cs281: Introduction to Computer Systems

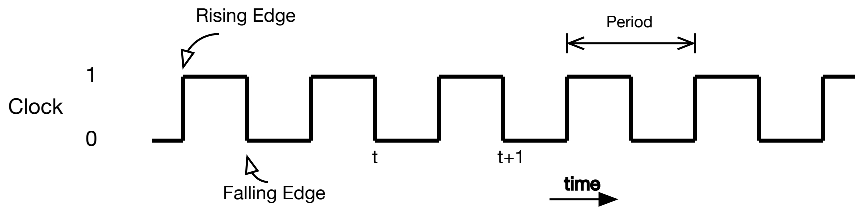
Lab04 – Introduction to Sequential Circuits

Overview

In this lab, we will learn about designing circuits that utilize *memory* (and a *clock*) in order to achieve some functionality. Through the prior labs, you have learned the fundamentals of creating *combinational* circuits, in which the outputs of the circuit are determined by the inputs ... they require no memory, nor any notion of the sequence of operations that may have happened in the past. When we introduce the idea of a clock, we are incorporating a model of *time* into our thinking, and so we more precisely say that a combinational circuit is one where the outputs at time $t + \epsilon$ is determined by the inputs at time t. The ϵ is intended to convey that, in real time, the change on the outputs does not happen instantaneously. When the inputs change, the changed current values takes time to traverse through the wires and gates of the circuit, and we use ϵ to denote that small delay time.

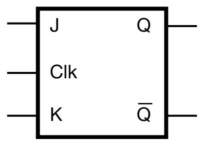
To understand how we design the CPU of a computer, we need the concept of a sequence of steps that occur over time. For instance, we can think of the CPU executing an instruction in one step, and then another instruction in the next step, and so forth. When we talk about circuits requiring memory in this context, we are not necessarily talking about large amounts of memory, like the RAM in a computer, nor even more moderate amounts of memory, like the Register File in the CPU. In the current lab, for instance, we require two *single bits* of memory storage, used to track where we are in a short sequence of possible states.

A *clock* is a digital signal that simply alternates between logical one and logical zero (aka HIGH/LOW or +5V/0V) with consistent, regular timing, yielding what we can depict with a square wave:



This is very much like the HIGH/LOW sequence we see with the Function Generator on the breadboard, and we can, in fact, use the Function Generator to provide a clock if we wish. The *rising edge* refers to the transition from 0 to 1, and the *falling edge* refers to the transition back down from 1 to 0. The *period* refers to the interval of time for a full cycle, so on the illustration, the period is shown as the time from one rising edge of the clock to the next rising edge of the clock.

A *clocked sequential circuit* is one in which we employ a clock as well as bits of memory storage along and the combinational circuit design process we have already learned about to achieve some desired functionality. The storage device to holding a single bit of information is called a *flip-flop*. The *J-K Flip-Flop* is one variant and can be represented pictorially as follows:



The left-hand side shows the inputs to the device, and the right-hand side shows the outputs. Since the device stores a single bit, we generically call that bit Q, and so the device gives both the stored bit and its negation as outputs. The bit stored remains stable with its current value for the duration of a clock period, but can change its value to store a (possibly) different bit based on its J and K inputs and does so at a certain point in the clock cycle. If the point in time when a flip-flop may take on a new value occurs on the falling edge of the clock, it is called *negative-edge-triggered*, and if the value change occurs on the rising edge of a clock cycle, it is called *positive-edge-triggered*. In the lab, we have negative-edge-triggered J-K flip-flops, so we will use that triggering as an example here[1].

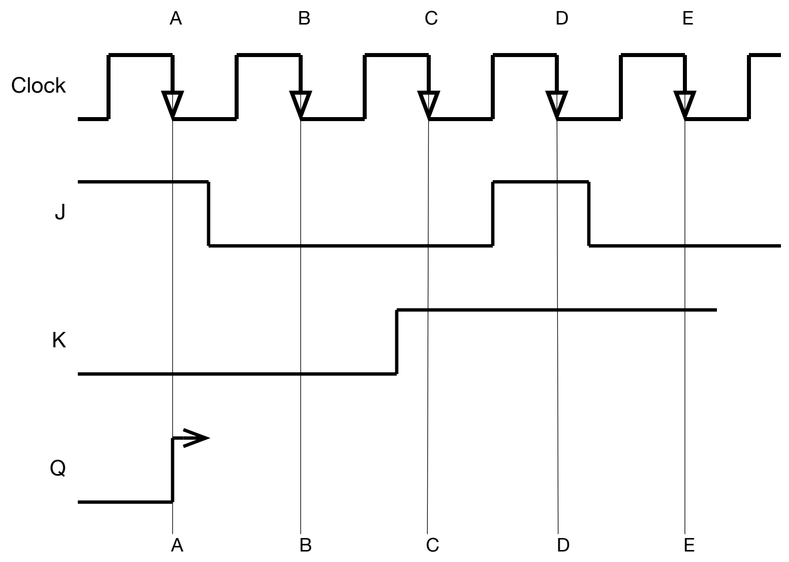
For a one-bit memory device, we want at least three possible manipulations of the bit stored, given a clock transition:

- If we want the device to store 0, we set J to 0, and K to 1.
- If we want the device to store 1, we set J to 1, and K to 0.
- If we want the device to maintain the previously stored value across a clock transition, we set both J and K to 0.

