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

EECS150 — J. Wawrzynek
Spring 2010 — 3/31/09

Midterm Exam - **Solutions**

This is a *closed-book, closed-note* exam. No calculators or any other electronic devices, please.

Read all the questions **before** you begin. Each question is marked with its number of points (one point per expected minute of time). Although you might not need it, you have until 9pm.

You can tear off the spare pages at the end of the booklet and/or use the backs of the pages to work out your answers. Neatly copy your answer to the places allocated for them.

Neatness counts. We will deduct points if we need to work hard to understand your answer. **Simplicity also counts.** In the design problems, correct simpler designs with fewer components will be awarded a higher score than more complex designs with more components.

Put your name and SID on each page.

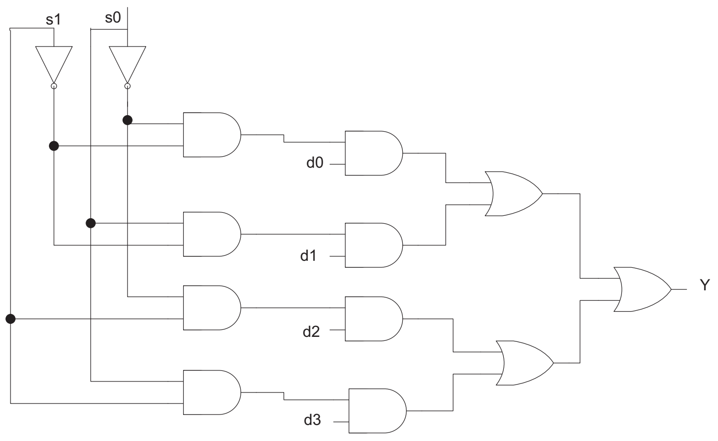
problem	maximum	score
1	8pts	
2	6pts	
3	10pts	
4	15pts	
5	12pts	
6	10pts	
7	24pts	
8	16pts	
9	19pts	
Total	120pts	

1. Multiplexor Implementation [8pts].

   Consider the design of a 4-to-1 multiplexor circuit with four data inputs, `d0`, `d1`, `d2`, and `d3`, two control inputs, `s0` and `s1`, and a single output, `y`.

   Using only simple logic gates (ANDs, ORs, NANDs, NORs, inverters), but no transmission gates, sketch the circuit diagram for a multiplexor circuit *optimized for minimum delay* from the data inputs, `d0`–`d3`, to the output, `y`. You may use gates with any number of inputs, but remember that the delay through a logic gate grows with the square of the number of inputs.

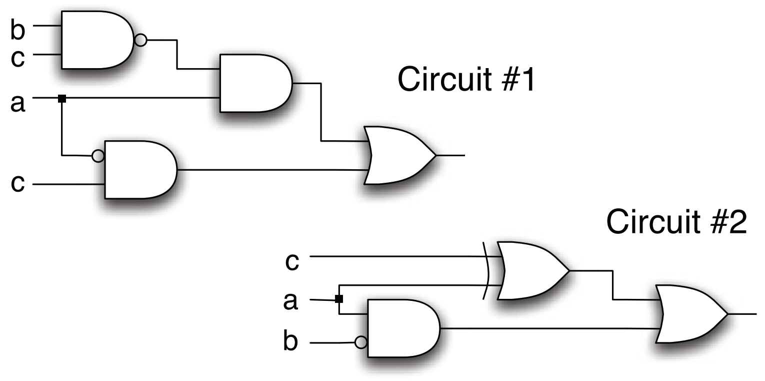
   Let us suppose for comparison that the delay through an x-input gate is x^2, based on the problem stating that the delay grows with the square of the number of inputs. A common functionally-correct answer was an implementation that uses two rounds of selection to pass the data: one with s0 followed by one with s1. This results in the d signals passing through at least 4 2-input gates. delay $= 4 * 2^2 = 16$. Answers that built a 4:1 Mux from three 2:1 muxes were equivalent to this case.
   Another functionally-correct implementation was to decode selects and pass in the data at the same gate, using a three-input AND gate. delay $= 1 * 3^2 + 1 * 4^2 = 25$ for 3-input AND, 4-input OR; or delay $= 1 * 3^2 + 2 * 2^2 = 17$ for 3-input AND, 2-input OR, 2-input OR. The most optimal design for minimum delay from $d_0 - d_3$ to y is shown below.



   Notice that the select signal is decoded entirely before the data inputs are introduced. This reduces the number of gate delays for the data inputs. delay $= 3 * 2^2 = 12$.
   This is a very important concept. If one of the d signals was part of the critical path then you would want them to go through less logic in the mux to reduce delay. A real example of this would be if your processor control was fast and created mux select signals early, but your datapath was slow and made the data available later: it would then be good if there was less delay through the mux for the data signals.

