3/16/12 Please turn in your check-in assignments.

The course space is now located at:

/egr/courses/unix/ECE/410

you need to run the "~~umask~~ umask 007" command every single time prior to launching Cadence.

- Lab 6 is now due (check-off) on Wed 3/21/12, and report is due Fri 3/23/12.
- Lab 7 is cancelled.

Begin work now on your Final Projects.

11.2 Binary Decoders

"Decodes" a control word, by activating one output line while the other lines are not affected.

Active-High Decoder – sets a 1 on the selected line while other lines remain at 0

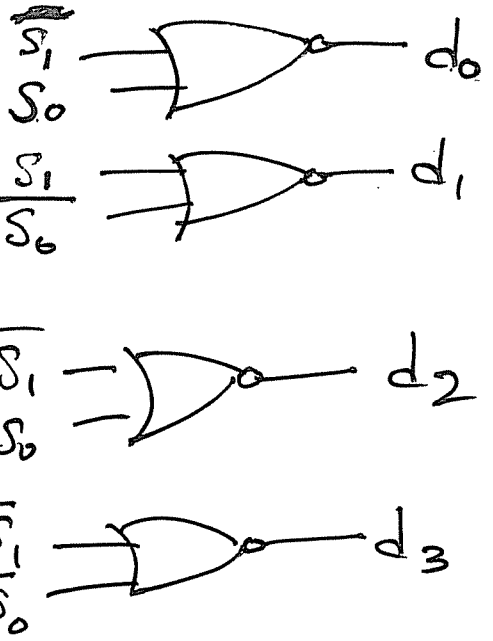
Active-Low Decoder – 0 on the selected line, 1 on the others

Active ~~Low~~ High Easily implemented with NOR-gates:

S_1	S_0	d_0	d_1	d_2	d_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

NOR:

a	b	f
0	0	1
0	1	0
1	0	0
1	1	0



11.5 Shift and Rotation Operations

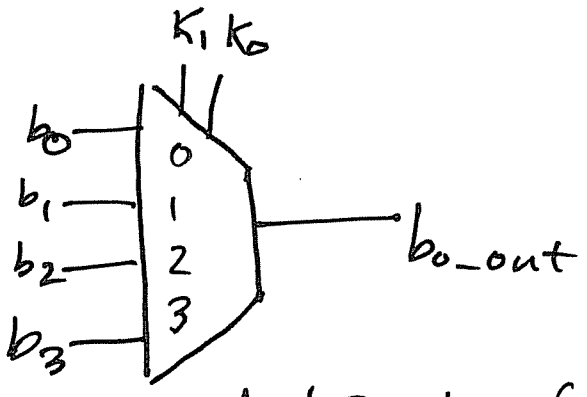
Constant shifts are trivial in hardware, requiring only wires.

Variable shifters take an N-bit input, a shift amount A, and control signals indicating shift type and direction.

In this class we will consider two of the three common types of variable shifts *(bonus knowledge: the third is called an arithmetic shift, and is more complicated in that it preserves the sign of the two's-complement input in a right shift, to avoid specific errors in rounding when using shifts as a divide by 2^n operation. The details are beyond the scope of this course)*

1. Rotate: Rotate numbers in a circle such that empty spaces are filled with bits shifted off the other end.
   Example: 1101 ROR 1 = 1110
   1101 ROL 1 = 1011
2. Logical Shift: Shift the number to the left or right, and fill the empty spaces with zeros.
   Example: 1101 LSR 1 = 0110
   1101 LSL 1 = 1010

How to connect inputs to correct outputs?
Consider N N-input multiplexers:

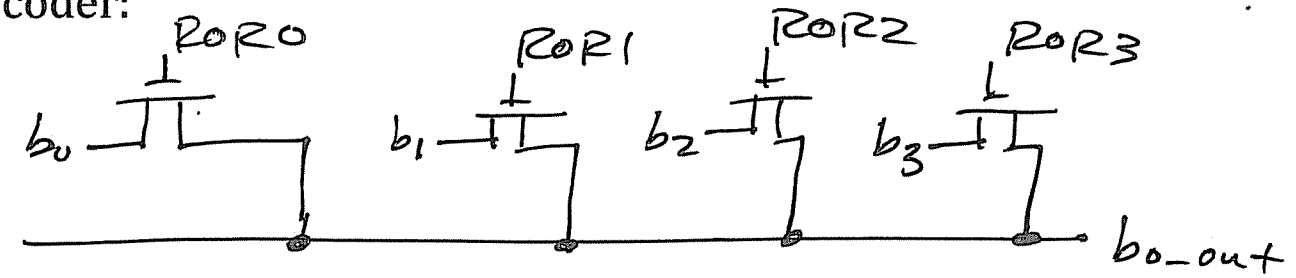


K_1 K_0	b_0_out
0 0	b_0 = ROR0
0 1	b_1 = ROR1
1 0	b_2 = ROR2
1 1	b_3 = ROR3

And similar for each bit

Multiplexors with more than 4-8 inputs become very slow.

Could also consider N switches connecting inputs to output lines, with the control bits ROR0:3 generated by a binary decoder:



This design could also use TGs, which, especially in modern processes will give much better performance at the cost of more complicated interconnect routing.

The n-FET example above is a bit-slice of what your book shows in Figures 11.25 and 11.26. These shifters can only shift right OR left, and the book suggests you use BOTH in your design, and multiplex the outputs with a left/right control bit.

For our Final Design Project, nFET designs similar to those in your book are perfectly acceptable implementations. Other barrel shifter types (TG, or MUXs like will be covered below) will be faster, but much more work. When choosing your Final Design Project Shifter design, consider carefully the extra work you may be adding on to the project if you choose a more complex layout.

Before moving forward, let's discuss something (not covered in your book.) Two types of shifters: array shifters and logarithmic shifters

1. *array shifter*: *N N*-input MUXs or switches select the correct input for each output from each of the possible inputs.
2. *logarithmic shifter:* instead of using *N N*-input MUXs, (for bit widths larger than 4-8 have very large parasitic capacitance) we can instead construct the shifter from $\log_v N$ levels of *v*-input MUXs. $v = 2$
   Example: 4-input shifter that produces the ROR logic from the previous example

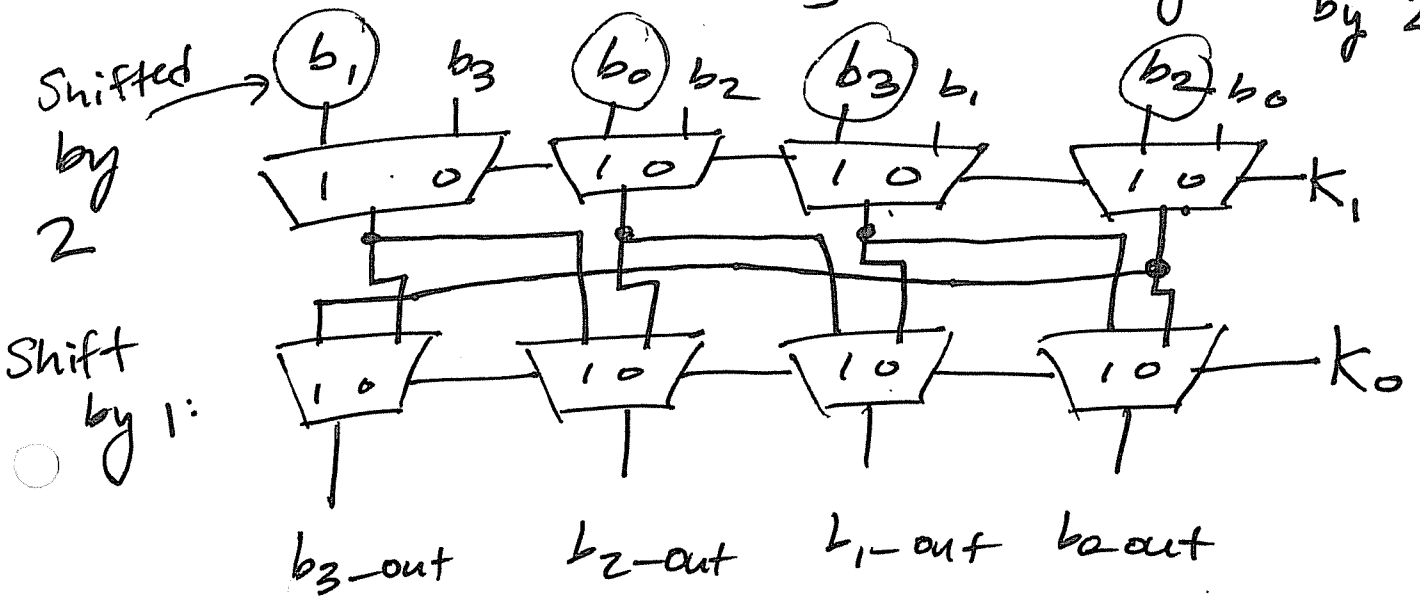
Shift	output	$k_1 k_0$
ROR0	$b_3 b_2 b_1 b_0$	0 0
ROR1	$b_0 b_3 b_2 b_1$	0 1
ROR2	$b_1 b_0 b_3 b_2$	1 0
ROR3	$b_2 b_1 b_0 b_3$	1 1

