	0	0	1
1	1	1	0

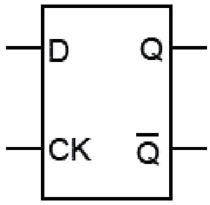


Fig. 5.3.1 Level Triggered D Type Flip-flop

D Type Flip-flops.

The major drawback of the SR flip-flop (i.e. its indeterminate output and non-allowed logic states) described in Digital Electronics Module 5.2 is overcome by the D type flip-flop. This flip-flop, shown in Fig. 5.3.1 together with its truth table and a typical schematic circuit symbol, may be called a Data flip-flop because of its ability to 'latch' and remember data, or a Delay flip-flop because latching and remembering data can be used to create a delay in the progress of that data through a circuit. To avoid the ambiguity in the title therefore, it is usually known simply as the D Type. The simplest form of a D Type flip-flop is basically a high activated SR type with an additional inverter to ensure that the S and R inputs cannot both be high or both low at the same time. This simple modification prevents both the indeterminate and non-allowed states of the SR flip-flop. The S and R inputs are now replaced by a single D input, and all D type flip-flops have a clock input.

Operation.

As long as the clock input is low, changes at the D input make no difference to the outputs. The truth table in Fig. 5.3.1 shows this as a 'don't care' state (X). The basic D Type flip-flop shown in Fig. 5.3.1 is called a level triggered D Type flip-flop because whether the D input is active or not depends on the logic level of the clock input.

Provided that the CK input is high (at logic 1), then which ever logic state is at D will appear at output Q and (unlike the SR flip-flops) $\overline{Q}$ is <u>always</u> the inverse of Q).

In Fig. 5.3.1, if D = 1, then S must be 1 and R must be 0, therefore Q is SET to 1.

Alternatively,

If D = 0 then R must be 1 and S must be 0, causing Q to be reset to 0.

The Data Latch?

The name Data Latch refers to a D Type flip-flop that is level triggered, as the data (1 or 0) appearing at D can be held or 'latched' at any time whilst the CK input is at a high level (logic 1).□As can be seen from the timing diagram shown in Fig 5.3.2, if the data at D changes during this time, the Q output assumes the same logic level as the D.

Ripple Through

Fig. 5.3.2 also illustrates a possible problem with the level triggered D type flip-flop; because Q changes in sympathy with D when the clock pulse is at its high level, it only 'remembers' the last input state that occurred during the clock pulse, (period RT in Fig. 5.3.2). This effect is called 'Ripple Through', and although this allows the level triggered D Type flip-flop to be used as a data switch, only allowing data through from D to Q as long as CK is held at logic 1, this may not be a desirable property in many types of circuit.

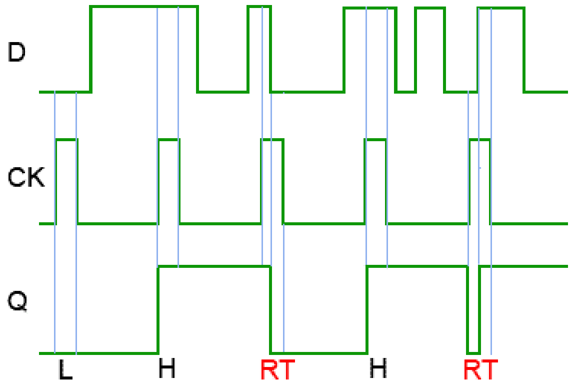


Fig. 5.3.2 Timing Diagram for a Level Triggered D Type Flip-flop

The Edge Triggered D Type Flip-flop

Fortunately ripple though can be largely prevented by using the Edge Triggered D Type flip-flop illustrated in Fig 5.3.3.

The clock pulse applied to the flip-flop is reduced to a very narrow positive going clock pulse of only about 45ns duration, by using an AND gate and applying the clock pulse directly to input 'a' but delaying its arrival at input 'b' by passing it through 3 inverters. This inverts the pulse and also delays it by three propagation delays, (about 15ns per inverter gate for 74HC series gates). The AND gate therefore produces logic 1 at its output only for the 45ns when both 'a' and 'b' are at logic 1 after the rising edge of the clock pulse.

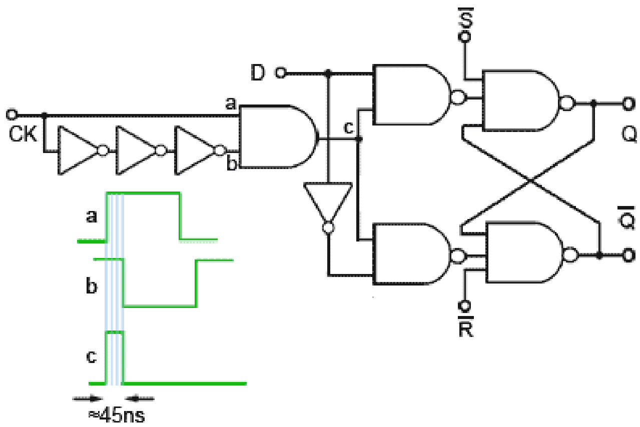


Fig. 5.3.3 Edge Triggered D Type Flip-Flop with Set and Reset

Synchronous and Asynchronous Inputs

A further refinement in Fig. 5.3.3 is the addition of two further inputs SET and RESET, which are actually the original $\overline{S}$ and $\overline{R}$ inputs of the basic low activated SR flip-flop.

Notice that there is now a subtle difference between the active low Set ($\overline{S}$) and Reset ($\overline{R}$) inputs, and the D input. The D input is SYNCHRONOUS, that is its action is synchronised with the clock, but the $\overline{S}$ and $\overline{R}$ inputs are ASYNCHRONOUS i.e. their action is NOT synchronised with the clock. The SET and RESET inputs in Fig 5.3.4 are 'low activated', which is shown by the inversion circles at the S and R inputs to indicate that they are really $\overline{S}$ and $\overline{R}$.

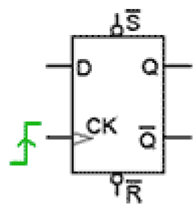


Fig. 5.3.4 Edge Triggered D Type Flip-Flop

The flip-flop is positive edge triggered, which is shown on the CK input in Fig 5.3.4 by the wedge symbol. A wedge accompanied by an inversion circle would indicate negative (falling) edge triggering, though this is generally not used on D Type flip-flops.

Timing Diagram

The 'Edge triggered D type flip-flop with asynchronous preset and clear capability', although developed from the basic SR flip-flop becomes a very versatile flip-flop with many uses. A timing diagram illustrating the action of a positive edge triggered device is shown in Fig. 5.3.5.

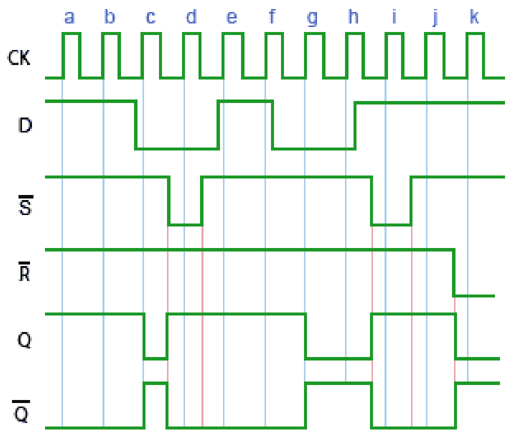


Fig. 5.3.5 Positive Edge Triggered D Type Timing Diagram

At the positive going edges of clock pulses a and b, the D input is high so Q is also high.

Just before pulse c the D input goes low, so at the positive going edge of pulse c, Q goes low.

Between pulses c and d the asynchronous $\overline{S}$ input goes low and immediately sets Q high.

The flip-flop then ignores pulse d while $\overline{S}$ is low, but as $\overline{S}$ returns high, and D has also returned to its high state before pulse e, Q remains high during pulse e.

D is still high at the positive going edge of pulse f, but because the flip-flop is positive edge triggered, the change in the logic level of D during pulse f is ignored until the positive going edge of pulse g, which resets Q to its low level.

At the positive going edge of pulse h, the low level of input D remains, keeping Q low, but between pulses h and i, the $\overline{S}$ input goes low, overriding any action of D and immediately making Q high.

Clock pulse i is again ignored, due to $\overline{S}$ being in its active low state and Q remains high, under the control of $\overline{S}$ until just before pulse j. At the positive going edge of pulse j, input D regains control, but as D is high and Q is already high, no change in output Q occurs.

Finally, just before pulse k, the asynchronous reset input ($\overline{R}$) goes low and resets Q to its low level (logic 0), which again causes the D input to be ignored.

Edge Triggered D Type Flip-flop Summary:

- At the positive going edge of a CK pulse, Q will assume the same level as input D, unless either asynchronous input has control.
- A logic 0 on the asynchronous input $\overline{S}$ at any time will cause Q to be set to logic 1 from the time $\overline{S}$ goes low, until the first CK pulse after $\overline{S}$ returns to logic 1.
- A logic 0 on the asynchronous input $\overline{R}$ will cause Q to be reset to logic 0 from the time $\overline{R}$ goes low, until the first CK pulse after $\overline{R}$ returns to logic 1.
- The action of the asynchronous inputs overrides any effect of the D input.
- Both asynchronous inputs should not be low at the same time, as both Q and $\overline{Q}$ will then be at logic 1. This is a non-allowed state.

The D Type Master Slave Flip-Flop

Yet a further version of the D Type flip-flop is shown in Fig. 5.3.6 where two D type flip-flops are incorporated in a single device (blue background), this is the D type master-slave flip-flop. Circuit symbols for the master-slave device are very similar to those for edge-triggered flip-flops, but are now divided into two sections by a dotted line, as also illustrated in Fig 5.3.6.

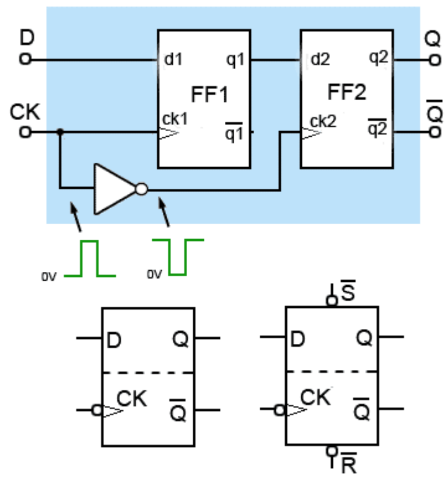


Fig. 5.3.6 The D Type Master Slave Flip-flop

The flip-flops FF1 (the master flip-flop) and FF2 (the slave flip-flop), are positive edge triggered devices, and an inverted version of the CK pulse is fed from the main CK input to FF2. Notice that although the clock inputs on the circuit symbols suggest that this is a negative edge triggered device, data is actually taken into FF1 on the POSITIVE going edge of the CK pulse. The data also of course appears at q1 at this time, but as the CK pulse is inverted at ck2, FF2 is seeing a falling edge at the same time, so ignores the data on d2.

After the positive going edge of the external CK pulse, FF1 ignores any further data at D, and at the negative going edge of the external CK pulse, the data being held at q1 is taken into the d2 input of FF2 which now sees a positive going edge of the inverted CK pulse. Therefore data is taken into D at the positive going (rising) edge of the CK pulse, and then appears at Q at the negative going (falling) edge of the CK pulse.

Considering the master slave flip-flop as a single device, the relationship between the clock (CK) input and the Q output does look rather like a negative edge triggered device, as any change in the output occurs at the falling edge of the clock pulse. However, as illustrated in Fig. 5.3.7 this is not really negative edge triggering, because the data appearing at Q as the clock pulse returns to logic 0, is actually the data that was present at input D at the RISING edge of the CK pulse. Any further changes that may occur in data at the D input during the clock pulse are ignored. D type master-slave flip-flops are also available with asynchronous inputs making it a very versatile device indeed.

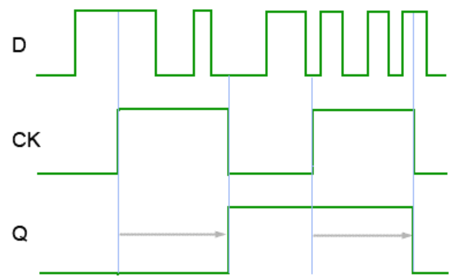


Fig. 5.3.7 Timing Diagram for a D Type Master-Slave Flip-flop

The Toggle Flip-flop

Toggle flip-flops are the basic components of digital counters, and all of the D type devices are adaptable for such use. When an electronic counter is used for counting, what are actually being counted are pulses appearing at the CK input, which may be either regular pulses derived from an internal clock, or they can be irregular pulses generated by some external event.

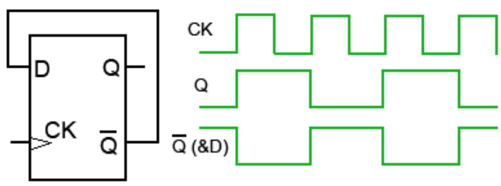


Fig. 5.3.8 An Edge Triggered D Type, Converted to a Toggle Flip-flop

When a toggle flip-flop is used as one stage of a counter, its Q output changes to the opposite state, (it toggles) high or low on each clock pulse. Most edge-triggered flip-flops can be used as toggle flip-flops including the D type, which can be converted to a toggle flip-flop with a simple modification. In theory all that is necessary to convert an edge triggered D Type to a T type is to connect the $\overline{Q}$ output directly to the D input as shown in Fig. 5.3.8.

The actual input is now CK. The effect of this mode of operation is also shown in the timing diagram in Fig. 5.3.8 using a positive edge triggered D type flip-flop.

Toggle Flip-flop Operation

Suppose that initially CK and Q = 0. Then $\overline{Q}$ and D must be 1. At the rising edge of a CK pulse, the logic 1 at D is allowed into the flip-flop and, at the end of the flip-flop's propagation delay, appears at Q, and $\overline{Q}$ changes to logic 0 at the same time.

This logic 0 is now fed back to D, but it is important that it is not immediately accepted into the D input, otherwise oscillation could occur with D continually changing between 1 and 0. However, because of the flip-flop's propagation delay, when the logic 0 from $\overline{Q}$ arrives at D, the very short edge-triggering period will have completed, and the change in data at D will be ignored.

At the next CK rising edge of the clock signal, the 0 at D now passes to Q, making $\overline{Q}$ and D logic 1 again. The Q output of the flip-flop therefore toggles at each positive going edge of the CK pulse.

Because the Q output changes state at each clock pulse rising edge, the 0 period and the 1 period of the Q output will always be of equal length, and the output will be a square wave with a 1:1 mark to space ratio, its frequency will be half that of CK.

To use toggle flip-flops as simple binary counters, a number of toggle flip-flops may be connected in cascade, with the Q output of the first flip-flop in the series, being connected to the CK input of the next flip-flop and so on. This is also the principle of frequency division. How counters and dividers can be constructed from toggle flip-flops is explained in Digital Electronics Module 5.6.

Data Timing

In practice however, using direct feedback from $\overline{Q}$ to D can cause problems as, to ensure stable operation and avoid unwanted oscillation, it is important in any digital circuit, that any changes in logic level taking place at D must be both stable, (free from any overshoot or ringing etc.) and at a valid logic level during a short period, before and after the clock signal causes a change. These periods are called the set up and hold times.

Although it is easy to think of the clock signal initiating a change at a particular **time,** e.g. when its rising edge occurs, data is actually clocked into input D when the CK waveform reaches a certain **voltage level**. In 74HC series gates, this level is 50% of V_{DD}, as illustrated in Fig 5.3.9. This shows in expanded time detail, the transitions taking place at the D and CK inputs of a D type positive edge triggered flip-flop.

