Lecture 9: Flip-Flops, Registers, and Counters

1. T Flip-Flops toggles its output on a rising edge, and otherwise keeps its present state.
   1.1. Since the toggle from high to low to high takes two clock cycles, the output frequency will be half of the clock frequency.
   1.2. Designing a T Flip-Flop (that toggles the output) from S-R Flip-Flops
   1.3. Karnaugh Maps for S-R Flip-Flops and T Flip-Flops, where *Q* is the present state, and *Q'* is the next state.

Q'	SR = 00	SR = 01	SR = 11	SR = 10
Q = 0	0	0	d	1
Q = 1	1	0	d	1

Q'	T = 0	T = 1
Q = 0	0	1
Q = 1	1	0

S	T = 0	T = 1
Q = 0	0	1
Q = 1	d	0

R	T = 0	T = 1
Q = 0	d	0
Q = 1	0	1

   1.4. To solve this we note what SR combination(s) will create each T next state for each Q.
      1.4.1. If $T = 0$ and Q = 0, then Q' = 0. If Q = 0 and Q' = 0, then $S = 0$, and R is in either state.
      1.4.2. If $T = 0$ and Q = 1, then Q' = 1. If Q = 1 and Q' = 1, then $R = 0$, and S is in either state.
      1.4.3. If $T = 1$ and Q = 0, then Q' = 1. If Q = 0 and Q' = 1, then $S = 1$ and $R = 0$.
      1.4.4. If $T = 1$ and Q = 1, then Q' = 0. If Q = 1 and Q' = 0, then S = 0 and $R = 1$
      1.4.5. Solving for S in terms of T and Q using the S K-map, $S = T\bar{Q}$
      1.4.6. Solving for R in terms of T and Q using the R K-map, $R = TQ$
      1.4.7. So to use an S-R flip-flop as a T flip-flop, we need to connect an AND of T and $\bar{Q}$ to the S input, and connect an AND of T and Q to the R input.

2. J-K Flip-Flop
   2.1. The J-K flip-flop augments the behavior of the S-R flip-flop (J=Set, K=Reset) by interpreting the $S = R = 1$ condition as a "flip" or toggle command.
   2.2. To create a J-K flip-flop from an S-R flip-flop, we'll create a truth table. The truth table starts with all the combinations of J, K, Q, and their resulting Q'. After filling the Q', we fill in the S and R that will create that Q' given the row's Q. Once the table is complete, generate the S K-map based on JKQ combinations and S value for each combination. Do the same thing for the R K-map.

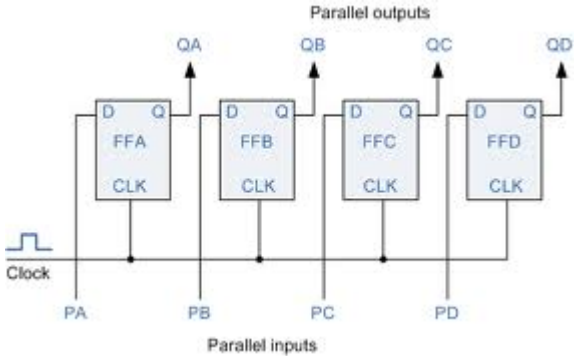
J	K	Q	Q'	S	R
0	0	0	0	0	d
0	0	1	1	d	0
0	1	0	0	0	d
0	1	1	0	0	(1)
1	0	0	1	(1)	0
1	0	1	1	d	0
1	1	0	1	(1)	0
1	1	1	0	0	(1)

S	JK = 00	JK = 01	JK = 11	JK = 10
Q = 0	0	0	[1]	[1]
Q = 1	d	0	0	d

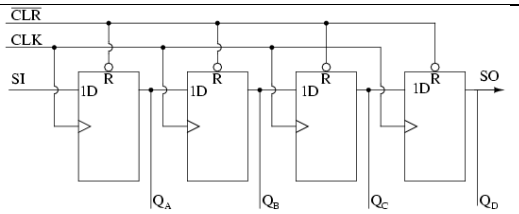
R	JK = 00	JK = 01	JK = 11	JK = 10
Q = 0	d	d	0	0
Q = 1	0	[1]	[1]	0

   2.3. Based on the S K-map, $S = J\bar{Q}$, and based on the R K-map, $R = KQ$. Again, the addition of two AND gates can transform an S-R Flip-Flop into a J-K flip-flop.

