**Score: \_\_\_\_\_**

**Lab04 – Sequential Circuits and Static Memory (SRAM)**

COMP256 – Computing Abstractions

Dickinson College

Spring 2023

Prof. Grant Braught

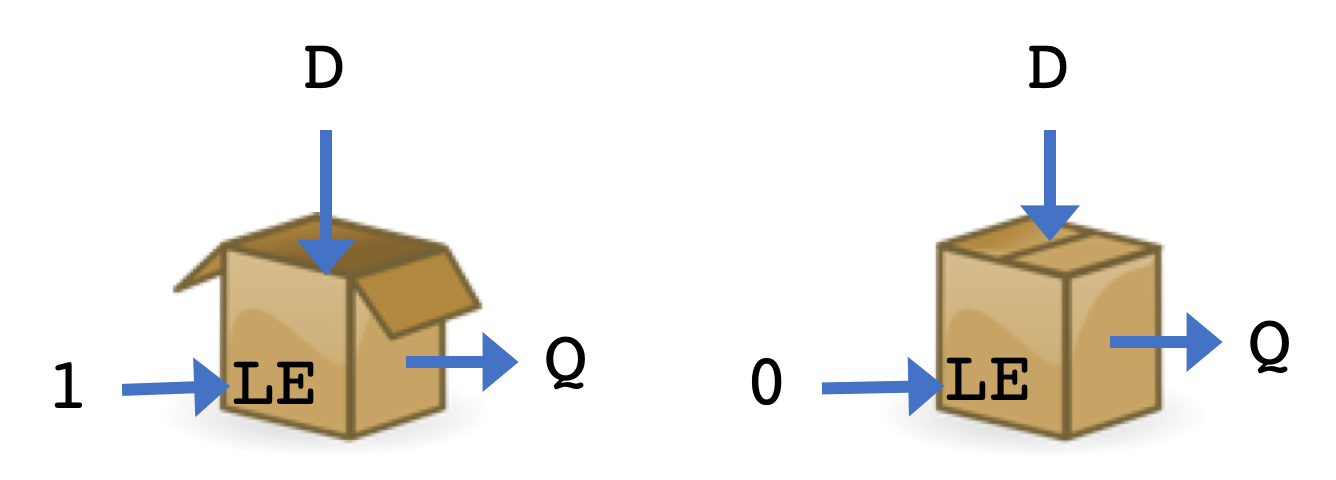
**Name(s):**

**Introduction:**

In the first three labs we worked our way up the hardware abstraction hierarchy from electrical signals, to 0’s and 1’s to transistors to logic gates and a circuit that can perform addition, and ultimately to a small “programmable” machine. That machine could be programmed to perform one of four different operations on its inputs using a 2-bit instruction. What is still missing is a sense of how the data and the program instruction are stored. For example, we know that in the Knob and Switch computer that we have been working with, the instructions and data are stored in the main memory cells and the registers. So, the natural question at this point is… how are those built from electronic components? In this lab you will learn about the behavior of memory cells and how one can be constructed using the logic gates with which we are already familiar.

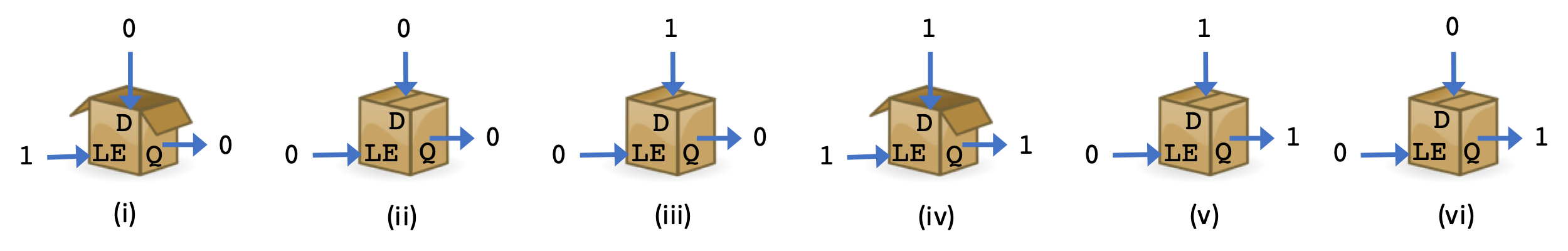
**An Abstract Memory Cell:**

A storage box provides an analogy that is often helpful in thinking about how memory cells work. Consider the storage boxes shown below. The value of the LE input controls whether the box is open or closed. The box will be open when LE = 1 (left box) and will be closed when the input LE = 0 (right box). When the box is open (when LE=1) the value of the D input (a 0 or a 1) will be placed into the box. But when the box is closed (when E=0), the value of D input is blocked from being able to be put into the box. The Q output always shows what is currently stored in the box (a 0 or a 1) regardless of whether the box is open (E=1) or closed (E=0).



