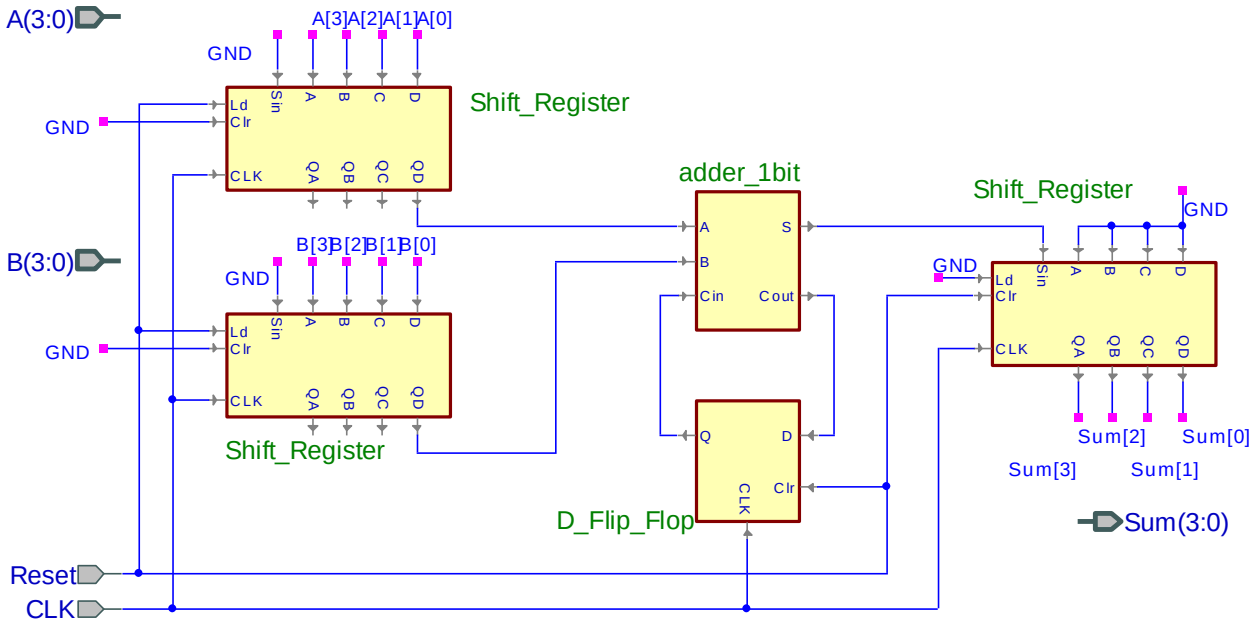
CSE370 HW7 Solutions (Winter 2010)

1. **CLD2e, 7.11**
   For this we basically have three shift registers, one for each operand (A & B) and one to accumulate the sum (S). For this problem we'll assume we don't need to worry about stopping things after 4 clock cycles once the result is calculated (there is some other counter or state machine that knows to read the sum at that point). The first part of the problem asks us to draw the circuit diagram for this adder.

   We are specifically asked to consider how to deal with the carry-out/carry-in. We need to remember the carry-out from the previous 1-bit addition so that we can use it for the next 1-bit addition. Therefore, we need a register to hold this value between clock cycles. The resulting circuit is shown below:



   To handle the reset of the machine there are two things we need to do. First, we need to reset the carry flip-flop so that we start with no carry. Also, on reset we should do a "parallel load" of the A and B operand shift registers to load the initial values that we are adding. Some people had a separate load signal, this is okay.