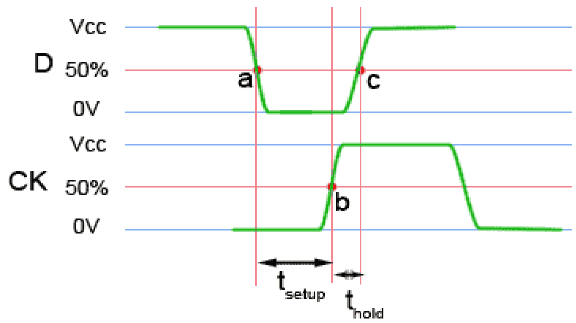


Fig. 5.3.9 Clocked Logic Set Up and Hold Times

To guarantee correct triggering, it is important that the data at the D input has settled at a valid logic level before the clock signal triggers any change. Therefore there must be some time allowed from when the D input first becomes valid to allow time for any slow rising pulse, any overshoot or ringing to occur before the clock pulse samples the logic level.

For example, the time between point (a) in Fig.5.3.9, where D initially falls below 50% of V_{DD} and the time when CK rises to its trigger threshold of 50% V_{DD} (point b) is called the set up time (t_{setup} or t_{su}), and in 74HC series ICs this will typically be between 5ns and 15ns.

After the trigger point there must be a further period (b to c in Fig. 5.3.9) where the data at D must remain at the same valid logic level to ensure that the correct logic level has been accepted. This is called the hold time (t_{hold} or t_h) and is typically around 3ns in 74HC series ICs.

In sequential logic circuits precise timing is vitally important. The design of a circuit must take into consideration not only set up and hold times but also the propagation times of gates or flip-flops in each path that a digital signal takes through a circuit. Failure to get the timing right can lead to problems such as 'glitches' i.e. sudden sharp spikes, as a device such as a flip-flop momentarily produces a change from one logic level to another and back again. Such glitches may be very short (a few nanoseconds) but sufficient to trigger another device to a wrong logic level.

With devices such as flip-flops using both triggering and feedback, incorrect timing can also lead to instability and unwanted oscillations. Avoiding such problems is a major reason for the use of edge triggering and master slave devices.

D Type Flip-flop ICs

A selection of D type Flip-flop ICs are listed below.

74HC74 Dual D Type Flip-flop with Set and Reset from ON Semiconductors.

74LS75 Quad D Type Data Latches from Texas Instruments.

74HC174 Hex D Type Flip-flop with Reset from NXP.

74HC175 Quad D Type Flip-flop with Reset from NXP.

74HC273 Octal D Type Flip-flop with Reset from Texas Instruments.

74HC373 Octal Transparent D Type Data Latches with 3-State Outputs from Texas Instruments.

74HC374A Octal 3-State Non-Inverting D Type Flip-flop from ON Semiconductors.

5.4 JK Flip-Flops

What you'll learn in Module 5.4

After studying this section, you should be able to:

Understand JK Flip-flop circuits and can:

- Describe typical applications for JK flip-flops.
- Recognize standard circuit symbols for JK flip-flops.
- Recognise JK Flip-flop integrated circuits.
- Describe alternative forms of JK flip-flops.

Understand timing diagrams to explain the operation of JK flip-flops.

Use circuit simulation software to construct JK flip-flops.

A Universal Programmable Flip-flop

The JK Flip-flop is a programmable flip-flop because, depending on the logic states applied to its inputs, J, K, S and R, it can be made to mimic the action of any of the other flip-flop types.

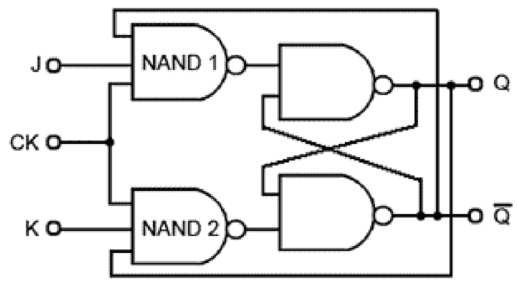


Fig.5.4.1 Basic JK Flip-flop Circuit

Fig. 5.4.1 shows the basic configuration for a JK flip-flop (without S and R inputs) using only four NAND gates. The circuit is similar to the clocked SR flip-flop shown in Fig. 5.2.7, (Digital Electronics Module 5.2) but in Fig. 5.4.1, it can be seen that although the clock input is the same as in the clocked SR flip-flop, gate NAND 1 in Fig. 5.4.1 is now a three input gate and the set input (S) has been replaced by an input labelled J, and the third input provides feedback from the $\overline{Q}$ output.

On NAND 2 the reset input (R) of Fig 5.2.7 has been replaced by input K and there is an additional feedback connection from Q. The purpose of this feedback is to eliminate the indeterminate state that occurred on the SR flip-flop when both inputs were made logic 0 at the same time.

Operation

As a starting point, assume that both J and K are at logic 1 and the outputs Q = 0 and $\overline{Q}$ = 1, this will cause NAND 1 to be enabled, as it has logic 1 on two (J and $\overline{Q}$) of its three inputs, requiring only a logic 1 on its clock input to change its output state to logic 0. At the same time, NAND 2 is disabled, because it only has one of its inputs (K) at logic 1, its feedback input is at logic 0 because of the feedback from Q.

On the arrival of a clock pulse, the output of NAND 1 therefore becomes logic 0, and causes the flip-flop to change state so that Q = 1 and $\overline{Q}$ = 0. This action enables NAND 2 and disables NAND 1.

As this change of state at the outputs occurs however, there is a problem. If the clock pulse is still high, or in its t_{hold} period when the flip-flop changes state, the output of NAND 2 may instantly go to logic 0 and the flip-flop will reset back to its original state. This can then set up a situation where the flip-flop will rapidly oscillate between its two states.

These problems caused by the output data 'racing' round the feedback lines from output to input before the end of the clock pulse are known as RACE HAZARDS and of course must be avoided. This can be done however, by using a more complex version of the circuit.

The JK Master Slave Flip-flop.

As well as minimising the race hazards problem, this type of flip-flop can also function as an SR, a clocked SR, a D type, or a Toggle flip-flop. The 'master slave' terminology refers to the device having two separate flip-flop stages, isolating the input from the output. As well as reducing the race hazards problem, it also has a further advantage over the simpler SR types, as its J and K inputs can be any value without causing any indeterminate state.

A typical circuit symbol is shown in Fig 5.4.2, and Table 5.4.1 shows how different logic combinations applied to the J and K inputs change the way the JK flip-flop responds to the application of a clock pulse on the CK input.

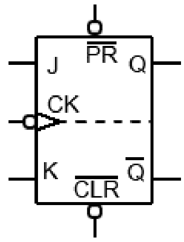


Fig. 5.4.2 JK Master-Slave Flip-Flop Symbol

JK Synchronous Inputs

- When J and K are both 0 the flip-flop is inhibited, Q is the same after the CK pulse as it was before; there is no change at the output.
- If J and K are at different logic levels, then after the CK pulse, Q and $\overline{Q}$ will take up the same states as J and K. For example, if J = 1 and K = 0, then on the trailing (negative going) edge of a clock pulse, the Q output will be set to 1, and if K = 1 and J = 0 then the Q output is reset to logic 0 on the trailing edge of a clock pulse, effectively mimicking the D type master slave flip-flop by replacing the D input with J.
- If logic 1 is applied to both J and K, the output toggles at the trailing edge of each clock pulse, just like a toggle flip-flop.

Table 5.4.1

J	K	Q Before CK Pulse	Q After CK Pulse	Comments
0	0	0	0	No Change
0	0	1	1	
1	0	0	1	Q = J
1	0	1	1	
0	1	0	0	$\overline{Q}$ = K
0	1	1	0	
1	1	0	1	Outputs Toggle
1	1	1	0	

The JK flip-flop can therefore be called a 'programmable flip-flop' because of the way its action can be programmed by the states of J and K.

Each of the above actions are synchronised with the clock pulse, data being taken into the master flip-flop at the rising edge of the clock pulse, and output from the slave flip-flop appears at the falling edge of the clock pulse.

Note: Although the above describes the action of a master slave JK flip-flop, there are also positive edge and negative edge triggered versions available.

Asynchronous Inputs

Asynchronous inputs, which act independently of the clock pulse, are also provided by the active low inputs $\overline{PR}$ and $\overline{CLR}$. These act as (usually active low) SET and RESET inputs respectively, and as they act independently of the clock input, they give the same facilities as a simple SR flip-flop. As with the SR flip-flop, in this mode some external method is needed to ensure that these two inputs cannot both be active at the same time, as this would make both Q and $\overline{Q}$ logic 1.

JK Master-Slave Operation

A theoretical schematic circuit diagram of a JK master slave flip-flop is shown in Fig 5.4.3. Gates G1 and G2 form a similar function to the input gates in the basic JK flip-flop shown in Fig. 5.4.1, with three inputs to allow for feedback connections from Q and $\overline{Q}$.

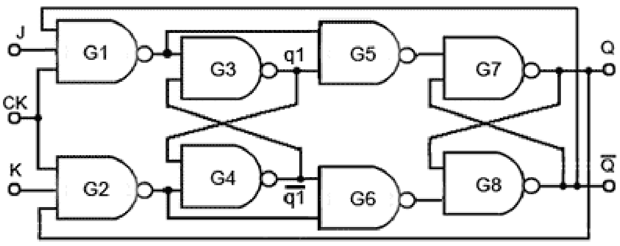


Fig 5.4.3 JK Master-Slave Flip-Flop Schematic Diagram

Gates G3 and G4 form the master flip-flop and gates G7 and G8 form the slave flip-flop. Two further gates, G5 and G6, are included between the master and slave flip-flops to transfer data from the master to the slave. The way this transfer happens is that the output of the master flip-flop is delayed for the duration of the clock pulse, by storing it, temporarily in the master flip-flop, whilst the CK pulse is high. The operation (in toggle mode) is as follows:

Loading the Master Flip-flop

With J and K both at logic 1 (the toggle mode setting), suppose that Q = 0 and $\overline{Q}$ = 1, gate G2 will be disabled as, although there are two logic 1 states on its J and CK inputs, the feedback (bottom) input of G2 will be at logic 0 due to the feedback from Q.

G1 however has a logic 1 fed back from $\overline{Q}$, which ensures that gate G1 is enabled, as all three of its inputs are now logic 1. G1 output will therefore be at logic 0 (NAND gate rules), which will cause the master flip-flop (G3 and G4) to set its q1 output to logic 1, and its $\overline{q1}$ output to logic 0.

During the time the CK input remains at logic 1, q1 and $\overline{q1}$ will remain at q1 = 1 and $\overline{q1}$ = 0, but the transfer gates G5 and G6 are inhibited because for example, if Q is currently at logic 0 and $\overline{Q}$ is at logic 1, gate G1 will have all three of its inputs at logic 1, and so its output will be 0. Because G1 output is also the active low $\overline{SET}$ input of G3, as the CK pulse went to logic 1, G3 output went to logic 0, setting the master flip-flop output q1 to logic 1.

Controlling the Transfer Gates

The logic 0 on G1 output will cause transfer gate G5 to be disabled, and combined with the logic 1 at q1 this will cause Q5 output to remain at logic 1 for the duration of the CK pulse. The input to G6 from G2 output however will be at logic 1, but as $\overline{q1}$ will now be at logic 0, transfer gate G6 will also be disabled, making its output logic 0. The data at the outputs q1 and $\overline{q1}$ cannot now be passed to the slave flip-flop for the duration of the clock pulse.

The Clock Pulse Falling Edge

Once the clock input goes low however, logic 0 is applied to the clock inputs of gates G1 and G2. The output of G1 now returns to logic 1, making both inputs to gate G5 logic 1, and causing its output to fall to logic 0. As $\overline{q1}$ is still at logic 0, gate G6 is still disabled, and so the output of G6 is at logic 1.

The Slave Flip-flop

With the output of Gate G5 at logic 0 and G6 output at logic 1, gates G7 and G8, which form a low activated SR flip-flop is set, and so Q becomes logic 1 and $\overline{Q}$ becomes logic 0.

The output conditions are now inverted, and this change is fed back to the input gates G1 and G2. However these are now both disabled because the clock input is already low, so the master flip-flop is not affected.

The arrival of the rising edge of the next clock pulse then allows the new logic levels at Q and $\overline{Q}$ into the feedback inputs to gates G1 and G2 to be fed into the master flip-flop as before, but this time Q is at logic 1, so it is gate G2 that will be enabled at the rising edge of the clock pulse.

Now, as the clock pulse goes to logic 1 the master flip-flop will be reset, q1 will go to logic 0 and at the falling edge of the clock pulse the transfer gates will pass the data to the slave flip-flop setting Q back to logic 0, so the Q and $\overline{Q}$ outputs toggle once more.

JK Flip-flop Circuit Variations

Although the standard JK flip-flop circuit shown in Fig. 5.4.3 works, the inclusion of the transfer gates limits the circuit's operation to level triggering. However Fig 5.4.4 illustrates a different method of transferring data from the master to the slave flip-flop. Instead of the transfer gates G5 and G6 used in Fig. 5.4.3, Fig. 5.4.4 uses a NOT gate to invert the positive going CK pulse triggering the master flip-flop, producing an inverted version of the clock pulse to trigger the slave flip-flop. With this modification, data is clocked into the master flip-flop at the rising edge of the CK input. Any further changes in data at J or K do not now affect the state of the master flip-flop whilst CK is high, because the feedback from Q and $\overline{Q}$ will always disable which ever of the two input gates could make a change to the master flip flop.

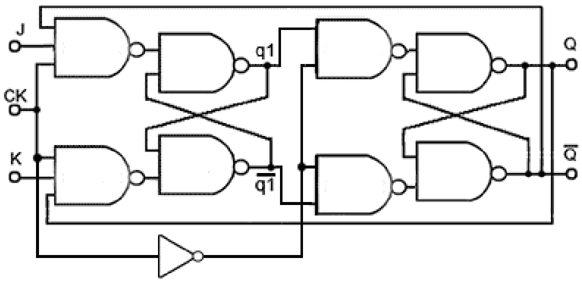


Fig. 5.4.4 Alternative Method for Clocking the JK Master Slave Flip-flop

Due to the CK inverter, at the falling edge of the CK pulse, the slave flip-flop now sees a rising edge, and the slave flip-flop accepts the data from q1 and $\overline{q1}$, toggling the states of Q and $\overline{Q}$. This master slave circuit therefore only accepts data from J and K at the rising edge of CK and outputs it on Q and $\overline{Q}$ at the falling CK edge. However, in Figs. 5.4.3 and 5.4.4, both the master and the slave flip-flops within the JK flip-flop are both simple level triggered clocked SR flip-flops. Both designs work as predicted for a JK flip-flop, in toggle mode. But in modes where J and K can change, the master flip-flop in Fig 5.4.3 accepts data from the J and K inputs whenever the CK pulse is high, allowing the master flip-flop outputs to change as long as the CK pulse is high. Therefore it is the data that is present at the instant before the CK falling edge, which is passed to the slave flip-flop. In Fig 5.4.4 the master flip-flop only accepts data at the rising edge of CK, and outputs that data at the falling edge of the CK pulse.

JK Flip-flops Using D Type Devices

Fig. 5.4.5 shows a positive edge triggered JK flip-flop (not master slave) constructed from a positive edge triggered D Type flip-flop, that uses a modified data select circuit to correctly steer the feedback from Q and $\overline{Q}$ outputs to the J and K inputs. This circuit also makes use of the asynchronous $\overline{SET}$ and $\overline{RESET}$ inputs of the D Type flip-flop, and because the D Type is edge triggered, this version of a JK flip flop is truly edge (not level) triggered. It is also possible to use a negative edge triggered D Type flip-flop to make a negative edge triggered JK flip-flop by this method.

