**DALHOUSIE UNIVERSITY**
Department of Electrical & Computer Engineering
Digital Circuits - ECED 2200

# Experiment 4 - Latches and Flip-Flops

**Objectives:**

1. To simulate and implement an RS latch memory element
2. To implement a basic JK latch and a Master-Slave JK flip-flop

**Theory:**

The ability to store information is essential in computing. Memory elements are used to store program information such as the results of calculations or user input. The information can later be read back from the memory elements to be used for new calculations or to display to the user.

Remember in lectures and in lab 2 how we constructed addition and subtraction circuits. The circuits only allowed two numbers to be added together at a time. What if you only had one Adder circuit, but needed to add three numbers together? Using a memory element, you can use the same adder circuit to add all three numbers together in two steps. In step one (Fig. 1 below) you (the user) input your first and second numbers (Augend and Addend) into the adder circuit and get the result (sum). The result is then stored in some memory circuitry. In the second step (Fig. 2) the adder circuit gets the result from the memory circuitry and uses it as the new Augend. You as the user input the third number as the new Addend. These numbers go through your adder circuit, and you receive your final result, which is the sum of all three numbers. Pocket calculators work somewhat like this when you are using them to perform long calculations.

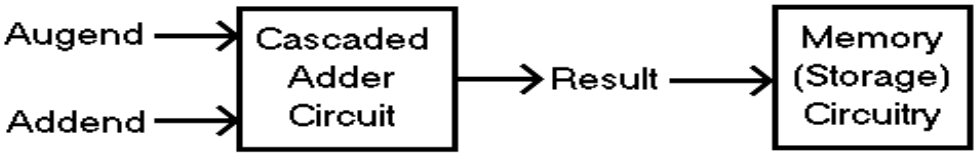



Fig. 1

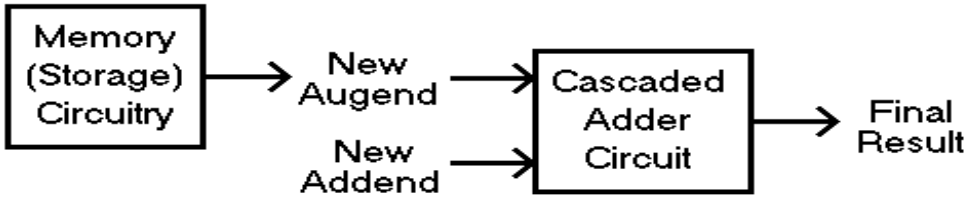



Fig. 2

In this lab we will be examining some basic memory elements, called latches and flip-flops.

**Procedure:**

Part 1) RS Latch

The RS latch is the basic memory element. Most of the other latches and flip-flops are derived from the RS latch. Two versions of the latch circuit are shown in the first schematic #1. The RS latch can be constructed with either NAND gates or NOR gates. Both versions are shown on your sheet. Notice that the output of each gate is wired to one of the inputs of the other. This is called feedback because the output is "fed back" to the input. The circuits are designed so that X2 and X4 are the output **Q**. X1 and X3 should always be the inverse of Q, called **Q NOT**. The RS latch has three functions it will perform on output Q. They are Set (**S**), Reset (**R**), and Hold. When the Set state is asserted, Q is '1' and Q NOT is '0'. When the Reset state is asserted Q is '0' and Q NOT is '1'. When neither set or reset are asserted then the latch is automatically in the Hold state. While in 'hold' mode, the latch simply keeps Q and Q NOT at whatever value they last were. The two input switches on each latch will trigger the Set (**S**) and Reset (**R**) functions. The two designs you have been given activate their set and reset functions differently. For one design you have to close a switch to assert its function (active high), and for the other you have to open the switch to assert its function (active low). For instance to activate the Set function in the active high design, close the Set switch. This sends a set signal through the logic and 'latches' output Q to '1'. If you then open the Set switch back up, Q should stay at '1' and the device has entered the 'hold' state. It is up to you to determine which switch is Set and which is Reset in each design, and which design is 'active high', and which is 'active low'.

a) Construct the circuits shown on the schematic, implement them using the digital board.
b) Which switch is 'Set' and which is 'Reset' for each circuit, and which is 'active high' and which is 'active low'?
c) What happens to Q and Q NOT when you assert Set and Reset at the same time? (E.g. for the 'active high' design, you close both the Set and Reset switches at the same time)
d) Simulate each design with different combinations of R and S. Construct a state transition table for each design. (See the end of the lab for an explanation of the state transition table and how to make one)

The RS Latch allows us to 'remember' a single bit. By activating the set function we make the output 'Q' go high (or '1'). If we then deactivate the set function the latch enters the hold state and keeps the output in its high state. The output 'Q' will only go low if the reset function is activated. So, until the latch is reset, it 'remembers' (or 'latches') a binary '1' for us. If we want to store a binary number, we would use an RS latch for each bit. For instance to store the nibble 1010 we would need four latches. Latch #1 and #3 would have Set asserted, and #2 and #4 would have Reset asserted. The latches would then be put into the hold state by de-asserting the Set and Reset functions. This stores (latches) our binary number in the four RS latches. To clear it we would just send a reset command to all four latches at once.