2. Combinational Logic Circuits [6pts].

   Using whatever means possible, prove or disprove that the two combinational logic circuit shown below have equivalent function. Explain your approach and show your work.



One solution was to simply show equivalence through the use of two truth tables. By enumerating every possibility, we exhaustively prove that the circuits are equivalent.

a	b	c	$\overline{bc}$	$\bar{a}c$	$a\overline{bc}$	$\bar{a}c + a\overline{bc}$
0	0	0	1	0	0	0
0	0	1	1	1	0	1
0	1	0	1	0	0	0
0	1	1	0	1	0	1
1	0	0	1	0	1	1
1	0	1	1	0	1	1
1	1	0	1	0	1	1
1	1	1	0	0	0	0

a	b	c	$a \oplus c$	$a\bar{b}$	$(a \oplus c) + a\bar{b}$
0	0	0	0	0	0
0	0	1	1	0	1
0	1	0	0	0	0
0	1	1	1	0	1
1	0	0	1	1	1
1	0	1	0	1	1
1	1	0	1	0	1
1	1	1	0	0	0

Another solution was to prove equivalence algebraically. Circuit #1 can be expressed as:

$$\begin{aligned} y_1 &= a\overline{bc} + \bar{a}c \\ &= a(\bar{b} + \bar{c}) + \bar{a}c \\ &= a\bar{b} + a\bar{c} + \bar{a}c \end{aligned}$$

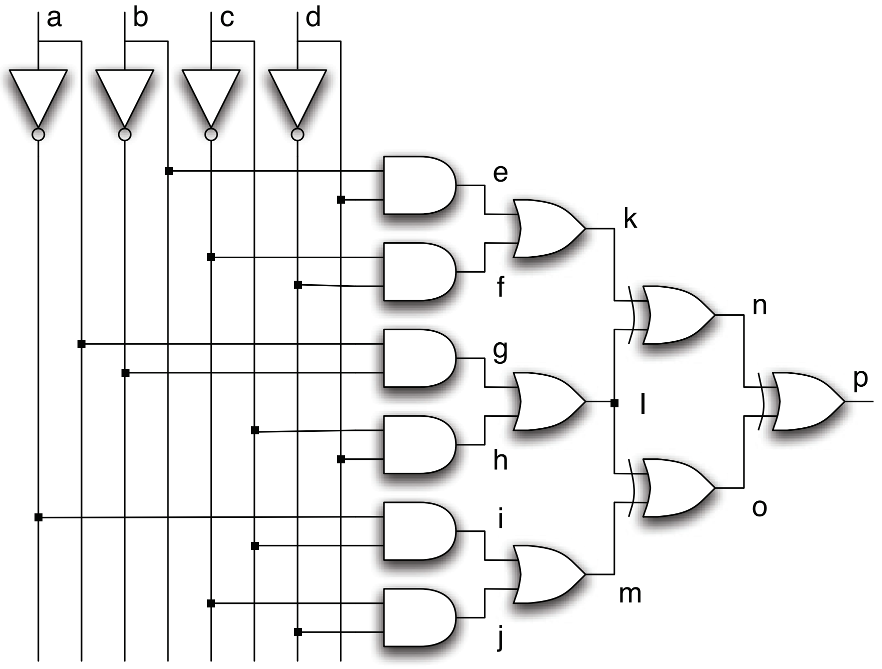
And, Circuit #2 can be expressed as:

$$\begin{aligned} y_2 &= (a \oplus c) + a\bar{b} \\ &= \bar{a}c + a\bar{c} + a\bar{b} \\ &= a\bar{b} + a\bar{c} + \bar{a}c \end{aligned}$$

Therefore, the circuits are equivalent.

3. FPGA Mapping [10pts].

Using only 3-input lookup tables (LUTs), partition the circuit shown below into as few LUTs as possible. *Do not attempt to simplify the gate-level circuit before mapping it to LUTs.* Indicate your answer by filling in the table. Fill in one row for each LUT, assigning node names from the circuit to LUT inputs and outputs. Mark unused LUT inputs with "X" (for unused).



