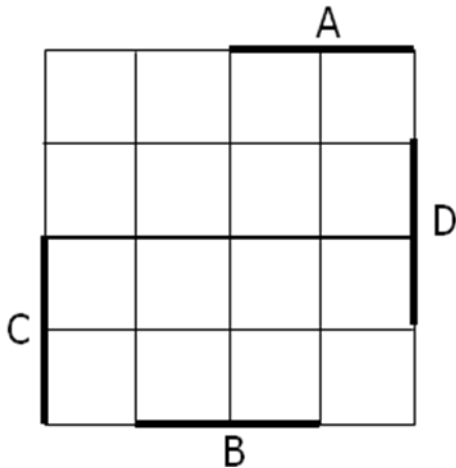


Sample CSE370 Final Exam Questions

Logic Minimization

(a – 10 pts) Minimize the function $F = \sum m(0, 6, 7, 8, 9, 11, 15) + d(1,13)$ using the K-map below. List all prime implicants, essential prime implicants, and the minimal solution in SOP form in the spaces provided.



Prime implicants:

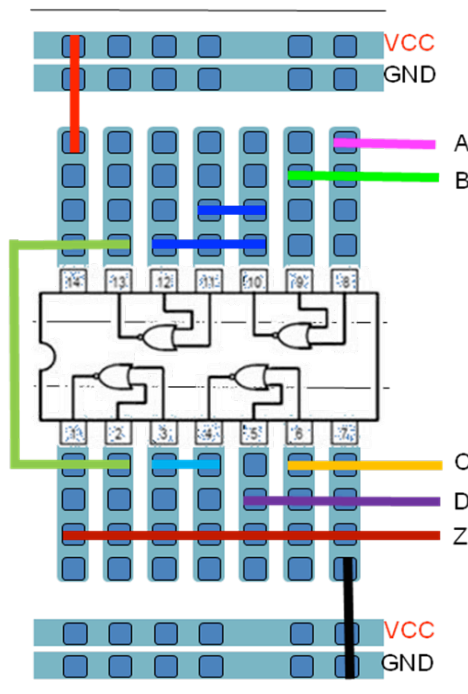
Essential prime implicants:

Minimal cover:

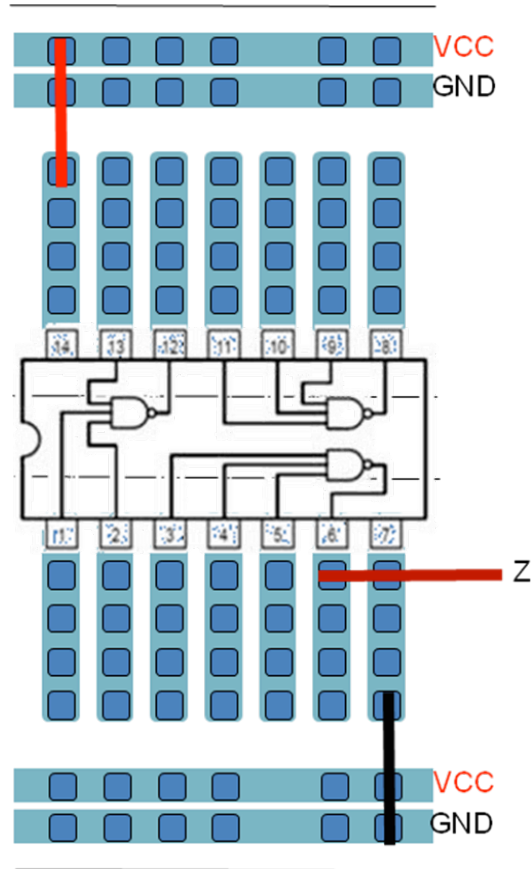
F =

Gates to Functions back into Gates

a. Given the following breadboard wiring diagram, derive a SOP equation for the single output, Z, through the use of Boolean algebra. The package in the center is a set of 4 NOR gates (the '02 part in your lab kit).



b. Remap the function you derived for Z in part (a) to the package of 3 3-input NAND gates shown below (the '10 part in your kit). You can assume that A', B', C', and D' are available as inputs if you should need them. Have the inputs and outputs on the right as in the diagram of part (a).



c. Assuming that '02 2-input NOR gates have a delay of 7ns and '10 3-input NAND gates have a delay of 9ns, what is the delay to Z of a change in an input (or its complement) for the circuit in part (a) and the circuit you wired up in part (b).

Delay of circuit of part (a) = _____

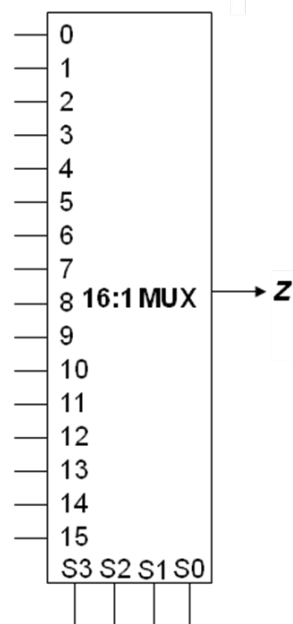
Delay of circuit of part (b) = _____

Multiplexer Logic

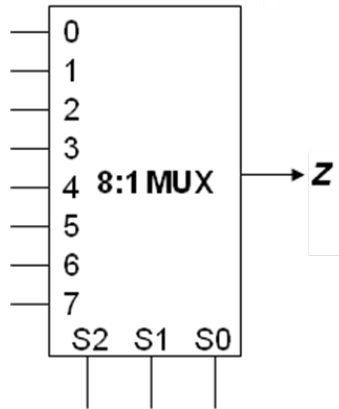
a) Given the function $Z = B'C + A'BD + AB'$, fill in the truth-table below.

