CS150 Fall 2012 — Solutions to Homework 4

September 23, 2012

Problem 1

43 CLBs are needed.

For one bit, the overall requirement is to simulate an 11-LUT with its output connected to a flip-flop for the state bit's register.

Using the general rule of LUT combination, it is possible to make an 11-LUT out of 8 8-LUTs and an 8:1 multiplexer. In order to satisfy the requirement of 8 8-LUTs, 8 slices may be used in the configuration to implement an 8:1 logic function, as discussed in class.

For one bit, there are 11 inputs to the logic function determining its next value. If these inputs are numbered I_0 through I_{10}, we may assign I_0 through I_7 to be connected to the 8-LUTs. Each of these 8-LUTs will implement the subset of the overall truth table corresponding with *one* potential state of the remaining inputs I_8 through I_{10}. These intermediate outputs can be numbered O_0 through O_8.

The final step is to select the correct signal from the set of intermediate outputs. This can be accomplished with an 8:1 multiplexer selecting based on the state of $\{I_8, I_9, I_{10}\}$. Since a 6-LUT has 6 inputs, it can implement any 6:1 logic function, including a 4:1 multiplexer, which has $2 + 4$ inputs including the select. Therefore, intermediate outputs O_0 through O_3 can be fed into a 6-LUT in another slice along with I_8 and I_9, while intermediate outputs O_4 through O_7 are fed into a second 6-LUT with I_8 and I_9.

Finally, the outputs of these two 6-LUTs (one of which contains the correct output for the 11-input overall logic function) can be multiplexed by one of the 2-input muxes connected to pairs of 6-LUT outputs, F7AMUX or F7BMUX.

However, since this 8:1 multiplexing and registering uses two 6-LUTs, along with one of the muxes used for 7-input functions and one output flip-flop, the slice has only half of its inputs and outputs used. If LUTs C and D are used, all logic associated with LUTs A and B are unused, and vice versa. Therefore, the 'glue logic' for *two* state bits can be made to fit in one slice. However, an acceptable answer could still use one whole slice for this 'glue logic' per bit.

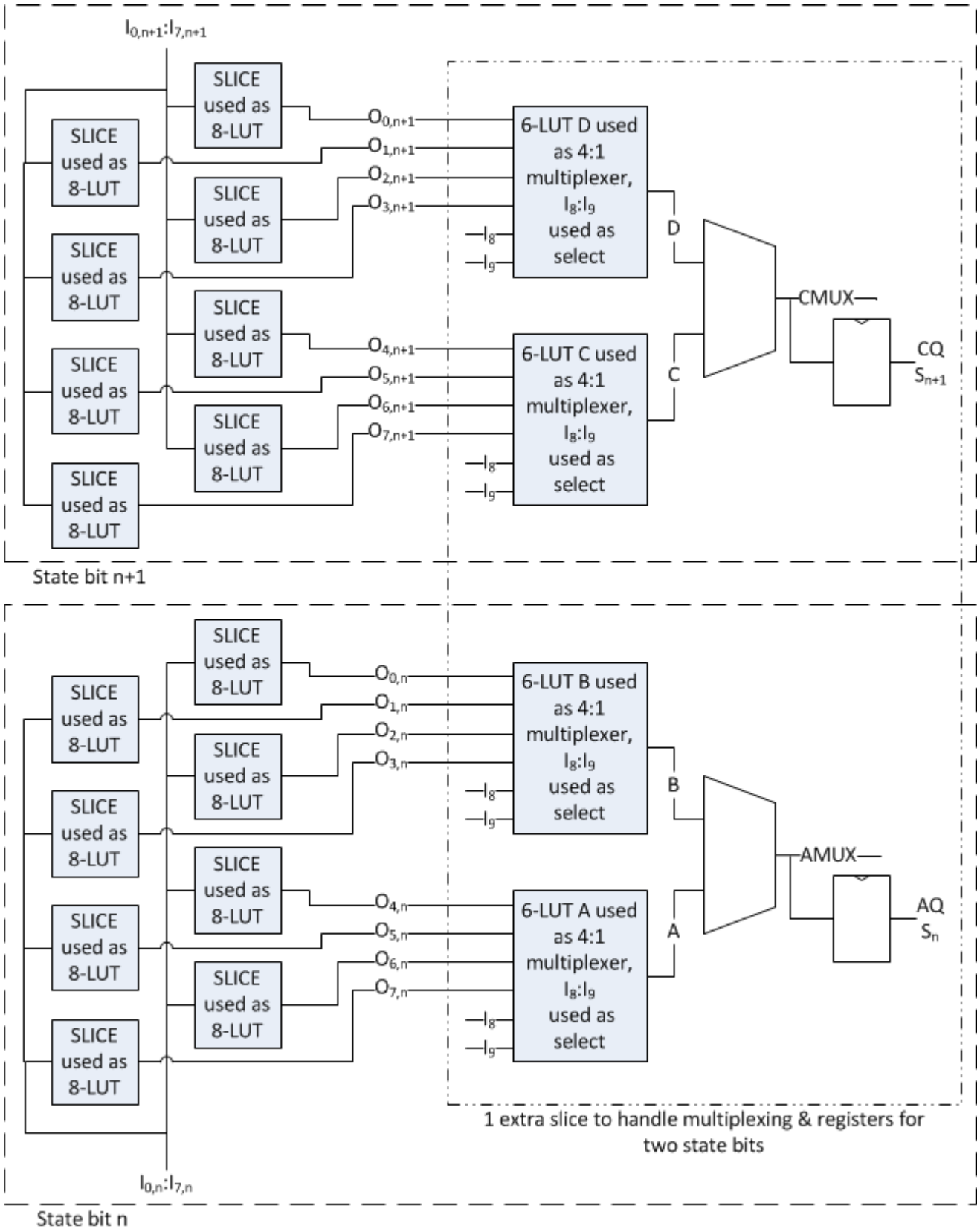
This results in $8 + 0.5 = 8.5$ slices used per bit. For the total number of CLBs, we take the ceiling of half the number of slices used for all ten bits.

