

Frank I Gomez

5/3/21

Lab 4, Section D.

ID: 1650550

figomez

CSE 100L

## Lab 4 Write Up

### 1. Description

The purpose of this lab was to create a 16bit counter, using flip flops to store the current value of the counter. We were allowed to use only the system clock obtained from the lab4\_clks file, not clkin or digsel, unless otherwise specified.

- Top Level

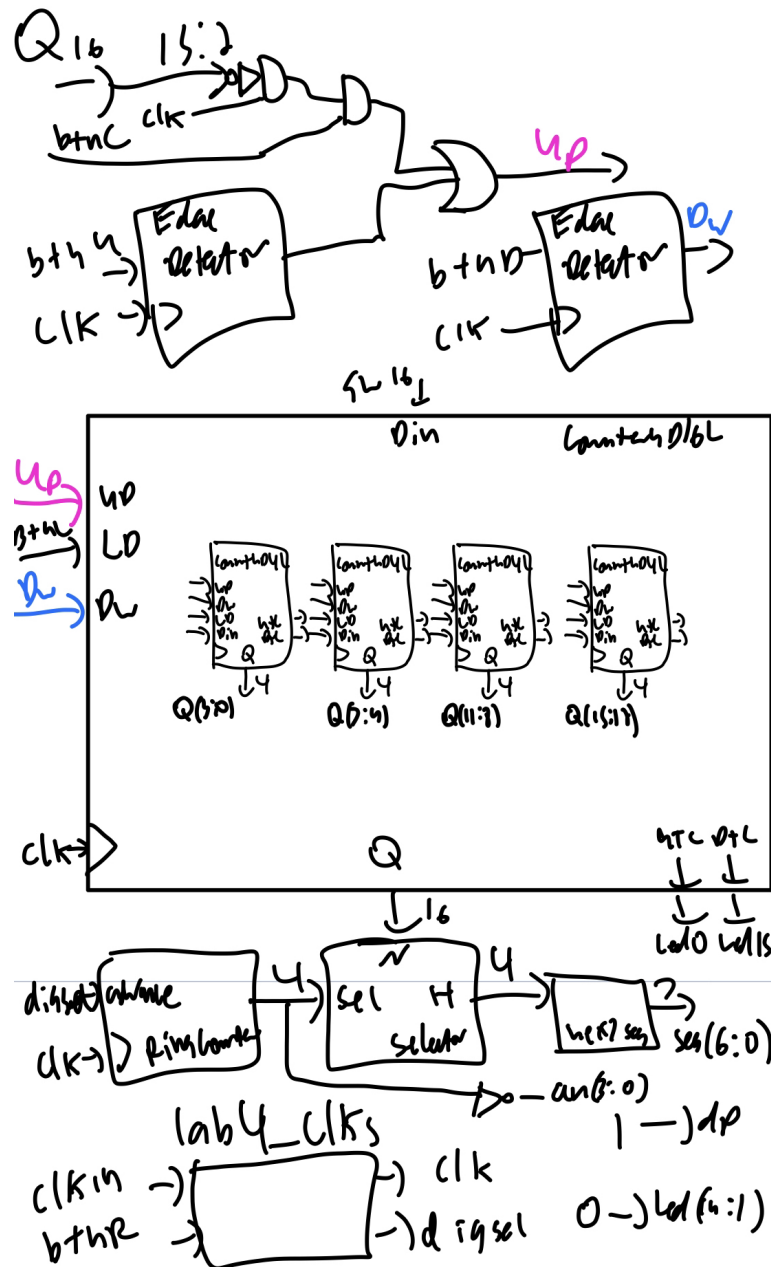


Diagram 1

Even more stuff going on this time around. First thing to note is lab4\_clks. That module creates the clk input used pretty much everywhere in this lab. Now, starting from the top down, those double-AND gates at the top handle when btnC is down, and they're OR'd with the output of the btnU edge detector, since they're both used for handling an "Up" input. Likewise, next to them, btnD uses an edge detector for the same reason, and they all input into counterUD16L, alongside

btnL. BtnL corresponds to LD, which, when active, will load the 16bit binary value stored in the 16 switches into the counter. Inside the 16bit counter are 4 4bit counters, which store the current counter value, and output UTC and DTC at F and 0 respectively. When all 4 counters return UTC or DTC, the corresponding leds (0 and 15) will be lit on the display. The 16 bit value then leaves the counter, being forwarded into the Selector. The selector iterates between 4 bits of the value, thanks to the input from RingCounter, and then passes said value to hex7seg, which does the same thing as the last lab, displaying the hex value. For brevity, two ring counters were technically used, as a second ring counter iterates the 4 AN values as well, changing which digit of the seven segment display is lit, but it's included in the diagram as it is to save space. Finally, leds 1-14 are set to 0, and dp is set to 1, powering them off.

