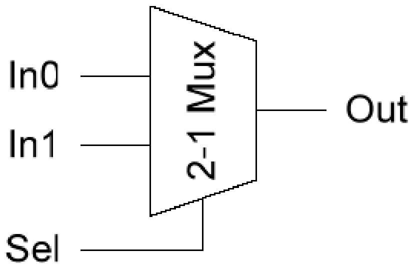
University of California at Berkeley
College of Engineering
Department of Electrical Engineering and Computer Science

EECS 150 D. E. Culler
Fall 2007

Problem Set #2: Programmable Logic and HDLs
Assigned 9/6/2007, Due 9/14/2007 at 2 PM

Problem 1. You are given a single 2-input MUX. How many of the 2-input boolean functions can you obtain by wiring the three inputs of this device to some combination of A, B, Logical 0, and Logical 1. You do not have the complements of A and B. (Show how.) How many more can you get with 2 MUXes? 3 Muxes?

In0	In1	Sel	Out
0	X	0	0
1	X	0	1
A	X	0	A
B	X	0	B
1	0	A	~A
1	0	B	~B
0	A	B	A & B
1	A	B	A | ~B
A	1	B	A | B
A	0	B	A & ~B
1	B	A	~A | B
B	0	A	~A & B

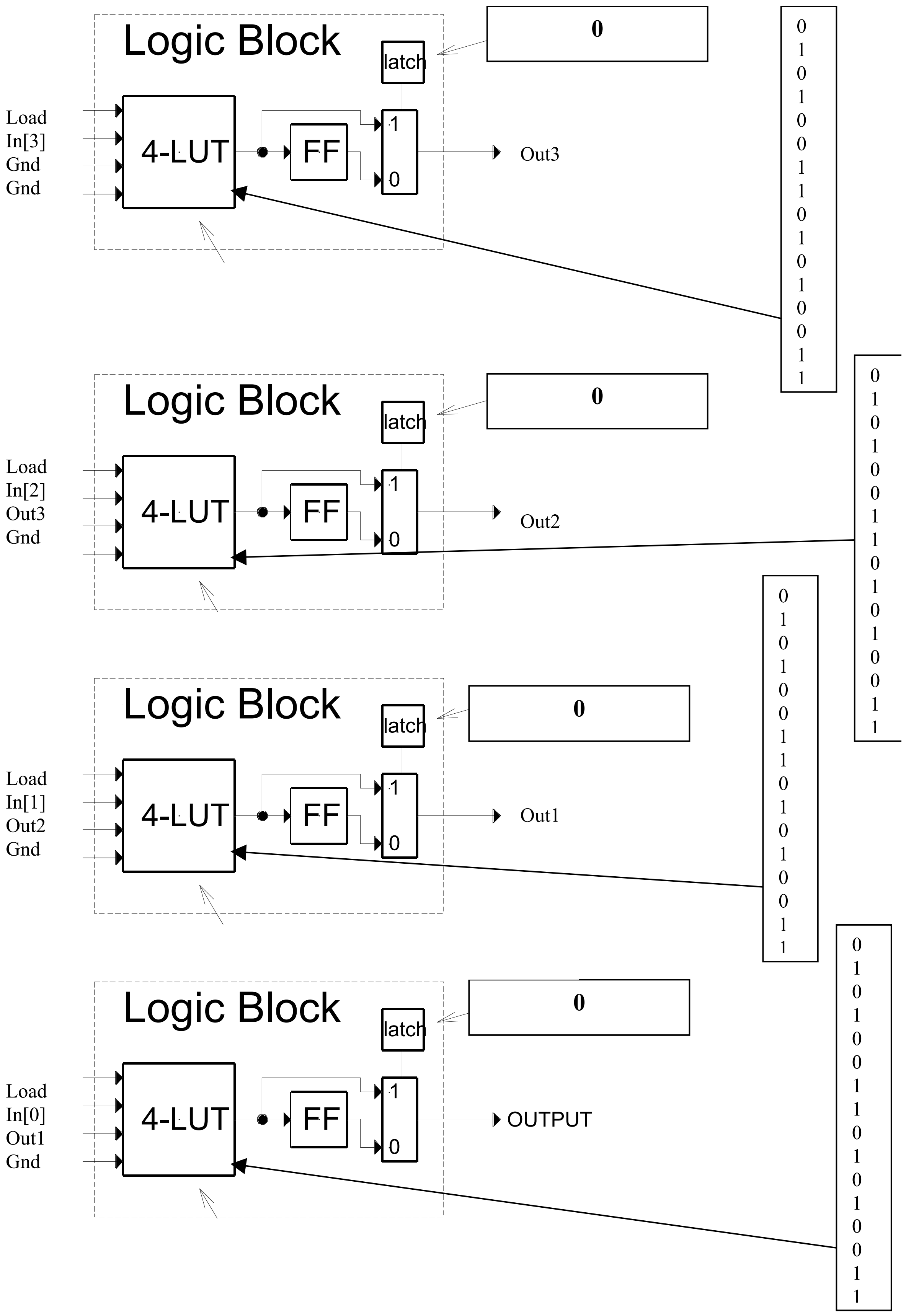


With a second MUX you can add NOR, NAND, XOR, XNOR, so you get them all. With a third, you can build a 2-LUT.

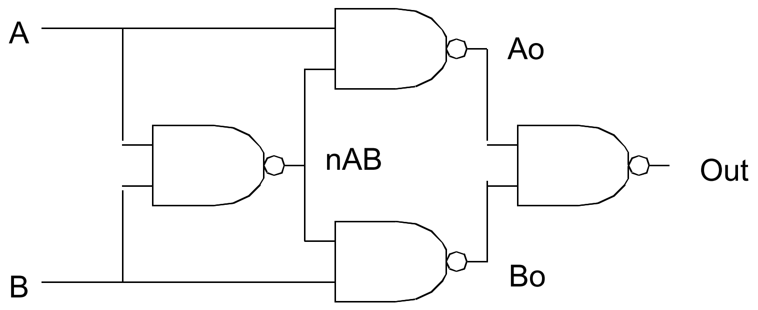
Problem 2. Based on the information in lecture and the data sheet on the Xilinx Vertex family, how does the number of User I/O pins grow with the number of LUTs? How do the BlockRAM Bits scale with LUTs?

BlockRam Bits scale about linearly with LUTs
User I/O pins grows as the log of the number of LUTs