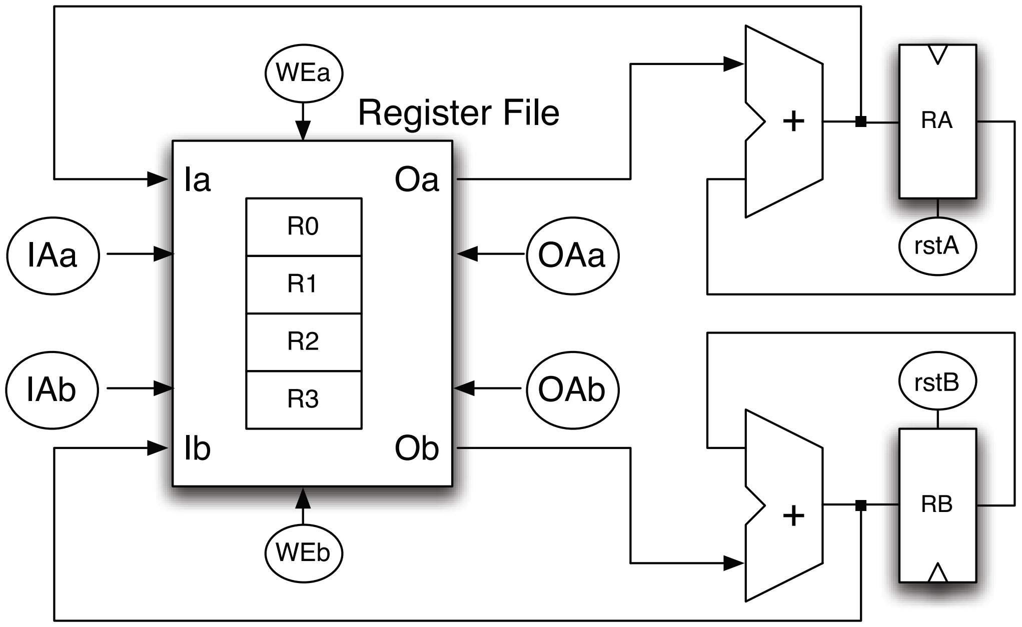
LUT #	input 1	input 2	input 3	output
1	b	c	d	k
2	a	b	X	g
3	g	c	d	l
4	a	c	d	m
5	k	l	m	p

Another acceptable solution was to show explain how any 4 input function can be represented with a 4LUT. Then, show how you would create a 4LUT form three 3LUTs. To get full credit for this solution, it was necessary to show exactly how you would connect the wires to the 3LUTs.

4. Register Transfers [15pts].

Consider the datapath shown below. A controller (not shown), is used to control operation of the datapath. It sets the value of all the control signals (circled in the datapath diagram).

The block labeled "Register File" stores four data registers, R0–R3, has two asychronous read ports, and two synchronous write ports. The four port addresses are IAa, IAb, OAa, and OAb. Writing to the register file is controlled by the WEa and WEb signals. The data registers at the output of the adders, RA and RB, have synchronous reset inputs, rstA and rstB, respectively.



(a) (11pts) Assume that the register file is initialized with registers R0–R3 holding the values a, b, c, and d, respectively. Registers RA and RB begin uninitialized.

Your task is to generate the control signal sequence which will result in a in R0, $a+b$ in R1, $a+b+c$ in R2, and $a+b+c+d$ in R3, and do so in the minimum number of clock cycles. Indicate your answer by filling in the table with the control signal values that the controller would generate on each clock cycle (for use on the next positive clock edge).

The following show a valid timing sequence:

cycle #	IAa	IAb	OAa	OAb	WEa	WEb	rstA	rstB
1	x	x	x	x	0	0	1	1
2	x	x	R0	R2	0	0	0	0
3	R1	x	R1	R3	1	0	0	0
4	R2	R3	R2	R1	1	1	0	0

And the actions given in each cycle are given below:

cycle #	Comment
1	$RA < -0, RB < -0$
2	$RA < -A, RB < -C$
3	$R1 < -A + B, RA < -A + B, RB < -C + D$
4	$R2 < -A + B + C, R3 < -A + B + C + D, RA < -A + B + C, RB < -A + B + C + D$

4 cycles minimum. Other similar optimal solutions are of course possible. Don't cares (x) are possible for OAa, OAb when the output registers are being reset and WEa, WEb are low because there will be no effect on state. IAa and IAb can be x when the corresponding WEa or WEb is low for the same reason. Full score did not require don't cares, just that circuit function was correct, but answers where incorrect don't cares messed up correct calculation were penalized. For example, if rstA is low but OAa is x, then RA will get garbage (an unknown value).

