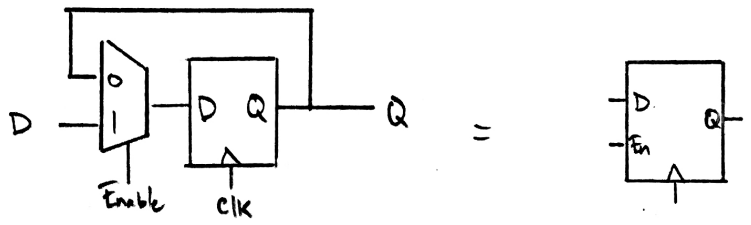
QUIZ 3, ECE 152A, Spring 2019 (Duration 20 minutes)

Name: Solution

(10 points) In class we studied the D Flip-Flop (D-FF). Often, it is desirable to have an enabled D-FF i.e. When an enable signal is high the system functions as a normal D-FF but when the enable is low the D-FF holds the data irrespective of the clock signal. Design a D-FF which has an enable signal implementing this functionality i.e Enable = 1, D->Q on clk rising edge, when Enable = 0, Q[n] = Q[n-1] (hold the previously stored value)

We want to use Enable to control the input to a normal DFF, so an easy way to implement this is to use a MUX

When Enable=1
Q[n] = D[n]
When Enable=0
Q[n]=Q[n-1]



- Note that it is a bad idea to gate the clk signal with an asynchronous enable, since it can create glitches and runt pulses.

2. Often we want to clock the data into a device serially and read it out parallelly (called a SPI: Serial-Parallel-Interface). Consider a 16 bit serial input register (consists of 16 D-FF connected serially to each other i.e D[i]=Q[i-1]).

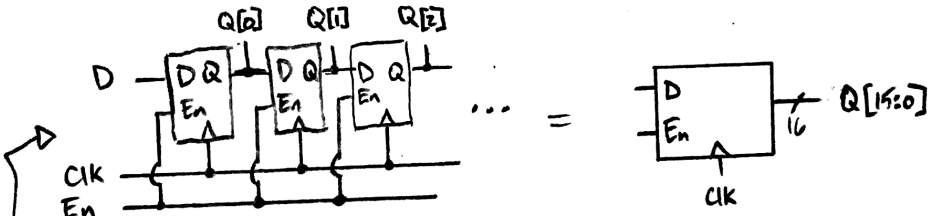
a) [5 points] How many clock cycles do you need to load the data into this register?

b) [10 points] Design a control state machine that can precisely clock in the data. Assume you assert a load signal for one cycle to start the loading process. You might want to use the enable D-FF you designed in question 1 (you can use a block diagram). Once the load signal is asserted high for 1 cycle, it is ignored until the number of clocks needed to load the data word has elapsed and then it waits an additional cycle to enable the load process again.

a) If we have 16 Serial FF's, it takes 16 clock cycles to fill up the register.

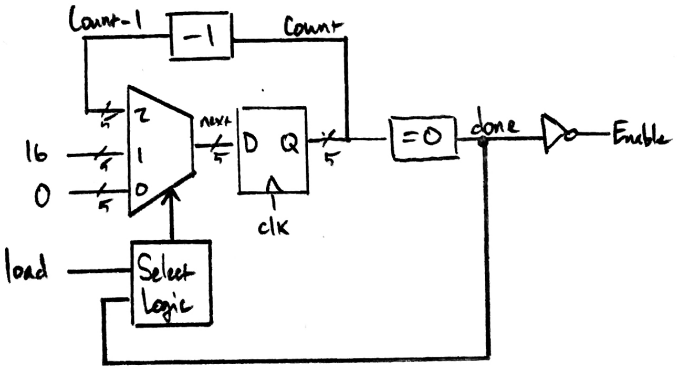
b) We only want the register to load data if the "load" signal is asserted, so the enabled DFF's from Problem 1 are useful. Once the circuit detects a load signal, the Flip Flops should be enabled for 16 cycles to load all the data. Then, the circuit waits another cycle (presumably so the parallel data can be read out). The load signal should therefore be ignored for 17 clock cycles once asserted high, while an enable signal to the register is pulled high for 17 cycles. This is simple to implement with a timer

For Simplicity, let the following represent a 16-bit Shift Register w/ common clk and Enable



17-Cycle Timer

- Need 5 bits since 4 bits can only represent 16 states
- If load signal asserted, set state to 16
  - Then, count down to 0 (takes 17 cycles)
  - while counting down, ignore load signal, and assert Enable to shift Register
- To implement, use mux to select input to an FF register. Put Decrement block in feed back to count down. Use CL to control mux. Hold mux input at 0 if done counting and load signal not asserted, keeping the timer primed for next load signal. Want to hold enable high during counting, so it can be the complement of the "done" signal.



Putting It Together