A	B	C	D	Z
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

b) Given the truth table of part (a), implement Z using a single 16:1 multiplexer shown below. Make sure to clearly label all inputs and outputs.



c) Given the truth table of part (a), implement Z using a single 8:1 multiplexer shown below. Try to use as few other gates as possible. Make sure to clearly label all inputs and outputs.

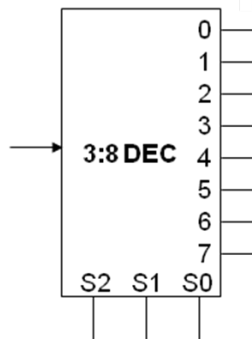


Decoder Logic

Given the function $W = (A \text{ xor } C) + B$, fill in the truth-table below.

A	B	C	W
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Given the truth table of part (a), implement W using a single 3:8 decoder shown below and no more than a single 2-input NOR gate. Make sure to clearly label all inputs and outputs.

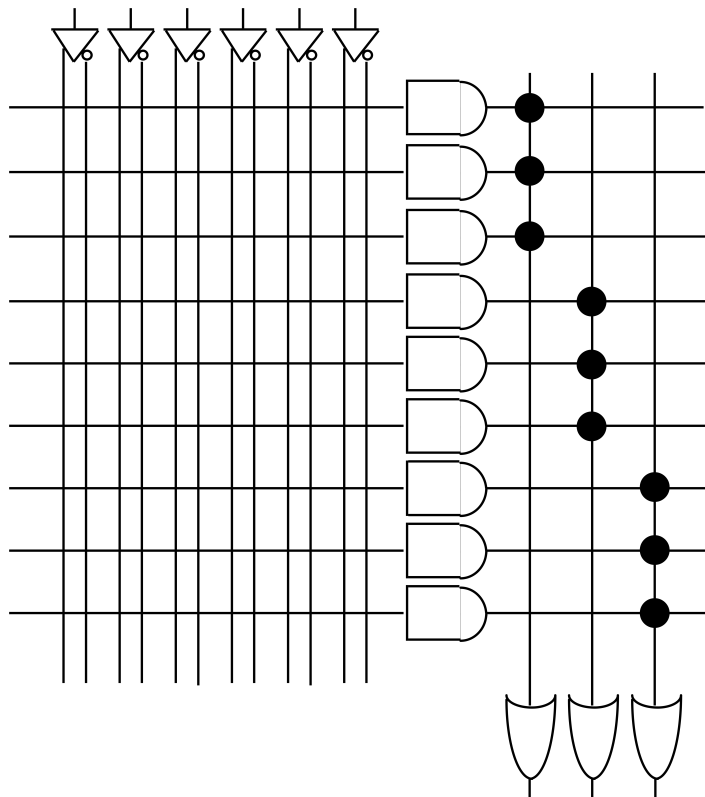


Programmable Logic

Given the following K-maps for two logic functions, X and Y, respectively, show how you would program the PAL on the next page (6 inputs, 3 outputs, 9 product terms overall) to implement them. Clearly label all you inputs, outputs, and product terms on the PAL.

		A			
		0	0	1	0
		0	0	0	1
	C	1	0	0	0
		0	1	0	0
				B	

		A			
		0	0	1	1
		0	0	0	1
	C	1	0	0	0
		1	1	0	0
				B	



Generate a circuit that implements the Verilog description below.

```
module problem_3_b (clk, A, B, C)
  output A, B, C;
  input  clk;

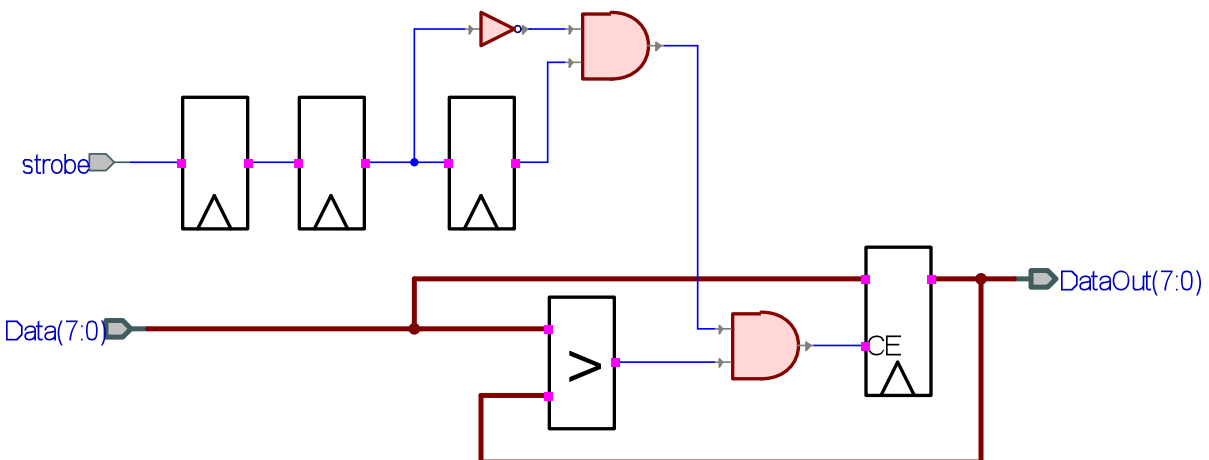
  reg    A, B, C;

  always @(posedge clk) B <= A;

  always @(posedge clk) C <= B;

  always @(posedge clk) A <= A ^ (B ^ C);
endmodule
```