So we can say that the value of Q “remembers” what D was the last time the box was open.

The following sequence of images illustrates how the box in our analogy stores one bit of information:



The above sequence represents time passing from left to right and goes as follows:

(i) LE=1 at the beginning so the box is open. The input D=0, so a 0 is stored into the box and appears on the output Q.

(ii) LE=0 now so the box is closed. Because the box is closed the value of the output Q=0 is stored in the box. You can think of the value of Q being “frozen” or “fixed” or “remembered” when the box is closed.

(iii) LE=0 so the box is still closed. So, even though the input D has changed to a 1 at this time, because the box is closed the value of D is blocked and cannot be put in. Thus, the value of Q “remembers” that the last thing put into the box was a 0.

(iv) LE=1 so the box is now open. Because the input D is 1, a 1 will be placed into into the box. The output Q reflects what is in the box, so Q is also a 1.

(v) LE=0 so the box is now once again closed. Note specifically that Q is not 1 because D=1 in this image. But rather that Q is a 1 because that’s what is in the box.

(vi) LE=0 so the box is still closed. The value of Q remains a 1 reflecting the value that is in the box even though the input D at this point has changed to a 0.

The above sequence also illustrates that we will need a fundamentally different type of circuit to implement a memory cell**. With the *combinational logic* circuits, we have seen up until this point, a circuit’s inputs uniquely determine its output. That is, if we put in the same combination of inputs, we will always get the same output.** That relationship is exactly what is expressed by a truth table. Each row of the truth table corresponds to one combination of the inputs and for a given circuit that row will have a specific output that is always the same. That is, the circuit will always generate the indicated output for the associated combination of inputs.

🔑 1. Now, consider the sequence of images above again.

a. Treating LE and D as inputs and Q as the output, identify a pair of images (give their Roman numerals) that show that a memory cell cannot be constructed using a combinational logic circuit. Hint: Read the paragraph above closely and use what it says about sequential circuits to identify something in the images that they cannot do.

b. Briefly explain why the pair of images you identify in part a shows that a memory cell cannot be built using a combinational logic circuit.

c. Identify a second pair of images (give their Roman numerals) that also shows that the memory cell cannot be built using a combinational logic circuit.

**Sequential Circuits:**

In this section you will learn about what are called ***sequential circuits***. Unlike combinational logic circuits, the same combination of inputs to a sequential circuit can generate a different output. We will see that sequential circuits provide us with a way to create a memory cell. Ultimately, you will build a circuit that implements a 1-bit memory cell that behaves just like the box in our analogy above.

**The unique aspect of a sequential circuit that allows us to build a memory cell is that the outputs of the circuit *“feedback”* and become additional inputs to the circuit.**

For example, consider the circuit shown below:

A close up of a logo

Description automatically generated 

Notice in this circuit how the values of P and Q both feedback into the circuit as inputs to the NOR gates. This creates a cycle, where the output Q must be defined in terms of itself. While that may seem problematic, we’ll see that with well-constructed sequential circuits it is not. And in fact, if we think about it, it makes sense that this should be a necessary condition for building a memory cell. If the output Q is supposed to remain the same regardless of the input D (i.e. when LE=0), then somehow the circuit must know (or remember) the previous value of Q. Having that value of Q feedback into the circuit as an input is what makes this possible.

**The SR-Latch:**

The above sequential circuit is known as an SR-Latch. The SR-Latch will not quite behave like the boxes in our analogy (i.e. like a 1-bit memory cell). However, once we understand the SR-Latch we will use it as a building block in another circuit that is a 1-bit memory cell that behaves just like the boxes in our analogy.