Part 2. JK latch and Flip-Flop

One problem with the RS latch is that when both Set and Reset are asserted at the same time both outputs become equal ($Q = \overline{Q} = 0$), which is an unwanted state. One solution to this is to add a pair of AND gates to the input to ensure that Set and Reset cannot receive an assert signal at the same time. The JK latch schematic, found in Schematic #2, shows this configuration. Since $Q$ and $\overline{Q}$ are (theoretically) never the same, it guarantees that only one of the AND gates will be activated at a time. *Note*: At least one of the asynchronous inputs CLR (clear) or PR (preset) needs to be asserted to make $Q \neq \overline{Q}$ when the circuit is initialized.

Feeding Q and Q NOT back to the inputs through the two AND gates cause another interesting effect: toggling. Toggling is when the output Q flips back and forth from '1' to '0'. This occurs whenever the inputs are both asserted at the same time. For instance when the inputs to the AND gates (called 'J' and 'K') are both set to '1', and Q is '1' (so Q NOT is '0') then the RS latch sees R = 1, S = 0. This resets the latch, changing Q to '0' and Q NOT to '1'. These outputs are brought back around to the input (feedback), which swaps the activation state of the two AND gates. Now the RS latch sees R = 0, S = 1. This sets the latch, changing Q back to '1' and Q NOT to '0'. These are fed back to the inputs, and the cycle begins again. So, when inputs 'J' and 'K' are both high, we get toggling.

a) Construct the basic JK latch with asynchronous PR & CLR inputs as shown in diagram Schematic #2, implement it on the DE1-SoC board.
b) After initializing the flip-flop by asserting one of the asynchronous inputs, verify that:
   i. The Set & Reset states set by the J & K inputs work properly by making J or K '1' (one at a time),
   ii. The outputs $Q$ and $\overline{Q}$ never become equal unless you assert both asynchronous inputs simultaneously.
   iii. If the Set & Reset inputs to the RS latch (determined by J & K when both PR & CLR are de-asserted) are set to '1' simultaneously the outputs $Q$ and $\overline{Q}$ toggle indefinitely.

The problem with the previous design is that it toggles continuously as long as 'J' and 'K' are '1'. We would like a design that only toggles once, then stops toggling. This would let us get the complement of a bit stored in the latch. If we had four JK flip-flops, we could use the set and reset functions to latch a bit onto each one. For example, setting latch 1, 3, and 4, and resetting latch 2 would store the binary nibble 1011. If we then sent a toggle command to all four latches (by asserting J and K at the same time), all the bits would change state (toggle), giving us the compliment of the number: 0100. If the latches continued to toggle then the output would keep changing from 1011 to 0100, which isn't very useful.

To make a device that only toggles once, two latches are cascaded creating a Master/Slave JK Flip-Flop (Schematic #3). Notice that it consists of two basic JK flip-flops with yet another pair of AND gates on the front and a single Inverter. The design uses three switches: 'J', 'K', and 'Clk'. The 'Clk' switch is called the Clock line and (with the extra AND gates and inverter) it prevents the uncontrolled toggling seen in the basic JK flip-flop.

a) Construct the Master-Slave JK flip-flop as shown on the schematic (Schematic #3) and implement it on the DE1-SoC board.

b) Construct a state transition table for the circuit.

c) Inputs 'J' and 'K' do the same things as 'R' and 'S'. That is, one line triggers the set function and the other triggers the reset function. Which is which?

d) How does the Master-Slave configuration prevent the circuit from continuously toggling the way the normal JK flip-flop does?
Hint: Look at the way the inputs J and K propagate from gate to gate as the Clk line goes from '1' to '0' and back to '1'