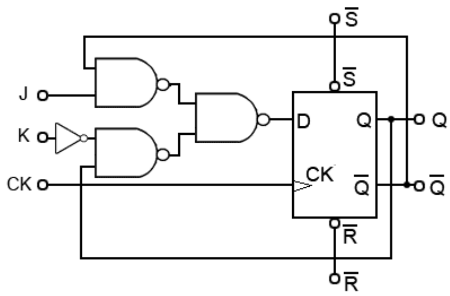


Fig 5.4.5 JK Flip-flop Using a Positive Edge Triggered D Type

JK Master Slave Flip-flop Using D Type Flip-flops

Fig. 5.4.6 shows a JK Master Slave Flip-flop using two positive edge triggered D Type flip-flops and inverting the clock pulse to convert the slave flip-flop to negative edge triggering. This design therefore, has true edge triggering on both rising and falling edges of the clock pulse, and is immune from any changes in data happening during the high or low level periods of the clock signal (except for any changes or disturbances that may occur during the t_{setup} or t_{hold} periods close to the clock pulse edges, as described in Sequential Logic Module 5.3).

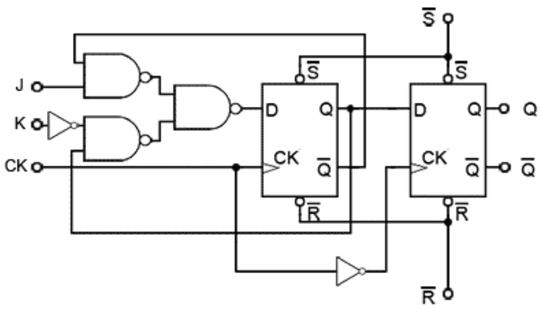


Fig. 5.4.6 JK Master Slave Flip-flop Using D Type flip-flops

5.5 CMOS Flip-flops

What you'll learn in Module 5.5

After studying this section, you should be able to:

Describe the differences between TTL and CMOS flip-flop circuits and can:

- Recognise a transmission gate (bi-lateral switch).
- Describe the action of a transmission gate.
- Describe the action of a bi-stable flip-flop using transmission gates.

Understand how CMOS circuits are used in flip-flop ICs.

- CMOS D Type flip-flops.
- CMOS JK flip-flops.

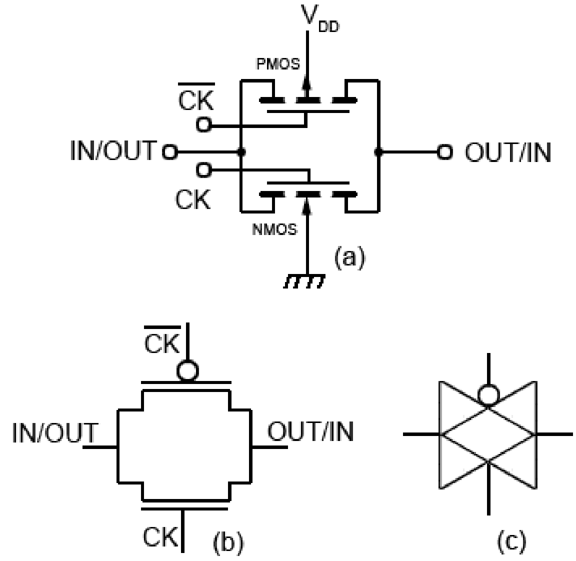


Fig. 5.5.1 The CMOS Transmission Gate

CMOS Transmission Gates

The flip-flops described so far in this module have been based on TTL technology, however many modern devices such as the 74HC and 74HCT series are CMOS ICs, which have radically different internal structures. The flip flops in CMOS ICs depend on a different type of gate, called a 'Transmission Gate' or 'Bi-lateral Switch', which make it possible to construct bi-stable flip-flops using less space within the IC, and have simpler structures than those used in TTL ICs.

Fig. 5.5.1(a) illustrates the basic structure of a transmission gate, which in some ways operates in a similar way to an electro-mechanical relay switch, except that it is much faster and very much smaller.

Like a relay, once it is energised, information can flow through the switch in either direction, therefore the signal terminals are dual purpose and can be labelled in/out and out/in. In a transmission gate this is because the signal path is via two metal oxide silicon (MOS) transistors, one of which is PMOS and the other is NMOS, connected in parallel. Signals, either digital or analogue, can pass between source and drain of these transistors in either direction when they are made to conduct by placing an appropriate voltage on the gate terminal of each transistor.

The switching signal in digital circuits is provided by the clock pulses CK and $\overline{CK}$. When the CK pulse is applied to the gate of the NMOS transistor and the $\overline{CK}$ to the PMOS transistor gate, the signal channel between the input and output terminal will conduct, and have a typical resistance of about 125Ω. In the absence of these pulse voltages however, or if they are reversed, with CK applied to the PMOS gate and $\overline{CK}$ applied to the NMOS gate, the conduction channel will exhibit an extremely high impedance (1×10^{12} ohms), virtually open circuit.

CMOS Flip-flop

Fig 5.5.2 shows a basic circuit for a single flip-flop, which operates as a level triggered D Type flip-flop. Apart from the NOT gate (N1) and the buffer (B1) controlling the CK input, the basic flip-flop uses only two NOT gates (N2 and N3) and two transmission gates (TG1 and TG2).

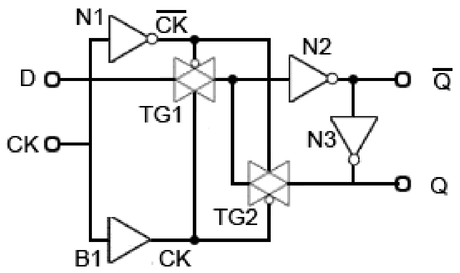


Fig. 5.5.2 Basic CMOS Flip-flop Circuit

CMOS Flip-flop Operation

The inverter N1 and the Buffer B1 create clock pulses CK and inverted clock pulses $\overline{CK}$, which (because N1 and B1 have identical propagation delays), will exactly coincide in time when applied to the transmission gates of the flip-flop circuit.

Initially, assuming that the CK and D are both at logic 0, $\overline{CK}$ will be at logic 1, so transmission gate TG1 will be in its high impedance state, preventing D from having any effect upon the flip-flop.

When CK is logic 1 and $\overline{CK}$ is logic 0, TG1 will conduct and the logic 0 from D will be inverted by N2, so the output $\overline{Q}$ will become logic 1. The logic 1 at $\overline{Q}$ will be inverted by N3 to become logic 0 at the Q output.

The logic 1 at Q will not affect the logic 0 at the input to N2 as TG2, connected in opposite polarity to the CK and $\overline{CK}$ clock signals will be turned off. This condition will remain stable irrespective of any further clock pulses being applied, as whenever TG1 is turned on, TG2 is turned off.

If input D is now changed to logic 1 between the occurrence of clock pulses, the rising edge of the first clock pulse after the change at D will turn on TG1, transmitting the logic 1 from D to the input of N2, causing $\overline{Q}$ to change to logic 1 and (via N3) Q to change to logic 0.

Whilst the CK input is high, any changes at D will be transmitted via TG1 and N2 to the outputs, indicating that the flip-flop is level triggered, but the moment the falling edge of the clock pulse occurs, TG1 will turn off and TG2 will turn on, isolating N1 and N2 from any further changes at the D input and leaving the output of N3 connected via TG2 to the input of N1.

As both these points will be at the same logic state (the logic state existing at D before the falling edge of the CK pulse) the flip-flop outputs will remain in a stable mode until the next clock pulse, when Q will take up the same state as input D once more.

Practical CMOS Flip-flop Circuits

Fig. 5.5.3 illustrates a CMOS D Type Positive Edge Triggered Master Slave Flip-flop. Notice that each pair of transmission gates TG1/ TG2 in the master flip flop, and TG3/TG4 in the slave flip-flop are connected to the clock lines in the opposite sense to each other, so that as soon as the master flip-flop accepts data from D at the rising edge of the CK pulse, the slave flip-flop is inhibited, preventing any further change at the outputs, effectively giving positive edge, rather than level triggering.

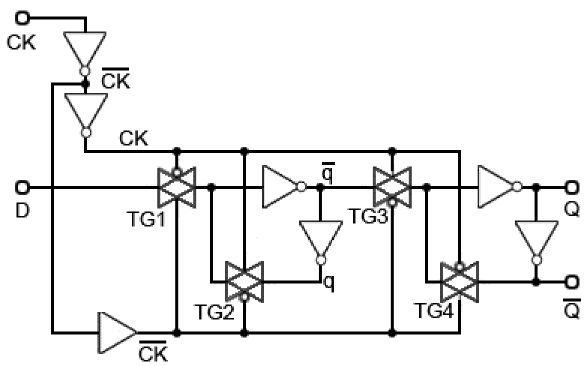


Fig. 5.5.3 CMOS D Type Positive Edge Triggered Master Slave Flip-flop

Buffered Inputs and Outputs

Buffered inputs and outputs, together with input static protection are common to most sequential ICs, but have been omitted from the schematic diagrams in this module for clarity of operation. The buffer gates and protection circuits used can be found in data sheets for individual IC designs that can be downloaded from the links at the end of this module.

CMOS D Type Flip-flop with SET and RESET

Fig. 5.5.4 shows how a CMOS D Type master slave flip-flop may be modified to include $\overline{S}$ and $\overline{R}$ inputs. In this version, NAND gates have replaced the inverters used in the master and slave flip-flops in Fig 5.5.3.

When logic 0 is applied to the $\overline{S}$ input, G3 output (and Q) is set to logic 1, (as a NAND gate output can only be logic 0 when all of its inputs are at logic 1).

Making $\overline{S}$ logic 0 also disables both the master and slave flip-flops by forcing both G3 and G2 outputs to logic 1. Therefore neither the clock nor the D inputs will have any effect on the Q and $\overline{Q}$ outputs whilst $\overline{S}$ is low.

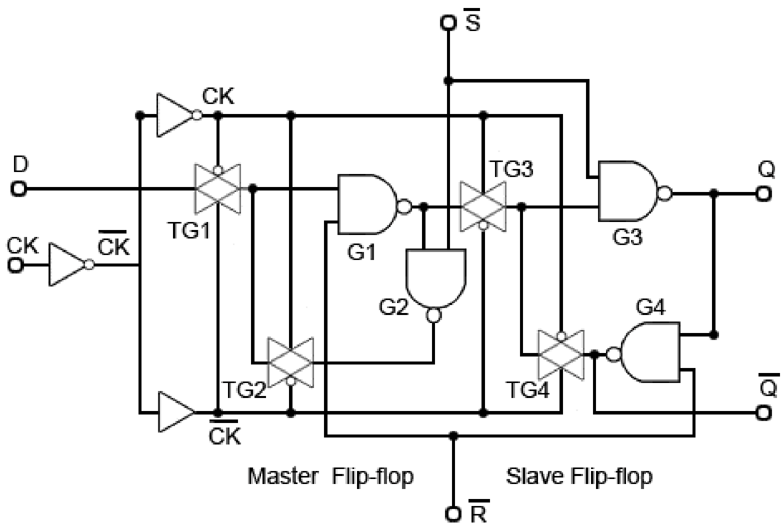


Fig. 5.5.4 CMOS Positive Edge Triggered D Type Flip-flop with SET and RESET

The RESET input ($\overline{R}$) works in the same way, by forcing the NAND gates G1 and G4 to have logic 1 outputs.

The CMOS JK Flip-flop

Converting the D Type flip-flop shown in Fig. 5.5.4 into the fully featured JK Flip-flop shown in Fig 5.5.5 is a simple matter of adding positive feedback lines from the Q and $\overline{Q}$ outputs to the two J and K input gates of the feedback steering circuit, which is simply a modified version of the basic data select circuit described in Digital Electronics Module 4.2.

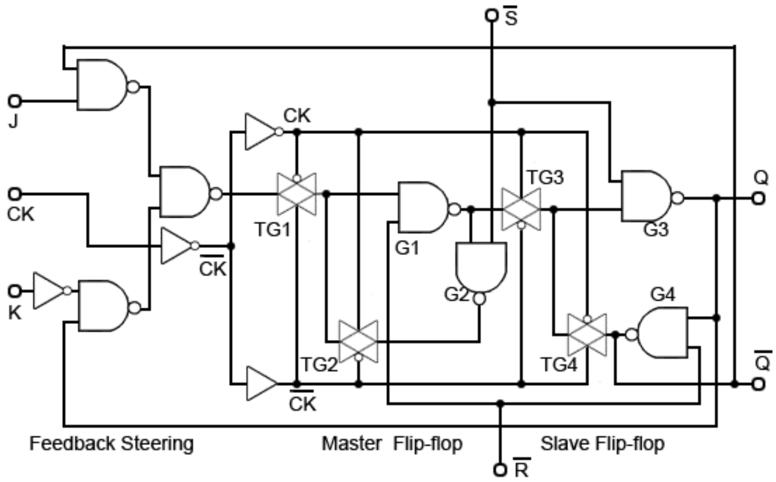


Fig. 5.5.5 CMOS Positive Edge Triggered JK Flip-flop with SET and RESET

JK Flip-flop ICs.

74HC107 Dual Negative Edge Triggered JK Flip-flop with RESET from NXP

74HC109 Dual Positive Edge Triggered JK Flip-flop with SET and RESET from NXP

74HC112 Dual Negative Edge Triggered JK Flip-flop with SET and RESET from NXP

HEF4027 Dual Positive Edge Triggered JK Flip-flop with SET and RESET from NXP

5.6 Counters

What you'll learn in Module 5.6

After studying this section, you should be able to:

Understand the operation of digital counter circuits and can:

Describe the action of asynchronous (ripple) counters using D Type flip flops.

- Up counters.
- Down counters.
- Frequency division

Understand the operation of Synchronous counters.

Describe common control features used in synchronous counters.

- BCD counters.
- Up/down control.
- Enable/disable.
- Preset and Clear.

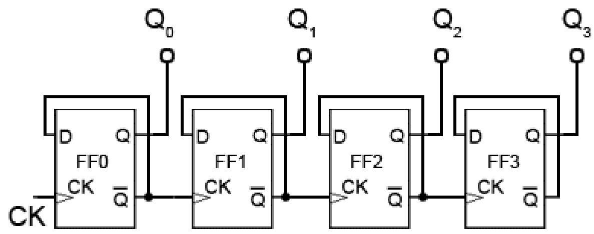


Fig. 5.6.1 Four-bit Asynchronous Up Counter

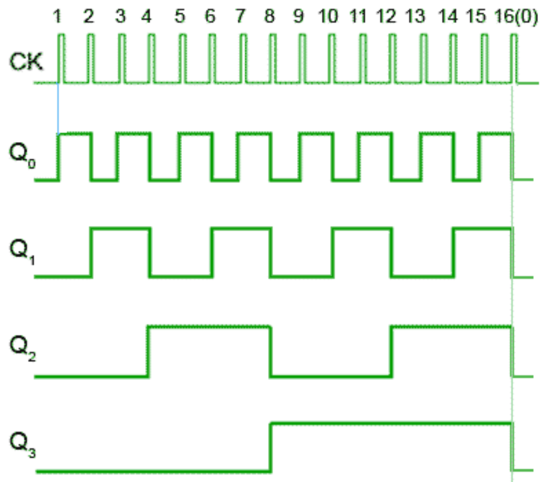


Fig. 5.6.2 Four-bit Asynchronous Counter Waveforms

Asynchronous Counters.