To understand how this circuit works we will need to *“****chase logic values around the circuit****.”* For example, consider the situation where S=1, R=0 and Q=0 as shown below:



In this situation we know that the inputs to the top NOR gate will be 1 0 (S=1 Q=0) so the output of the top NOR gate will be 0 (P=0). This tells us that the input to bottom NOR gate will be 0 0 (P=0 R=0) which makes the output of the bottom NOR gate change from Q=0 to Q=1. This change then has a ripple effect changing the inputs to the top NOR gate to 1 1 (S=1 Q=1). With these inputs the output of the top NOR gate remains P=0. So, at this point, all of the inputs and outputs have *“settled down”* (i.e. further “chasing of the logic values around the circuit” will not have any effect).

One good way to “chase the logic values” is to draw out the circuit on some scrap paper. Then write the starting values next to S, R and Q. Then “chase the logic values” around the circuit by crossing out old values and filling in new ones until all of the values “settle down.” The result of “chasing the logic values” for S=1, R=0, Q=0, as described above, is illustrated below:



🔑 2. As just described, the result of “chasing the logic values S=1, R=0, Q=0” results in a new output of Q=1. In the truth table below, this “settled down” or next value of Q is labeled QNext. The example from above with inputs S=1, R=0, and Q=1 is shown in yellow highlight in the table below. Once you are sure you understand the highlighted line, complete the rest of the table below by “chasing” the specified input values around the circuit until all of the values “settle down”.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  | **S** | **R** | **Q** | **QNext** |  |
|  | 0 | 1 | 0 |  |  |
|  | 0 | 1 | 1 |  |  |
|  | 1 | 0 | 0 | **1** |  |
|  | 1 | 0 | 1 |  |  |
|  |  |  |  |  |  |

3. Looking at the table above we can get some insight into why this circuit is called an SR-Latch. The S stands for Set and the R stands for Reset. Your table should show that **when S=1 the value of Q is *Set* to 1 and when R=1 the value of Q is Reset to 0.**

*No answer is required here, but you need to ensure that you see the expected behavior in your table. If not, revisit questions #2 and resolve any issues before going on.*

🔑 4. As mentioned, this circuit is known as an SR-Latch. We now understand that the S and R part of the name come from their effect of **S**etting and **R**esetting the output value Q.

But why Latch? The term ***Latch*** is commonly used to describe a circuit that exhibits memory (i.e. that a 0 or 1 can be “*latched into*” or saved by the circuit). Complete the truth table below by “chasing” the given input patterns around the circuit.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  | **S** | **R** | **Q** | **QNext** |  |
|  | 0 | 0 | 0 |  |  |
|  | 0 | 0 | 1 |  |  |
|  |  |  |  |  |  |

5. Looking at the truth table in #4 you should see that when S=0 and R=0 the value of QNext is always the same as Q. That is, the output Q is does not change when you “chase” the values.

*No answer is required here, but you need to ensure that you see the expected behavior in your table. If not, revisit questions #2 and resolve any issues before going on.*

**One way to interpret the table in question #4 is to think of it as showing that Q *remembers* what it had been before S and R were changed to 0 0.** For example, imagine that we make S=0, R=1 which will reset Q=0. If we now change the inputs so that S=0, R=0 then Q will *remember* that it had been reset to 0. Similarly imagine we make S=1, R=0 to set Q=1. If we now change the inputs so that S=0, R=0 then Q will *remember* that it had been Set to 1. Summarizing, if we make S=0, R=0 then Q remembers if it has most recently been set to 1 or reset to 0.

Note again that this behavior cannot be accomplished with a combinational logic circuit because as we are seeing same inputs (S=0, R=0) has two possible outputs (Q=0 or Q=1).

**Building an SR-Latch:**

Okay, that all may seem a little strange, but let’s build it and see that it does actually work.

6. Create a new circuit in TINKERCAD named Lab04-SRLatch and setup a full-size breadboard as shown below. Be sure to add labels to the slide switches for the inputs and the LED for the output. These will correspond to the inputs and outputs of the SR latch. **Be sure to put the 74HC02 (NOR) chip to the right edge of the board and the switches to the left. This will leave room for things that you’ll be adding to this circuit later in the lab.**



*No answer is required here, you will provide a link to your TINKERCAD circuit when it is complete.*

🔑 7. Wire up the 74HC02 chip to create an SR latch with the inputs S and R provided by the slide switches and the output Q displayed by the LED (off = 0, on = 1). The value P will exist in your circuit, but is not displayed.  **The pin-outs for all of the chips are again included at the end of the lab for reference.**

Give a link to your completed TINKERCAD circuit here:

8. Experiment with your circuit to complete the following truth table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  | **S** | **R** | **Q** |  |
|  | 0 | 1 |  |  |
|  | 1 | 0 |  |  |
|  |  |  |  |  |

9. The table above you should agree with what you predicted in question #3. That is, **S=1 *Sets* the value of Q to 1 and R=1 *Resets* the value of Q to 0**.