The J-K flip-flop also defines what should happen if both J and K are 1 ... it sets the stored value to its complement. We can represent this in tabular form, and use the notation Q^+ to denote the value of Q at the next transition ($t + 1$), while Q represents the stored bit from time t up to time $t + 1$.

J	K	Q^+	
0	0	Q	Unchanged
0	1	0	Reset
1	0	1	Set
1	1	$\overline{Q}$	Complement

To illustrate, we will draw the clock, values for inputs J and K, and begin the output of a J-K flip-flop, Q with the signal levels aligned vertically. We will use A, B, C, D, and E to represent the time points associated with the five falling edges of the clock in the picture.



[1]Note that the LogiSim software package that we will use for simulating more complicated circuits uses positive-edge-triggered storage devices. That will not affect our design process, but will affect when things change as we observe the circuits.

Q1 In the above figure, complete the drawing depicting the value of Q across the rest of the example.

Q2 Explain why Q transitions from 0 to 1 at time A.

Q3 Explain why Q has the value it does at time B.

Q4 Explain why Q has the value it does at time C.

Q5 Explain why Q has the value it does at time D.

Q6 Explain why Q has the value it does at time E.

In a clocked sequential circuit system, suppose we have a single (external) binary input X and a single (external) binary output Y. If we consider a clock period to represent a *step*, then we can refer to the value of the input X at any given step, and the value of the output Y at any given step. We can also consider the *sequence* of input values for X. We can then characterize the output Y at a given point in time based on the sequence of inputs for X. Suppose, for example, the characterization that Y should be 1 if the current and last values in the input were both the same (0 and 0, or 1 and 1) and 0 otherwise.

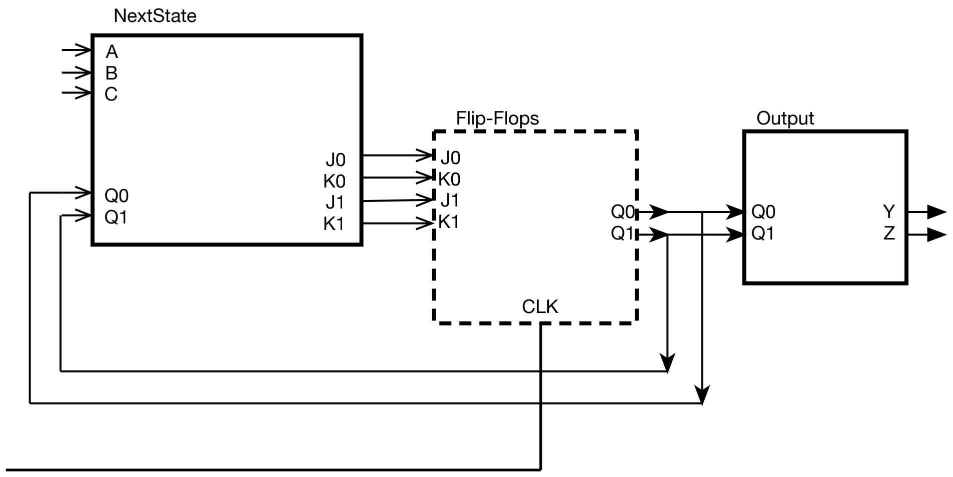
Q7 Suppose we can determine a way to use a single J-K Flip-Flop so that it stores the *last* value of X. Describe the use of just a few boolean logic gates so that, given the output of the J-K Flip-Flop and the current value of X, you compute the current value of Y.

Q8 If you were successful in Q7, then all that remains is, based on the value of the current input X, to determine values for the J and K to the flip-flop so that, at the next clock transition, the flip-flop stores the value of X. Draw the truth table and fill it in. (Hint: There is only one input, X, so the table has two rows. There are two outputs from this combinational circuit subproblem that we have defined, in addition to the one input, so there are three columns in the table.)