Counters, consisting of a number of flip-flops, count a stream of pulses applied to the counter's CK input. The output is a binary value whose value is equal to the number of pulses received at the CK input.

Each output represents one bit of the output word, which, in 74 series counter ICs is usually 4 bits long, and the size of the output word depends on the number of flip-flops that make up the counter. The output lines of a 4-bit counter represent the values 2^0, 2^1, 2^2 and 2^3, or 1,2,4 and 8 respectively. They are normally shown in schematic diagrams in reverse order, with the least significant bit at the left, this is to enable the schematic diagram to show the circuit following the convention that signals flow from left to right, therefore in this case the CK input is at the left.

Four Bit Asynchronous Up Counter

Fig. 5.6.1 shows a 4 bit asynchronous up counter built from four positive edge triggered D type flip-flops connected in toggle mode. Clock pulses are fed into the CK input of FF0 whose output, Q_0 provides the 2^0 output for FF1 after one CK pulse.

The rising edge of the $\overline{Q}$ output of each flip-flop triggers the CK input of the next flip-flop at half the frequency of the CK pulses applied to its input.

The Q outputs then represent a four-bit binary count with Q_0 to Q_3 representing 2^0 (1) to 2^3 (8) respectively.

Assuming that the four Q outputs are initially at 0000, the rising edge of the first CK pulse applied will cause the output Q_0 to go to logic 1, and the next CK pulse will make Q_0 output return to logic 0, and at the same time $\overline{Q}_0$ will go from 0 to 1.

As $\overline{Q}_0$ (and the CK input of FF1 goes high) this will now make Q_1 high, indicating a value of 2^1 (2_{10}) on the Q outputs.

The next (third) CK pulse will cause Q_0 to go to logic 1 again, so both Q_0 and Q_1 will now be high, making the 4-bit output 1100_2 (3_{10} remembering that Q_0 is the least significant bit).

The fourth CK pulse will make both Q_0 and Q_1 return to 0 and as $\overline{Q}_1$ will go high at this time, this will toggle FF2, making Q_2 high and indicating 0010_2 (4_{10}) at the outputs.

Reading the output word from right to left, the Q outputs therefore continue to represent a binary number equalling the number of input pulses received at the CK input of FF0. As this is a four-stage counter the flip-flops will continue to toggle in sequence and the four Q outputs will output a sequence of binary values from 0000_2 to 1111_2 (0 to 15_{10}) before the output returns to 0000_2 and begins to count up again as illustrated by the waveforms in Fig 5.6.2.

Four Bit Asynchronous Down Counter

To convert the up counter in Fig. 5.6.1 to count DOWN instead, is simply a matter of modifying the connections between the flip-flops. By taking both the output lines and the CK pulse for the next flip-flop in sequence from the Q output as shown in Fig. 5.6.3, a positive edge triggered counter will count down from 1111_2 to 0000_2.

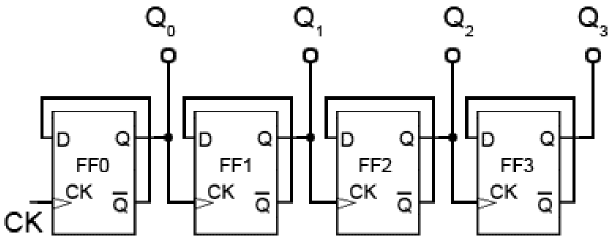


Fig 5.6.3 Four-bit Asynchronous Down Counter

Although both up and down counters can be built, using the asynchronous method for propagating the clock, they are not widely used as counters as they become unreliable at high clock speeds, or when a large number of flip-flops are connected together to give larger counts, due to the clock ripple effect.

Clock Ripple

The effect of clock ripple in asynchronous counters is illustrated in Fig. 5.6.4, which is a magnified section (pulse 8) of Fig. 5.6.2.

Fig. 5.6.4 shows how the propagation delays created by the gates in each flip-flop (indicated by the blue vertical lines) add, over a number of flip-flops, to form a significant amount of delay between the time at which the output changes at the first flip flop (the least significant bit), and the last flip flop (the most significant bit).

As the Q_0 to Q_3 outputs each change at different times, a number of different output states occur as any particular clock pulse causes a new value to appear at the outputs.

At CK pulse 8 for example, the outputs Q_0 to Q_3 should change from 1110_2 (7_{10}) to 0001_2 (8_{10}), however what really happens (reading the vertical columns of 1s and 0s in Fig. 5.6.4) is that the outputs change, over a period of around 400 to 700ns, in the following sequence:

$1110_2 = 7_{10}$

$0110_2 = 6_{10}$

$0010_2 = 4_{10}$

$0000_2 = 0_{10}$

$0001_2 = 8_{10}$

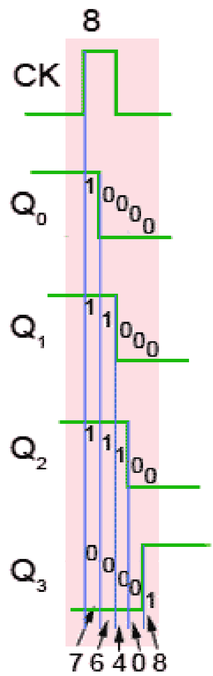


Fig.5.6.4 Timing Diagram Detail Showing Clock Ripple

At CK pulses other that pulse 8 of course, different sequences will occur, therefore there will be periods, as a change of value ripples through the chain of flip-flops, when unexpected values appear at the Q outputs for a very short time. However this can cause problems when a particular binary value is to be selected, as in the case of a decade counter, which must count from 0000_2 to 1001_2 (9_{10}) and then reset to 0000_2 on a count of 1010_2 (10_{10}).

These short-lived logic values will also cause a series of very short spikes on the Q outputs, as the propagation delay of a single flip-flop is only about 100 to 150ns. These spikes are called 'runt spikes' and although they may not all reach to full logic 1 value every time, as well as possibly causing false counter triggering, they must also be considered as a possible cause of interference to other parts of the circuit.

Although this problem prevents the circuit being used as a reliable counter, it is still valuable as a simple and effective frequency divider, where a high frequency oscillator provides the input and each flip-flop in the chain divides the frequency by two.

Synchronous Counters

The synchronous counter provides a more reliable circuit for counting purposes, and for high-speed operation, as the clock pulses in this circuit are fed to every flip-flop in the chain at exactly the same time. Synchronous counters use JK flip-flops, as the programmable J and K inputs allow the toggling of individual flip-flops to be enabled or disabled at various stages of the count. Synchronous counters therefore eliminate the clock ripple problem, as the operation of the circuit is synchronised to the CK pulses, rather than flip-flop outputs.

Synchronous Up Counter

(Note: In the diagrams under this heading only the connections essential to toggle operation and the clock are shown.)

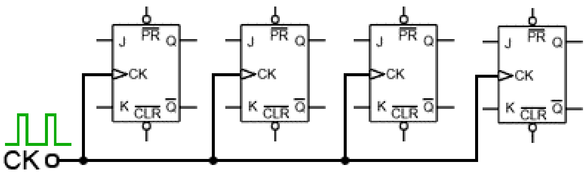


Fig.5.6.5 Synchronous Clock Connection

Fig. 5.6.5 shows how the clock pulses are applied in a synchronous counter. Notice that the CK input is applied to all the flip-flops in parallel. Therefore, as all the flip-flops receive a clock pulse at the same instant, some method must be used to prevent all the flip-flops changing state at the same time. This of course would result in the counter outputs simply toggling from all ones to all zeros, and back again with each clock pulse.

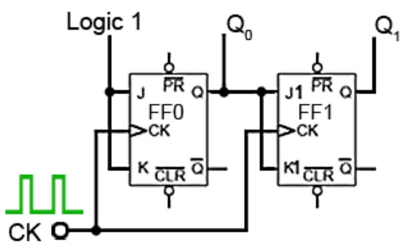


Fig. 5.6.6 The First Two Stages of a Synchronous Counter

However, with JK flip-flops, when both J and K inputs are logic 1 the output toggles on each CK pulse, but when J and K are both at logic 0 no change takes place.

Fig. 5.6.6 shows two stages of a synchronous counter. The binary output is taken from the Q outputs of the flip-flops. Note that on FF0 the J and K inputs are permanently wired to logic 1, so Q_0 will change state (toggle) on each clock pulse. This provides the 'ones' count for the least significant bit.

Table 5.6.1

CK	Q_0	Q_1	J1	K1	After the CK pulse
0	0	0	0	0	No CK pulses yet
1	1	0	1	1	Q_0 toggles making J1 & K1 = 1
2	0	1	0	0	Q_0 & Q_1 toggle making J1 & K1 = 0
3	1	1	1	1	Only Q_0 toggles making J1 & K1 = 1
4	0	0	0	0	Q_0 & Q_1 toggle making J1 & K1 = 0

On FF1 the J1 and K1 inputs are both connected to Q_0 so that FF1 output Table 5.6.1will only be in toggle mode when Q_0 is also at logic 1. As this only happens on alternate clock pulses, Q_1 will only toggle on even numbered clock pulses giving a 'twos' count on the Q_1 output.

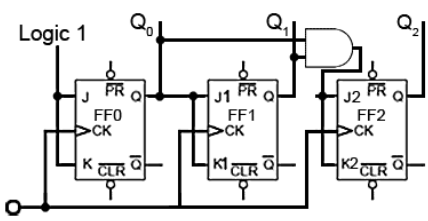


Fig. 5.6.7 Adding a Third Stage

Table 5.6.1 shows this action, where it can be seen that Q_1 toggles on the clock pulse only when J1 and K1 are high, giving a two bit binary count on the Q outputs, (where Q_0 is the least significant bit).

In adding a third flip flop to the counter however, direct connection from J and K to the previous Q_1 output would not give the correct count. Because Q_1 is high at a count of 2_{10} this would mean that FF2 would toggle on clock pulse three, as J2 and K2 would be high. Therefore clock pulse 3 would give a binary count of 111_2 or 7_{10} instead of 4_{10}.

To prevent this problem an AND gate is used, as shown in Fig. 5.6.7 to ensure that J2 and K2 are high only when both Q_0 and Q_1 are at logic 1 (i.e. at a count of three). Only when the outputs are in this state will the next clock pulse toggle Q_2 to logic 1. The outputs Q_0 and Q_1 will of course return to logic 0 on this pulse, so giving a count of 001_2 or 4_{10} (with Q_0 being the least significant bit).

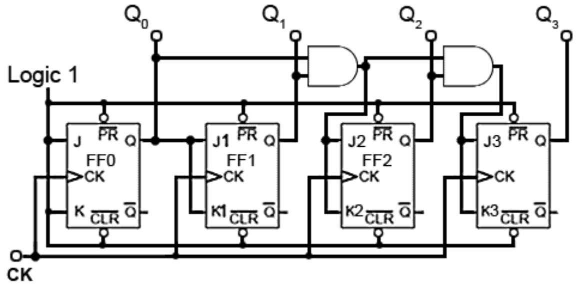


Fig. 5.6.8 Four Bit Synchronous Up Counter

Fig. 5.6.8 shows the additional gating for a four stage synchronous counter. Here FF3 is put into toggle mode by making J3 and K3 logic 1, only when Q_0 Q_1 and Q_2 are all at logic 1.

Q_3 therefore will not toggle to its high state until the eighth clock pulse, and will remain high until the sixteenth clock pulse. After this pulse, all the Q outputs will return to zero.

Note that for this basic form of the synchronous counter to work, the $\overline{PR}$ and $\overline{CLR}$ inputs must also be all at logic 1, (their inactive state) as shown in Fig. 5.6.8.

Table 5.6.2

CK	Q_3	Q_2	Q_1	Q_0	$\overline{Q_3}$	$\overline{Q_2}$	$\overline{Q_1}$	$\overline{Q_0}$
0	0	0	0	0	1	1	1	1
1	0	0	0	1	1	1	1	0
2	0	0	1	0	1	1	0	1
3	0	0	1	1	1	1	0	0
4	0	1	0	0	1	0	1	1
5	0	1	0	1	1	0	1	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	0	0
8	1	0	0	0	0	1	1	1
9	1	0	0	1	0	1	1	0
10	1	0	1	0	0	1	0	1
11	1	0	1	1	0	1	0	0
12	1	1	0	0	0	0	1	1
13	1	1	0	1	0	0	1	0
14	1	1	1	0	0	0	0	1
15	1	1	1	1	0	0	0	0

Synchronous Down Counter

Converting the synchronous up counter to count down is simply a matter of reversing the count. If all of the ones and zeros in the 0 to 15_{10} sequence shown in Table 5.6.2 are complemented, (shown with a pink background) the sequence becomes 15_{10} to 0.

Down Counter Circuit

As every Q output on the JK flip-flops has its complement on $\overline{Q}$, all that is needed to convert the up counter in Fig. 5.6.8 to the down counter shown in Fig 5.6.9 is to take the JK inputs for FF1 from the $\overline{Q}$ output of FF0 instead of the Q output. Gate TC2 now takes its inputs from the $\overline{Q}$ outputs of FF0 and FF1, and TC3 also takes its input from FF2 $\overline{Q}$ output.

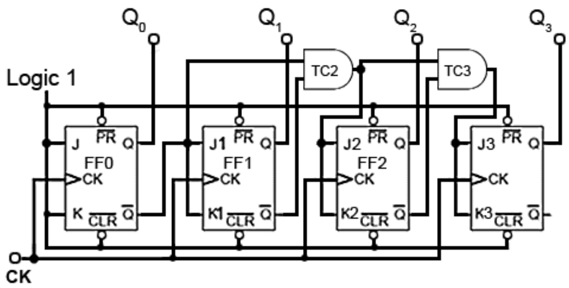


Fig. 5.6.9 Four Bit Synchronous Down Counter

Up/Down Counter

Fig. 5.6.10 illustrates how a single input, called (UP/$\overline{DOWN}$) can be used to make a single counter count either up or down, depending on the logic state at the UP/$\overline{DOWN}$ input.

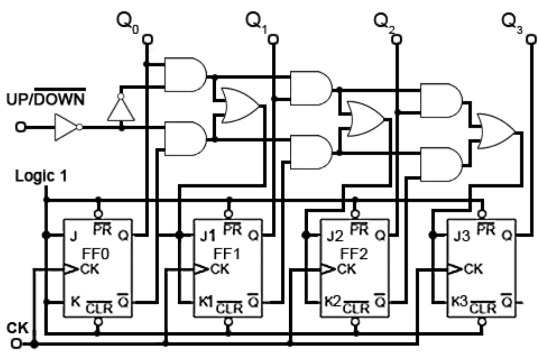


Fig. 5.6.10 Four-bit Synchronous Up/Down Counter

Each group of gates between successive flip-flops is in fact a modified data select circuit described in Combinational Logic Module 4.2, but in this version an AND/OR combination is used in preference to its

DeMorgan equivalent NAND gate circuit. This is necessary to provide the correct logic state for the next data selector.

The Q and $\overline{Q}$ outputs of flip-flops FF0, FF1 and FF2 are connected to what are, in effect, the A and B data inputs of the data selectors. If the control input is at logic 1 then the CK pulse to the next flip-flop is fed from the Q output, making the counter an UP counter, but if the control input is 0 then CK pulses are fed from $\overline{Q}$ and the counter is a DOWN counter.