← Rotate Right

k_1 shifts by 2

k_0 shifts by 1

$\Rightarrow k_n$ shifts by 2^n



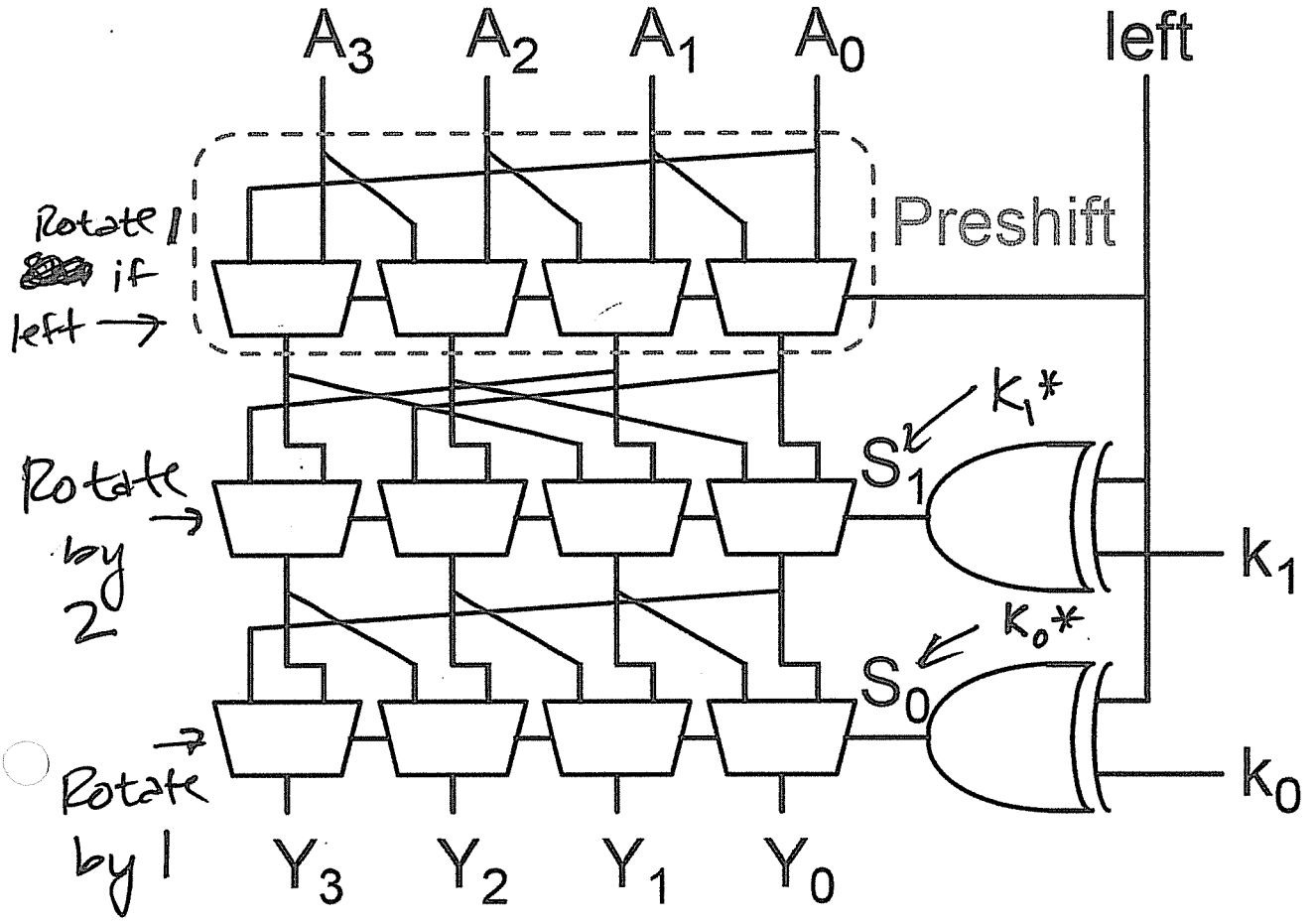
What if I want to rotate left?