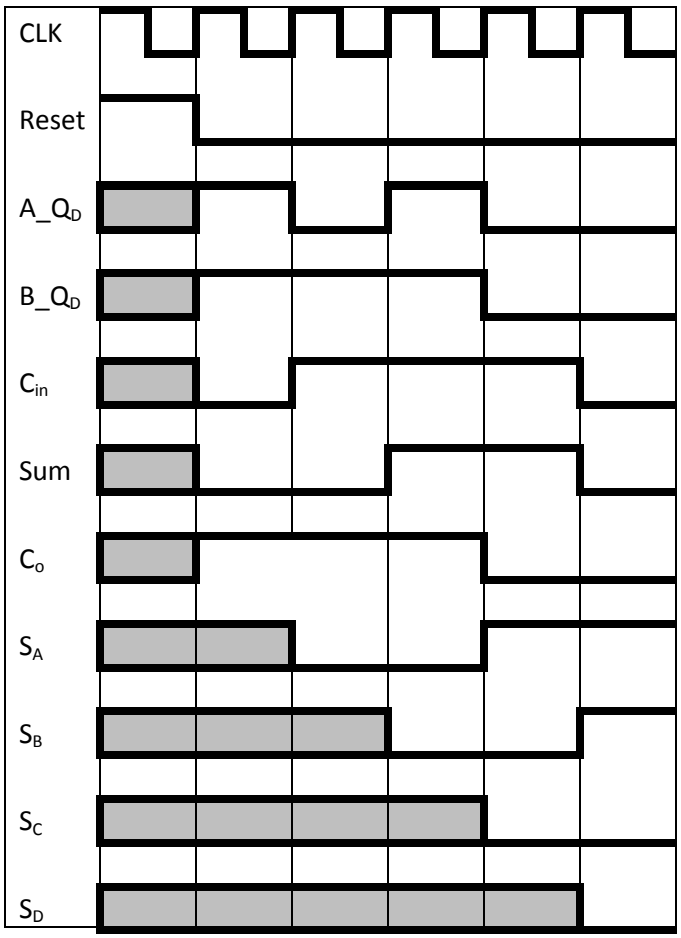
   Note: Some people tried to connect the carry-out connected directly to carry-in w/o a register. This won't necessarily work because the adder is just combinational logic and so the carry-out being rerouted into the carry-in could cause a logic loop. E.g. If A,B,C_{in} = 1,1,0 then S=0 C_o=1…but C_o is C_{in} so A,B,C_{in} becomes 1,1,1, so the result becomes S=1, C_o=1!

   Part (b) of the question asks about the control signals for the adder. For this part it is safe to assume that the registers have a "load" input. Also, we assume that the flip-flop has a reset input. Therefore when reset goes high we will do the "parallel load" into the A & B shift registers and reset the carry flip-flop.

To illustrate how things would work we will assume that the A and B operand shift registers have the following values at their parallel load inputs:

A[3:0] = 0101
B[3:0] = 0111

The following timing diagram shows how the control signals and clock affect the internal values in our circuit:



Notice that the sum is correctly calculated as 1100 (S_A-S_D) after four clock cycles (after reset).

2. CLD2e, 7.18

Let L=3 be the number of state bits
Let M=2 be the number of input bits
Let N=6 be the number of output bits

a) **# of states** – minimum: 8 maximum: 8
Because we are told that we cannot have any don't cares, we know that every possible combination of the registers must represent unique states. This means that there must be exactly $2^L = 2^3 = 8$ states and so the minimum and maximum number of states is 8.

Note: A lot of people said 1 was the minimum number of states. This would normally be true if we allowed for don't cares.