Q9 Give the boolean expression for J:

Q10 Give the boolean expression for K:

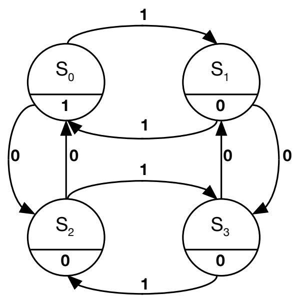
What if we had three inputs to a clocked sequential circuit system (A, B, C) and two flip-flops (with outputs Q_0 and Q_1) to store the state needed for our desired functionality, and two output bits from the system, Y and Z? We could generalize the strategy we used above and design a combinational logic circuit whose inputs were both the system inputs (A, B, and C) and also the bits representing the current state (i.e. Q_0 and Q_1) and whose outputs were the J's and K's to be fed to the two flip-flops (call them J_0 and K_0, and J_1 and K_1 for storage elements 0 and 1, respectively). We generically call this the *NextState* circuit, and the same approach works even if we have a different number of system inputs and a different number of bit-storage elements. This is the generalized equivalent of what you did in Q8. Similarly to what we did in Q7, we could also define a combinational circuit to calculate the outputs of Y and Z, given the storage bits from the flip-flops. A pictorial representation of the solution strategy is:



To solve more general problems, what we need now is a mechanism to describe a solution to a problem by defining a set of states and, for the steps needed to solve the problem, the transitions, based on the inputs, from one state to another. We also need a way of annotating the states with the desired outputs of the system, if indeed the output is determined solely by the state.

The needed mechanism is a graphical representation of a set of states and the transitions between them, and is known as a *Finite State Machine* (FSM). We will represent nodes/vertices in the graph with circles and label the circles with a designation identifying a unique state (in general, we will simply start with state S_0, and create as many new states $S_1, S_2, ...$ as we need to solve the problem). Directed arcs will show transitions between the states, and each arc will be labeled with the input(s) that cause the transition. To allow the annotation of the outputs associated with each state, we will add the output beneath the label within the circle for each state.

An example will help clarify. Consider the following pictorial:



In this example, we have four states, S_0 through S_3. The output (the value below the dividing line in the node), which we will call Z, has value 1 if, at a point in the clock-timed sequence, we are "in" state S_0 and Z has value 0 if we are in any other state. The labels on the arcs are the values of the input, call it X, that, at each cycle of the clock, cause us to transition from one state to the next. Since our single bit of input X can only have value of 0 or 1 at any point in time, each state should have two outgoing transitions. Note that, although not shown in this example, a transition from a state back to itself for a given value of the input is perfectly legal.

Q11 : Assuming we start in state S_0, what state is the system in if the value of X over clock periods is "0 1 0 0 1 1"? What is the output at that time?

Q12 : What state and output if the input is "1 0 1 1 0 1"?

Q13 Given four non-trivial sequences such that the system ends in state S_0 (and so has output 1)?

Q14 Give an English description of the patterns by which a sequence ends in state S_1? S_2? Evaluate your description against a variety of patterns. Does your description continue to hold up?

Now let us develop a circuit that uses 2 JK flip flops of memory and a clock to implement the above FSM. We first observe that we have four states (0 to 3), and we need to map from our two bits of memory to encode the four states. If we use subscripts 1 and 0 to refer to the two flip-flops, and consider their outputs Q_1 and Q_0, one way to encode our current state is given by the following natural *state assignment*:

Q_1	Q_0	Associated State
0	0	S_0
0	1	S_1
1	0	S_2
1	1	S_3

So, for example, whenever flip-flop 1 has value $Q_1 = 1$ and flip-flop 0 has value $Q_0 = 0$, the system is in state S_2. With our FSM model, given by the picture above, and the state assignment given by this table, we can now design our two needed combinational circuits, the *NextState* circuit, and the *Output* circuit.

For the *NextState* circuit, we have two bits of input coming from the flip-flops, (i.e. Q_1 and Q_0), and 1 bit of input for X. The outputs are the values needed for $J_1^+, K_1^+, J_0^+, K_0^+$, where the superscript '+' indicates this is the *next* value for each, so that, on the next clock transition, the flip-flops take on the appropriate values needed to put the system "in" the desired state. Fill in the remainder of the NextState truth table below.

As an example, the first line represents the situation that the system is currently in state S_0 because Q_1 is 0 and Q_1 is 0 and, since X is 0, our next state needs to be S_2. This means flip-flop 1 must take on the value 1 and flip-flop 0 must take on the value 0. Q_1 has value 0, and flip-flop 1 needs to take on value 1. To do this, we can either toggle the current value, so J/K would be $1/1$ or we can "set" with J/K of $1/0$. Thus we get J_1^+ of 1 and "don't care" for K_1^+.

row	Q_1	Q_0	X	J_1^+	K_1^+	J_0^+	K_0^+
0	0	0	0	1	X	0	X
1	0	0	1				
2	0	1	0				
3	0	1	1				
4	1	0	0				
5	1	0	1				
6	1	1	0				
7	1	1	1				

Q15 Describe why J_0^+ is 0 and K_0^+ is X in the first line of the truth table.

Fill in the K-maps, circling a minimal set of prime implicants for each of the output variables in the NextState circuit. Then, to the right, give a boolean expression for the circuit.

J_1^+

	Q_0X			
Q_1	0 0	0 1	1 1	1 0
0				
1				

K_1^+

	Q_0X			
Q_1	0 0	0 1	1 1	1 0
0				
1				

J_0^+

	Q_0X			
Q_1	0 0	0 1	1 1	1 0
0				
1				

K_0^+

	Q_0X			
Q_1	0 0	0 1	1 1	1 0
0				
1				