Problem 3: You are synthesizing logic (by hand) for and idealized FPGA consisting of many CLBs as shown in the following figure and interconnect that ties them together. Show how to use a collection of these blocks to implement the 4-bit Parallel-to-Serial converter discussed in lecture (slide 10 of Lecture 3). For each block, specify the contents of the 4-LUT and the latch. Show how the blocks are wired together.


Logic Block
latch
0
Load
In[3]
Gnd
Gnd
4-LUT
FF
1
0
Out3
0 1 0 1 0 0 1 1 0 1 0 1 0 0 1 1
Logic Block
latch
0
Load
In[2]
Out3
Gnd
4-LUT
FF
1
0
Out2
0 1 0 1 0 0 1 1 0 1 0 1 0 0 1 1
Logic Block
latch
0
Load
In[1]
Out2
Gnd
4-LUT
FF
1
0
Out1
0 1 0 1 0 0 1 1 0 1 0 1 0 0 1 1
Logic Block
latch
0
Load
In[0]
Out1
Gnd
4-LUT
FF
1
0
OUTPUT
0 1 0 1 0 0 1 1 0 1 0 1 0 0 1 1

Problem 4: In lab 0 you implemented an XOR using 4 nand gates. Write the structural verilog module corresponding to the schematic below.



Write the same behavior as a behavioral module using continuous assignment.

```
module xor ( out, a, b );
 input     a, b;
 output    out;
 wire      nAB, Ao, Bo;

 nand_gate  (nab, a, b);
 nand_gate  (Ao, a, nAB);
 nand_gate  (Bo, b, nAB);
 nand_gate  (out, Ao, Bo);
endmodule
```

Write the same behavior as a behavioral module using Blocking assignment.