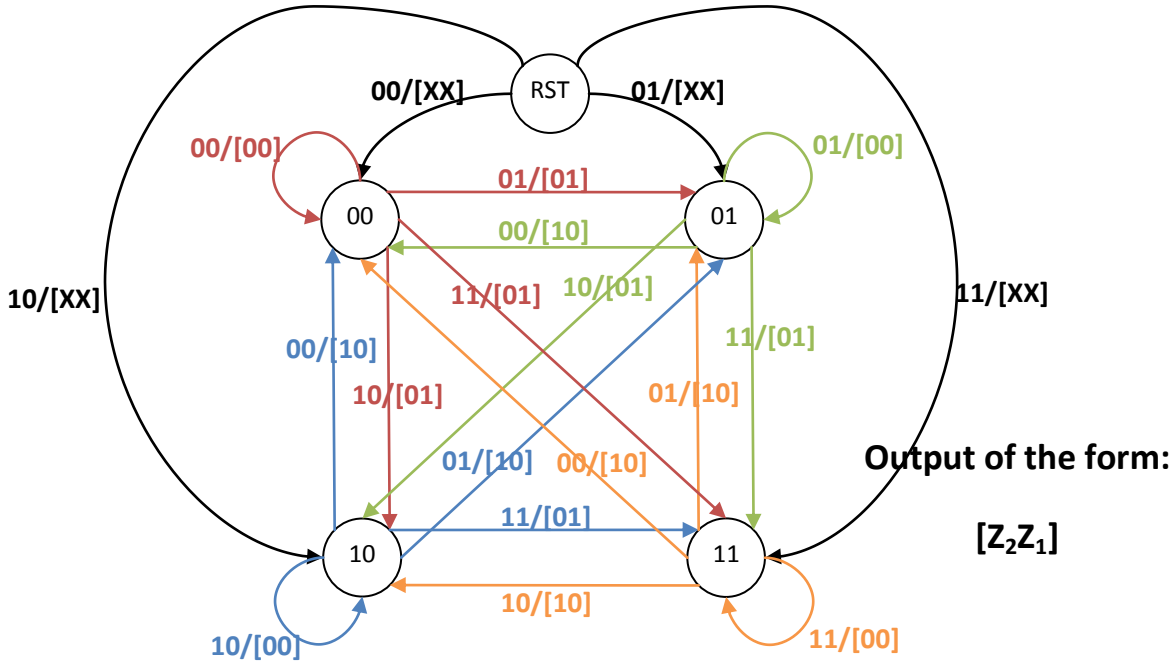
b) **# of transitions per state** – minimum: 4 maximum: 4
Again, we know that each possible input must correspond to a transition and there are no don't cares. Since there are two input bits we know that there must be exactly $2^M = 2^2 = 4$ different transitions from every state. Thus, the minimum and maximum is 4.

c) **#of possible paths into a state** – minimum: 0 maximum: 32
There are no restrictions on where the arrows can go so at minimum there can be 0 transitions into a state. Furthermore, all transitions might go to the same state and so There are $2^{L+M} = 2^5 = 32$ transitions maximum into a state.

d) **# of unique outputs** – minimum: 1 maximum: 32
There are exactly 6 output bits so this would be 2^6 or 64 potential outputs; however, the output in a Mealy machine is based on the current input and state of which there are only 5 bits, this means that the maximum outputs can be at most 2^5 or 32. The exact formula would be $\min(2^{L+M}, 2^N) = \min(2^5, 2^6) = 2^5 = 32$. There must be at least one output since the output must be some value for every state.