The output registers start uninitialized, so we need to reset them to a known value before using them to accumulate useful numbers. This reset takes a cycle because at the first positive edge when the reset is high the register will be reset. (If you were to assume the registers had asynchronous reset and still be able to use the first cycle for useful work, then you would have to state that the rstA,rstB would have to be pulsed sometime during the first cycle but go low before the clock edge; otherwise RA and RB may not actually store the first data values).
This problem is an example of a parallel prefix calculation (sum in this case) with two parallel functional units (adders in this case). The key to optimizing for minimum number of cycles is to recognize that parts of sums can be calculated in parallel and intermediate results can be shared.
One mistake students made was to assert WE a cycle late. Remember that if cycle i starts with clock edge i and ends with edge i+1, then during cycle i, WE must be high, IA have write address, and dataIn have write data, to perform a synchronous write on edge i+1. Also note that the new data will be used during cycle i+1.

(b) (4pts) Now assume that the Register File has a read access delay of 2ns, a write setup time of 1ns, and a write delay of 1ns, i.e., the written data appears in the proper register 1ns after the clock edge. There is no register file bypassing.

The adders have a combinational logic delay of 4ns, and the output registers have a setup time, 1ns and clock-to-q delay of 1ns, and a hold time of 1ns. Ignore wire delay and clock skew.

What is maximum clock frequency for this circuit?

This question asks for the T_{min} constraint, in terms of frequency.

$$T_{min} >= t_{c-q} + t_{logic,max} + t_{setup}$$

The critical path is the longest delay path from a state element eynchronous output to a state element synchronous input. Looking at just one branch of the circuit (both are equivalent here), we find four paths: rfile→adder→register, rfile→adder→rfile, register→adder→register, and register→adder→rfile. Since the logic delay and setup is the same for all paths, we look for the path with largest t_{c-q}. The output register has $t_{c-q} = 1ns$. The register file has a read delay of 2ns; however, if the same location is being written as read, then on the rising edge of the clock, the register file picks up the data at the data input and takes the write delay (1ns) to store the value successfully in the register. This means that since the rfile value is not valid for 1ns after the clock edge, the read delay will be added to this, for a total of 3ns. One way to convince yourself of this is to consider that the rfile may be implemented by registers and a mux for the read port. The write delay would then be from the clock-to-q of the rfile registers, and the read delay would be from the mux logic to output the value. So, the critical path is rfile→adder→rfile.

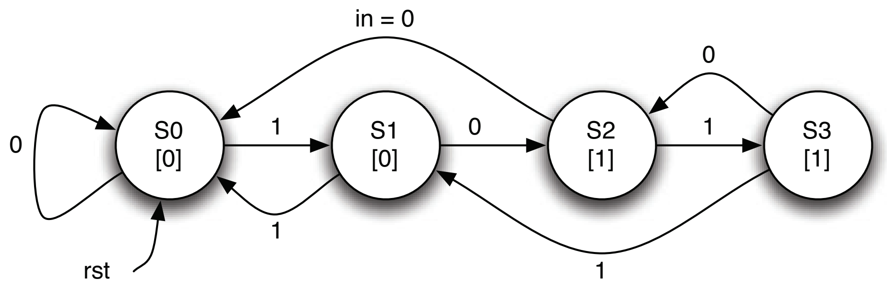
$$T_{min} = (t_{write} + t_{read}) + t_{logic,max} + t_{setup}$$
$$T_{min} = 1ns + 2ns + 4ns + 1ns = 8ns$$
$$f_{max} = 1/T_{min} = 1/8ns = 125MHz$$

Note1: Some answers included setups and clock-to-q for both the rfile and the output register. Keep in mind that the register file is asynchronous read, but unlike your MIPS cpu project, there is not a combinational path through it in this problem. The reason is that the combinational path through the register file would be from the output address (OAa) to the output, but the path in this circuit instead includes the data write port, which only "takes" data on the clock edge, making it a path a endpoint. Note2: Some answers included the hold time in the calculation of T_{min}. Hold time is a separate timing constraint, which says that the minimum time from the clock edge to the endpoint flipflop input changing has to be greater than the hold time of the flipflop. $t_{hold} <= t_{c-q} + t_{logic,min}$.

5. Verilog and Finite State Machines [12pts].

For the state transision diagram shown below:

(a) Complete the Verilog description, following the CS150 style rules.



