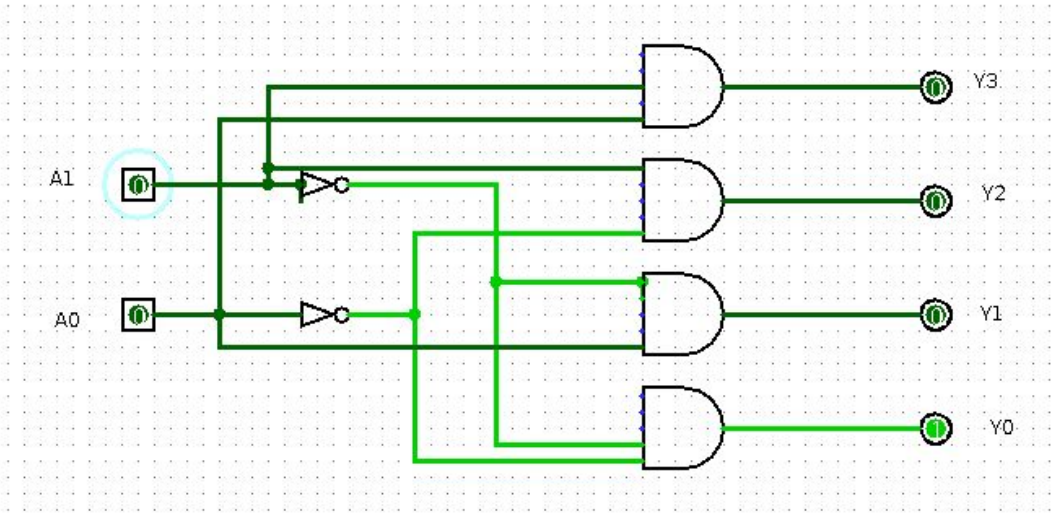
Almir Aljic
2018-11-01

Assignment 1: Decoder

Task 1.1



We were able to verify the correctness of this design by setting different values on our input bits A0 and A1. When both input bits are false (zero), all output bits except Y0 are false. When A0 is true, only Y1 is true and so on.