3. CLD2e, 7.26

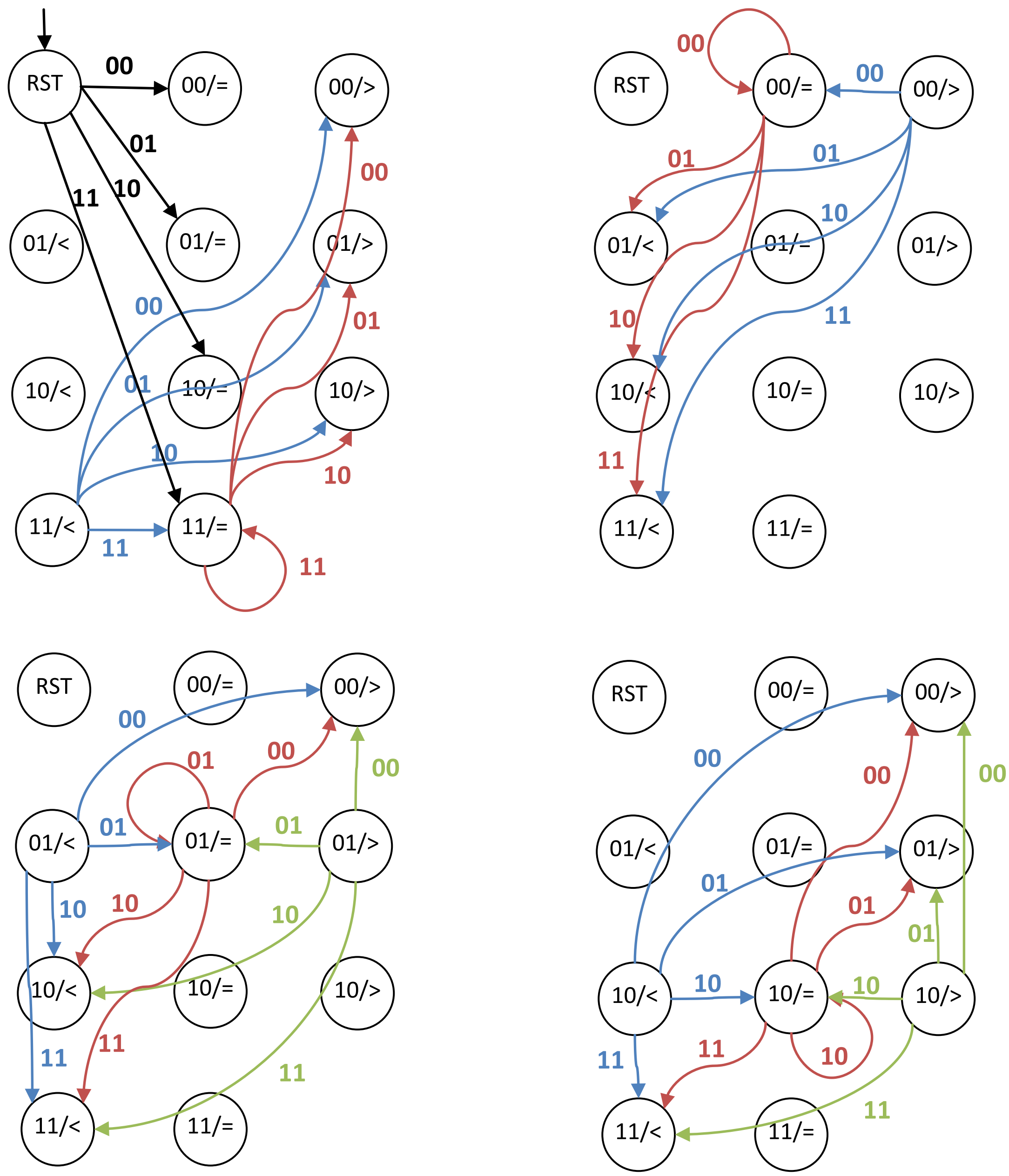
a) There are 5 states, a reset state and a state that represents the current value of the input. The transitions represent the next value of the input and the output for each transition is based on whether the state we are transitioning from and the value of the transition.



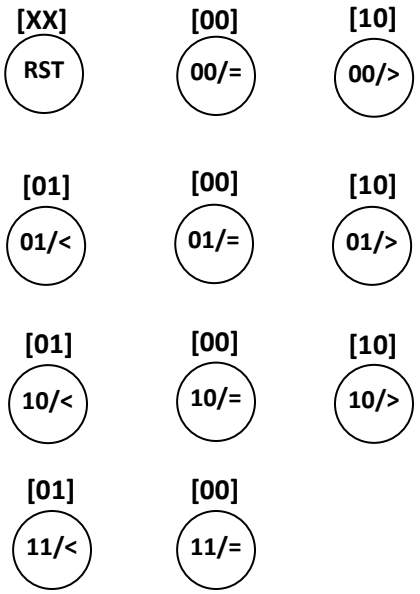
b) There are 11 states. One state is the reset state so we only really have 10 interesting states. You can think of the problem as follows: What information do we need to store to determine the output? Well, we could do it if we knew the value of the current number and

the value of the previous number. This would create 16 states...however, we actually need less information, we really only need to know the current number and whether the previous number was >/</= to the current number. This is 4x3 or 12 states...but wait, if the current value is 00 then the previous number couldn't have been less than 00, so for 00 we only have >/=. Similarly for 11 the previous value couldn't have been >, so we only have </=. That’s how we get the 10 states (plus 1 reset state).

This machine has a lot of transitions so I have split up things to make it easier to see the various transitions and outputs for each of the states:



The Moore output for each state is as follow:



Output of the form: $[Z_2Z_1]$

Note: The reason that we need the RST states is because when the machine is first starting up there is no previous or current value so we need some state to start in. The output from this reset state might be "don't care". We also make the simplifying assumption that the first number we get transitions us to the appropriate "=" state.