*No answer is required here, but you need to ensure that your circuit behaves as expected. If your circuit does not behave as expected revisit questions #3-#8 and resolve any issues before continuing.*

10. Now let’s check the latching feature of the SR-latch as well.

a. Use your circuit to complete each row in the truth table. Use S or R to set the value of Q needed for the row. Then set S=0 and R=0 to find the value of QNext.

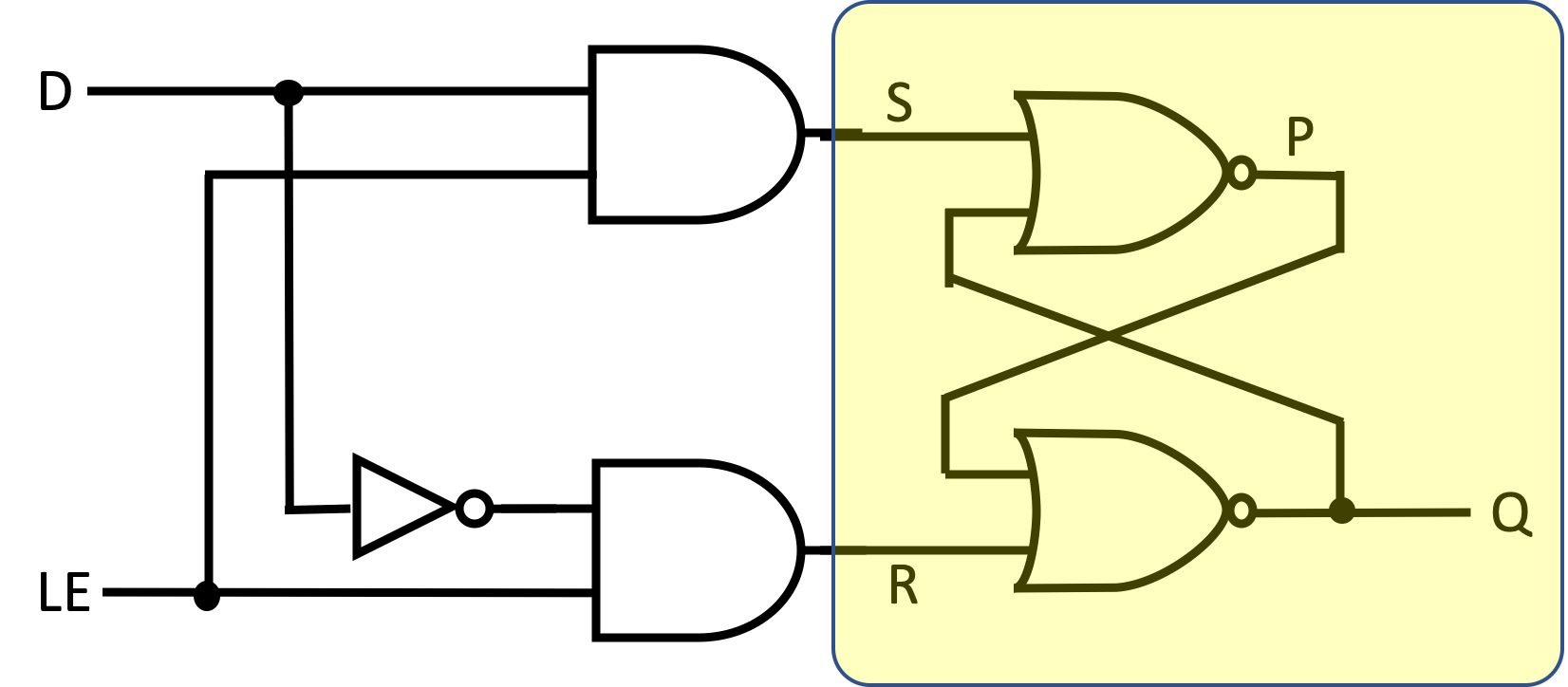
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  | **S** | **R** | **Q** | **QNext** |  |
|  | 0 | 0 | 0 |  |  |
|  | 0 | 0 | 1 |  |  |
|  |  |  |  |  |  |

b. What you observe in the table above should agree with what you predicted in question #4.