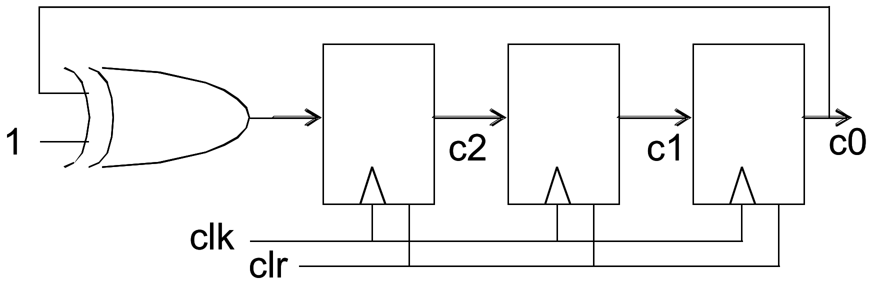
```
module xor ( out, a, b );
 input     a, b;
 output    out;

 assign out = a ^ b;
endmodule
```

Write the same behavior as a behavioral module using non-Blocking assignment.

```
module xor ( out, a, b );
 input     a, b;
 output    out;
 reg out
 always (a OR b)
     out <= a ^ b;
endmodule
```

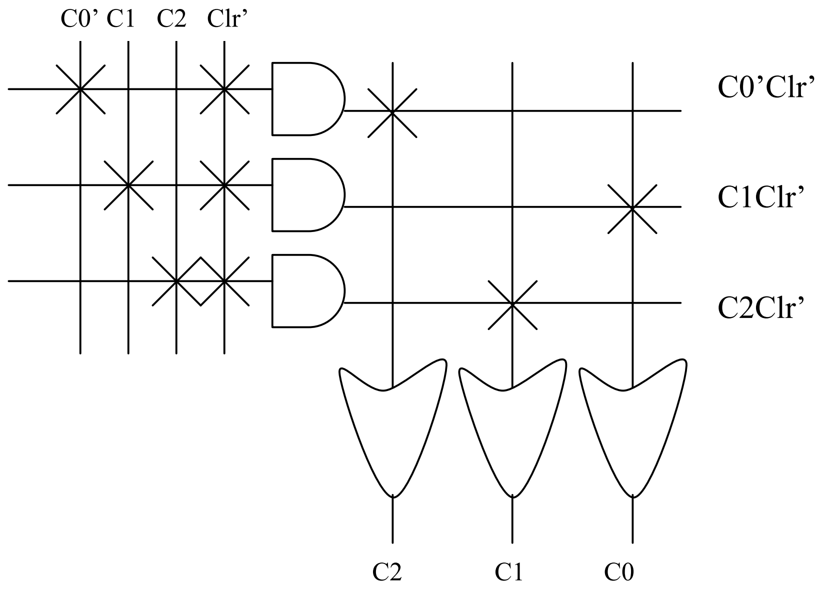
Problem 5. Some aspects of digital design are deeply rooted in the mathematical elegance of Boolean algebra. Others are motivated by clean geometrical structure that yields simple physical layouts. One example of the latter in the Johnson counter shown in schematic below. With n bits it is possible to represent 2^n values. This counter only uses 2n of those possible values, so it has a few more flip-flops, but almost no logic and no carry propagation, so it is fast and easy to route. The clr input resets all the flip-flops to zero.



Generate the truth table how this counter "increments". This has four input columns (clr, c2, c1, and c0) and three output columns (the new c2, c1, and c0).

Inputs				NextState		
clr	C2	C1	C0	C2	C1	C0
1	X	X	X	0	0	0
0	0	0	0	1	0	0
0	1	0	0	1	1	0
0	1	1	0	1	1	1
0	1	1	1	0	1	1
0	0	1	1	0	0	1
0	0	0	1	0	0	0
0	0	1	0	1	0	1
0	1	0	1	0	1	0

Show how this logical would be implemented on a 4-input, 3-output PLA.