4. CLD2e, 7.28

Given the timing for this problem we get the following chart for the lights:

N	Red	Red	Gr →	Ye →	Green	Yellow
S	Red	Red	Red	Red	Green	Yellow
E	Green	Yellow	Red	Red	Red	Red
W	Green	Yellow	Red	Red	Red	Red
0	45	60	80	90	135	150

This timing gives us exactly 6 potential light configurations.

We can also see that we will need a 45 second time, a 20 second timer, a 15 second timer, and a 10 second timer.

Inputs	**Outputs**
Timer45Done, Timer20Done, Timer15Done, Timer10Done, Reset, Clock	ResetTimer45, Reset Timer20, ResetTimer15, ResetTimer10, NorthLights[4:0] SouthLights[2:0] EastLights[2:0] WestLights[2:0]

The inputs are going to be a reset signal, clock signal, and signals for when the timers complete. The outputs are going to be signals to reset each timer and whether each light should be on or off.

In our Mealy state machine we would have 6 states, one for each configuration and when we transition we would reset the appropriate timer for leaving the next state. We will leave a state when the appropriate timer expires (which was reset when we transitioned into this state).

I would probably use an output encoding for the states of the form:

state = {NorthLights[4:0], SouthLights[2:0], EastLights[2:0], WestLights[2:0]}

This makes it so we don't need any logic for the output.

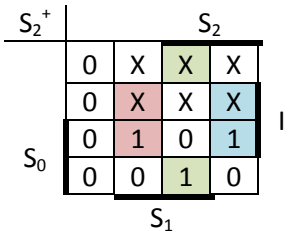
5. CLD2e, 8.18

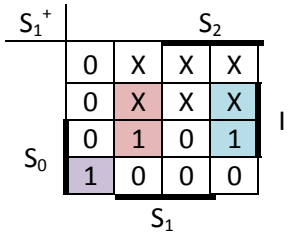
First, notice that this is a Moore machine since the output is based solely on the state. Also, notice that we need three flip-flops because the states are encoded as three bits.

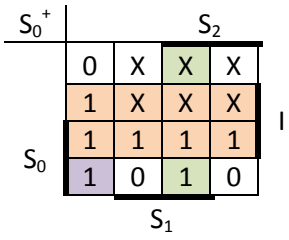
From the state diagram we can see that the state transition table is encoded as follows:

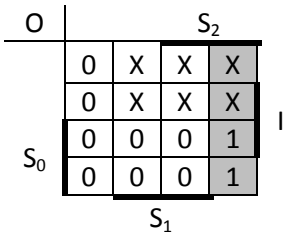
State	S			I	S^+			O
A	0	0	0	0	0	0	0	0
	0	0	0	1	0	0	1	0
B	0	0	1	0	0	1	1	0
	0	0	1	1	0	0	1	0
	0	1	0	0	X	X	X	X
	0	1	0	1	X	X	X	X
C	0	1	1	0	0	0	0	0
	0	1	1	1	1	1	1	0
	1	0	0	0	X	X	X	X
	1	0	0	1	X	X	X	X
E	1	0	1	0	0	0	0	1
	1	0	1	1	1	1	1	1
	1	1	0	0	X	X	X	X
	1	1	0	1	X	X	X	X
D	1	1	1	0	1	0	1	0
	1	1	1	1	0	0	1	0

This gives the following K-maps:









Notice that the problem asked us to use the fewest gates possible. To do this we need to use as many duplicate terms amongst the equations as possible. I have done that above and have a total of 5 terms to computer S^+ and one more term for the output (O).