3. Register = a set of flip-flops used to store *n* bits of information. A common clock is used for each flip-flop in the register.
   3.1. Parallel Register = a set of 1-bit memories that can be read or written simultaneously, e.g. CPU data registers, AX in Intel.

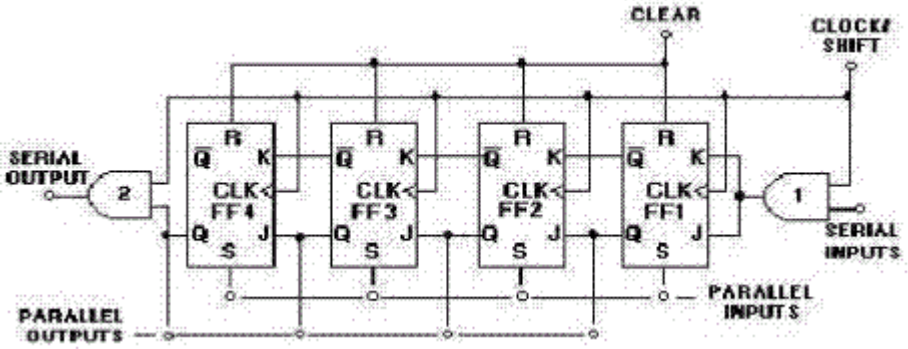


   3.2. Shift Register = a register that accepts and/or transfers information serially.
      3.2.1. Used to interface to serial I/O devices, to delay signals, and to implement a logical shift within an ALU.



Serial-in/ Parallel out shift register details

SIPO

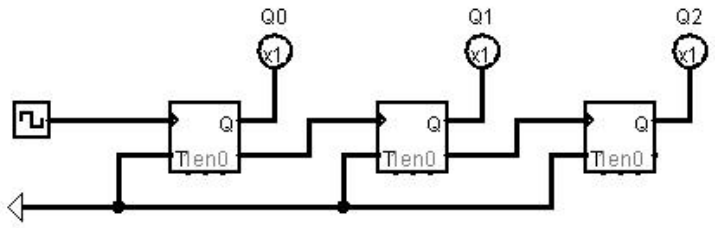


This circuit can be SIPO, PISO, or act as logical shift (PIPO)!

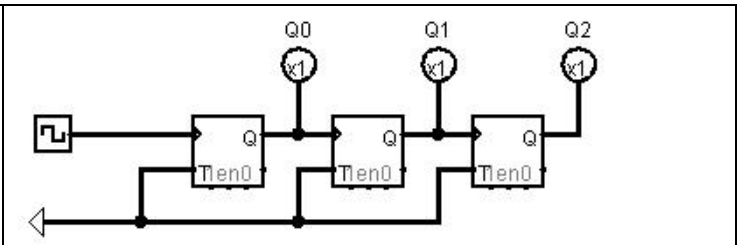
4. Counter = a register whose value is easily incremented by 1 modulo the capacity of the register.