$$\text{number of CLBs} = \lceil [10 * (8 + 0.5)] / 2 \rceil = \lceil 42.5 \rceil = 43$$

Using one whole extra slice on top of the 8 to implement 8 8-LUTs per bit, one could alternately use:

$$\text{number of CLBs} = \lceil [10 * (8 + 1)] / 2 \rceil = \lceil 45 \rceil = 45$$

Below is a block diagram of the 85-SLICE or 43-CLB configuration for two bits. This structure repeated five times would yield the entire FSM state logic; this portion uses 17 slices.



Problem 2

Since we are given the information that 8 inputs and 8 state bits (and therefore 8 output registers) are used, the only requirement is that the number of product terms overwhelms the number of inputs on the or gate.

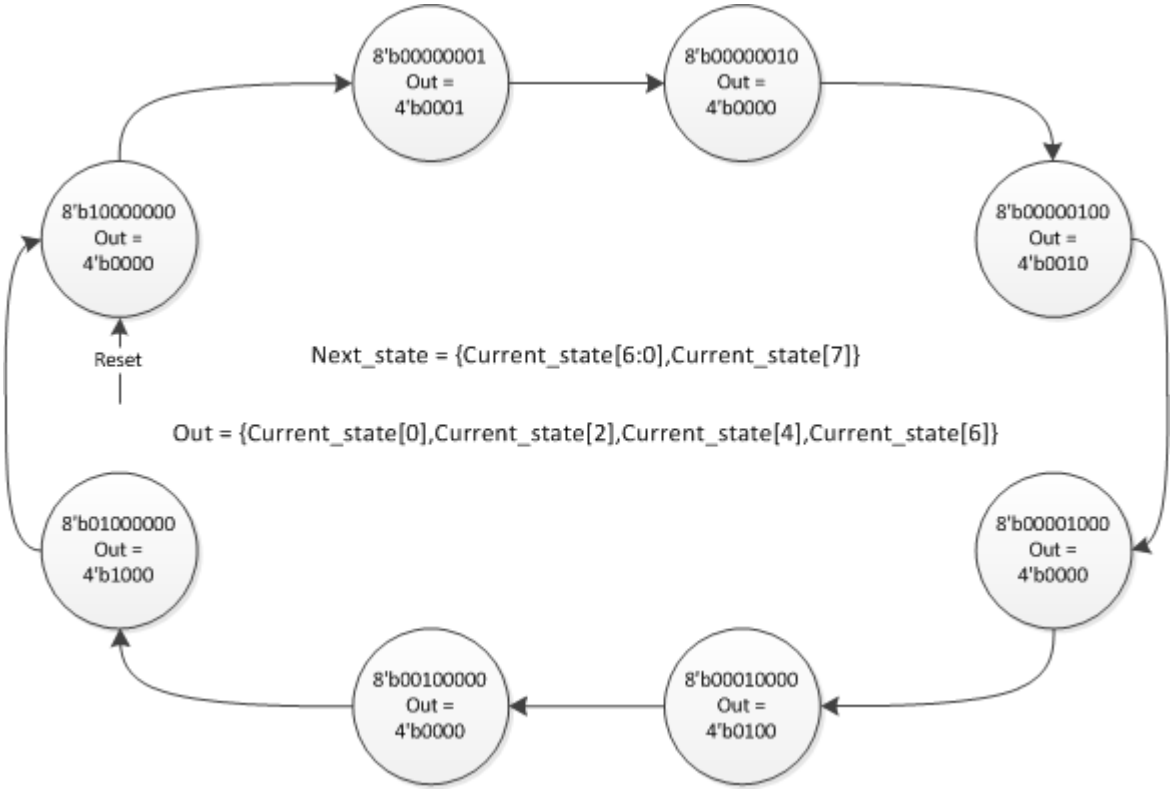
Here, we will use the pin numbering to subscript the signals. Therefore, the inputs will be I_2 through I_9 and the state bits will be S_{12} through S_{19}.