Synchronous BCD Up Counter

A typical use of the $\overline{CLR}$ inputs is illustrated in the BCD Up Counter in Fig 5.6.11. The counter outputs Q_1 and Q_3 are connected to the inputs of a NAND gate, the output of which is taken to the $\overline{CLR}$ inputs of all four flip-flops. When Q_1 and Q_3 are both at logic 1, the output terminal of the limit detection NAND gate (LD1) will become logic 0 and reset all the flip-flop outputs to logic 0.

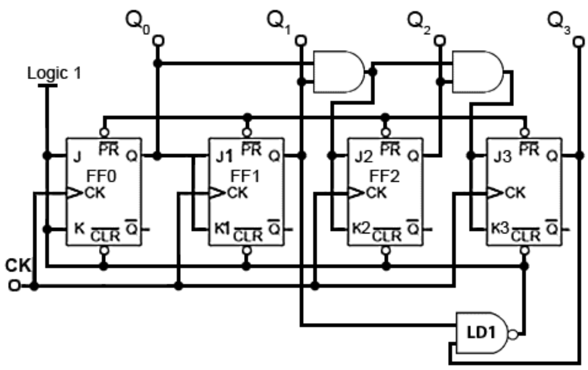


Fig. 5.6.11 Synchronous BCD Up Counter

Because the first time Q_1 and Q_3 are both at logic 1 during a 0 to 15_{10} count is at a count of ten (1010_2), this will cause the counter to count from 0 to 9_{10} and then reset to 0, omitting 10_{10} to 15_{10}.

The circuit is therefore a BCD_{8421} counter, an extremely useful device for driving numeric displays via a BCD to 7-segment decoder etc. However by re-designing the gating system to produce logic 0 at the $\overline{CLR}$ inputs for a different maximum value, any count other than 0 to 15 can be achieved.

If you already have a simulator such as Logisim installed on your computer, why not try designing an Octal up counter for example.

Counter IC Inputs and Outputs

Although synchronous counters can be, and are built from individual JK flip-flops, in many circuits they will be ether built into dedicated counter ICs, or into other large scale integrated circuits (LSICs).

For many applications the counters contained within ICs have extra inputs and outputs added to increase the counters versatility. The differences between many commercial counter ICs are basically the different input and output facilities offered. Some of which are described below. Notice that many of these inputs are active low; this derives from the fact that in earlier TTL devices any unconnected input would float up to logic 1 and hence become inactive. However leaving inputs un-connected is not good practice, especially CMOS inputs, which float between logic states, and could easily be activated to either valid logic state by random noise in the circuit, therefore ANY unused input should be permanently connected to its inactive logic state.

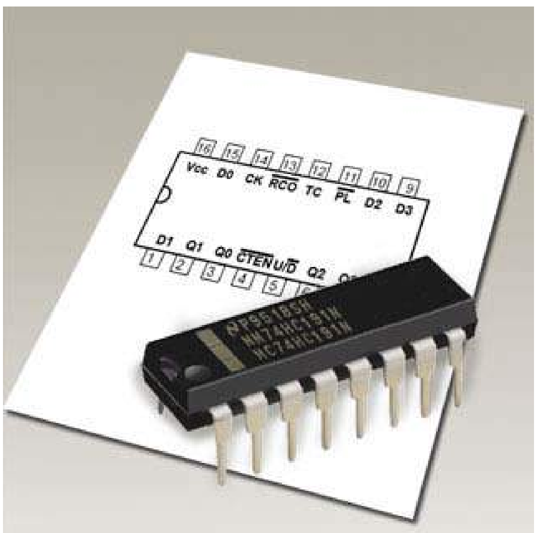


Fig. 5.6.12 Counter IC Inputs and Outputs

Enable Inputs

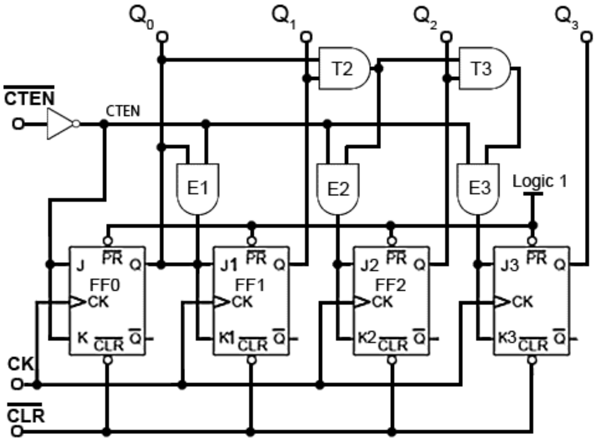


Fig. 5.6.13 Synchronous Up Counter with Count Enable and Clear Inputs

ENABLE ($\overline{EN}$) inputs on counter ICs may have a number of different names, e.g. Chip Enable ($\overline{CE}$), Count Enable ($\overline{CTEN}$), Output Enable ($\overline{OE}$) etc., each denoting the same or similar functions.

Count Enable ($\overline{CTEN}$) for example, is a feature on counter integrated circuits, and in the synchronous counter illustrated in Fig 5.6.13, is an active low input. When it is set to logic 1, it will prevent the count from progressing, even in the presence of clock pulses, but the count will continue normally when $\overline{CTEN}$ is at logic 0.

A common way of disabling the counter, whilst retaining any current data on the Q outputs, is to inhibit the toggle action of the JK flip-flops whilst $\overline{CTEN}$ is inactive (logic 1), by making the JK inputs of all the flip-flops logic 0. However, as the logic states of the JK inputs of FF1, FF2 and FF3 depend on the state of the previous Q output, either directly or via gates T2 and T3, in order to preserve the output data, the Q outputs must be isolated from the JK inputs whenever $\overline{CTEN}$ is logic 1, but the Q outputs must connect to the JK inputs when $\overline{CTEN}$ is at logic 0 (the count enabled state).

This is achieved by using the extra (AND) enable gates, E1, E2 and E3, each of which have one of their inputs connected to CTEN (the inverse of $\overline{CTEN}$). When the count is disabled, CTEN and therefore one of the inputs on each of E1, E2 and E3 will be at logic 0, which will cause these enable gate outputs, and the flip-flop JK inputs to also be at logic 0, whatever logic states are present on the Q outputs, and also at the other enable gate inputs. Therefore whenever $\overline{CTEN}$ is at logic 1 the count is disabled.

When $\overline{CTEN}$ is at logic 0 however, CTEN will be logic 1 and E1, E2 and E3 will be enabled, causing whatever logic state is present on the Q outputs to be passed to the JK inputs. In this condition, when the next clock pulse is received at the CK input the flip-flops will toggle, following their normal sequence.

Preset and Clear

Fig. 5.6.13 also makes use of the $\overline{CLR}$ input to allow the counter outputs to be reset to zero at any time. Asynchronous PRESET ($\overline{PR}$) and CLEAR ($\overline{CLR}$) are inputs available on most JK and D type flip-flops and are very useful in the design of counters. The $\overline{CLR}$ inputs of several flip-flops can be used together to reset the count to zero by making the $\overline{CLR}$ inputs of the flip-flops logic 0 at any time, independently of the logic state of the clock input. This was done in the BCD up counter in Fig. 5.6.10.

Alternatively the $\overline{PR}$ inputs of the flip-flops could be used in a BCD down counter to set all of the Q outputs to logic 1 when a count of 0000 is sensed.

However, using the asynchronous inputs in a synchronous circuit design must be done with care, as it is possible, depending on the exact time during the clock cycle at which either asynchronous input causes a change, that unwanted spikes, and/or timing errors could be generated. A preferable, and more flexible method of setting or resetting each of the Q outputs separately uses a parallel loading method, where the individual flip-flops within the counter can be set or reset, depending on the logic states at data inputs (one per flip-flop). ICs are available that support either synchronous or asynchronous loading to suit various design requirements.

Asynchronous Parallel Load

While common $\overline{PR}$ and $\overline{CLR}$ inputs can produce outputs of 0000 or 1111, a PARALLEL LOAD ($\overline{PL}$) input will allow any value to be loaded into the counter. Using a separate DATA input for each flip-flop, and a small amount of extra logic, a logic 0 on the $\overline{PL}$ will load the counter with any pre-determined binary value before the start of, or during the count. A method of achieving asynchronous parallel loading on a synchronous counter is shown in Fig. 5.6.14.

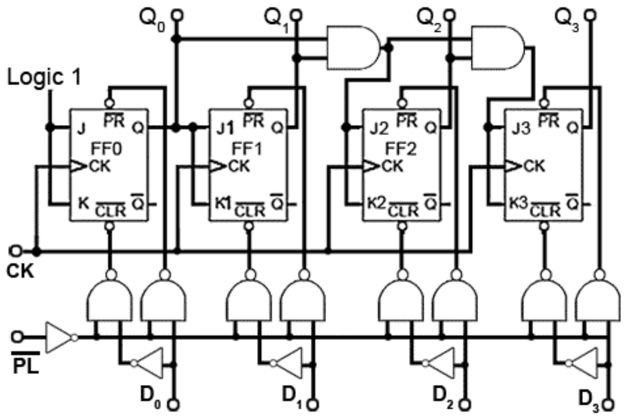


Fig. 5.6.14 Asynchronous Parallel Load

Load Operation

The binary value to be loaded into the counter is applied to inputs D_0 to D_3 and a logic 0 pulse is applied to the $\overline{PL}$ input. This logic 0 is inverted and applied to one input of each of the eight NAND gates to enable them. If the value to be loaded into a particular flip-flop is logic 1, this makes the inputs of the right hand NAND gate 1,1 and due to the inverter between the pair of NAND gates for that particular input, the left hand NAND gate inputs will be 1,0.

The result of this is that logic 0 is applied to the flip-flop $\overline{PR}$ input and logic 1 is applied to the $\overline{CLR}$ input. This combination sets the Q output to logic 1, the same value that was applied to the D input. Similarly if a D input is at logic 0 the output of the left hand NAND gate of the pair will be Logic 0 and the right hand gate output will be logic 1, which will clear the Q output of the flip-flop. Because the $\overline{PL}$ input is common to each pair of load NAND gates, all four flip-flops are loaded simultaneously with the value, either 1 or 0 present at its particular D input.

Multiple Inputs and Outputs

Modifications such as those described in this module make the basic synchronous counter much more versatile. Both TTL and CMOS synchronous counters are available in the 74 series of ICs containing usually 4-bit counters with these and other modifications for a wide variety of applications. Fig 5.6.15 shows how all the input functions described above, plus some important outputs such as Ripple Carry ($\overline{RC}$) and Terminal Count (TC) can be combined to form a single synchronous counter IC.

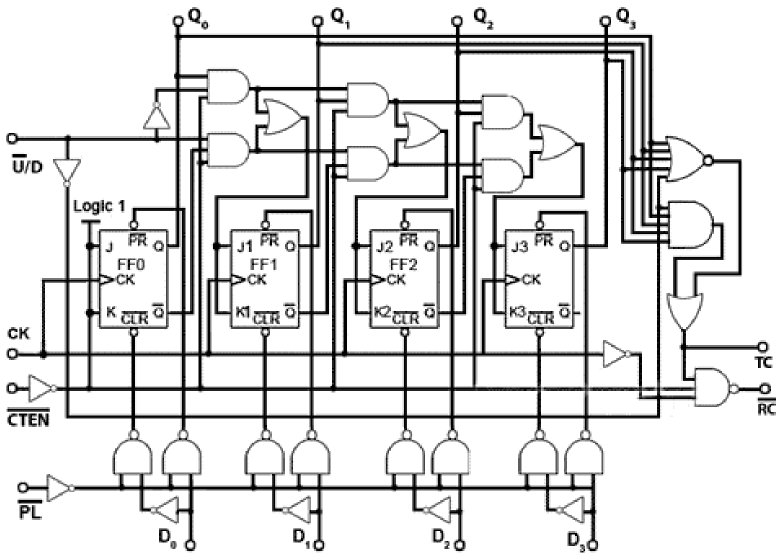


Fig. 5.6.15 Synchronous Up/Down Counter with Multiple Inputs and Outputs

A typical single synchronous IC such as the 74HC191 four-bit binary up/down counter also uses these input and output functions, which are designated on NXP versions (Fig. 5.6.16) as follows:

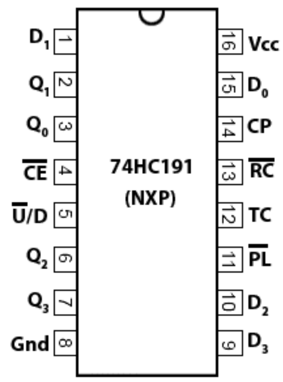


Fig. 5.6.16
74HC191 Pinout

Inputs

- D_0, D_1, D_2 and D_3 (Load inputs) - A 4 bit binary number may be loaded into the counter via these inputs when the Parallel Load input $\overline{PL}$ is at logic 0.
- $\overline{CE}$ (Count Enable) - Allows the count to proceed when at 0. Stops count without resetting when at logic 1.
- $\overline{U}$/D (Up/Down) - Counts up when 0, down when at logic 1.
- CP - Clock Pulse input.

Outputs

- Q_0, Q_1, Q_2 and Q_3 - Four bit binary output.
- TC (Terminal Count) - Also called MAX/MIN in some versions, gives a logic 1 pulse, equal in width to one full clock cycle, at each change over of the most significant bit (signifying that the count has overflowed beyond the end of an up or down count). TC can be used to detect the end of an up or down count, and as well as being available as an output, TC is used internally to generate the Ripple Carry output.
- $\overline{RC}$ (Ripple Carry) - Outputs a logic 0 pulse, equal in width to the low portion of the clock cycle at the end of a count, and when connected to the clock input of another 74HC191 IC it acts as a 'carry' to the next counter.

Cascading Synchronous Counters

Connecting Synchronous counters in cascade, to obtain greater count ranges, is made simple in ICs such as the 74HC191 by using the ripple carry ($\overline{RC}$) output of the IC counting the least significant 4 bits, to drive the clock input of the next most significant IC, as show in red in Fig. 5.6.17.

Although it may appear that either the TC or the $\overline{RC}$ outputs could drive the next clock input, the TC output is not intended for this purpose, as timing issues can occur.

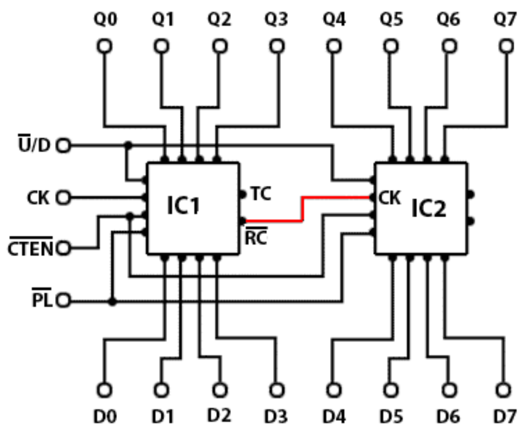


Fig. 5.6.17 Connecting the 74HC191 in Cascade

Synchronous vs. Asynchronous Counters

Although synchronous counters have a great advantage over asynchronous or ripple counters in regard to reducing timing problems, there are situations where ripple counters have an advantage over synchronous counters.

When used at high speeds, only the first flip-flop in the ripple counter chain runs at the clock frequency. Each subsequent flip-flop runs at half the frequency of the previous one. In synchronous counters, with every stage operating at very high clock frequencies, stray capacitive coupling between the counter and other components and within the counter itself is more likely occur, so that in synchronous counters interference can be transferred between different stages of the counter, upsetting the count if adequate decoupling is not provided. This problem is reduced in ripple counters due to the lower frequencies in most of the stages.