Write a Verilog module that implements the following circuit. Exact syntax is not important.



In the carry-lookahead adder, the propagate function is sometimes defined as:

$$p_i = a_i + b_i$$

and sometimes as:

$$p_i = a_i \oplus b_i$$

Explain why both functions work in the carry-lookahead adder, and why one might be preferred over the other. I'm looking for a succinct, precise answer, not a lot of words.

Here is a Verilog description of a circuit. Design a circuit using just logic gates and registers that implements this specification. Show all your work.

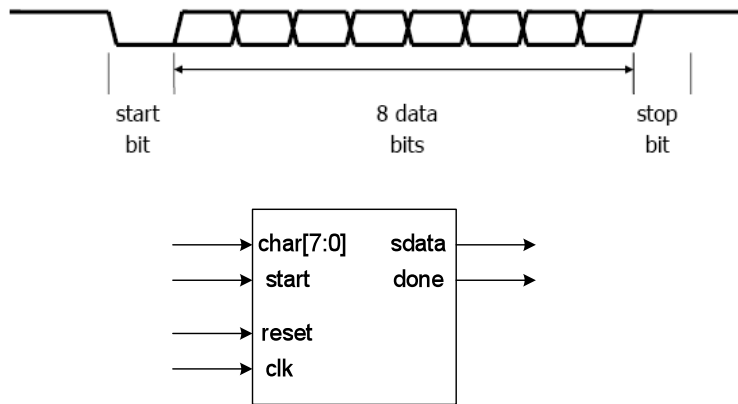
```
module what (clk, reset, A, B, out);
    input clk, reset;
    input A, B;
    output out;
    reg    out;

    reg    state[1:0], nxtState[1:0];

    always @(posedge clk) begin
        if (reset) state <= 0;
        else state <= nxtState;
    end

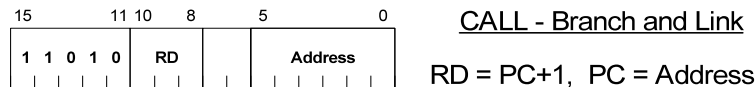
    always @(*) begin
        out = 0;
        nxtState = state;
        case (state)
            0: begin
                if (A) nxtState = 1;
                else if (B) out = 1;
            end
            1: begin
                if (B) nxtState = 2;
                else if (!A) out = 1;
            end
            2: begin
                if (!A) nxtState = 3;
                else if (!B) out = 1;
            end
            3: begin
                if (!B) nxtState = 0;
                else if (A) out = 1;
            end
        endcase // case(state)
    end
endmodule
```

In your project, you converted a serial input stream into a character that was displayed on the LCD screen. In this problem, you will design a circuit that does the opposite: take an 8-bit character as input and send it in bit-serial form to the output. The module interface is shown below. The start input is asserted for one clock cycle when a new character is available on the input. The circuit then sends this character one bit per clock, low-order bit first, to the sdata output. First a start bit (0) is sent, followed by the 8 data bits, and finally a stop bit (1), as shown in the timing diagram. The done output signal should be asserted for one clock cycle when the character has been sent and the circuit is ready to accept a new character. The components you can use are registers, counters, adders, comparators, multiplexors, decoders, and logic gates.



- Draw a block diagram of your design in the datapath/control style. Hint: Don't try to do too much in the control part of the circuit (FSM).
- Your design should have a state machine controller: Draw the state diagram that describes precisely how it operates.
- Design the entire circuit in detail using the components listed above. You may write equations for logic functions instead of drawing logic gates.

On the next page is the top-level schematic for an implementation of the x370 processor along with the Verilog for the control module. Describe clearly how to implement a new instruction: CALL (also known as Branch and Link), which allows the program to branch to a procedure by remembering the address of the instruction following the CALL instruction in a register. The procedure can then return back to the main program by branching to the address saved in the register. (This would be another instruction, which you don't have to worry about.)



For example, if the instruction '11010_010_00_001010' at address 000101 is executed, then the next instruction would be executed from address 001010 and the value 000110 would be written to register 2.