Any logic function with more than 8 product terms in its most reduced form will prevent implementation using one macrocell. One example is:

$$Out = I_2 + I_3 + I_4 + I_5 + I_6 + I_7 + I_8 + I_9 + S_{12}$$

Problem 3

a.) State transition diagram:

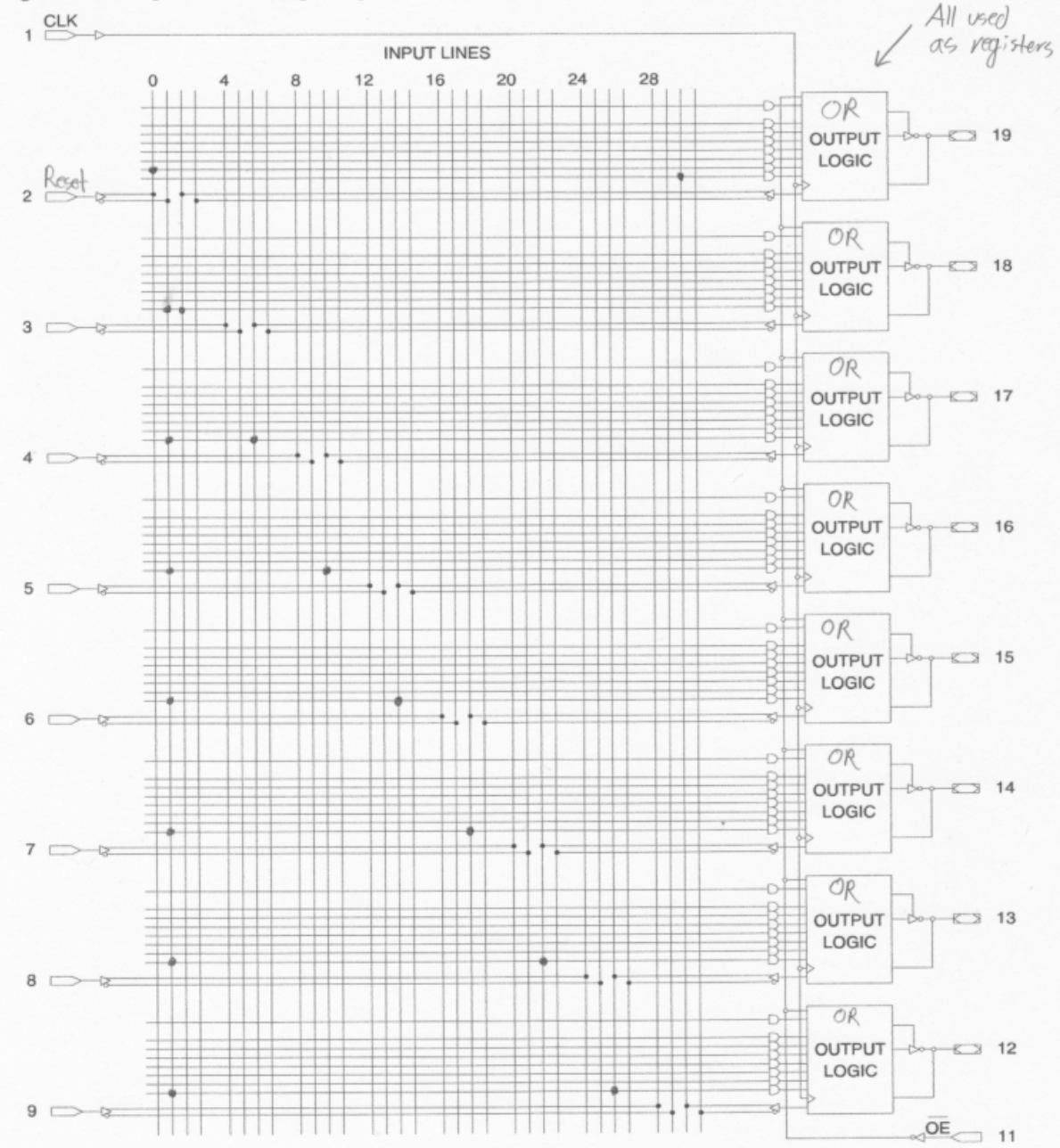


b.) The four output phases will be pins 12, 14, 16, and 18, respectively.

Pins [19 : 12] will represent the eight state bits; on the following the positive edge where reset is high, pin 19 will be high. The overall mode will be registered, and all the macrocells will have output logic used as a register, with the output permanently enabled from the FF.

c.) Fuse table:

Figure 11-3. Registered Mode Logic Diagram



Problem 4

a.) This circuit will implement a linear feedback shift register to generate a maximal-length sequence of output bits. Because we have a total of 8 flip-flops available, we may only implement a maximum 8-bit LFSR (regardless of feedback logic). Therefore, using the Wikipedia-sourced polynomial of $o = x^8 + x^6 + x^5 + x^4 + x^0$, we may derive the value of the next input bit as a function of the current 8-bit state.

However, calculating $o = x^8 + x^6 + x^5 + x^4 + x^0$ has quite a lot of computation. Luckily, our next input is a single bit, so we need only use a logic function with 1-bit parity output! Wikipedia has also done the CS70 for us; therefore, we find that for the generator polynomial:

$$o = x^{n_1} + x^{n_2} + \cdots + x^{n_k} + x^0$$

The one-bit output *next* is simply given as the XOR of the bit positions corresponding to (power-1) for each term.

$$next = x[n_1 - 1] \oplus x[n_2 - 1] \oplus \cdots \oplus x[n_k - 1]$$

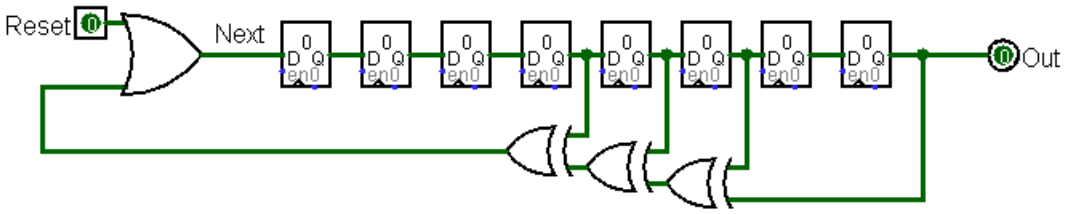
Using the polynomial above, we have the following logical equation.

$$next = reg[7] \oplus reg[5] \oplus reg[4] \oplus reg[3]$$

The only remaining consideration is adding an OR with reset signal to ensure that some bit is nonzero after reset – the state $8'b0$ will result in a constant output!

$$next = (reg[7] \oplus reg[5] \oplus reg[4] \oplus reg[3]) + res$$

The following circuit implements this function.



b.) If we want a non-repeating sequence of greater than 60000 bits, we need a 16-bit LFSR per the Wikipedia article. Since only 8 signals may be registered in one Atmel ATF16V8 unit, the design would need to be changed to incorporate two ATF16V8s with the shift register spanning the two devices.

Problem 5

a.) First, the output function must be mapped to an empty K-map.

		AB			
		00	01	11	10
CD	00	X	0	1	1
	01	X	X	1	0
	11	0	X	1	1
	10	X	0	X	X

Then, the K-map is filled out with three rectangles.