Assignment 2: Multiplexer

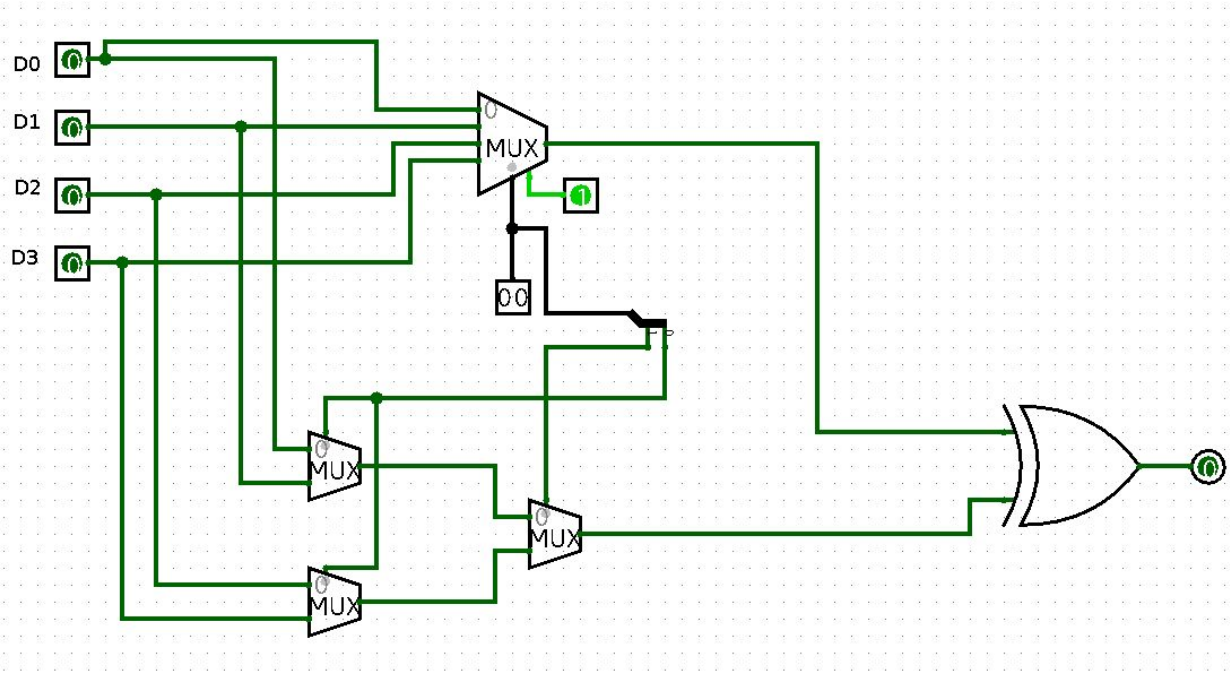
Task 2.1

The output MUX out is blue because the enable bit is zero. If we “turn it on”, i.e. make it true, i.e. set the enable bit to 1, the MUX out will light up in a proper dark green or light green color (color depending on the values of our input bits). Equal is red because it cannot determine a proper output, because we aren’t really feeding the gate any input values. The select signal of the 4:1 multiplexer is black because two bits are travelling through that particular wire (wires carrying multiple bits are colored black).

Task 2.2

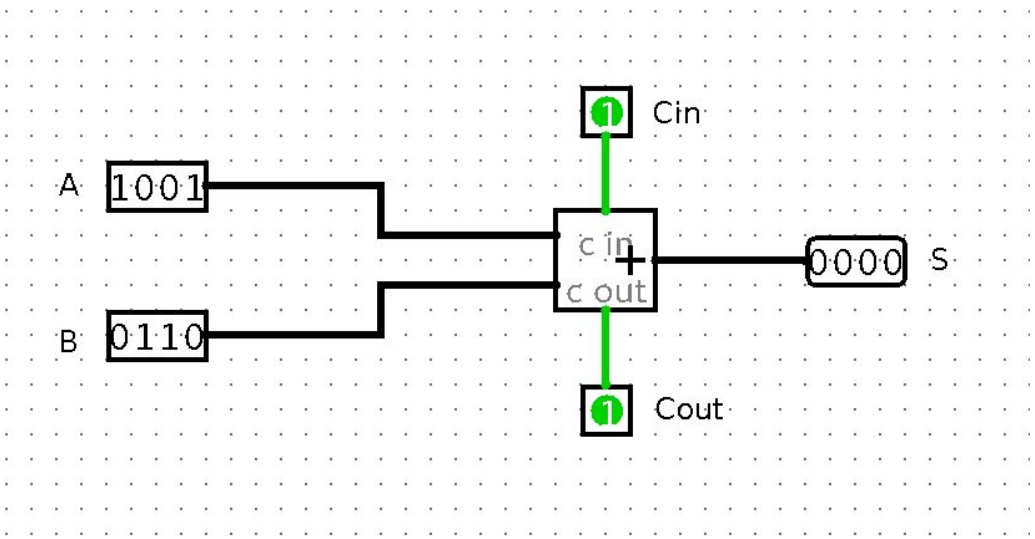
The XOR gate acts in a way that if both the 4:1 MUX and the three 2:1 MUX:s output the same bit, our equals output will be zero. However, should the two MUX constructions output different bits, our equals output would be 1, indicating that for the same set of input bits, we get different results based on which construction we consult. Thus, we are able to try the whole set of possible input combinations, and so long as our equals output remains zero, we know that the two MUX constructions return the same result. In other words, in using the XOR gate, we can test the correctness of our implementation.