operation	output	K_L: $K_1 K_0$	$\overline{K_L}$	K_R of eg. output:
ROL0	$b_3 b_2 b_1 b_0$	00	11	$00 = \overline{K_L} + 1$
ROL1	$b_2 b_1 b_0 b_3$	01	10	$11 = \overline{K_L} + 1$
ROL2	$b_1 b_0 b_3 b_2$	10	01	$10 = \overline{K_L} + 1$
ROL3	$b_0 b_3 b_2 b_1$	11	00	$01 = \overline{K_L} + 1$

So, to shift left, we need a "preshift" stage to shift right by 1 controlled by a "left" bit, and we can use "left" bit to modify K_1 and K_0 to get $\overline{K_L}$ when needed:

	left	K_0	K_0^* ← Actually used K_0	
right rotate	0	0	0	XOR
	0	1	1	
left rotate	1	0	1	
	1	1	0	

Logarithmic Barrel Shifter that rotates left or right:



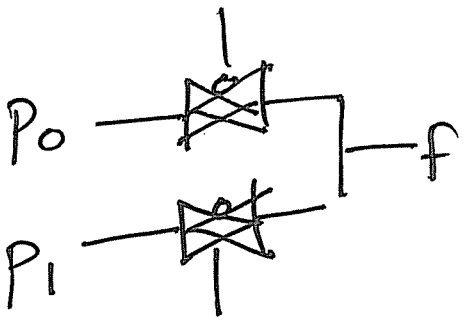
To shift you need circuitry to "mask" off the outputs where 0's are inserted

(For example: ROR 1: $b_0 b_3 b_2 b_1$)

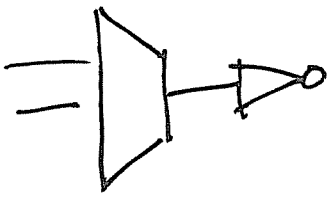
↓

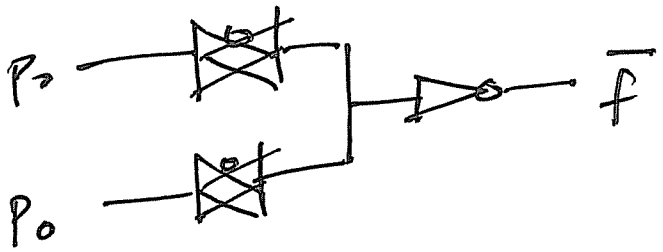
$0\ b_3\ b_2\ b_1$

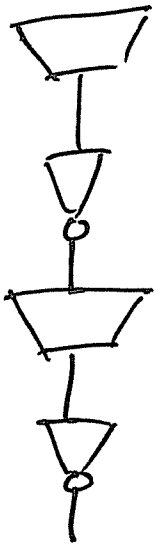
TG muxes actually work well here, with inverters:

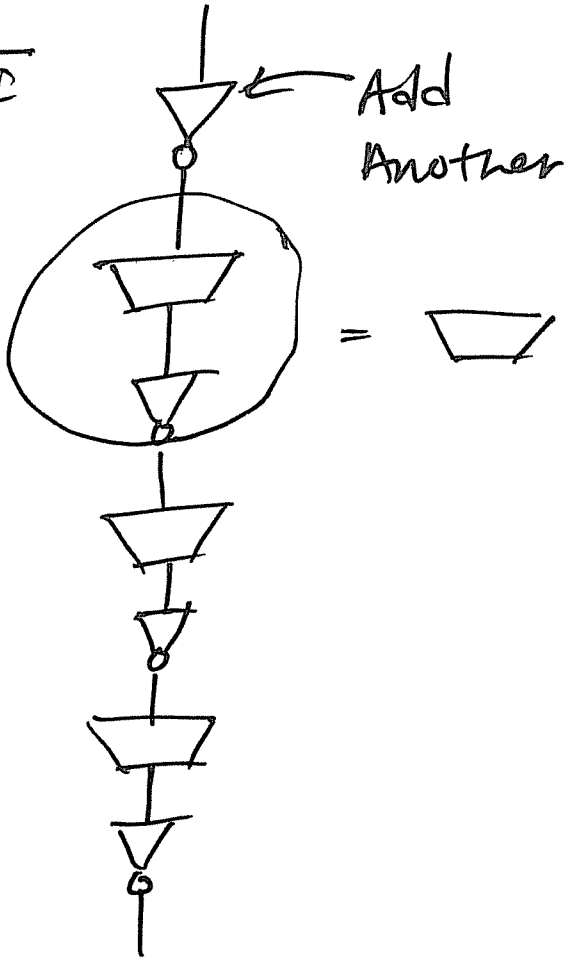


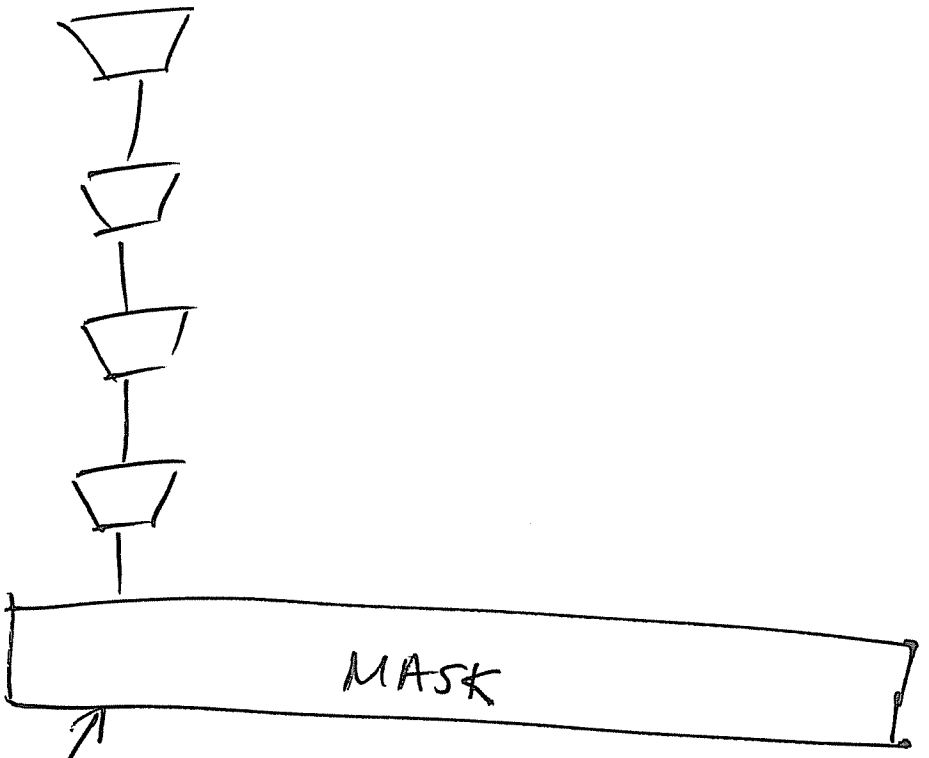
TG MUX











We will talk about this on Monday

Reading Assignments: Finish Ch 11

Check In: "Define transparent as applies to latches"