4.1. Ripple (Asynchronous) Counters = each FF waits for the previous FF before processing the signal.

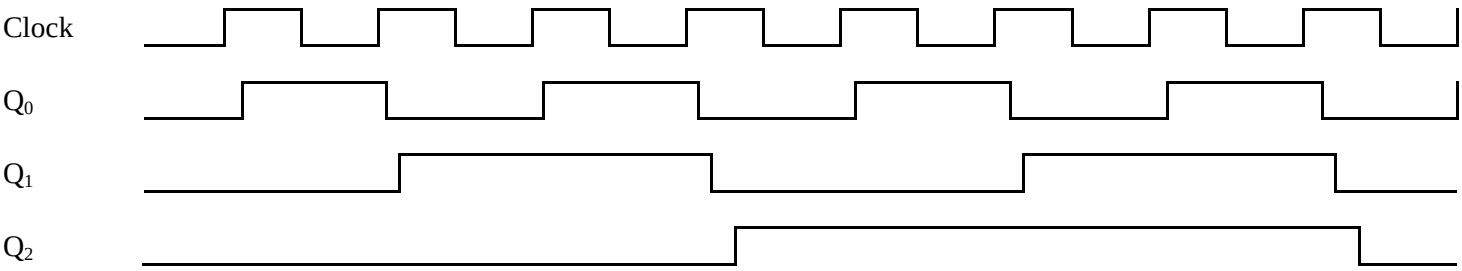
4.1.1. Example: 3-bit Counters



Three-bit Up-counter

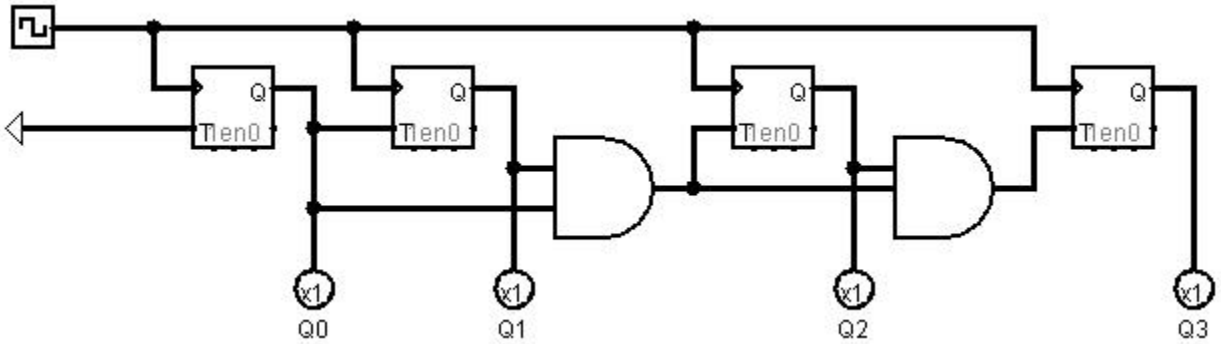


Three-bit Down-counter



Timing Diagram for Three-bit Up-counter

4.2. Synchronous Counter = a counter that has all of the FF change at the same time. Makes it faster than ripple counter.



Four-bit Synchronous Up-counter

Parallel Register

Picture, Figure A.30 in Stallings using SR FF's again. S's gated in, R is reset line, Z's anded with output strobe.

Shift Register (Draw on board, talk about how important this circuit is. Point out this is central to modem-type communication, taking a parallel word and sending it out serially).

Pictures - Figure A.31 in Stallings made out of D FF's, as is Figure 7.18 in Digital book. Figure 7.19 shows how to make a parallel load shift register, which can either shift individual stuff in or load a whole word.

Counter
These can be either Synchronous or Asychronous.
(Draw using JK ff's, with both J and K inputs of left-most tied high, and all the K inputs of the rest tied high as well. For each of the other FF's, the J input is the output of the previous FF. Draw the timing diagrams, show how the 8 cycles on top lead to 4 below, 2 below that, etc.)

Above is synchronous. Asynch is where you connect the output of one stage to the clock of the next stage. Using T FF's, for example, tie all T's high and connect the Zbar outputs to the next stage's clock.

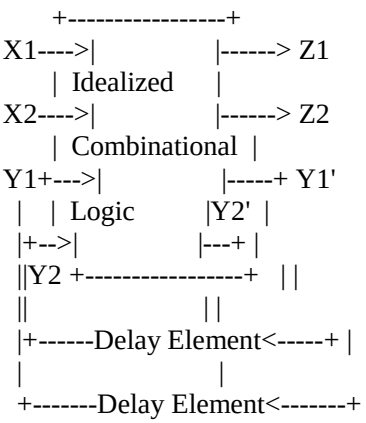
Pictures - Figure A.32 in Stallings is an asynchronous one, and section 7.9 in Digital book presents both kinds. A.32 = 7.20(a)

Register -

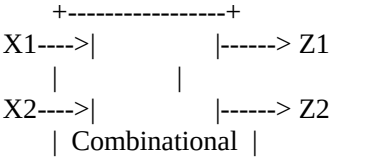
This is the end of the intro to sequential design. Next step is to use this stuff, start creating state diagrams and solving word problems.