Almir Aljic
2018-11-01



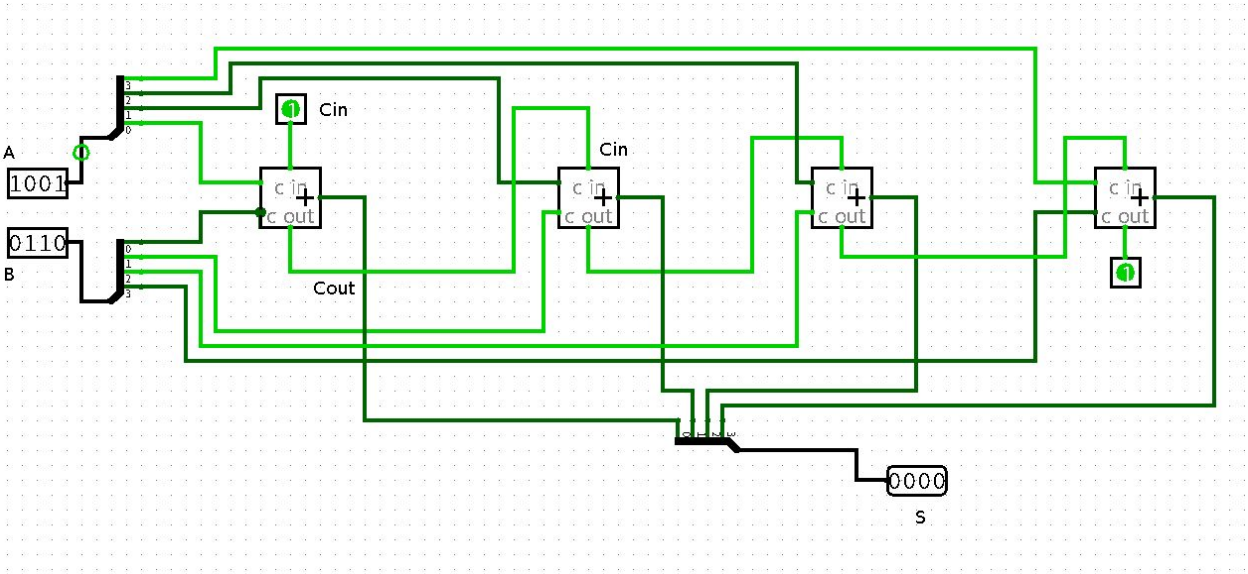
Assignment 3: Adder

Task 3.1



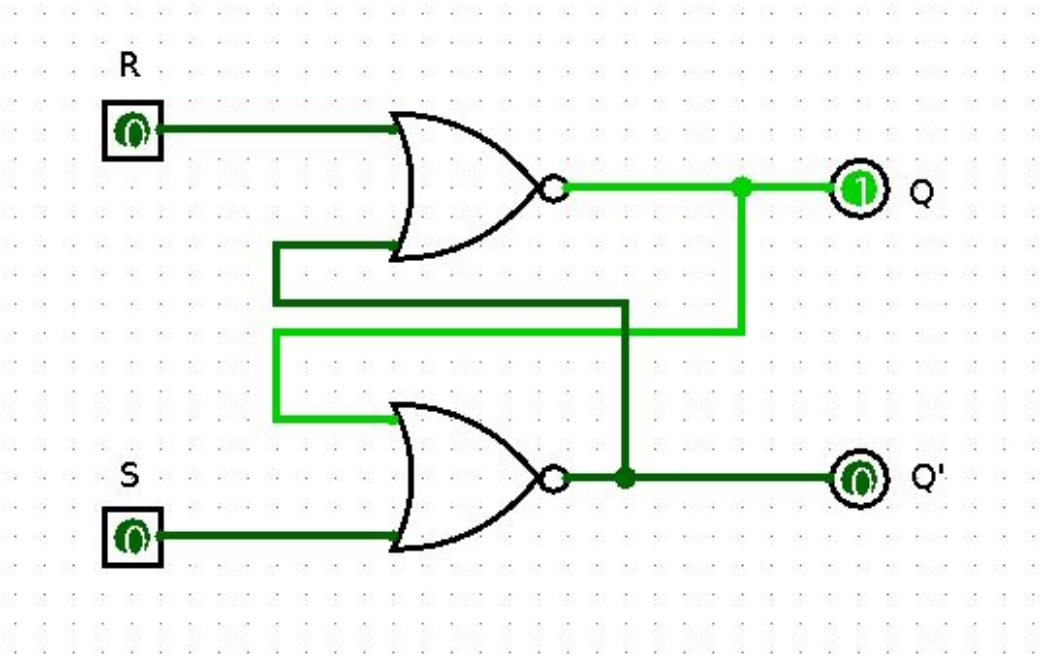
Almir Aljic
2018-11-01

Task 3.2



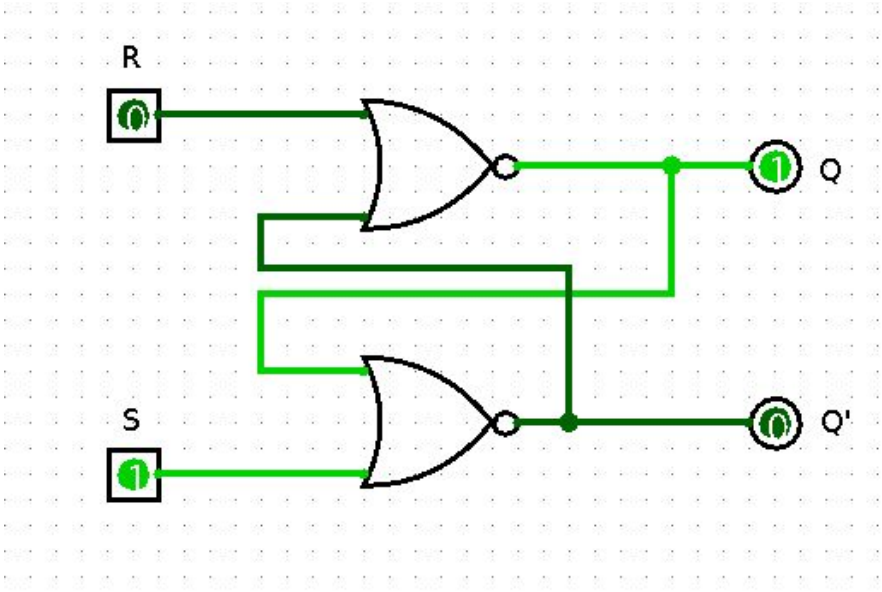
Assignment 4: Latches

Task 4.1



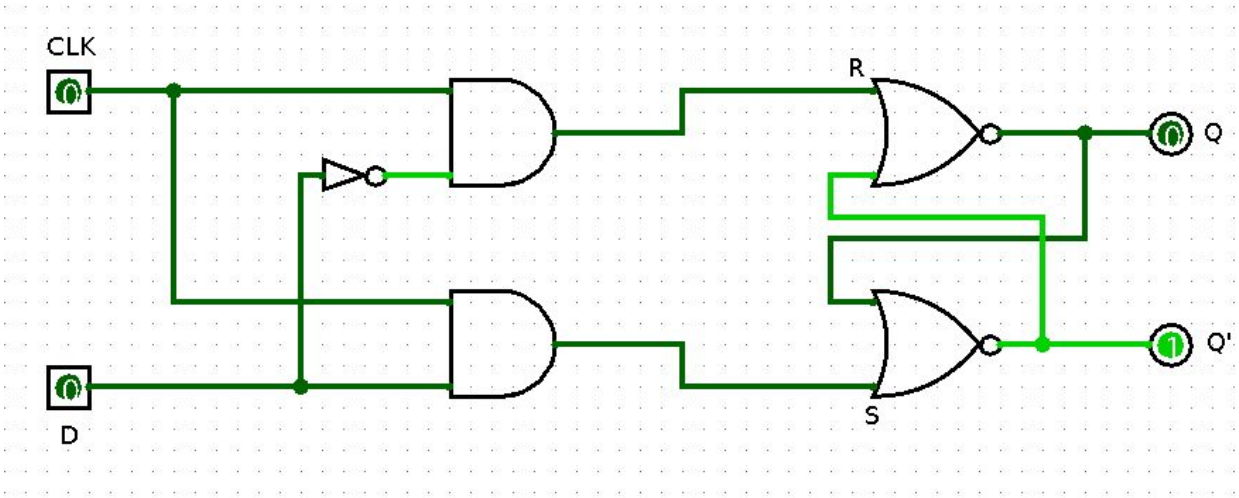
The set of possible values that Q and Q' can have are {0, 1}, such that Q' = Q + 1 (binary addition that disregards the carry bit). Their values depend on if R or S was previously set to 1,

before they both became 0. If S was previously set to 1, then Q = 1. If R was previously set to 1, then Q = 0. Below is a screenshot where S = 1 and R = 0.



If S = 1, then Q = 1 and Q' = 0. Those are the only possible values for Q and Q' given S = 1 and R = 0. If you just toggle S and switch between 0 and 1, Q and Q' will not change. Remember, the S ("set") signal merely sets the value of Q in a given moment. Then Q retains that value regardless of if you turn the S signal on or off. That is why to perform a reset (which resets Q from 1 to 0), you must enable the R ("reset") signal. However, one must be careful to ensure that the set signal is zero upon enabling the reset signal, for otherwise Q and Q' are both set to zero, a clash we definitely want to avoid.