- counterUD16L

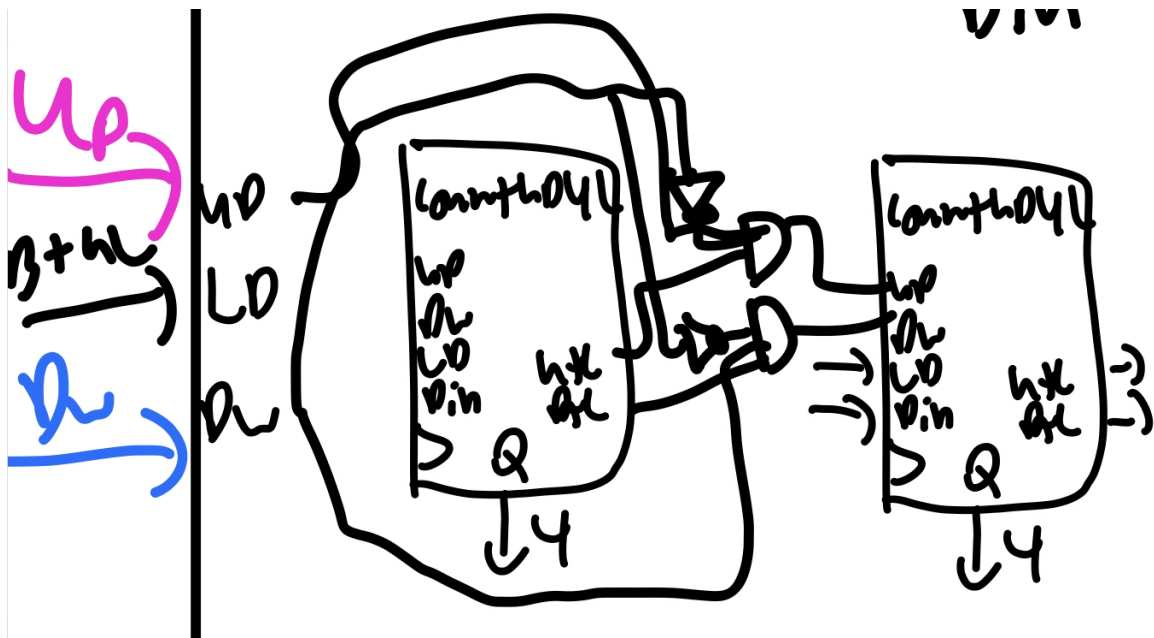


Diagram 2

The part I want to explain here is the difficulty behind getting counters beyond the first to increment/decrement. They only increase when an Up value is received alongside the ALL the previous UTC values, and only if Down is not being pressed at the same time. The same follows for Down, DTC, and not Up. While it makes sense, it leads to the diagram looking like this mess, three times over by the end.

- countUD4L

As for counting Up and Down at the local level, there's actually a really nice equation provided in class for this exact thing.  $D_n = Q_n \oplus (Q_{n-1} * Q_{n-2} * \dots * Up)$  and

$D_n = Q_n \oplus (!Q_{n-1} * !Q_{n-2} * \dots * Dw)$ . Using this, each bit will only flip when all the values before it are 1, or when all the values before it are 0. There's a little more logic, checking if LD is pressed, which overrides all iteration, in addition to doing nothing if Up and Down are held at the same time. These values are then loaded into the flip flops. UTC and DTC are high at all 1s and all 0s respectively.