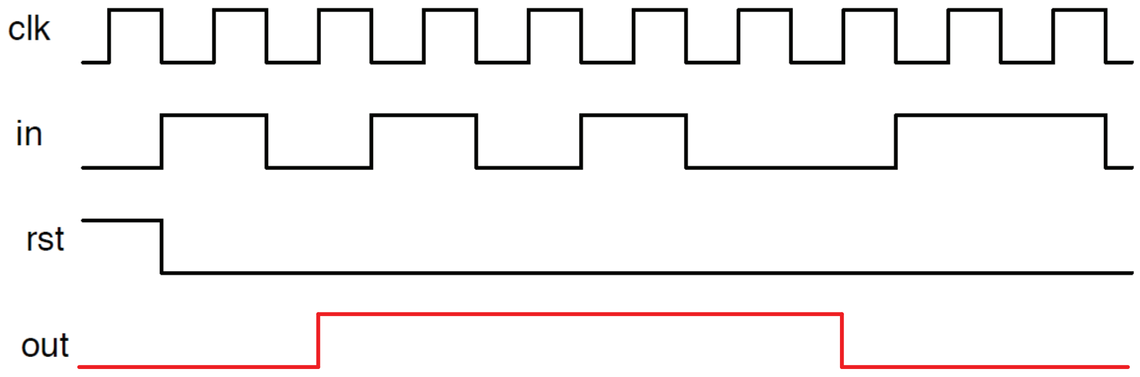
The solution is given here:

```
module FSM(clk, rst, in, out);
    input clk, rst;
    input in;
    output out;

    parameter S0 = 2'b00, S1 = 2'b01, S2 = 2'b10, S3 = 2'b11;

    reg [1:0] CS, NS;

    always @(posedge clk) begin:
        if (rst) CS <= S0;
        else CS <= NS;
    end

    assign out = (CS == S2) | (CS == S3);

    always @( * ) begin
        NS = CS;
        case (CS)
            S0: if (in) NS = S1;
            S1: if (in) NS = S0; else NS = S2;
            S2: if (in) NS = S3; else NS = S0;
            S3: if (in) NS = S1; else NS = S2;
        endcase
    end
endmodule
```

This part was given 9 points. In addition to grading down for functionally incorrect Verilog, solutions which might have been functionally correct but heavily deviated from the FSM style shown in class lost points.

(b) Complete the waveform for “out” corresponding the input signals shown below:

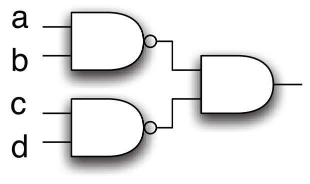
The solution waveform is given here:

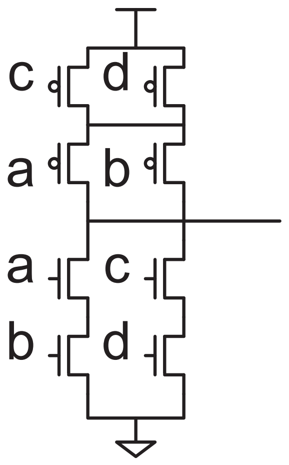


This part was given 3 points. Each cycle whose output value was incorrect lost $\frac{1}{3}$ *points, rounded up at the end to the nearest point.*

6. Transistor Circuits [10pts].

Draw a transistor level circuit diagram for the function depicted in the gate level circuit below. Minimize the total number of transistors. No "pass-transistor logic", use only "static cMOS" circuits.





After "pushing the bubbles", the expression becomes much simpler to map to CMOS.

Many students implemented the function by translating each gate to its transistor implementation, and composing the resulting circuits. This approach yields a far less optimal solution, although a functionally correct one. Full credit was given only for a correctly optimized transistor schematic.

Up to 5 points were awarded for a functionally correct implementation. Up to 5 points were awarded for an implementation optimized at the transistor level.

7. Stack Machine Design [24pts].

A stack machine is a type of CPU that uses a hardware stack instead of a register file. All instructions take their operands from the stack and leave their result on the top of the stack. Remember, a stack is a "LIFO" (last in first out data structure) and typically supports "push" and "pop" operations.

In this problem you will design the datapath and specify the control for the following subset of a stack machine instruction set:

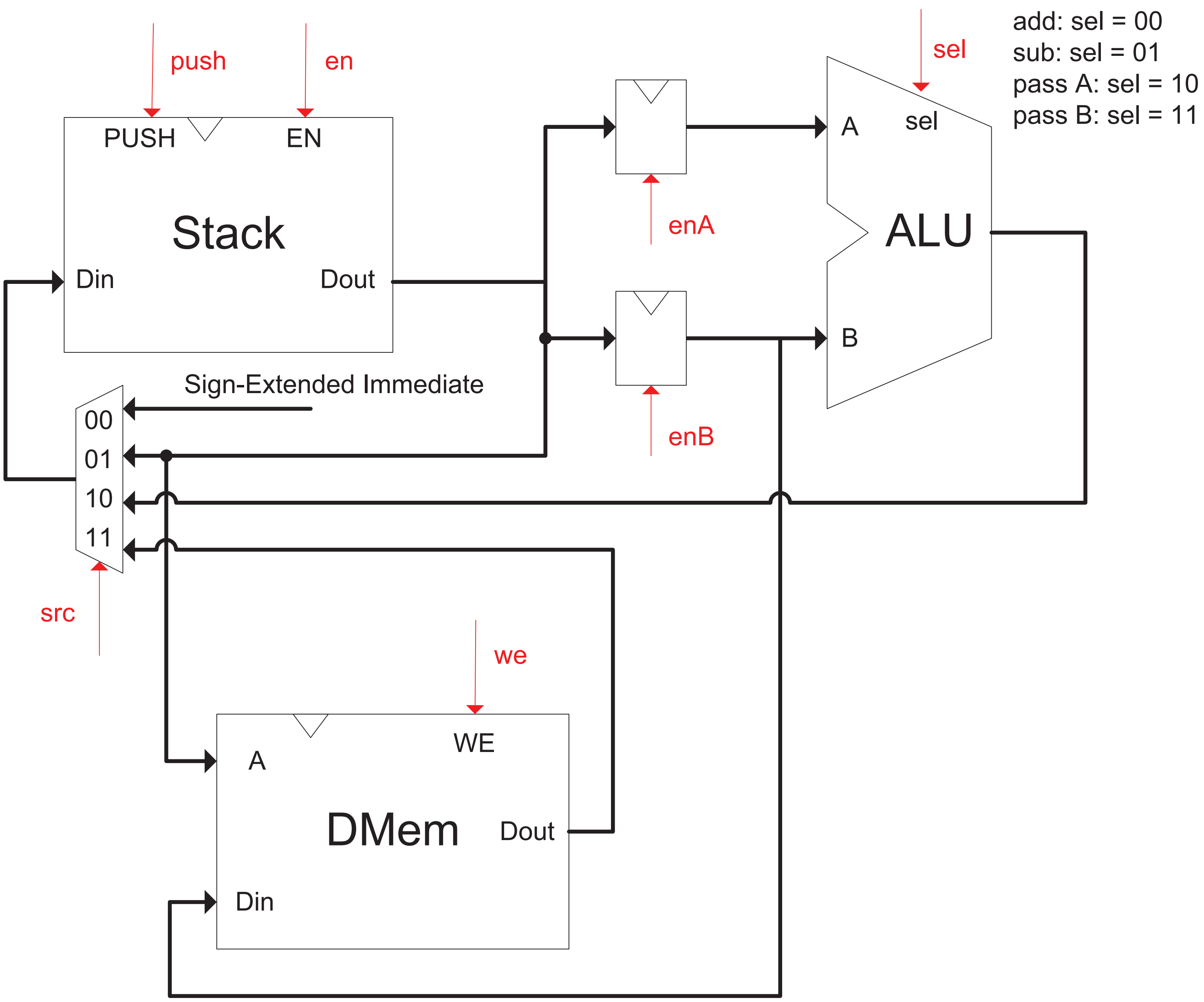
Instruction	Description
add	Pops two elements, forms their sum, and pushes the result.
sub	Pops two elements, subtracts the first popped from the second, and pushes the result.
dup	Duplicates the top of the stack.
swap	Exchanges the first two elements on the stack.
load	Loads a word from data memory using the top of the stack as the memory address (address is popped, the data from memory is pushed).
store	Stores the top of the stack in data memory using the next element on the stack as the address (both are popped).
const	Pushes a sign-extended immediate value from the instruction.

Shown below are a few blocks that you will need for your datapath. The data memory block has asynchronous read and synchronous write. The stack block has a data input and data output, along with two control signals, push (PUSH), and enable (EN). The top of the stack is always available on the data output signal Dout, i.e., it is always possible to peak at the top of the stack. On the positive edge of the clock, if PUSH=1 and EN=1 then the value on the data input is pushed onto the stack; if PUSH=0 and EN=1 then the top of the stack is popped; if EN=0, then the stack is not changed.

(a) In the space provided below add wires and any other blocks that you need for the above instructions. Be neat, and avoid crossing wires when possible. Circle your control signals. You will assign control signal values in part (b), coming up.

Your top priority is to minimize the number of cycles per instruction, followed by making the datapath as simple as possible. You may ignore instruction fetch for this problem.

Solution



Up to 10 points were awarded.

(b) Label the table columns with your control signal names. Fill in the table with the control signal values for each instruction. Use "X" to indicate "don't care". Note that some instructions may take more than one cycle. In those cases, use one row for each cycle of the instruction.

	push	en	enA	enB	sel	we	src
add	0	1	1	0	X	0	X
	0	1	1	1	X	0	X
	1	1	X	X	00	0	10
sub	0	1	1	0	X	0	X
	0	1	1	1	X	0	X
	1	1	X	X	01	0	10
dup	1	1	X	X	X	0	01
swap	0	1	1	0	X	0	X
	0	1	0	1	X	0	X
	1	1	X	0	10	0	10
	1	1	X	X	11	0	10
load	0	1	X	X	X	0	X
	1	1	X	X	X	0	11
store	0	1	X	1	X	0	X
	0	1	X	X	X	1	X
const	1	1	X	X	X	0	00

Up to 10 points were awarded.