Task 4.2



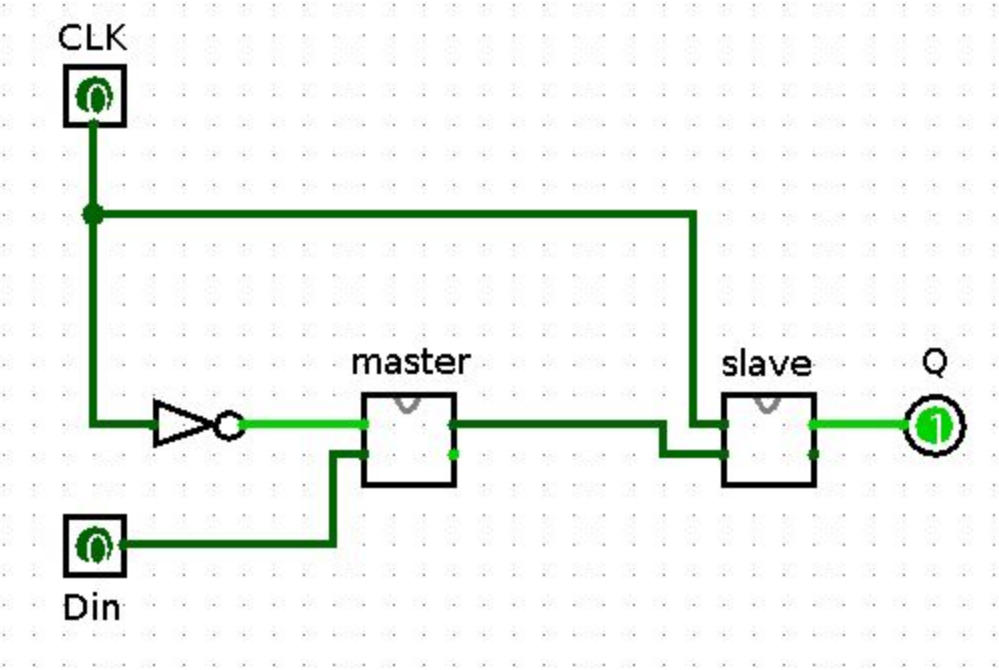
If we toggle CLK while D = 0, all that happens is we let the signal D go through to Q. If Q = 1 in a state of D = 0 and CLK = 0, then upon enabling CLK, Q will assume the value 0 (because D =

0). But if Q = 0 in a state of D = 0 and CLK = 0, then upon enabling CLK, Q will continue to be equal to 0 (because D = 0). Similarly, if CLK = 0 and we toggle D, then nothing at all happens. We can think of it as CLK being an enable signal, if CLK is off then D simply won't come through to Q. If CLK is on, Q assumes the value of D. The reason for this is because regardless of whether D = 0 or D = 1, the AND gates will both output the same bits: 0. So then R = 0 and S = 0 for any D in {0, 1}, given the condition that CLK = 0 (latch is *opaque*). In other words, nothing really changes.

Now if CLK = 1, then D acts as both a "setter" and a "resetter". This is also called that the latch is *transparent*. If D = 0, Q = 0. If D = 1, Q = 1. So we can set and reset the value of Q by toggling D. The problems with the SR latch were that if S = 1 and R = 1, then both Q and Q' were zero. We can't have that with the D latch, because there is essentially only one signal (D) that manages the set and reset operations. Also, the SR latch had the problem of mixing the issues of what and when updates are made. In the D latch, we have control over this - we can choose when updates are made (by enabling the CLK signal) and we can accurately choose which value we wish to store (by simply enabling or disabling the D signal). Synchronous sequential circuits are meant to operate under a global clock. The problem with the D latch is that is that it is asynchronous. When CLK is equal to one, then toggling D will change the value of output Q. These so-called "races" are not desirable in the context of why D latches are not used in synchronous sequential logic. Essentially, we want the inputs to all registers to settle before the next clock edge. This eliminates races and unstable behavior.