- edge\_detector

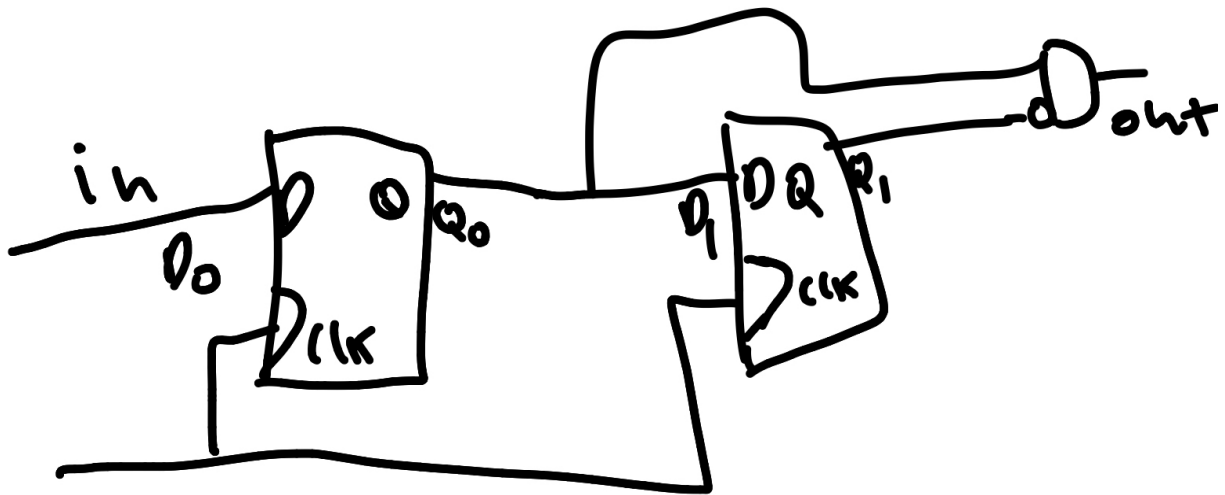


Diagram 3

Now for something a little simpler, and feasible to display, the edge detector. Simply put, when the edge detector receives a 0, followed by a 1, it will return a 1. Two 1s, two 0s, and a 1 then a 0 will return 0.

- ring\_counter

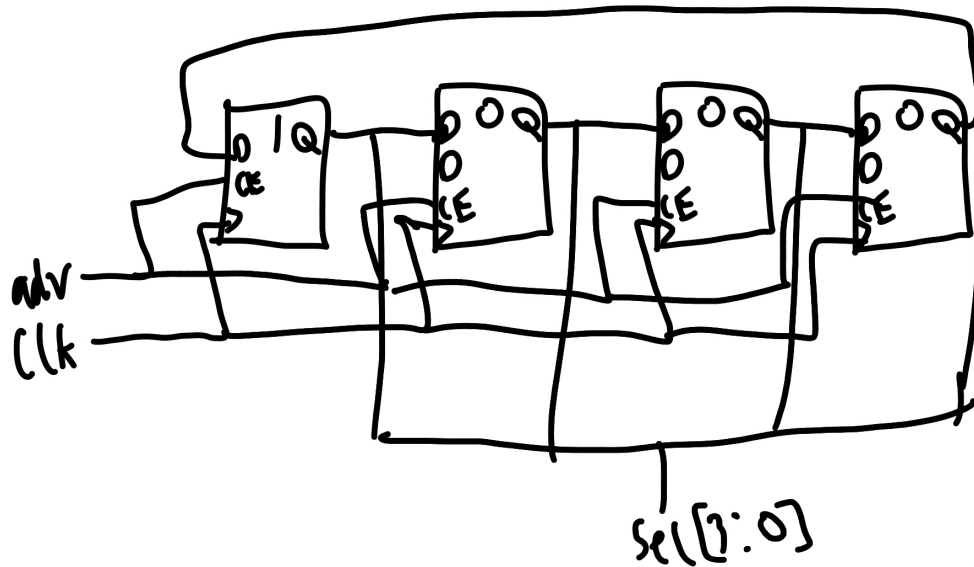


Diagram 4

Also an easy explanation. There's 4 flip flops, with one of them initialized with a value of 1. Every clock cycle, the output of each flip flop will be passed to the next, with the last flip flop passing back to the first flip flop, shifting the single bit around between them. This is the only module to make use of digsel instead of clk.

- Selector

Here's how the selector works; it takes a 16bit bus from the 16bit counter, and a 4bit bus from the ring counter. The ring counter will always be 0001, 0010, 0100, or 1000. Using these 4bits, the selector divides the 16bit input into 4 groups of 4bits, and passes those 4bits to the hex7seg. It does this fast enough so that on the hardware, all 4 values are displayed at once.



- lab4\_clks

Takes in verilog's clock and breaks it into a cycle of rising and falling edges, to allow the entire file to function on a cycle. Also provides digsel for the ring counter.

### 3. Testing and Simulation

This lab contained some especially frustrating bugs including, but not limited to: btnC adding an additional count beyond FFFC, btnC working on the simulator but not on the board, misunderstanding of how flip flops worked, creating race conditions, and more of the sort. In the end, nearly all of them were sorted thanks to the assistance of the lab TAs or the professor's lecture slides, as many of my issues (such as my logic for counting), were actually given by the professor in lecture, and if I had been paying closer attention, I could have saved myself a lot of time. Otherwise, the simulation went alright once I rewrote all the parts I didn't understand the first time around.

#### 4. Conclusion

In this lab we learned how to make use of flip flops to store values, allowing us to create a counter that doesn't need to have itself as an input, accidentally causing a race condition.

Honestly, all my gripes with this lab would have been solved if I had just paid more attention in lecture, as my 4bit counter took up more time than any other module, and it was literally given to us in lecture. No comments on what should be changed, if a student doesn't listen in lecture then it's their loss. Makes one wonder if my problems with labs 2 and 3 could have been for the same reason.

## 5. Appendix

(Note: Waveform Viewer cannot be printed to PDF, and as such, it is added here as a screenshot.)

(Note: hex7seg and m8\_1e are from lab 3, and as such, will not be included.)