*No answer is required here, but you need to ensure that your circuit behaves as expected. If your circuit does not behave as expected revisit questions #4-#8 and resolve any issues before continuing.*

**The D-Latch:**

You may have noticed that thus far we have looked at every pattern of inputs for our SR-Latch except those when S=1 and R=1. This should make some sense, because it is unclear what it would mean to both Set and Reset the value of Q at the same time! For this reason, **the SR-Latch is often augmented with some additional circuitry to ensure that S and R cannot both be 1 at the same time. This configuration is shown below and is called a *D-Latch*.** Notice that the D-Latch contains an SR-Latch as a part of its circuit (highlighted in yellow).



Let’s explore the behavior of this D-Latch and check that it accomplishes its purpose.

11. One way to understand the behavior of a D-Latch is to consider how the inputs D and LE set the values of S and R, and then use our high-level abstract understanding of the SR-Latch to understand the behavior of the D-Latch. To facilitate that, complete the following truth table that shows the values of S and R for each pattern of the inputs D and LE.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  | **D** | **LE** | **S** | **R** |  |
|  | 0 | 0 |  |  |  |
|  | 0 | 1 |  |  |  |
|  | 1 | 0 |  |  |  |
|  | 1 | 1 |  |  |  |
|  |  |  |  |  |  |

12. Recall that the added circuitry in the D-Latch was intended to ensure that the situation where the SR-Latch is being both set (S=1) and reset (R=1) at the same time never occurs. Briefly explain how your table in #11 demonstrate that the D-Latch achieves this.

**The D-Latch is a 1-bit Memory:**

At the start of the lab, we used a box analogy to describe the desired behavior of a 1-bit memory cell. This section will have you explore the behavior of a D-Latch more fully and connect it back to the box in that analogy. The conclusion should be that the D-Latch is in fact behaving exactly like a 1-bit memory!

13. Let’s first consider the cases where LE=1. Use the table from question #11 and your high-level abstract understanding of how the S and R inputs to the SR-Latch set and reset its output Q to complete the following table showing the next output Q of the D-Latch (QNext). When you have filled in the value of QNext for a row identify the box image (i - iv) from the start of the lab that corresponds to the row.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  | **D** | **LE** | **S** | **R** | **QNext** | **Box Image** |  |
|  | 0 | 1 |  |  |  |  |  |
|  | 1 | 1 |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

14. From the table in question #13 you should see that when LE=1, the value of D becomes the next value of Q. Just as was the case in our box analogy.

*No answer is required here, but you need to ensure that you see the expected behavior. If you do not, revisit questions #11-13 and resolve any issues before continuing.*

Note: This is question highlights why we use the label LE. When LE=1 it *enables* the input D to be stored into the attached SR-Latch. For this reason, this input is usually called the ***Latch Enable*** and thus why the abbreviation LE is used to refer to it.

15. Now let’s consider the cases where LE=0. The table below shows all of the posibilites for D and Q when LE=0. Complete the table below showing the values of S and R based on the inputs D and LE. Then use your high-level abstract understanding of how the output of an SR-Latch is set and reset to fill in QNext. Finally, match each row to a box image (i - iv) from the start of the lab.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |
|  | **D** | **LE** | **S** | **R** | **Q** | **QNext** | **Box Image** |  |
|  | 0 | 0 |  |  | 0 |  |  |  |
|  | 0 | 0 |  |  | 1 |  |  |  |
|  | 1 | 0 |  |  | 0 |  |  |  |
|  | 1 | 0 |  |  | 1 |  |  |  |
|  |  |  |  |  |  |  |  |  |

16. From the table in question #15 you should see that when LE=0, the value of Q is not affected by any changes in D. Just as was the case in our box analogy.

*No answer is required here, but you need to ensure that you see the expected behavior. If you do not, revisit questions #11-13 and resolve any issues before continuing.*

17. Briefly explain how your answers to questions #13 and #15 provide evidence that the D-Latch in fact implements the 1-bit memory that we set out to create at the start of the lab.