Our equations are as follows:

$\mathbf{S_2^+ = S_2'S_1I + S_2S_1'I + S_2S_1I'}$

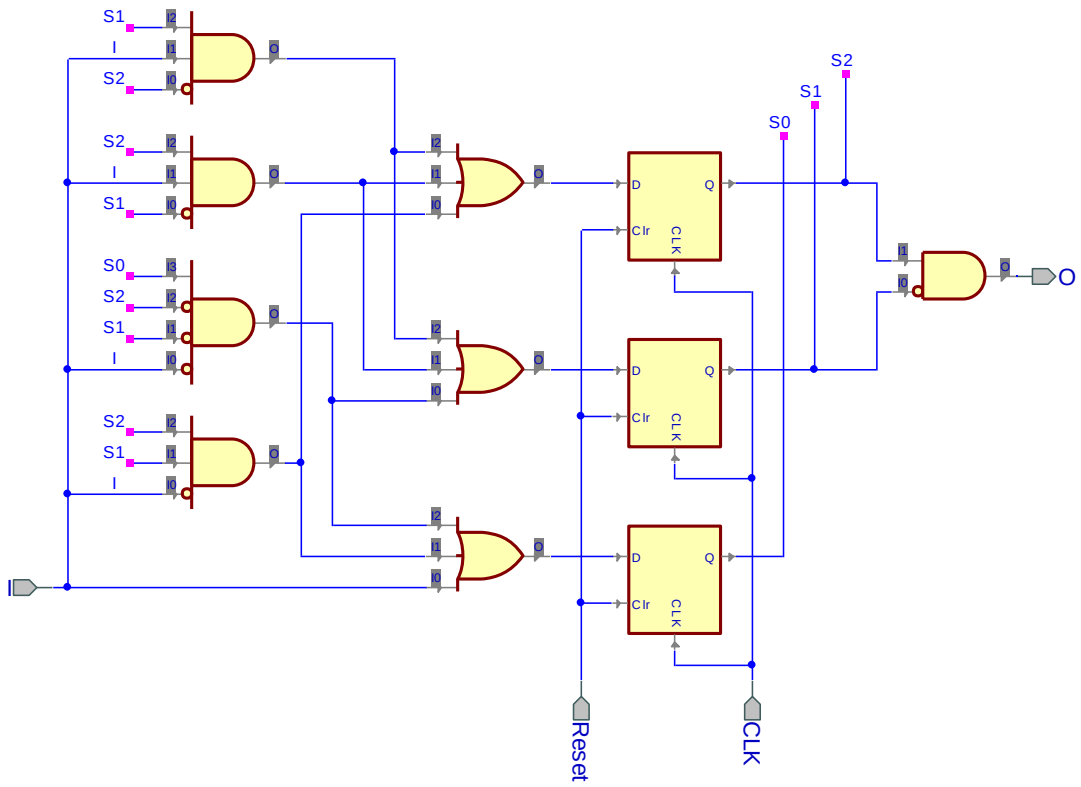
$\mathbf{S_1^+ = S_2'S_1I + S_2S_1'I + S_2'S_1'S_0I'}$

$\mathbf{S_0^+ = S_2S_1I' + S_2'S_1'S_0I' + I}$

$\mathbf{O = S_2S_1'}$

Note: A lot of people minimized each K-map independently and thus ended up with more total terms and thus more logic.

We can then map the equations to the following circuit:



I have chosen to handle the reset signal by using flip-flops with a reset input. If our flip-flops do not have a reset input, then we can add an AND-gate to the input of each flip-flop such that the new value becomes:

$S_X^{++} = S_X^{+}$ & !Reset (for X = 0..2)

6. CLD2e, 8.23

This problem is asking you to make the state machine robust to missing the output TS in state T_{04}. This could happen for many reasons in real hardware that we won't go into here. But basically we want to make our machine robust to this type of error.

The fix is really simple. We just have all states T_{04}-T_{19} output TS, not just T_{04}. This change doesn't add any states or transitions to either diagram. Now even if we miss TS for a while we will eventually get it and transition out of the state and reset the timer.

Note: People had other solutions to this. One set of solutions involved additional signals to communicate back and forth between the two state machines. This adds to the complexity of the solution and the second state machine no longer is just a simple counter.

Note: Another set of solutions made TS output high for several states (e.g. T_{04} and T_{05}), but this has the drawback of not being robust is TS is lost several times in a row.

Note: A final set of solutions had TS high again in T_{19} or changed the transition out of the yellow light state to be on TS or TL. These will prevent the problem but have the downside that if the first TS is missed we will have to wait at least another 14 cycles before we transition.