John Gangemi
Alexander Holst
Raj Patel
also known as **TEAM RAJ**
Friday, October 17, 2014
CMOS VLSI Lab 4

## Introduction & Background

In this lab, our task was to create the CMOS layouts for a D flip-flop, a bit slice shift register, and a bit slice bidirectional counter. The idea behind the lab was to introduce us to layout and simulation of basic sequential circuits, which are prone to more error and faulty behavior than combinational circuits. Each circuit is described below.

### D Flip-Flop

A flip-flop is a circuit that uses feedback to hold one bit of information. We were required to lay out a positive edge-triggered D-type flip-flop with asynchronous preset and clear. A D-type flip-flop means a flip-flop that chooses what state to store from a single-bit data input. Positive edge-triggered means a flip-flop that updates its state on the rising edge of a one-bit clock input. Asynchronous preset and clear mean a flip-flop with additional inputs that override the current state of the flip-flop. When *preset* is active, the state of the flip-flop will be forced high regardless of the data or clock inputs. When *clear* is active, the flip-flop likewise will be forced low.

### Bit Slice Bidirectional Shift Register

A shift register is an array of flip-flops for which, in our case on a rising clock edge, data in one flip-flop is shifted either left or right, if we imagine the array to be arranged horizontally. Bidirectional means that the shift register must support both methods of operation, toggled by an additional bit of input. A bit slice means that we are to lay out a single bit of a shift register such that it can be modularly combined with other bits to form a bidirectional shift register of arbitrary length. The shift register must also support asynchronous preset and clear, as described above.

### Bit Slice Bidirectional Counter

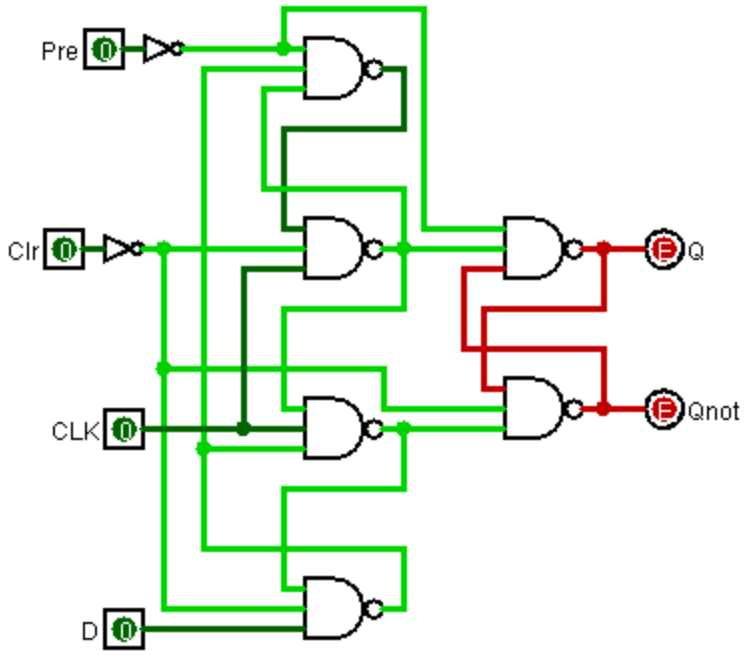
A counter is an array of flip-flops representing a binary number that increments or decrements by one on a rising clock edge, rolling over to 0 and $2^n$ on overflow and underflow, respectively, where $n$ is the number of flip-flops. Bidirectional means we must support both methods of operation, toggled by an additional bit of input. A bit slice means we must lay out a single bit counter that can be modularly combined to form a counter of arbitrary length. The counter must also support preset and clear, as described above.