**Building a D-Latch:**

Now that we understand that a D-Latch is a 1-bit memory, let’s build one.

18. Make a copy of your Lab04-SRLatch in TINKERCAD and name it Lab04-DLatch. Add to this circuit the necessary components to create a D-Latch. The two slider switches that you had should now be used as the D and LE inputs to the D-Latch. Be sure to update their labels.

Give a link to your completed TINKERCAD circuit here:

🏆 19. Experiment with your circuit to complete the table below. For each row, use the D and LE inputs to set the current output of the circuit to the indicated Q value. Then set the D and LE inputs to their values for the row. Then record the value of QNext that is generated by the circuit. Note that the two cells that are shaded will not get a box image\*\*.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
|  | **D** | **LE** | **Q** | **QNext** | **Box**  **Image** |  |
|  | 0 | 0 | 0 |  |  |  |
|  | 0 | 0 | 1 |  |  |  |
|  | 0 | 1 | 0 |  |  |  |
|  | 0 | 1 | 1 |  | **↑** |  |
|  | 1 | 0 | 0 |  |  |  |
|  | 1 | 0 | 1 |  |  |  |
|  | 1 | 1 | 0 |  | **↓** |  |
|  | 1 | 1 | 1 |  |  |  |
|  |  |  |  |  |  |  |

Note: **So why the two shaded cells?** All of the states to which you assigned a letter are called ***stable states***. That is, for the values of D, LE and Q in that row QNext is the same as Q. So, nothing is changing in the circuit – it is stable and will say that way until an input is changed. For the two shaded cells however QNext is different than Q. This represents the situation where things are still changing in the circuit (i.e. like when you did the value “chasing” in the SR-Latch until the values “settled down”). These states where the output is changing (i.e. Q≠QNext) are called ***transient states***. If you look closely at these transient states, you will see that the values of D, LE and QNext correspond to a different row of the table. The arrows in the shaded cells indicate the stable state that the circuit will “settle down” into.

**Making a Register:**

How does this D-Latch relate to a real computer? Well, consider the registers in the Knob & Switch computer. Each register has to store 16 bits. To do that the hardware could simply contain 16 D-Latches, one for each bit. To store a value into a register, the LE input for all 16 of those D-Latches would be set to 1 and the D input of each one would be set to the bit to be stored. Then the LE input would be set back to 0 to save (i.e. latch in) all of the values. This is very close to what happens in the registers in a real computer.

🏆 21. TINKERCAD includes a 74HC75 chip which contains four D-Latches (they call it a 4-bit latch). Use your favorite search engine to find a pin-out diagram for the 74HC75 chip and paste it below along with the URL where you found it.

🏆 22. Create a new TINKERCAD circuit named LAB04-74HC75. Connect two slide switches (one for D and one for LE), an LED (for Q) and one of the D-Latches on the 74HC75 chip. Experiment with your circuit to confirm that the D-Latch on the 74HC75 chip behaves just like the D-Latch that you built using logic gates earlier. Note: Some of the pin outs for the 74HC75 chip on-line use G instead of LE for the latch enable inputs. These pin outs usually also contain a output for each latch. This output will always be the logical opposite of Q. For our purposes you can simply ignore and use Q.

Give a link to your TINKERCAD circuit here:

🏆 🏆 23. **Optional Extra:** Create a new TINKERCAD circuit named LAB04-4Bit-Register. Build a circuit that uses all four of the D-Latches on the 74HC75 as a 4-bit register. Your circuit should have:

* 4 slide switches to specify the 4 bits (D values) to be saved into the register.
* 4 LEDs to show the contents of the register (Q values).
* A slide switch for LE that allows new values to be saved into the register.
* Use labels so that all switches and LEDs are easily identifiable.

Give a link to your TINKERCAD circuit here:

24. **Optional Extra:** Use your circuit from #23 to store the value 1110 into the register using unsigned binary. Paste a screen shot of your circuit with this value saved in the register as your answer to this question.

**Chip Pin-Out Reference:**

A picture containing clock

Description automatically generated

Optional: To help me improve and scope these activities for future semesters please consider providing the following feedback.

a. Approximately how much time did you spend on this Lab (include the 2-hour lab period in your total)?

b. Please comment on any particular challenges you faced in completing this Lab.

**Acknowledgements:**

Box images from Open Clip Art and used under a CC0 license.