Also, because the clock pulses applied to synchronous counters must charge, and discharge the input capacitance of every flip-flop simultaneously; synchronous counters having many flip-flops will cause large pulses of charge and discharge current in the clock driver circuits every time the clock changes logic state. This can also cause unwelcome spikes on the supply lines that could cause problems elsewhere in the digital circuitry. This is less of a problem with asynchronous counters, as the clock is only driving the first flip-flop in the counter chain.

Asynchronous counters are mostly used for frequency division applications and for generating time delays. In either of these applications the timing of individual outputs is not likely to cause a problem to external circuitry, and the fact that most of the stages in the counter run at much lower frequencies than the input clock, greatly reduces any problem of high frequency noise interference to surrounding components.

Counter ICs

Asynchronous (Ripple) Counters:

74HC390 - Dual decade ripple counter.

74HC393 - Dual 4-stage binary ripple counter.

74HC4040 - 12-Stage binary ripple counter.

74HC93 - 4-Bit binary ripple counter.

CD4060 - 14-Stage binary counter plus oscillator.

HEF4042B - 7-Stage binary ripple counter.

Synchronous Counters:

74HC160 - Pre-settable synchronous BCD counter with asynchronous reset.

74HC161 - 4-Bit synchronous BCD counter with asynchronous reset and synchronous load.

74HC163 - 4-Bit synchronous binary counter with asynchronous reset and synchronous load.

74HC191 - 4-bit synchronous binary up/down counter with asynchronous reset and load.

74HC192 - 4-Bit synchronous BCD counter with asynchronous reset and load.

74HC193 - 4-Bit synchronous binary counter with asynchronous reset and load.

CD4017/4022B - 4-Stage synchronous counters with Decade (1 of 10) or Octal (1 of 8) outputs.

5.7 Registers

What you'll learn in Module 5.7

After studying this section, you should be able to:

Understand the operation of digital parallel in/parallel out (PIPO) registers.

Describe the action of serial and parallel shift registers.

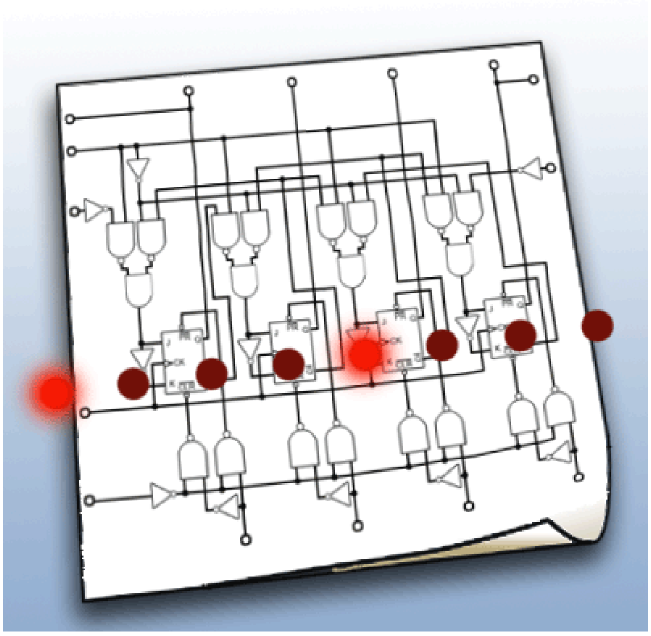
- Serial in/Serial out (SISO).
- Serial in/Parallel out (SIPO).
- Parallel in/Serial out PISO.

Understand the operation of

bi-directional shift registers.

Recognise common features used in shift registers.

Recognise register ICs



Registers

An electronic register is a form of memory that uses a series of flip-flops to store the individual bits of a binary word, such as a byte (8 bits) of data. The length of the stored binary word depends on the number of flip-flops that make up the register. A simple 4-bit register is illustrated in Fig. 5.7.1 and consists of four D Type flip-flops, sharing a common clock input, providing synchronous operation ensuring all bits are stored at exactly the same time.

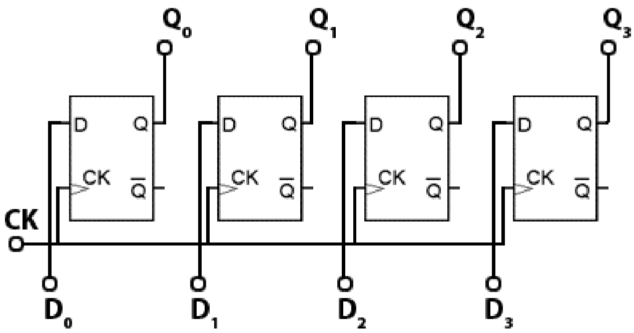


Fig. 5.7.1 Parallel In/Parallel Out (PIPO) Register

The binary word to be stored is applied to the four D inputs and is remembered by the flip-flops at the rising edge of the next clock (CK) pulse. The stored data can then be read from the Q outputs at any time, as long as power is maintained, or until a change of data on the D inputs is stored by a further clock pulse, which overwrites the previous data.

Different types of register are generally classified by the method of storage and readout used; this basic form of register is therefore classified as a 'Parallel In/Parallel Out' (PIPO) register.

Shift Registers.

Shift registers have a similar structure to the PIPO register but have the added ability to shift the stored binary word left or right, one bit at a time. This makes them extremely useful for many applications. They are used in handling serial data and converting it to parallel form or back again to serial form, and therefore are an essential component in communication systems. Shift registers are also essential in arithmetic circuits where binary numbers may be shifted right (and so divided by two), or left (multiplied by two) as part of a calculation. Shift registers can be used to delay the passage of data at a particular point in a circuit. As the data is shifted one bit at a time from input to output, the amount of delay will depend on the number of flip-flops in the register and the frequency of the clock pulses driving the shift register. Because a number of serial bits of data are stored as they enter the input, and are then recovered from the output at some later time, this action can also be described as a serial memory, or as a digital delay line.

The simple storage register shown in Fig. 5.7.1 can be modified to a shift register by connecting the output of one flip-flop into the input of the next, as shown in Fig. 5.7.2. The basis of shift register circuits is the D-type flip-flop, but the clocked SR or the JK flip-flop may also be converted to D-types by the inclusion of an inverter between S and R or between J and K. In all cases the clock input is in synchronous mode.

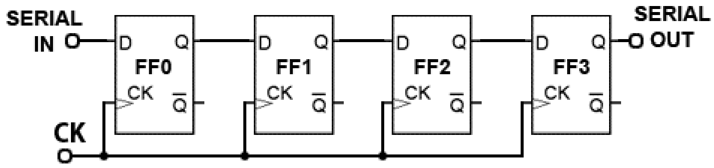


Fig. 5.7.2 Serial In/Serial Out (SISO) Shift Register

The serial input of the shift register in Fig. 5.7.2 is the D input of the first flip-flop, and the serial output is the Q output of the last flip-flop in the chain. The logic state at the serial input appears at the output, a number of clock pulses (equal to the number of flip flops) later.

Modes of Shift Register Operation.

SISO

Referring to Fig. 5.7.2, if the serial input goes from 0 to 1 just before the first clock pulse, the Q output of flip-flop FF0 will go high at the rising edge of clock pulse 1. At the next clock pulse rising edge, the logic 1 will be transferred to FF1 and so on until it reaches FF3, the serial output. A timing diagram of this operation is shown in Fig. 5.7.3, and Table 5.7.1 is a state-table showing the states of the four outputs after each clock pulse. After each CK pulse one more flip-flop output is set to 1 until (after 4 pulses) all Q outputs, including the serial output, are at logic 1. This form of operation is called ‘serial in/serial out’ or SISO.

Table 5.7.1

	After CK Pulse				
	0	1	2	3	4
Q_0	0	1	1	1	1
Q_1	0	0	1	1	1
Q_2	0	0	0	1	1
Q_3	0	0	0	0	1

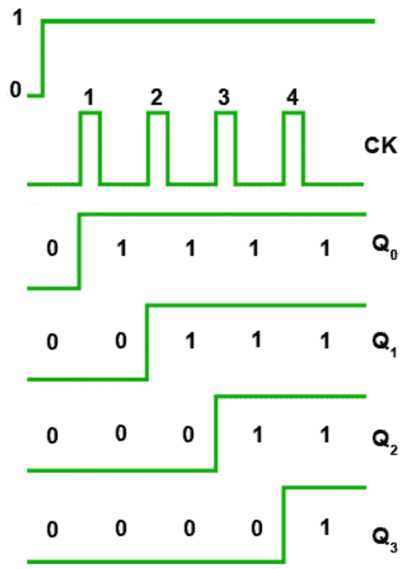


Fig. 5.7.3 Timing Diagram for SISO Operation

SIPO

In Fig. 5.7.4 the shift register is modified to include additional Q outputs from each flip-flop, so allowing the register to output data in parallel form. The register could therefore now be called a ‘Serial In/Parallel Out’ or SIPO register. This format is the basis for converting serial data to parallel data.

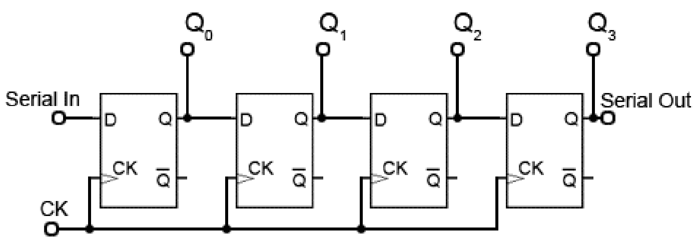


Fig. 5.7.4 Serial In/Parallel Out (SIPO) Shift Register

PISO

If use is also made of the $\overline{Q}$ output, and the additional preset and clear inputs available on many flip-flops, the shift register could be made more versatile still.

Fig. 5.7.5 shows a shift register modified to enable it to be loaded with a 4-bit parallel number, which may then be shifted right to appear at the serial output one bit at a time. As the 'Parallel In/Serial Out' or PISO register also has a serial input, it can also be used as a SISO register, and if extra outputs from each Q output were also included, the register would also have Serial In/Parallel Out (SIPO) operation.

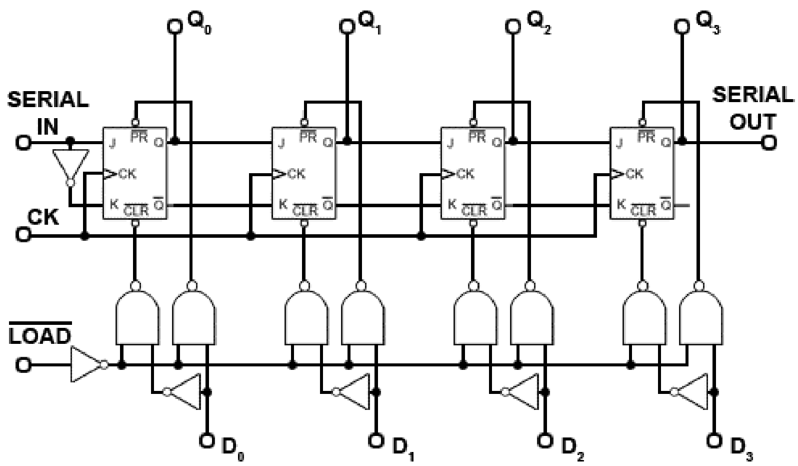


Fig. 5.7.5 Multiple Mode (SISO, SIPO, PISO, PIPO) Shift Register

Loading Parallel Data

If the $\overline{LOAD}$ input is taken to logic 0, the LOAD control line connected to the four pairs of NAND gates associated with the four flip-flops will be at logic 1, and all four pairs of NAND gates will be enabled. Therefore a logic 1 appearing on any of the D inputs will be inverted by the NOT gate connected to the D input, making the inputs to the left hand NAND gate of the relevant pair of gates, logic 1 and logic 0. This will cause logic 1 to be applied to the $\overline{CLR}$ input of the flip-flop.

The right hand NAND gate of the pair will have both inputs at logic 1, due to the logic 1 on LOAD line and logic 1 on the D input, and so will output logic 0 (NAND gate rules) to the $\overline{PR}$ input of the flip-flop, setting the Q output to logic 1.

If the D input is at logic 0, the left hand gate of the NAND gate pair will output logic 0 and the right hand NAND gate will output logic 1, causing the $\overline{CLR}$ input to clear the Q output of the relevant flip-flop to logic 0.

Notice that as JK flip-flops are being used in this design, a NOT gate is connected between J and K of the first flip-flop of the chain to make the JK flip-flop mimic a D Type. The remaining flip-flops of the shift register have J and K connected to the previous Q and $\overline{Q}$ outputs, so will also be at opposite logic states.

A 4-bit reversible shift register.

The shift register in Fig 5.7.5 could be operated as:

- A parallel in/parallel out register. (PIPO)
- A Serial in/serial out register. (SISO)
- A serial in/parallel out register. (SIPO)
- A parallel in/serial out register. (PISO)

However Fig 5.7.5 can only shift data in one direction, i.e. left to right. To be truly versatile it could be an advantage to be able to shift data in both directions and in any of the four shift register operating modes. Fig. 5.7.6 achieves this by adding data steering circuitry.

The gating arrangement at the bottom of Fig 5.7.6 (gates G1 to G13) is exactly the same as that described above in Fig. 5.7.5, and these gates control the loading of parallel data.

Gates G14 to G28 in Fig 5.7.6 control the direction of data flow through the register. The JK flip-flops use the inverter gates G29 to G32 to ensure that J and K are at opposite logic states, so the flip-flops are mimicking D Type operation, with J being used as the data input. Notice also that the clock is connected in the familiar synchronous mode.

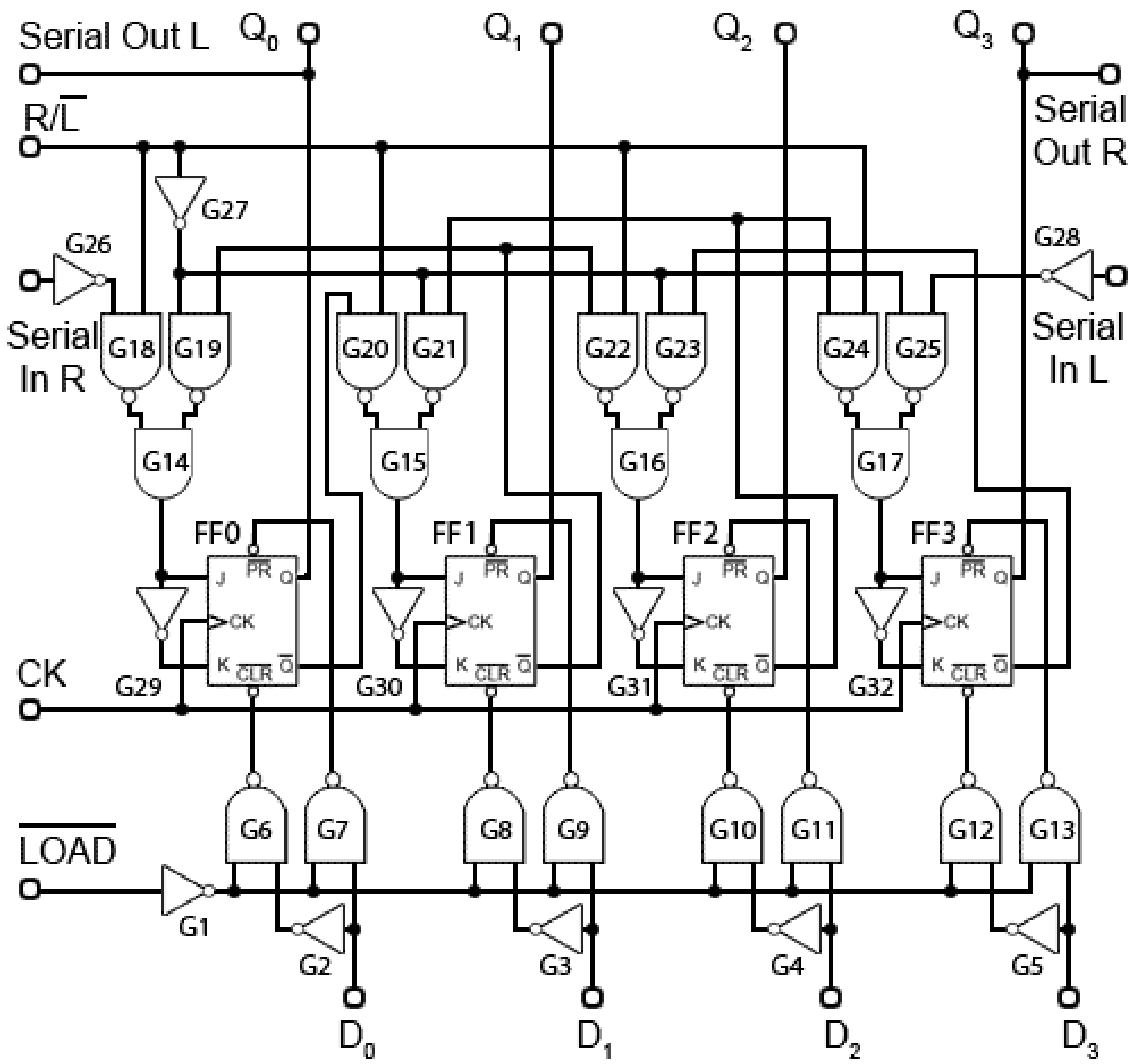


Fig. 5.7.6 4-Bit Reversible PIPO/PISO/SISO/SIPO Shift Register

Operation.