## Design & Layout

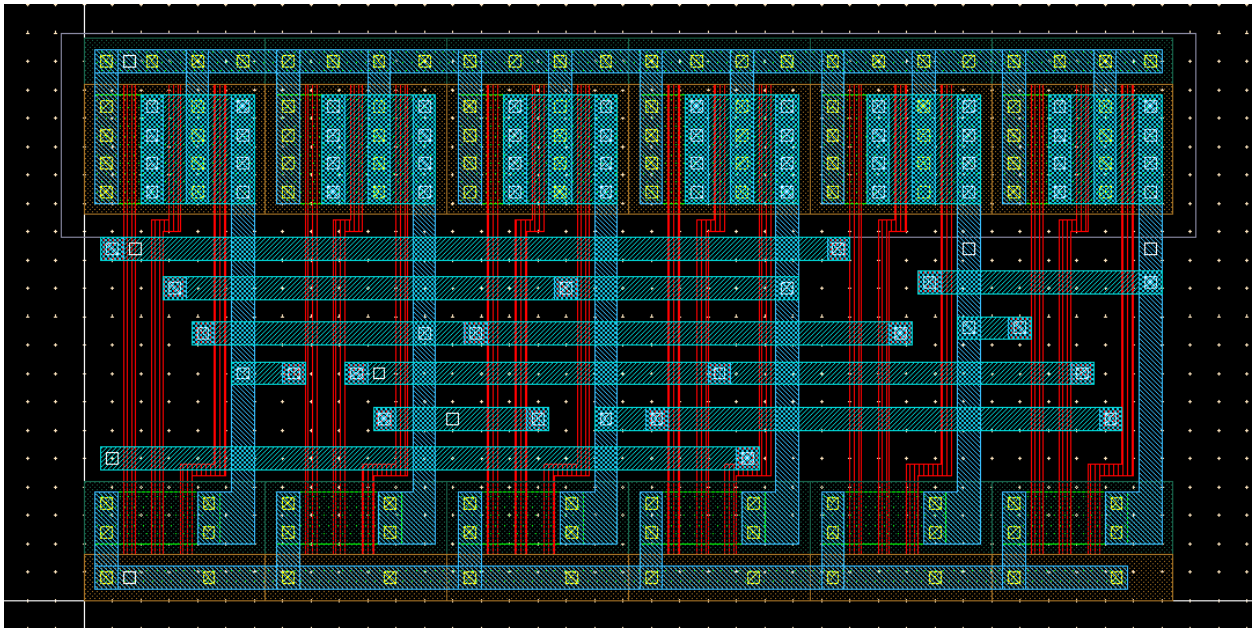
We opted to use a standard cell approach to produce layouts from our broadly NAND-based logic. We used the standard multiplexer created in the previous lab whenever it was needed, and we used the D flip-flop from the first part as a standard cell instance in the second and third parts.

**D Flip-Flop**

For our D-type flip-flop, we opted to use a six gate NAND approach with *preset* and *clear* as active low inputs. Clock and data remain active high. Having both *preset* and *clear* high at the same time is an illegal input, but in the case that *preset* and *clear* are both active at the same time, the outputs *q* and *q_not* will both transition high. The logic is shown here:

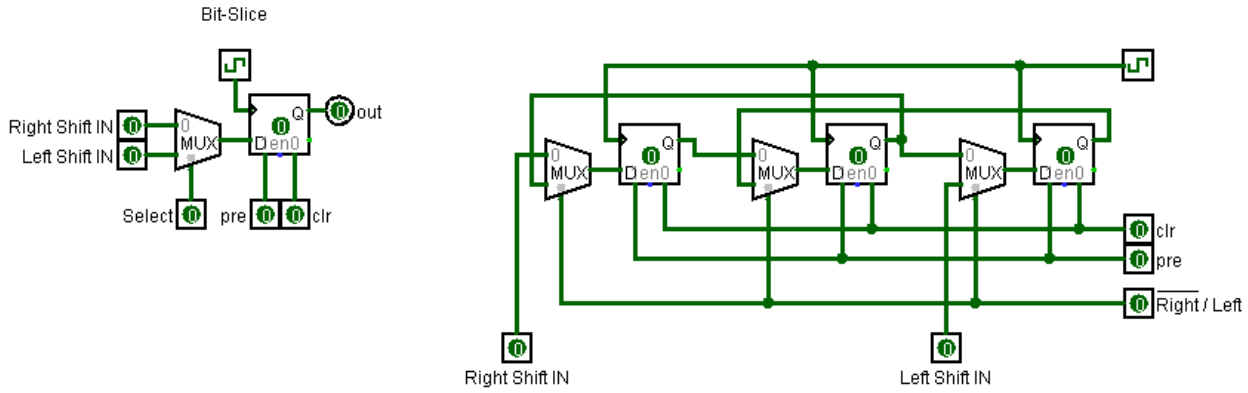



Layout was a simple matter of using six three-input NAND cells with interconnects established appropriately. The three-input NAND cells have bounding areas of 30 by 9.6, and the flip-flop has a **total bounding area of 30 by 57.6**. The layout is shown below:

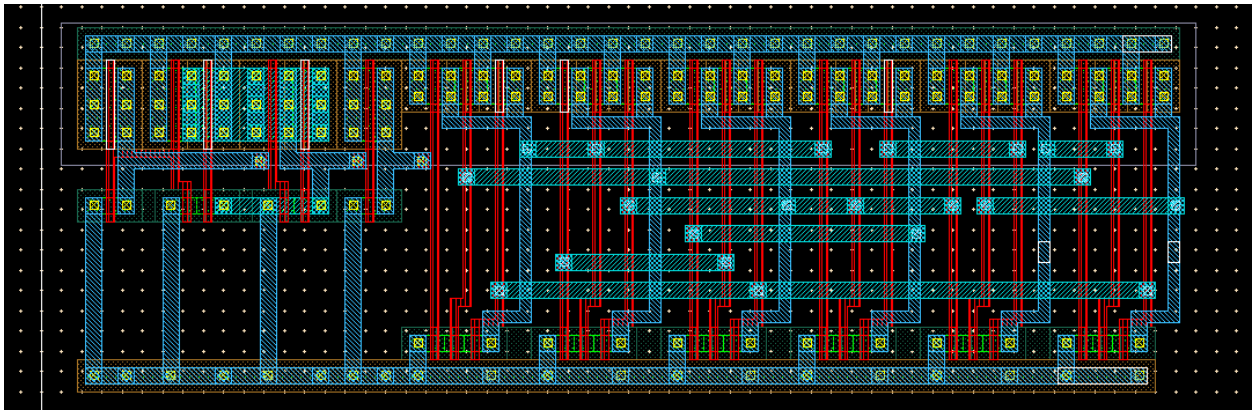


**Bit Slice Bidirectional Shift Register**

Our bit slice shift register takes two data inputs, one each for the outputs of the bit slices to the immediate left and right. It is assumed that the rightmost and leftmost inputs will be appropriately tied so valid inputs are shifted in. The bit slice takes a *select* input, for which low selects right shift and high selects left shift. The bit slice also takes active low *preset* and *clear* inputs and delivers *q* and *q_not* outputs. The logic is shown below:

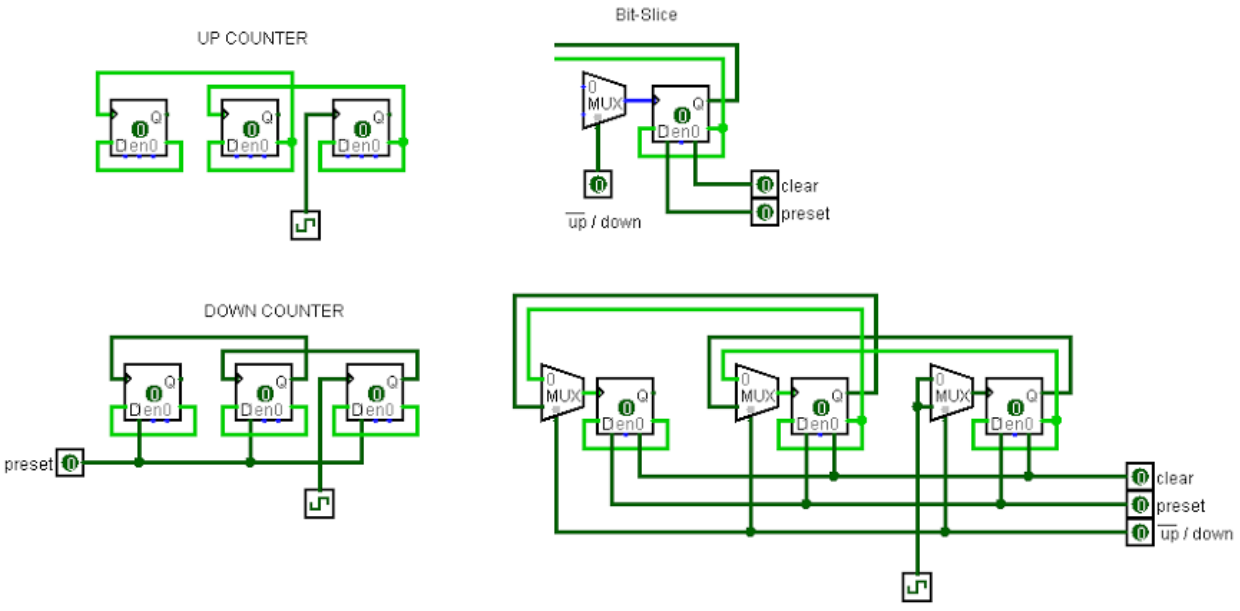



For layout, we used the D flip-flop and multiplexer we had already created and wired them together. Creating a shift register is simply a matter of wiring together multiple bit slices and connecting inputs appropriately. The bit slice has a **total bounding area of 27.3 by 84** and the layout is show below:



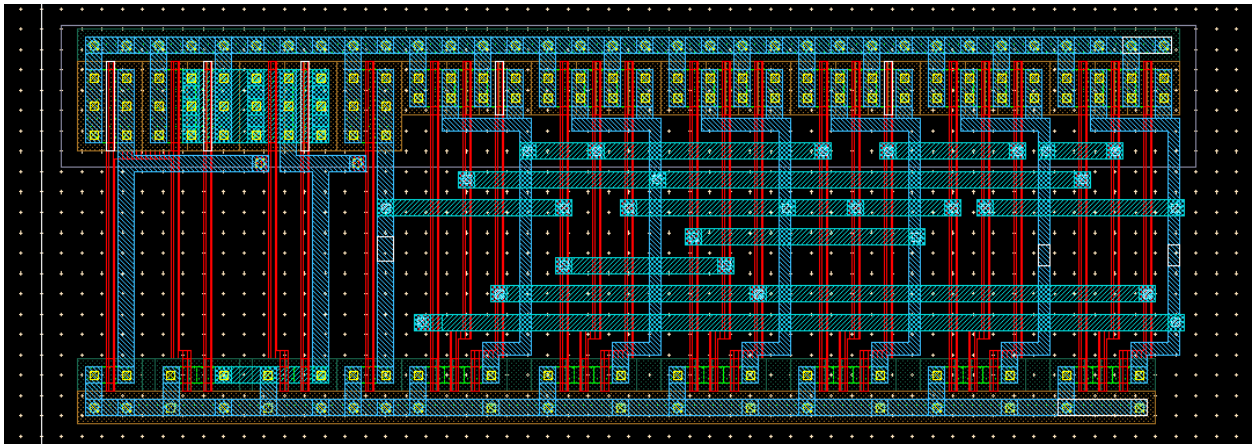
**Bit Slice Bidirectional Counter**

Our bit slice counter takes five inputs, a single-bit *select* for which low selects increment and high selects decrement, two single-bit data inputs of *q* and *q_not* into the multiplexer from the previous logic block, and *preset* and *clear* inputs. The counter, like the shift register, uses the D flip-flop and multiplexer we had already made. The main difference from the shift register bit-slice is that the output of the multiplexer feeds into the clock input of the flip-flop. In a n+1 stage counter using this design the clock is only sent to the first stage as the two inputs of the multiplexer . The data input of the flip-flop is constantly fed *q_not* therefore creating a 'toggle' flip-flop (the most common bit-storage element used in implementing counters). The logic is shown below:




For layout we wired instances of other layouts we had already used together (mux and D flip-flop), making sure to leave the inputs and outputs in places where it would be easy to wire

another bit slice in a chain together. The bit slice has a **total bounding area of 29.4 by 84** and the layout is shown below:



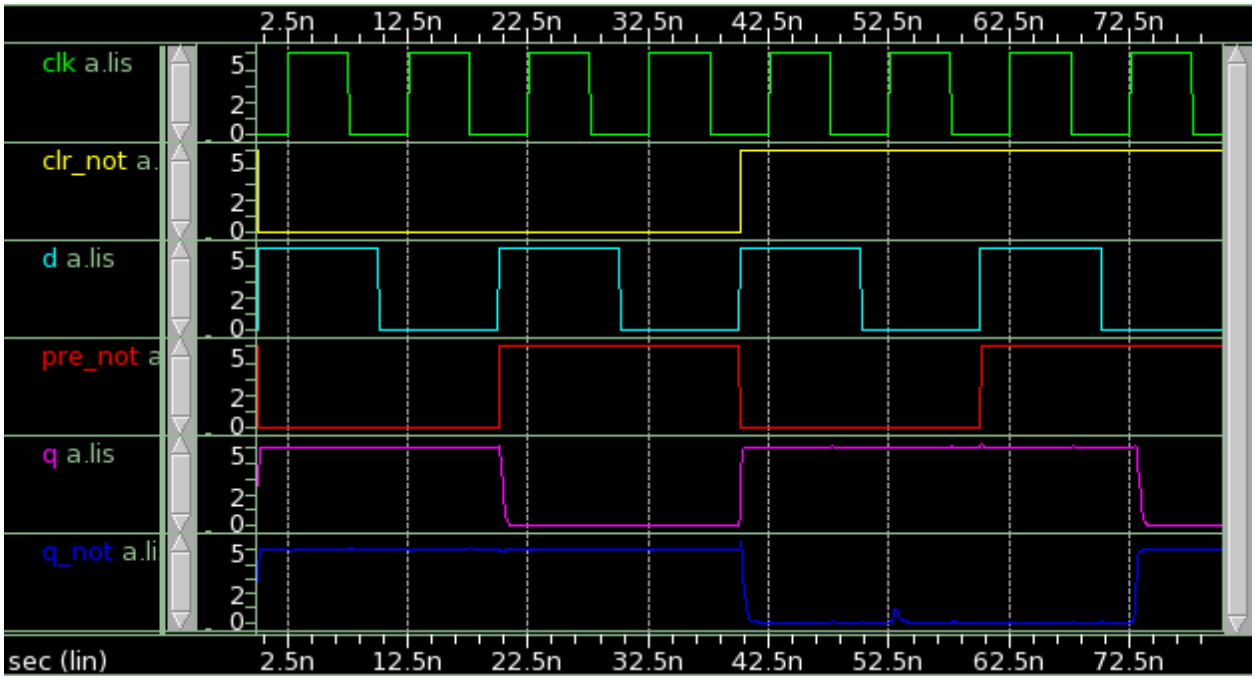
A side note on our design decision for the bit-slice counter:
In order to meet the design requirements listed in the lab documentation and to appropriately practice the theory of bit-slicing our implementation of the bit-slice counter is in all aspects a slightly modified toggle flip-flop (can be shown with a two state Moore diagram with transitions from 0 to 1). This design does have some operating conditions such as asserting clear or preset while changing the up/down control signal and therefore the current values in the flip-flops will not be stored. However this can be circumvented using another set of storage elements that will save the current state of the flip-flops when the up/down signal is asserted and load the counter flip-flops on the next clock cycle.
Had the design documentation for lab suggested we create a 4-bit counter (e.g. 74169 TTL IC) then the encompassing combinational logic from stage to stage could be included to correctly handle design limitations like not saving state when changing direction. Also the addition of a ‘ripple carry’ signal to assert the maximum binary value has been reached would allow the cascading of multiple 4-bit counters.

## Testing & Results

With combinational circuits, testing consisted of running through every possible input combination. With sequential logic, we must now factor in the state of the component to be thorough. To do so, we ran through every possible input combination for every possible state, and checked to see if the output waveform was what we expected. We made sure to give each input signal a rise and fall time of 0.1 ns and each output a capacitive load of 0.01 pF.

### D Flip-Flop

The D flip-flop has two possible internal states: $q$ = 0 and $q$ = 1. There are four inputs, meaning $2^4$ binary combinations. We ensured thorough checking by having the clock run twice as fast as the data input and out of phase to ensure transitions did not occur at the same time. The output waveforms are shown below:

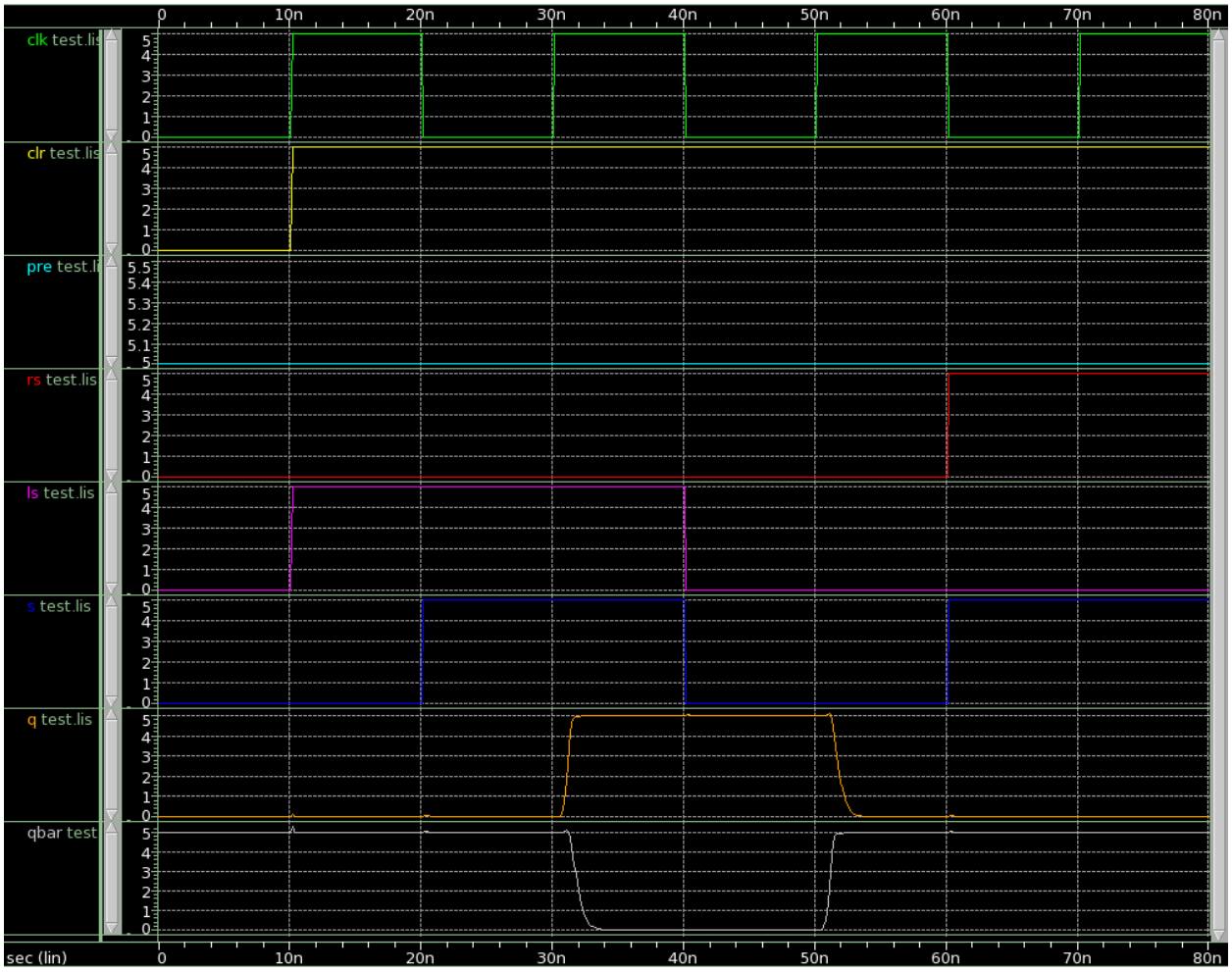



We can see that when *preset* and *clear* are both high (inactive), the flip-flop latches saves the correct data, and that *preset* and *clear* ignore the clock and data inputs as they are supposed to. We also see that when *preset* and *clear* are both low (active), the outputs behave as we described. From this, we can be confident that our flip-flop will behave as expected.

### Bit Slice Bidirectional Shift Register

Testing a shift register is a little more tricky. We tested by first clearing the circuit, then running through a series of shifts (altering ‘rs’ and ‘ls’ values while changing the mux control signal) to see whether changing shift direction worked or not. We had *preset* and *clear* (active low inputs)

for each bit slice tied to unified rails, so we tried clearing the circuit midway through. The output waveforms are shown below:

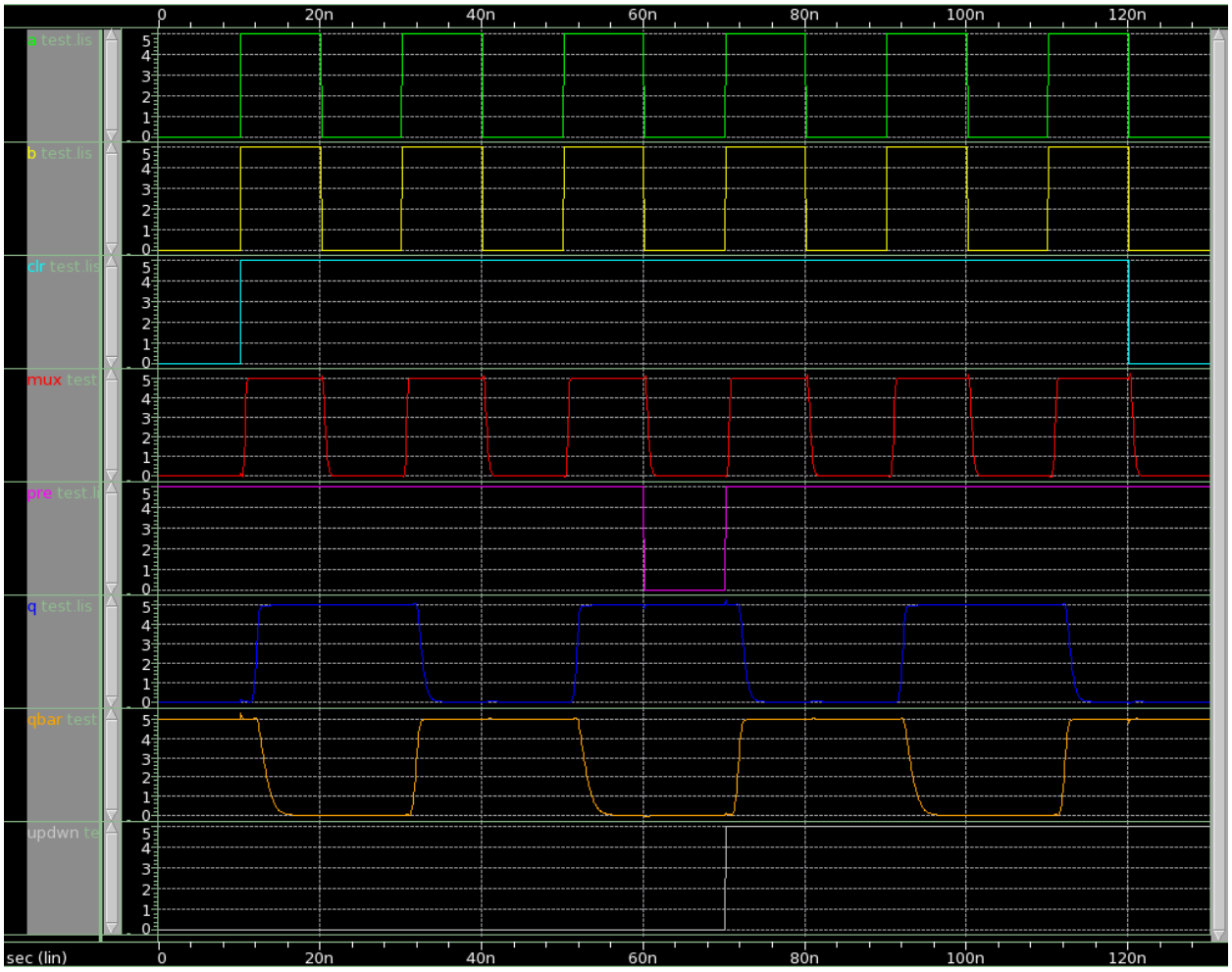



We can see that the shift register is indeed shifting in the values it should, that *preset* and *clear* override any state, and that changing shift direction works. We can conclude that our bit slice is functional.

**Bit Slice Bidirectional Counter**

Given the bit-slice implementation is by all means a toggle flip-flop we decided to test the circuit in such a manner that the output would toggle appropriately. Assuming this was the first stage of a n+1 counter then the clock would be sent to both inputs of the 2:1 multiplexer so inputs ‘a’ and ‘b’ have the same signal pattern and when asserted the flip-flop will change state (toggle).

The output is shown below:




From this, we can see that the counter will increment, decrement, reverse direction, and overflow and underflow as we intended. Also the clear and preset signals being active low work properly.

**Conclusion & Feedback**

Our results indicate that our implementation of a standard cell flip-flop and bidirectional bit slice shift register and counter built on that implementation are functionally correct. Our bounding areas were also uniform and of decent size , so we can consider this lab a success.

**Trivia**

A flip-flop while crucial to computer logic design is also a tool used to protect one's feet from blistering hot sand found in the vicinity of all coastal regions. The implementation of flip-flops should never be done in conjunction with socks, eww.