(c) Most stack machines also include a rotate instruction:

Instruction	Description
rot	"Rotate" the top 3 elements; the third element moves to the top and the top two elements move down one.

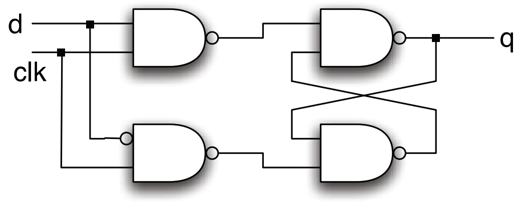
Is it possible to execute a rotate instruction on your machine without modifying the datapath? If so, how many cycles will it take?

No, it is not possible to implement the rotate instruction without modifying the datapath. To perform a rotate, three elements must be popped from the stack, and then pushed back onto the stack in a modified order (pop A, pop B, pop C, push A, push B, push C) This requires at least enough state to store the three words being rotated. Note that a correct solution is not allowed to modify the data memory: such an implementation breaks the ISA! The datapath shown in this solution only provides two registers to store intermediate state, and is thus insufficient to implement the rotate instruction.

Up to 4 points were awarded based on correctness (relative to your part a) and reasoning.

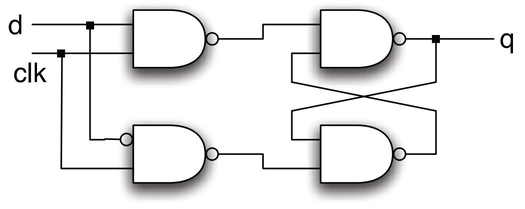
8. Flip-flop Implementation [16pts].

Consider the circuit shown below. Its function is a level-sensitive "high transparent" latch—transparent when the clock level is high.

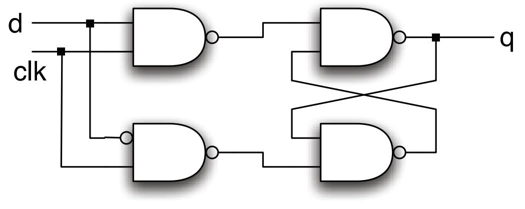


If possible, modify the circuit to achieve the following functions by adding bubbles (inversions) and extra gate inputs (no extra gates or transistors are allowed). If the desired function is not possible, indicate it by writing "not possible".

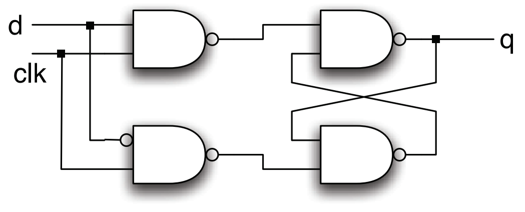
(a) Clock enable (CE) input.



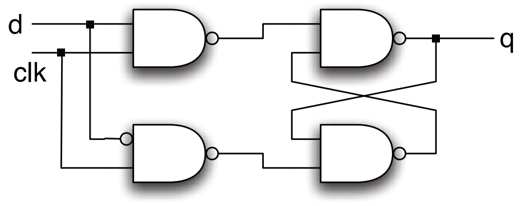
(b) Asynchronous Reset (clear)—sets the latch value to 0 independently of the clock.



(c) Synchronous Reset (reset)—set the latch value to 0 if the clock is high.

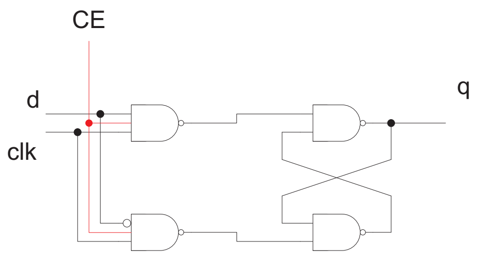


(d) Negative Edge-triggered Flip-Flop. For this part you may add another copy of the latch, if needed.