In any of the modes involving serial operation, data may be shifted left or shifted right by the application of a suitable logic level at the shift control (R/$\overline{L}$) input.

With a logic 1 at this input the register is in the shift right mode, and data is taken into the 'Serial in R' input to be shifted right by application of successive clock pulses, appearing as parallel data, changing with each clock pulse, on the flip flop Q outputs. After four clock pulses the data begins to appear in serial form on the Q_3 output, which is also the 'Serial Out R' output.

The logic 1 on the shift control (R/$\overline{L}$) enables gates G18, 20, 22 & 24, but because the logic 1 is inverted by G27, gates G19, 21, 23 & 25 are disabled.

The path of serial data (e.g. a logic 1) from left to right is as follows; the logic 1 appearing at the input to G26 is inverted and passes through G18 which re-inverts it to logic 1 and, as g19 is disabled its output must also be at logic 1. Both inputs to the AND gate G14 are at logic 1 and therefore so is its output, making the J input of FF0 logic 1.

On the arrival of a clock pulse, the logic 1 input to FF0 will appear on the output Q_0. Its inverse (logic 0) will also appear on the $\overline{Q}$ output of FF0. This logic 0 forms the input to the next multiplexer arrangement, gates G20, 21 & 15. As G20 is enabled (and G21 disabled) the logic 0 becomes logic 1 at G15 output and so is fed to the J input of FF1. This method is used to transfer data to each flip-flop in the chain.

To achieve shift left operation, the shift control (R/$\overline{L}$) is set to logic 0 and so enables gates G19, 21, 23 & 25 while disabling gates G18, 20, 22 & 24. Therefore the $\overline{Q}$ output of FF3 is connected via G23 and G16 to the D input of FF2, the $\overline{Q}$ output of FF2 is connected to the J input of FF1 via G21 and G15 (remember that G24 is disabled, so FF3 is isolated from this path). Finally, the $\overline{Q}$ output of FF1 is connected via G19 and G14 to the J input of FF0, the Q_0 output of which is also the 'Serial Out L' output. The ability to shift data in either direction, together with the parallel input and output facilities make this register a very versatile device.

It is common to connect shift register ICs in cascade, using the serial output of one register to connect to the serial input of the next register in the chain. For this reason both the data and clock inputs and outputs of register ICs are normally buffered.

Some examples from the many commercially available IC registers using these and similar methods, available in both CMOS and TTL versions, are listed below.

74HC164 8-Bit SIPO Shift register from NXP

74HC594 8-Bit SIPO/SISO with PIPO output storage register and dual clocks - from NXP.

74HC595 8-Bit SIPO/SISO with tri-state output PIPO storage register and dual clocks - from NXP.

HEF4014B PISO Register with 8-bit synchronous parallel LOAD and outputs from Q_5, Q_6 & Q_7 only – NXP

CD4031B 64 Stage SISO shift register with re-circulation mode – from Texas Instruments.

5.8 Arithmetic & Logic Unit

What you'll learn in Module 5.8

After studying this section, you should be able to:

Understand the operation of a basic ALU circuit.

Understand the relationship between Binary Arithmetic and digital circuits.

- Twos complement arithmetic.
- Adders & error flags.
- Logic operations with shift registers.

Use free software to investigate ALU operations

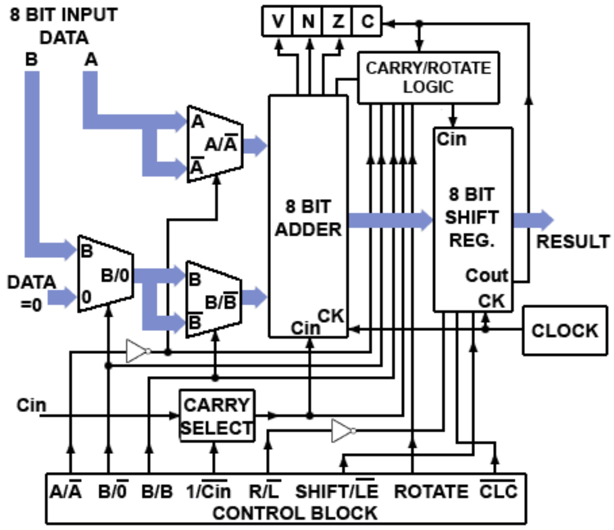


Fig. 5.8.1 ALU Block Diagram

Connecting Digital Circuits Together.

Digital Electronics Modules 2 to 5 have described how basic logic gates may be combined, not only to perform standard logic functions, but to build circuits that can perform complex logic tasks. Both small scale integrated (SSI) and medium scale integrated (MSI) chips are available in many forms, that can be directly connected together to make very complex circuits. It is this inter-connectivity that makes digital electronics so powerful and so versatile.

The standard circuits described in modules 2 to 5, both combinational and sequential, can be used to perform arithmetic operations such as addition, subtraction and counting, as well as logical operations such as combining data sources (multiplexing) and shifting bits left or right within a binary word.

As explained in Module 1, binary arithmetic is normally carried out electronically by using twos complement notation. The most common and versatile method of carrying out such operations is in an Arithmetic and Logic Unit (ALU), a circuit that forms the heart of any calculating or computing system.

The Arithmetic and Logic Unit

A simplified ALU is illustrated in Fig 5.8.1, which uses an arrangement of both combinational and sequential circuits from those described in modules 2 to 5. Their purpose is to perform the basic (though still complex) binary arithmetic described in Module 1.

Data passing through the ALU circuit does so on a system of buses, shown by the broad arrows in Fig. 5.8.1. These buses consist of groups of wires (usually as 8 parallel bits in simple systems) each carrying a single byte of binary data. In this system, data word A is the primary data source, and data word B is the secondary data source that may be added to, or subtracted from word A.

The ALU can also perform other operations. It can increment, add 1 to word A, or decrement, subtract 1 from it. By complementing (inverting) the logic value of individual bits of the data word A and adding 1 to the result, it is possible to use twos complement arithmetic to perform subtractions.

The shift register at the ALU output can also perform a 'logical shift-left' on word A by shifting the 8 bits consecutively into the carry bit, alternatively the shift register can create a rotating pattern of bits, rotating left, and using the carry bit as a ninth bit in the sequence, or rotate the 8 bits right ignoring the carry bit. Any of these functions can be selected by the control block, using various combinations of the eight control lines shown in Fig. 5.8.1.

Putting the correct pattern of 1s and 0s (the control word) on the control lines will cause the ALU to perform the required arithmetic or logical operation on the data being input at A and B. With a control word of 8-bits, this could potentially allow up to 256 different combinations, or control words, which would be more than ample, even for very complex microprocessors or micro controllers. However this basic ALU needs only eight control words to control the different operations available.

To see the ALU operate as described below, you can download our free, fully interactive Logisim ALU circuit (assuming you have the free Logisim Digital Simulator installed on your desktop or laptop computer), see our extra Logisim page for details.

Simulating the ALU

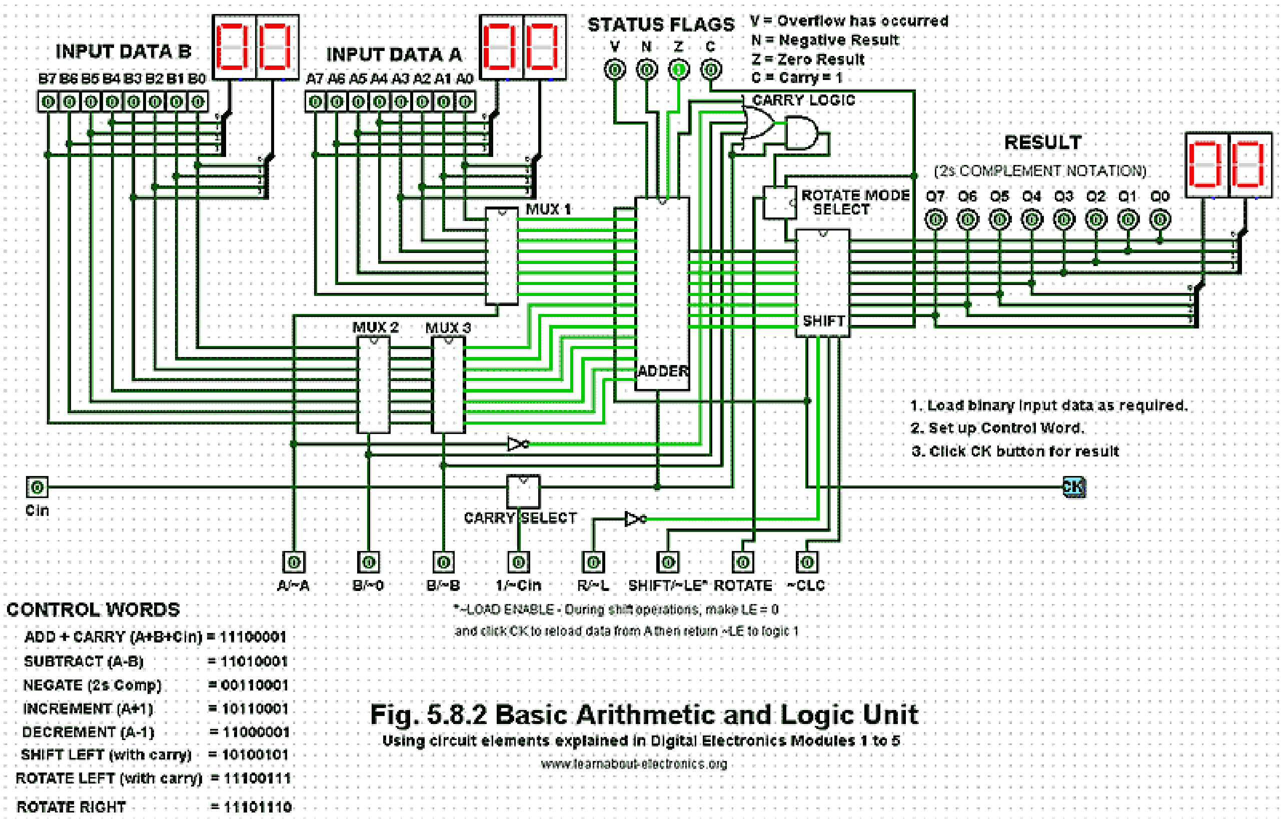


Fig. 5.8.2 Basic Arithmetic and Logic Unit – Logisim Simulation

The Component Parts

Any of the component parts of the Logisim design can be examined in detail by double clicking on the component (in simulation mode). To return to the main document, click 'main' in the component menu at the left of the screen.

Note: In this section the tilde character ~ is used where necessary to indicate NOT (e.g. ~LE = $\overline{LE}$) to match the usage in the Logisim simulations.

Multiplexers

MUX 1 and MUX 3 are identical 8 bit multiplexers that select either the input data word A (MUX 1) or data word B (MUX 3) or their internally generated complement, as shown in Fig. 5.8.3.

MUX 2 is a similar design but selects either the data word B or the zero value 00_{HEX} , as shown in Fig. 5.8.4.

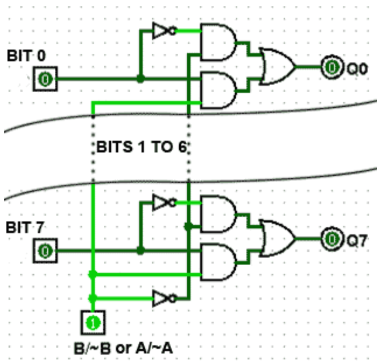


Fig. 5.8.3 MUX 1 and MUX 3

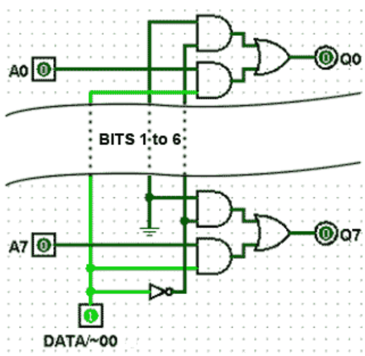


Fig. 5.8.4 MUX 2

8-Bit Adder

The adder component is an 8-bit ripple carry adder; real ALUs would normally feature a ‘carry-look-ahead’ adder, allowing for high-speed operation. However for this example the much simpler ripple carry adder is adequate, as the operation is totally manual.

The adder component is illustrated in Fig. 5.8.5 and consists of eight full-adder circuits with additional logic consisting of an XOR gate to detect overflow errors, and an 8-input NOR gate to detect a zero result.

Negative results are indicated by sampling the most significant bit of the ‘sum’ output, and a ‘carry’ is indicated by sampling the carry output of the most significant full adder.

Four D type flip-flops are used as ‘flag’ outputs to indicate the current state of the ALU after each operation.