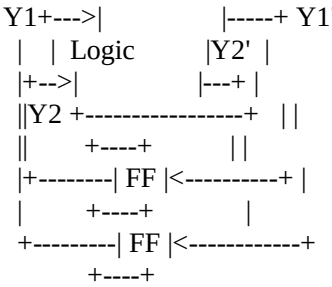
Now - how do we work with FF's? How do we design sequential circuits?

Model of Sequential circuit: Remember our model?



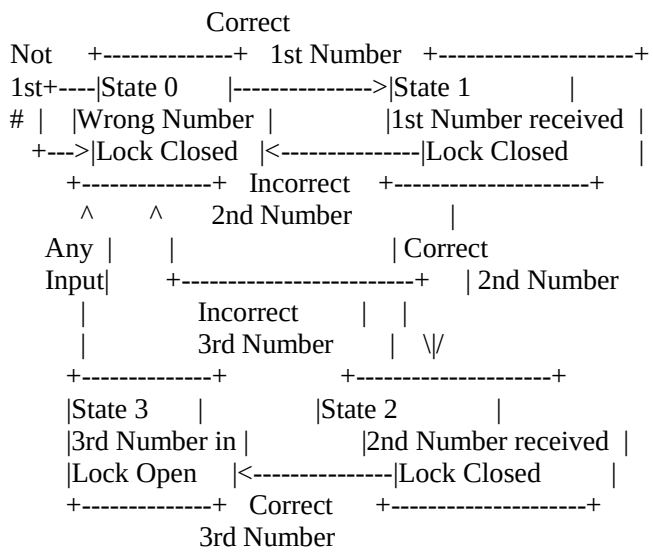
Well, the delay elements are replaced by FF's, and you have this:





The set of FF's is a register, for example, and the combinational logic might be the adder. (I have left off all the ...'s that should be in here - obviously there can be more than just 2 inputs and 2 outputs.)

OK - designing sequential circuits. We need a way to convert a specification (a word description) into the circuitry necessary to make it happen. We start by developing a state transition diagram - from this we can figure out what needs to be done. So, for example, think of an electronic (or non-electronic, for that matter) lock. We need to describe the machine:



This State Transition diagram describes the behavior of the machine. Remember, State implies memory. We get to a given state based on a pattern of inputs. In digital design, memory = FF's. So we get to a given state based on the state we are currently in and the inputs at this time.

Assume the lock has two buttons, only one button can be pushed at a time, and the combination is "LRL". To implement this, we need a couple more things - we need an input alphabet, an output alphabet, and a state alphabet. By convention, the state alphabet is usually in terms of Y's, the input alphabet is in terms of X's, and the outputs are in terms of Z's.

What states? Y0, Y1, Y2, etc.
What inputs? X0, X1, X2, etc.
What outputs? Z0, Z1, Z2, etc.

So our lock has 4 states, Y0, Y1, Y2 and Y3.

(Draw 4 circles here, corresponding to the 4 boxes above. Put Y0 in upper left, Y1 in upper right, Y2 in lower right, Y3 in lower left. Now, draw in the arcs between circles indicating transitions, and write above each arc the input

values. In the Moore model, the output value is put inside the circle. So an arc with 10 indicates left button high right button low, 01 is left low right high.)

The state transition diagram points out an interesting question - should the output be a function of the current state only, or a function of the current state and the current inputs? This is up to the designer. The two different possibilities have distinct names:

Mealy model - Outputs (Z's) are a function of the state we are in and the inputs at a given time.

Moore model - outputs are a function purely of the current state.

In the case of the lock, for instance - do you want the output to be high while you are pushing the 3rd number, or high until you push the next number (or for a fixed amount of time). Having it high only while pushing the last correct digit might make it tricky to get in the door if the lock is not right next to it. However, Mealy models almost always require less circuitry.

The ripple counter and shift register examples we did last time are Moore circuits - the output comes directly out of the FF's, and is independent of the current value of the input.

Now show how the lock could be written with only 3 circles if Mealy model is used. (When writing Mealy model, you write the inputs above the arc as above, but the output is written immediately below the inputs. So

10/1 would indicate the left button is high, right button low, and output is 1.
01/0 would indicate the left button is low, right button high, and output is 0).