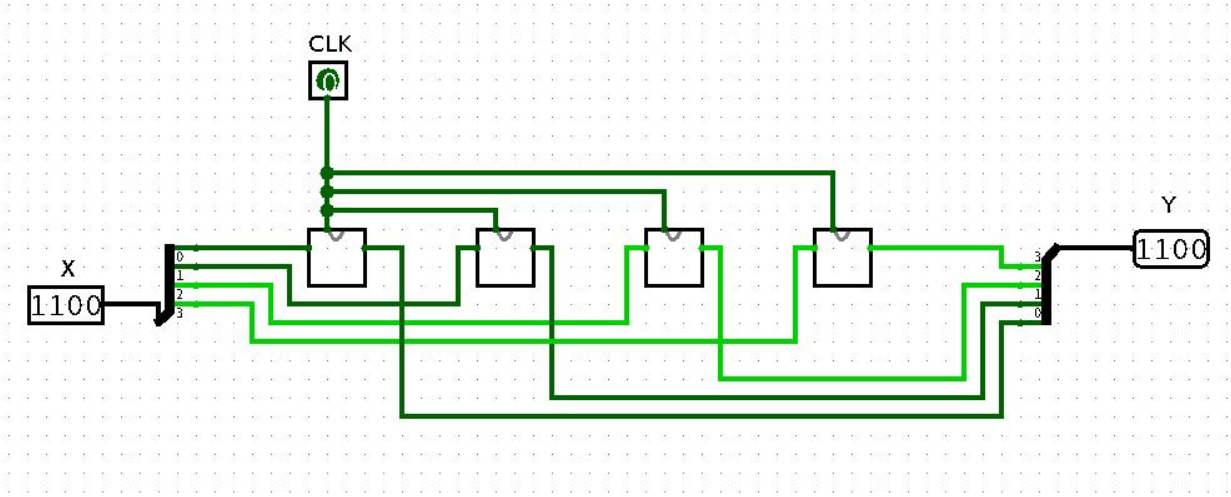
Assignment 5: D Flip-Flops and Registers

Task 5.1



When CLK = 0, the value of Din flows to the master D-Latch. When CLK = 1, this intermediate value continues flowing to the slave D-Latch. At that point, Q assumes the value of Din. So if Q = 1 and CLK = 0 and Din = 0, then the value of 0 is stored at the master D-Latch as an intermediate value. Then, if we were to set CLK to 1, this bit in the intermediate part of the circuit (the zero) will continue to flow to the slave D-Latch, which ultimately assigns the zero to Q. As a result, the value of Q would be set to 0 from its previous value of 1.

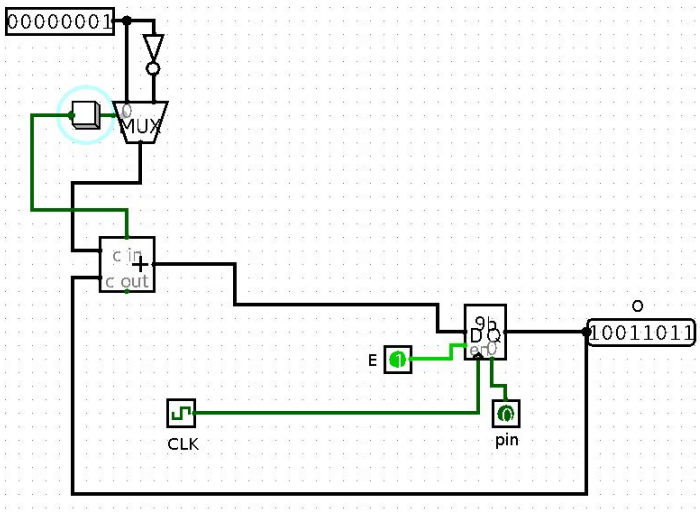
Task 5.2



Assignment 6: Design of a Synchronous Sequential Circuit

Almir Aljic
2018-11-01

Task 6.1



When pressing the button down (next to the MUX), the input bits will be sent through the NOT gate with 8-bit width, and those are the bits that will be selected by the MUX. This is a way of using an adder for subtraction - all we have to do is flip the bits of one of the inputs (think in terms of two's complement). If the button is not pressed down, our adder works as it usually does. Sequential means that a circuit retains past information - it has memory. Synchronous means that updating past information (memory) is regulated by one global clock. This assignment and the tasks in assignment 5 are examples of synchronous sequential circuits.