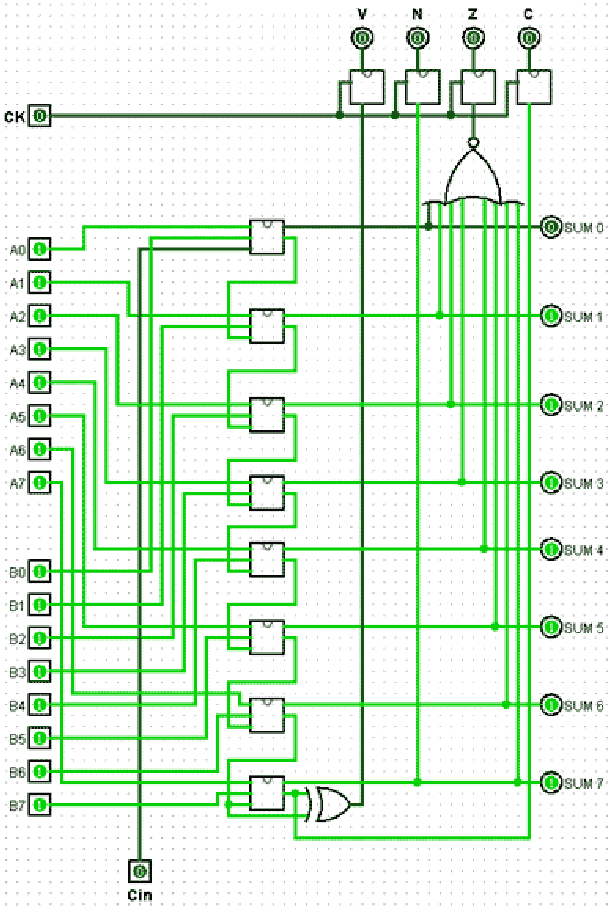


Fig. 5.8.5 The ALU Adder Component

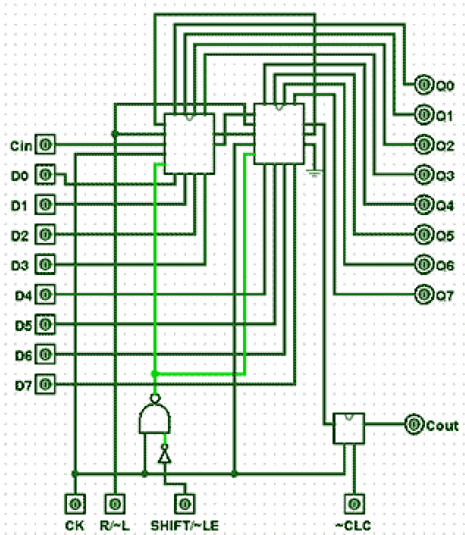


Fig. 5.8.6 The ALU Shift Register Component

The Shift Register

This component uses two 4-bit shift registers (from Module 5.7) connected in cascade as shown in Fig. 5.8.6. Inputs are provided for clock pulses, (CK), a right/left shift control (R/~L) and an input to control whether the shift register is in shift, or load-enable modes (SHIFT/~LE).

If ~LE is chosen temporarily during shift operations, the shift register can be reloaded from the data placed on the 8-bit ‘Data A’ and ‘carry-in’ (C_{IN}) inputs. This action is synchronised to the CK pulse by the external NAND and NOT gates connecting the SHIFT/~LE input to the two ~LOAD inputs of the 4-bit shift registers.

An additional JK flip-flop (mimicking a D type flip-flop) is placed between the 'serial-right' output of the shift register and C_{OUT} to allow the 'clear carry' input (~CLC) to clear the carry flag.

Carry Logic and Rotate Select

The carry logic circuit shown in Fig. 5.8.7 prevents the carry flag being set in rotate right mode, as bits rotate from bit 0 and re-enter the shift register at bit 7, therefore allowing correct carry flag operation in both left and right rotate modes.

When the ROTATE input is at logic 1, the Rotate Select circuit in Fig 5.8.7 allows C_{OUT} from the shift register to be fed back to the shift register C_{IN} input for continuous bit rotation.

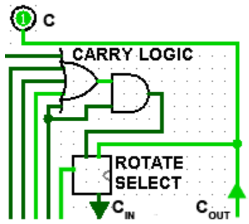


Fig.5.8.7 ALU Carry Logic

ALU Operation

Addition

To perform an addition, input data B is added to A. This is achieved by putting logic 1 on the control inputs of multiplexers 1, 2 and 3. This causes data A and B to be applied to the adder inputs. Also, to allow any carry bit from the C_{IN} input to be included in the addition, the 1 bit carry multiplexer must have logic 0 on its control input. The shift register is only used as a PIPO register in addition mode, so its input lines R/~L and ROTATE must be at logic 0. SHIFT/~LE must also be at logic 0 to enable parallel loading of the shift register, which will hold the result of the addition (A plus B) after the application of a single CK pulse.

The Status Flags

The Flag flip-flops are special outputs from the adder circuit. They consist of four separate D type flip-flops, each of which can be set to 1 or cleared to 0. They are set or cleared by the result in the adder. They signal, or 'flag' to the user, that a particular event has occurred.

The Carry flag (C)

The carry flag will be set if the result of any arithmetic or logic event causes a logic 1 to be carried over from bit 7 into the 'carry bit', (which is the carry flag). The carry flag can be cleared at any time by making the 'clear carry' input (~CLC) logic 0.

The Overflow flag (V)

When carrying out twos complement arithmetic, errors can occur if large numbers are involved. For example if two positive numbers less than 127_{10} are added and produce a negative result (any value greater than 127_{10}). This would cause the sign of the result (indicated by bit 7) to be wrong. The overflow flag gives an indication that an error has occurred by being set to 1 to indicate an 'overflow error'. An error is sensed and the overflow flag is set when either of two conditions occurs.

- There is a carry of logic 1 from bit 6 to bit 7 of the result, but the carry flip-flop is not at logic 1.
- There is no carry from bit 6 to bit 7 of the result, but the carry flip-flop is at logic 1.

By using the carry-out from bit 6 and the carry-out from bit 7 of the result as inputs to an XOR gate, the output of the gate will be set to logic 1 for either of the above error conditions, signalling an overflow error at the overflow (V) flag.

The Zero flag (Z)

This flip-flop is set when every bit of the result is zero.

The Negative flag (N)

A negative result, i.e. bit 7 = 1 sets this flip-flop to logic 1.

The Flag Register

The status flags are individual bits of a register called the Flag Register, and are operative not only when the ALU is in addition mode, but also in all other arithmetic modes, the C flag is also operative in shift and rotate left modes. In microprocessors the flag register not only indicates ALU results, but can also be used in decision-making. For example the ALU can be used to compare (by subtracting) two values and take various actions depending on the state of particular flags; e.g. after comparing two values, A and B, an action may be taken if A = B, indicated by the zero flag being set to 1, otherwise (if the zero flag is set to 0) take no action.

Subtraction.

Subtraction is performed using twos complement arithmetic. That is, to subtract B from A, input B is complemented and 1 added to the complemented value to form the twos complement. Then the twos complement of B is added to A in the adder to find the result. To achieve this action with data A and data B present at the inputs, logic 1 is applied to the control inputs of MUX 1 and MUX 2. MUX 3 has logic 0 applied to its control input to complement data B, while the CARRY MUX has a logic 1 applied to its control line so that the carry-in (C_{IN}) to the adder is forced to logic 1. This adds 1 to the result so that the twos complement of data B is added to data A. The result at the adder output is a twos complement number representing A - B. The flags are again set by the result as in the addition operation.

Counting with the ALU

Although the ALU does not include a binary counter circuit, it can also be used to count, by INCREMENTING or DECREMENTING, i.e. to add 1 to data A (incrementing), or subtract 1 from data A (decrementing). To count using this method would normally be carried out using (machine code or assembly language) software. A typical use could be to initiate a time delay by loading the ALU with some number, and then execute a looping routine to count down to zero by repeatedly decrementing data A. The zero result would be detected from the zero flag being set. However this would not be a common method, as the ALU (and therefore the CPU) would be occupied during the delay, and therefore not usable for other purposes. Most computer systems would also have dedicated counters for implementing similar time delays.

Incrementing.

Data A can be incremented if logic 1 is applied to the control inputs of MUX 1 and MUX 3. This will add B to A, with data B made zero by applying logic 0 to the control input of MUX 2. The 1 that must be added to data A is supplied by making the control input of the CARRY SELECT block logic 1, causing the carry input to the adder to be logic 1. The result at the adder output is therefore A + 1, again the flags are set by the result.

Decrementing.

To decrement data A, 1 must be subtracted from A. Because the ALU uses twos complement arithmetic, the twos complement of 1 added to A will in effect subtract 1 from A.

The twos complement of 1 is minus 1, which in 8-bit twos complement notation is 11111111_2. Therefore to subtract 1 from data A, data B must equal minus 1 (all bits = 1). To do this, and to make sure that the correct result is not changed by any data appearing on the data B input, logic 0 is applied to the control input of MUX 2 to make sure all data B bits = 0.

Logic 0 is also applied to the control input of MUX 3. This inverts data B, (which is 00000000_2) to give 11111111_2 at the adder input.

MUX 1 must have logic 1 on its control line, to apply data A to the other adder input. The adder's carry input is set to 1 by applying logic 0 to the control line of the CARRY MUX. This ensures that, provided there is no carry-in on the C_{IN} input, the correct result at the adder output will be A – 1.

Negation

Negation is simply the inverse of a value; therefore any value and its inverse will add to produce zero. In binary arithmetic the additive inverse of a value is its twos complement. The ALU can be used to negate (find the twos complement of) data A by complementing data A and then adding 1. This involves a similar process to decrementing, except that data B is treated differently, as follows:

The control input of MUX 1 is set to logic 0, which complements data A, also data B is made zero by putting logic 0 on MUX 2 control, and logic 1 on MUX 3. The Carry Select control input is set at logic 1, to add 1 to data A in the adder.

The shift register is used as a simple PIPO register by applying logic 0 to the three shift controls and logic 1 to the ~CLC input to make sure the carry is not cleared. This gives a final result of $\overline{A} + 1$, which is the twos complement of A.

The Shift Operations

Shift operations are controlled by the four lower order control lines, R/~L controls the direction of shift or rotation, SHIFT/~LE has the dual purpose of enabling the shift operations if logic 1 is applied, or acting as a LOAD ENABLE when at logic 0, allowing the shift register to be loaded or reloaded with appropriate data. Each action of the shift register (shift, rotate or load) is actuated by a single CK pulse. Also note that the shift register in this design does not affect the V, N or Z flags.

Shift Left (with Carry)

In this mode (with control word 10100101) input data B is kept at zero and, after the shift register is loaded by temporarily making SHIFT/~LE logic 0 to move data from input A into the shift register, shift is enabled by returning SHIFT/~LE to logic 1, and both ROTATE and ~CLC are disabled. The data in the shift register will now shift one bit to the left with each CK pulse applied. This appears to multiply the value of the data by two for each shift left, but it is a very limited multiplication operation, because the result is reduced each time the left most bit is lost as it passes through the carry bit. This action is therefore considered a logical, rather than an arithmetic shift.

Rotate Left (with Carry)

If rotate is activated by applying logic 1 to the ROTATE control input with SHIFT/~LE and ~CLC also at logic 1, the data being shifted left from bit 7 and through the carry flag, is returned via the C_{IN} input of the shift register to re-enter at bit 0 by the action of the ROTATE MODE SELECT data selector.

Rotate Right

When data in the shift register is rotated right, it leaves the register via bit 0 and is returned directly to bit 7 via an internal link, without passing through the carry flag.

There are a number of other operations, such as performing 8 bit logic functions, commonly found on microprocessors that this ALU is not designed to do. The purpose of this design is to illustrate how the circuits described in Digital Electronics Modules 1 to 5 are really just part of a bigger picture, they can be inter-connected in many ways to make many different circuits. This ALU design is one example, but how you use what you learn from the pages of learnabout-electronics and how you fit that knowledge into your own imagination is up to you.

5.9 Sequential Logic Quiz

Try our quiz, based on the information you can find in Digital Electronics Module 5 – Sequential Logic. Check your answers at http://www.learnabout-electronics.org/Digital/dig59.php and see how many you get right. If you get any answers wrong. Just follow the hints to find the right answer and learn more about sequential logic as you go.

1.
Which of the listed flip-flops is also known as a programmable flip-flop?

a) Clocked SR flip-flop.

b) JK flip-flop.

c) T-type flip-flop.

d) D-type flip-flop..

2.
Which of the following is a major advantage of the D-type flip-flop over the clocked SR type?

a) A shorter propagation delay.

b) Negative edge triggering.

c) No indeterminate output state.

d) Only one input and one output.

3.
Which of the following is a major advantage of synchronous counters over asynchronous counters?

a) They do not suffer from ripple through problems.

b) They use JK flip-flops.

c) The flip-flops operate consecutively when a clock pulse is received.

d) They are cheaper to implement than asynchronous counters.

4.
Which of the following actions must an ALU perform to NEGATE the input data?

a) Subtract 1 from the input data.

b) Complement the data bits and add 1.

c) Complement the data bits and subtract 1.

d) Covert the data to its ones complement.

5.
In a simple low activated SR flip-flop, the Q and NOT Q will be indeterminate (unknown) if:

a) Both S and R are at logic 0 together.

b) Both S and R change from logic 0 to logic 1 together.

c) R changes to logic 0 while S is at logic 1.

d) S changes to logic 0 while R is at logic 1.

6.
What type of device is shown in Fig. 5.9.1 if the waveforms illustrated are present at its CK and Q terminals?

a) Monostable.

b) Astable.

c) Bi-stable.

d) Latch.

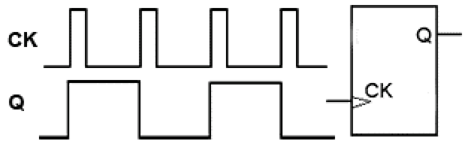


Fig 5.9.1

7.
What type of circuit is illustrated in Fig. 5.9.2?

a) PIPO shift register.

b) PIPO/SIPO shift register.

c) PIPO/SIPO/SISO shift register.

d) PIPO/SIPO/SISO/PISO shift register.

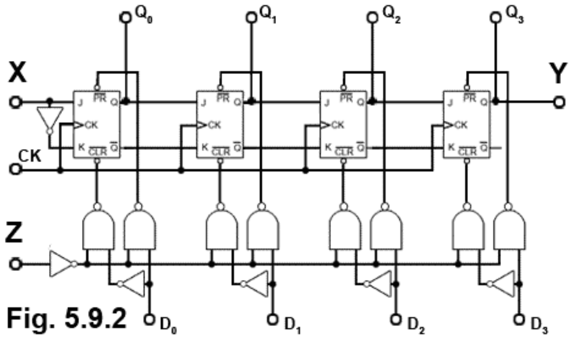


Fig. 5.9.2

8.
What type of circuit is illustrated by Fig. 5.9.3?

a) Synchronous up counter.

b) Parallel loading PIPO register.

c) Reversible shift register.

d) Synchronous up/down counter.

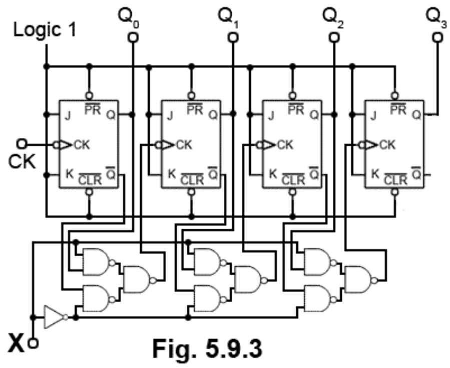


Fig. 5.9.3

9.
Refer to Fig. 5.9.3. What is the function of the input marked X?

a) The LOAD input.

b) The UP/ $\overline{\text{DOWN}}$ Control.

c) The $\overline{\text{UP}}$ /DOWN Control.

d) The $\overline{\text{LOAD}}$ input.

10.
Refer to Fig. 5.9.4. After 2 more clock pulses what will be the logic states on outputs A, B, C, D and E?

a) A=1, B=0, C=0, D=0, E=1.

b) A=0, B=0, C=0, D=1, E=1.

c) A=1, B=1, C=1, D=0, E=0.

d) A=1, B=0, C=1, D=0, E=0.

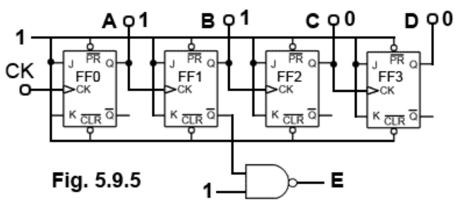


Fig. 5.9.5