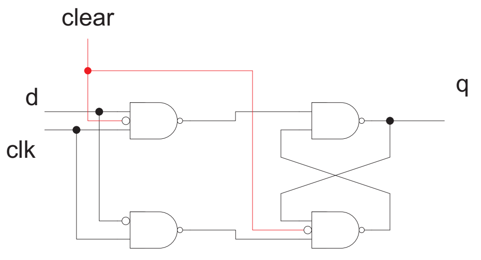


The solution to each part is shown below:

(a)

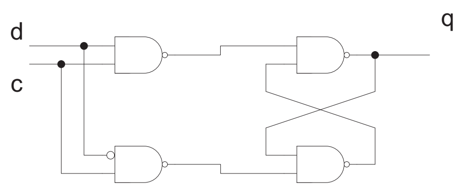


(b)

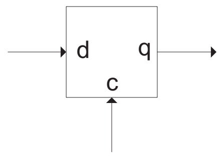


(c) Not Possible

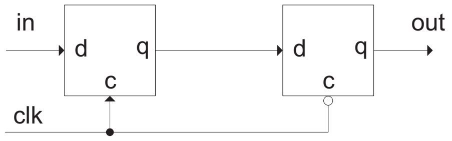
(d) We abstract:



Into a level-sensitive latch block:



Then a negative Edge-triggered Flip-flop is given by:



Each part was worth 4 points. Partial credit was given based on whether you handled the case where the added signal was logic 0 or logic 1, but not both. No credit was given to solutions that said an implementation was "Not Possible" when it was possible, and vice versa.

9. Short Answers [19pts].

(a) [1pt] List the primary reason why FPGAs have lower performance than ASICs. *Critical path delay on FPGAs is dominated by interconnect. Because the wiring is "general purpose" and not optimized for a particular logic function, the individual wires are longer and therefore higher capacitance. Furthermore, the connections between the wires are made with transistors, which add resitance. The net result is connections with a longer RC delay constant.*

(b) [4pt] Over the years, the size of LUTs have been increased by the FPGA manufacturers (from 3-LUTs to, now, 6-LUTs).

List two reasons why larger LUTs are a good idea: *Larger LUTs can result in higher performance and lower power for some functions, as the function can be mapped with less reliance on the interconnection fabric.*

Larger LUTs help amortize the area expense of extra circuitry that exits at the per LUT or per CLB-level, such as connections to the interconnection fabric, flip-flops, muxes, etc.

List two reasons why smaller LUTs are a good idea: *When a simple logic function is needed and it cannot be combined with another such function, mapping to a larger LUT would result in not utilizing the entire LUT (internal fragmentation). In other words, smaller LUTs are easier to pack full. In cases where the larger LUTs could not be fully packed, a smaller LUT would lead to better area density and higher performance.*

(c) [3pt] Assume a product development based on an FPGA has a NRE cost of $1M and a per unit FPGA cost of $100. An ASIC version of the product has an NRE cost $10M and a per unit cost of $10. Your company plans to ship 10,000 units of the final product. Would you choose FPGA or ASIC, and why? *The total cost to the company for the FPGA approach is:*

$$\$1M + 10,000 \ units \times \$100/unit = \$2M$$

And for the ASIC approach:

$$\$10M + 10,000 \ units \times \$10/unit = \$10.1M$$

Therefore the FPGA approach is the least expensive for the company.

(d) [2pt] List two reasons why you might choose to use Block RAM (BRAM) over distributed RAM (LUT-RAM) in an FPGA based design. *BRAMs have higher bit density and therefore are more area efficient for a relatively large number of bits. BRAMs have two true dual ports built in support for FIFO generation, built in ECC, two independent clocks.*

(e) [1pt] List one reason why you might choose to use LUT-RAM over BRAM in an FPGA based design. *LUT-RAMs allow asynchronous reads and are more area efficient when a relatively small number of bits need to be stored.*

(f) [4pt] Suppose you are given a 256 X 8 simple-dual-ported memory block. How many such memory blocks would it take to implement a 512 X 16 memory with two read ports and two write ports? *2 blocks for 16-bit width* $\times$ *2 for 512 word depth* $\times$ *2 for two read ports* $\times$ *2 for two write ports* = *16 blocks total.*

(g) [3pt] Consider a video system with a 1K by 1K image, 100 frames/second, and 3 Bytes per pixel. We would like to send a video stream over Ethernet and have available 10Mbps, 100Mbps, and 1000Mbps connections. Which of these connections would provide sufficient bandwidth?

$$1K\ pixels \times 1K\ pixels \times 100\ frames/sec \times 3Bytes/pixel$$

$$= 300MB/s\ \ (Million\ Bytes\ per\ second$$

The highest of the three Ethernet connections is 1000Mpbs (Million bits per second), which is approximately 125 Bytes per second. Therefore, none would provide sufficient bandwidth.

(h) [1pt] Put your name and SID on each page.

Spare page. *Will not be graded.*

Spare page. *Will not be graded.*

Spare page. *Will not be graded.*

Spare page. *Will not be graded.*

Spare page. *Will not be graded.*