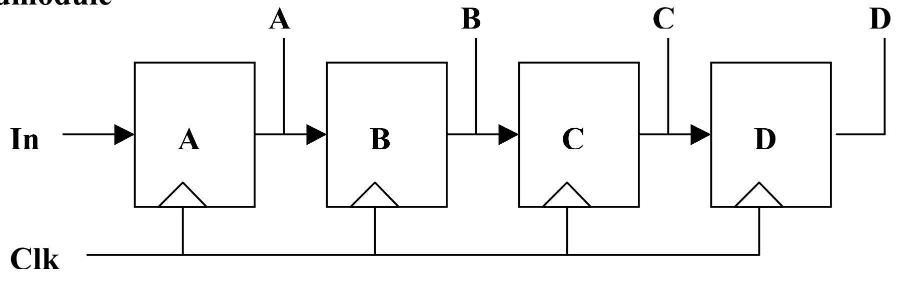


Write a behavioral Verilog module for this counter using a bit vector for the counter value.

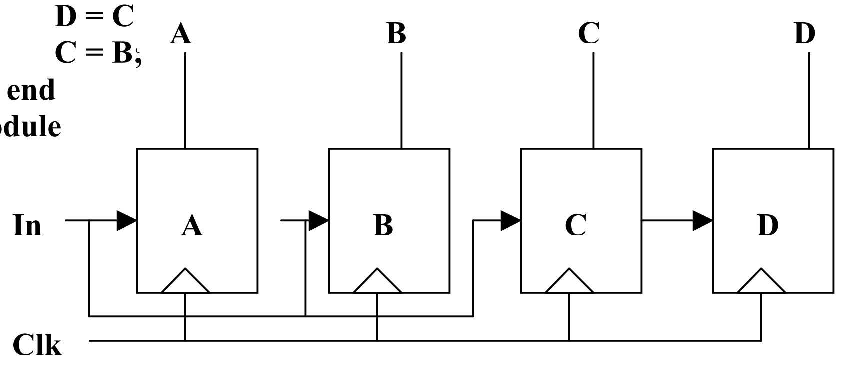
```
module JohnsonCounter( clk, clr);
  input     clk, clr;
  wire [2:0]   nextstate, current;
  always @(posedge clk) begin
    if(clr) current <= 3’b000;
   else current <= nextstate;
end
 always @(posedge clk) begin
   nextstate[0] <= current[1];
   nextstate[1] <= current[2];
   nextstate[2] <= not current[0];
 end
endmodule
```

Problem 6: Draw the a schematic that would implement the behavior described be each of the follow verilog modules.

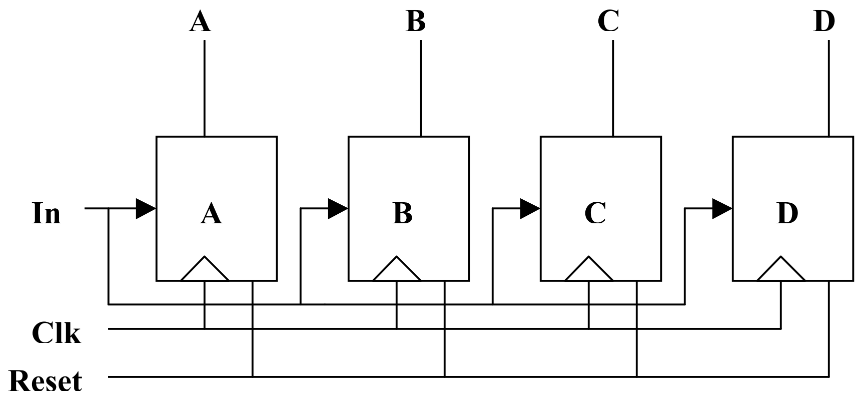
```
module sifter1 (in, A,B,C, D, clk);
        input in, clk;
        output A,B,C;
        reg A, B, C;
        always @ (posedge clk) begin
            A <= in;
            B <= A;
            D <= C
            C <= B;
        end
endmodule
```



```
module sifter2 (in, A,B,C, D, clk);
        input in, clk;
        output A,B,C;
        reg A, B, C;
        always @ (posedge clk) begin
            A = in;
            B = A;
            D = C
            C = B;
        end
endmodule
```



```
module sifter2 (in, A,B,C, D, clk);
        input in, clk;
        output A,B,C;
        reg A, B, C;
        always @ (posedge clk OR reset) begin
            A = in;
            B = A;
            C = B;
            D = C;
            if (reset) begin A = 0; B = 0; C=0; D=0; end
        end
endmodule
```


A
B
C
D
In
A
B
C
D
Clk
Reset