**State Transition Tables:**

State Transition Tables are truth tables that show how a circuit's outputs change over time. Basic truth tables are good for showing the behavior of any circuit that is not clocked; the circuit is time independent. For instance an 'AND' gate has no clock line so the inputs immediately affect the output. A circuit that is clocked, such as the Master-Slave JK flip-flop will not change its output until the circuit has received a complete clock pulse on the CLK line. The inputs can be changed but no reaction will be observed on the outputs until the clock has been triggered. Because of this dependence on the clock line, State Transition tables show the before and after states of the outputs. For example, say we have a JK flip-flop with **Q** = 1, **J** = 0, **K** = 1. This setting should reset **Q** to zero once CLK receives a pulse. The output **Q** will remain at 1 until the pulse is received. Shown in a table we have:

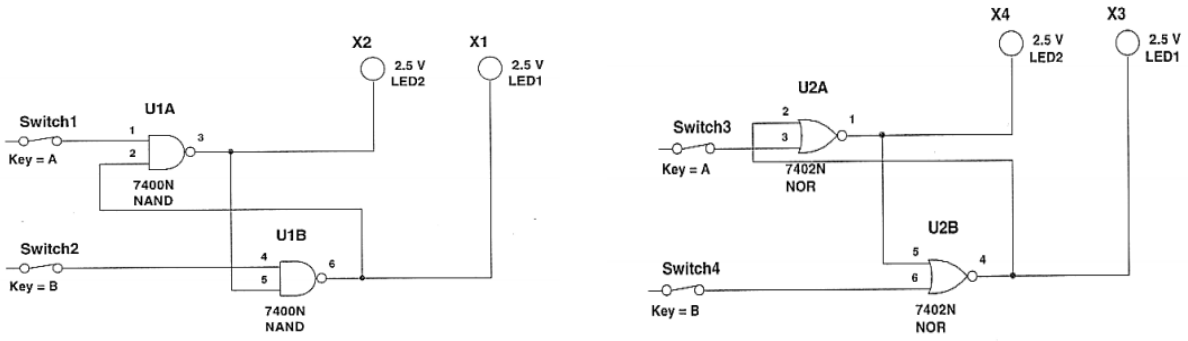
| J | K | | Q | Q+ |
|---|---|---|---|---|
| 0 | 1 | | 1 | 0 |

This shows the affect of the inputs on the output when the output is already '1'. This is one line of the State Transition table for the JK flip-flop. 'J' and 'K' show the states of the inputs. 'Q' shows the output before the clock pulse, and 'Q+' shows the output after the clock pulse. To complete the table you would give all possible inputs and output combinations. For instance the next line might look like:

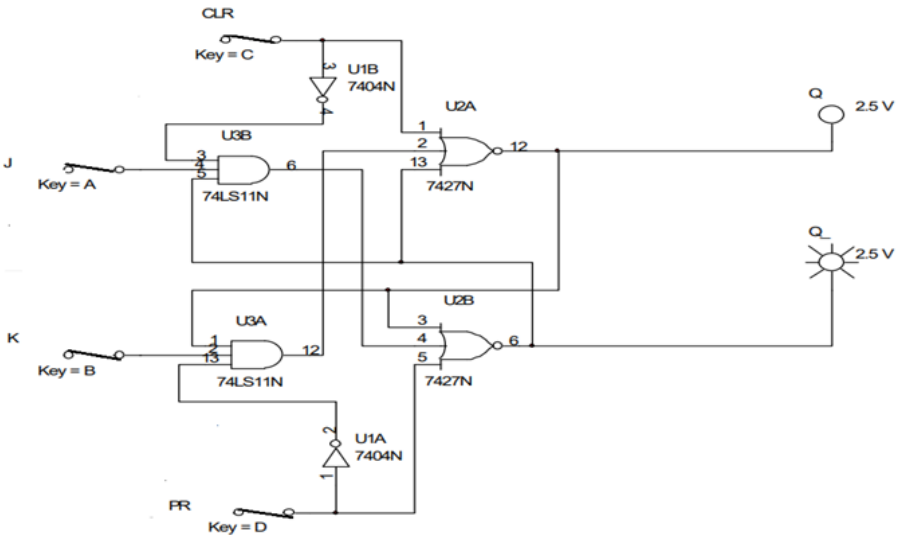
| J | K | | Q | Q+ |
|---|---|---|---|---|
| 0 | 1 | | 0 | 0 |

This line would show how the inputs affect the output when the output starts as a '0'. Since these circuits incorporate feedback, it is important to note that previous output state often affects the next output state. In effect, the output 'Q' is like an input for 'Q+'. A full State Transition table should contain all the different input and output combinations. Looking at the table you create should give you all the information you need to answer most of the questions in the lab.

RS Latch (Schematic #1)




JK Latch (Schematic #2)




Master-Slave JK Flip-Flop (Schematic #3)