		AB			
		00	01	11	10
CD	00	X	0	1	1
	01	X	X	1	0
	11	0	X	1	1
	10	X	0	X	X

This output function corresponds with $Y = AC + A\bar{D} + BD$.

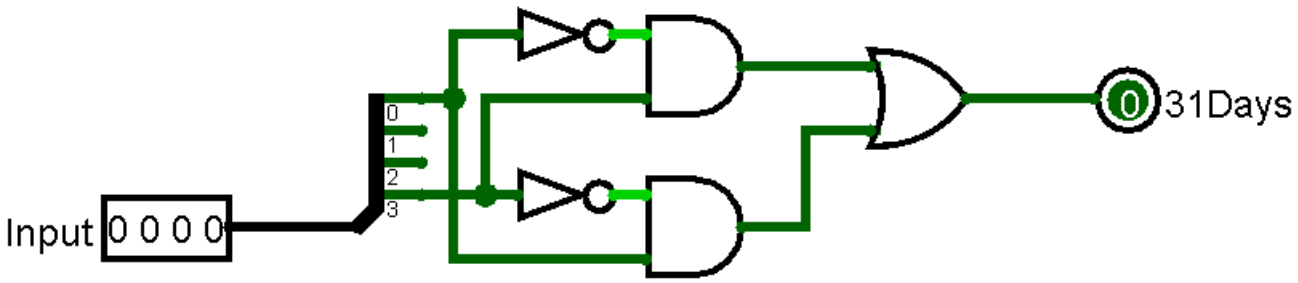
b.) Since there are only 12 months which are numbered from 1, the output function is 'don't care' for inputs $4'b0000$ and $4'b1101$ through $4'b1111$. Therefore, we have the following blank K-map.

		Input[3:2]			
		00	01	11	10
Input[1:0]	00	X	0	1	1
	01	1	1	X	0
	11	1	1	X	0
	10	0	0	X	1

It can then be filled with two rectangles.

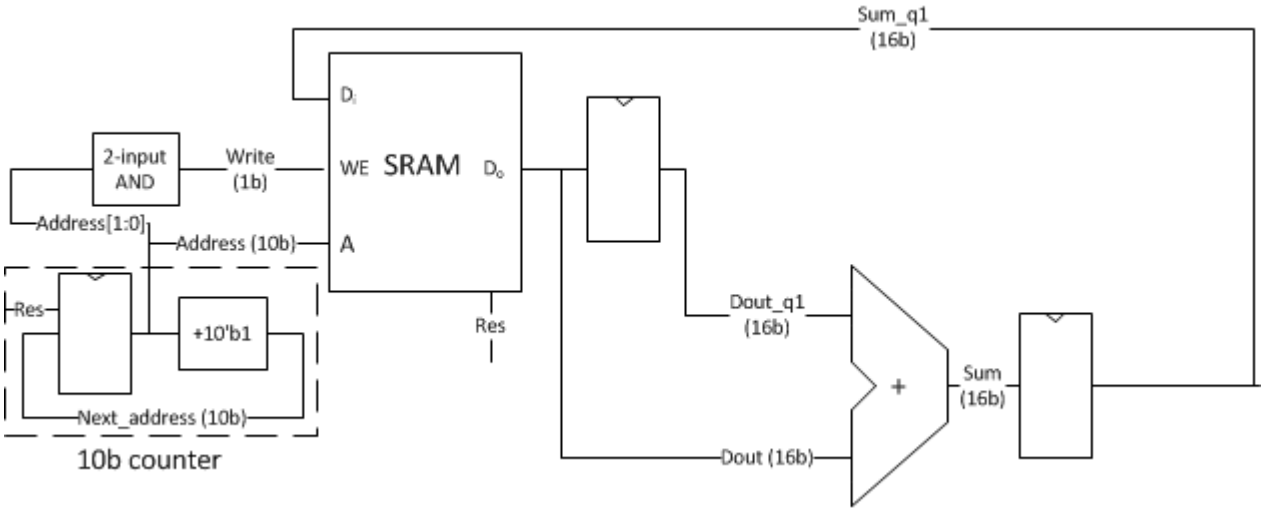
		Input[3:2]			
		00	01	11	10
Input[1:0]	00	X	0	1	1
	01	1	1	X	0
	11	1	1	X	0
	10	0	0	X	1

This corresponds with $Y = \bar{I}_3 I_0 + I_3 \bar{I}_0$. Below is a circuit implementation:



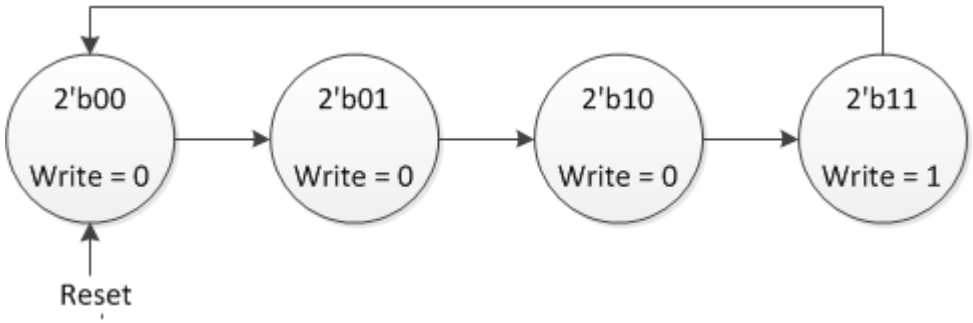
Problem 6

a.) Block diagram:



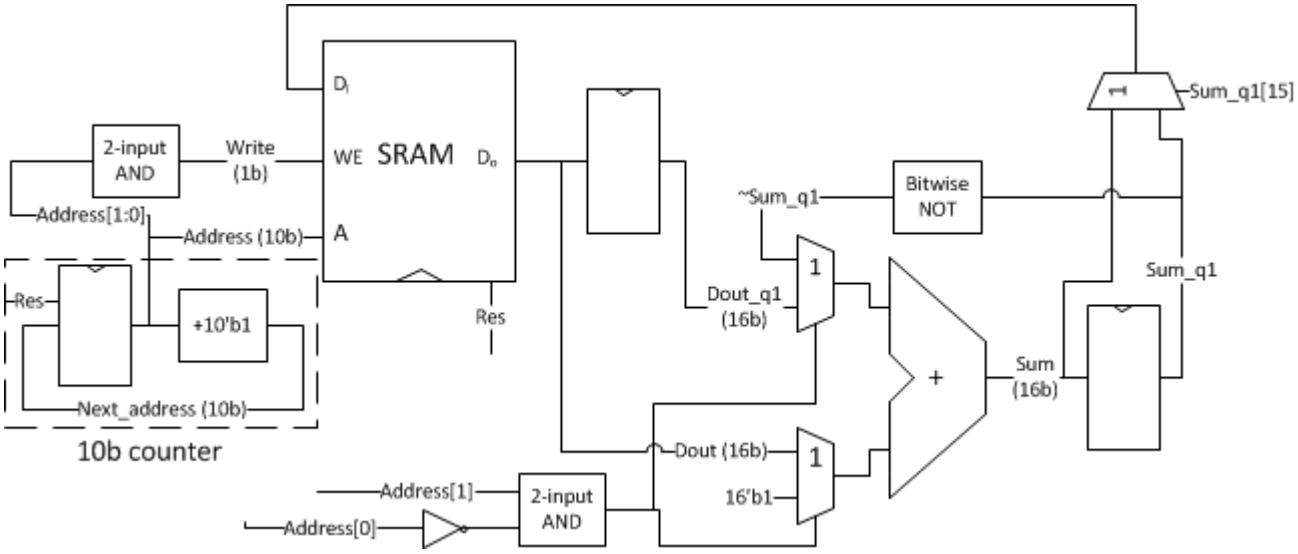
b.) The control signals are the address, the next address, and the write enable.

c.) The state bits of the control FSM are simply the lowest two bits of the address, $Address[1:0]$.



d.) To use one adder to get the absolute value, we could use the dead time that results from not using the data from $D[4n+2]$. During this time, we could feed back the delayed result of the adder into the adder with a bitwise inversion and add $10'b1$ – this would result in the additive inverse of the result.

Then, the SRAM writeback value could conditionally take this additive inverse or the delayed sum depending on whether the delayed sum was negative.


D_i
WE
SRAM
D_o
A
2-input AND
Write (1b)
Address[1:0]
Address (10b)
Res
+10'b1
Next_address (10b)
10b counter
Res
~Sum_q1
Bitwise NOT
Dout_q1 (16b)
1
Dout (16b)
1
16'b1
Address[1]
Address[0]
2-input AND
+
Sum (16b)
1
Sum_q1[15]
Sum_q1