

You may hand in your solution via email (cs141-2009-staff@eecs.harvard.edu) or in person at lecture.

1 Required Reading

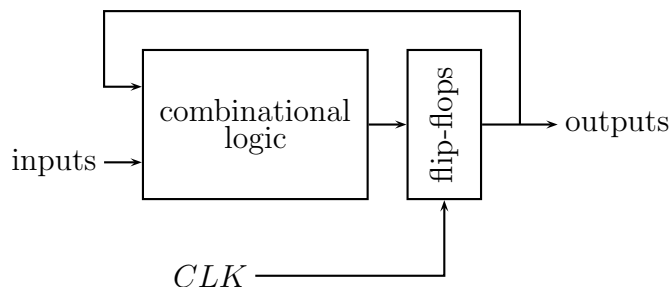
None - review last week's reading and do study for the midterm.

2 Problems (55 total points)

Please show your work in your solutions.

Optional midterm preparation questions will be *marked like this*. You don't need to turn them in, but they will help prepare for the midterm.

1. (5 points) Given the sequential logic circuit below, where the flip-flops have worst-case setup times of 20ns, propagation delays of 13ns, and hold times of 5ns, answer the following two questions:



- Assuming a zero propagation delay through the combinational logic block, what is the maximum allowable frequency of the clock that controls this sub-system?
- Assuming a typical combinational logic delay of 75ns and a worst-case delay of 100ns, how does your answer to part (a) change?

*(Optional) It is important to learn efficient Verilog coding. Using the techniques we learned in lab, implement a 32-bit shift register in 15 lines of Verilog or less (merging two or more lines into a single line is not allowed). Your module will have inputs *in*, *clk*, *load*, and*

$d[31:0]$, as well as output $q[31:0]$. When $load$ is low, you should shift the value of in into $q[0]$, the value of $q[0]$ into $q[1]$, and so on, on every clock edge. When $load$ is high, load the value of $d[31:0]$ into $q[31:0]$ on the next clock edge. You do not have to initialize $q[31:0]$.

2. (10 points) In addition to the simple FSMs you have been designing and learning in class, it is possible to use complex modules in your FSM design, for instance counters or adders. To give you a taste of such design, we ask you to design a FSM with input X and output Z with the following rules:

While $X = 0$, Z should also be 0.

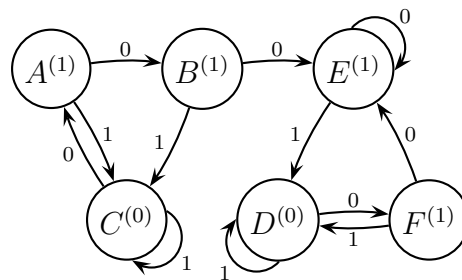
While $X = 1$, Z should also be 1, but for a maximum of sixteen clock cycles. After then, Z should go back to 0

For example, study the following simulation:

```
X 00011110001111111111111111111111000...
Z 00011110001111111111111111111111000000...
```

A typical FSM for this design would need at least 16 states. But if you have access to a 4-bit binary counter, you can do it with much fewer states. Design such an FSM and draw the circuit for it using D flip-flops and a 4-bit binary counter.

3. (10 points) Starting with the state diagram below, use row reduction and the implication chart method to find the minimum state diagram. The number in parentheses next to node names indicate the output values (this is a Moore machine).



4. (15 points) Design a Mealy finite state machine with input X and output Z . Z should be asserted for one clock cycle whenever the subsequence 0111 or 1000 has been input on X . The patterns may overlap. Study the following example

```
X 110000111000000...
Z 000010001001000...
```

Draw a state diagram, complete the state table, minimize it, assign the states and give minimized boolean equations that correspond to an implementation of this machine using D flip-flops.

5. (15 points) Congratulations! You just got a job at U-mobile, a popular cell phone provider. Given your extensive experience with computer hardware and digital design (there have been rumors that U-mobile has been taken over by Silicon7, which may have helped you get the job...), your employer put you in charge of an important project.

U-mobile base stations are using a programmable Zilinx FPGA to propagate the incoming signal through to the other base stations. Unfortunately, Zilinx input ports are only 4-bits wide while the data is 8-bit wide. So, what happens is that data that coming in is *serialized* in transmission, that is, each 8-bit chunk is split into two 4-bit chunks $a_3a_2a_1a_0$ and $b_3b_2b_1b_0$ and the chunks are sent to the base stations one after another. Moreover, since wireless communication channels are error-prone, each set of two chunks is *preceded* by a *control signal* – a set of four bits of the form

$$1\ 1\ cs_1\ cs_0 \quad (1)$$

where $cs_{1-0} = \{a_3 \oplus a_2 \oplus b_1 \oplus b_0, b_3 \oplus b_2 \oplus a_1 \oplus a_0\}$ is the checksum useful for error-checking.

The beginning of any transmission *BOT* is marked with a control signal of 0100_2 .
The end of any transmission *EOT* is marked with a control signal of 1000_2 .

The inputs to your FSM are `iclk`, a master clock for the FSM, `reset` and `in[3:0]`. The outputs are `out[7:0]`, `strobe` and `err`.

Your task is to design a FSM which does the following:

- While `in` is 0000_2 , the machine is in the `wait` state – waiting for the control signal
- When the *BOT* control signal is read, the machine transitions into the `ready` state, waiting for the data to come in
- While in the `ready` state, the machine accepts *three* sets of four bits – the control signal, and the two chunks. As the final chunk is accepted, the machine checks whether the checksum code from the previous control signal agrees with the checksum calculated from the data and outputs the 8-bit combined data on `out` and sets `strobe` to 1 if the two agree. Otherwise, the machine sets `err` to 1 and moves into the `halt` state (loops forever)
- The machine keeps accepting the data until an *EOT* control signal is received, upon which it goes to the `wait` state.

Assume that `out`, `strobe` and `err` are zero while the machine is reading intermediate data.

Your machine should do some error checking: for example, while the machine is in the `ready` state, if something other than `BOT` appears on input, `err` should be set to 1 and the machine should halt. Below are some examples of the expected behavior of your FSM:

	wait	wait	BOT	cs	a	b	cs	a	b	EOT	wait
in	0000	0000	0100	1110	1101	1001	1100	1100	0011	1000	0000
out	00h	00h	00h	00h	00h	d9h	00h	00h	c3h	00h	00h
strobe	0	0	0	0	0	1	0	0	1	0	0
err	0	0	0	0	0	0	0	0	0	0	0

Notice that above, the two 8-bit pieces of data transmitted were $11011001_2 = d9_{16}$ and $11000010_2 = c2_{16}$. The checksum for the first data is 10_2 because $a_3a_2b_1b_0 = 1101_2$ and $b_3b_2a_1a_0 = 1001_2$.

Below is an example of an incorrect checksum code (11_2 instead of 10_2), which causes the machine to halt:

	wait	BOT	cs	a	b	err	err
in	0000	0100	1111	1001	1000	xxxx	xxxx
out	00h	00h	00h	00h	00h	00h	00h
strobe	0	0	0	0	0	0	0
err	0	0	0	0	1	1	1

Below is an example of an incorrect control signal, which also causes the machine to halt:

	wait	BOT	cs	a	b	????	err
in	0000	0100	1100	1001	1001	0110	xxxx
out	00h	00h	00h	00h	00h	00h	00h
strobe	0	0	0	0	1	0	0
err	0	0	0	0	0	1	1

Design this FSM, describing its states and transitions. Explain all possible error conditions. **Don't** calculate the transition equations. Feel free to use other components in your design (counters, adders, multiplexers...).