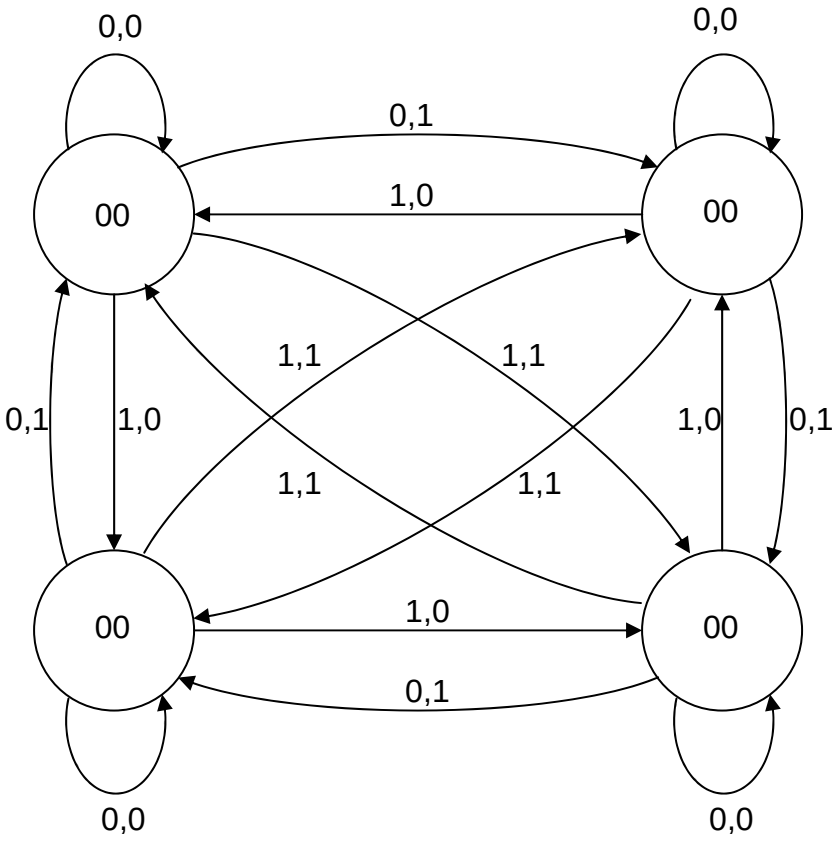
Exercise 7.1

(a) State diagram and transition table:

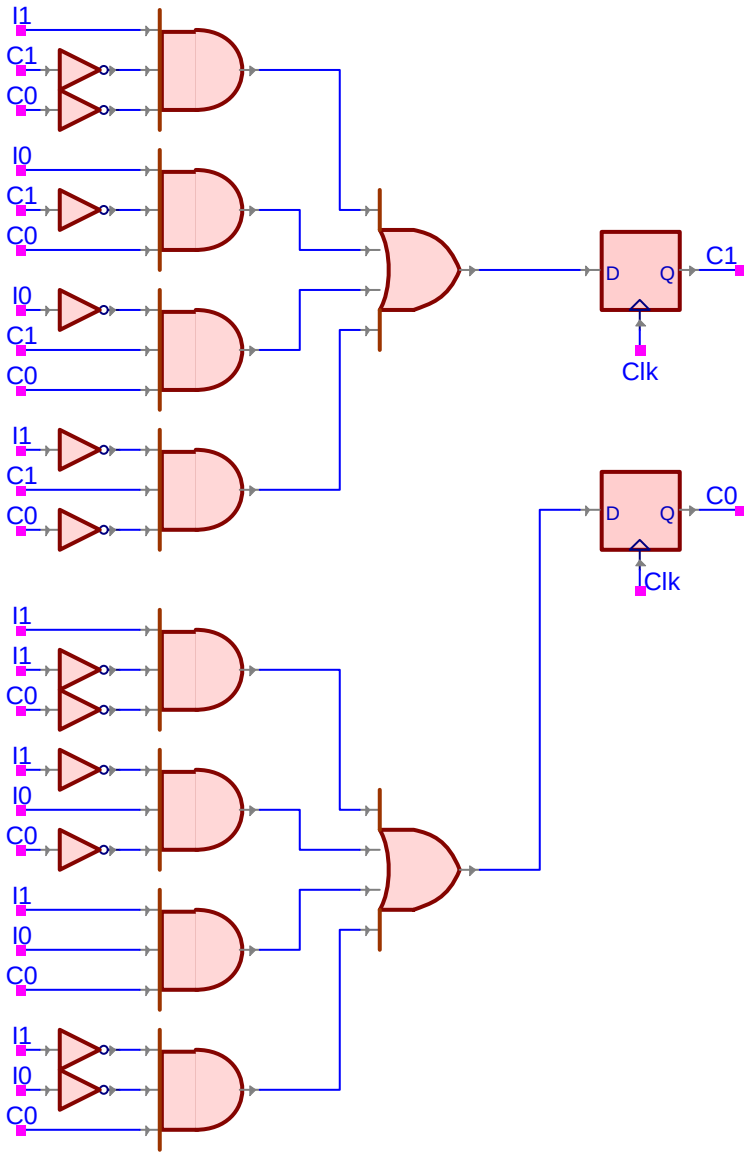


I_1	I_0	C_1	C_0	C_1+	C_0+
0	0	0	0	0	0
		0	1	0	1
		1	0	1	0
		1	1	1	1
0	1	0	0	0	1
		0	1	1	0
		1	0	1	1
		1	1	0	0
1	0	0	0	1	1
		0	1	0	0
		1	0	0	1
		1	1	1	0
1	1	0	0	1	0
		0	1	1	1
		1	0	0	0
		1	1	0	1

(b) Circuit schematic:

$C_1 = I_1C_1'C_0' + I_0C_1'C_0 + I_0'C_1C_0 + I_1'C_1C_0'$

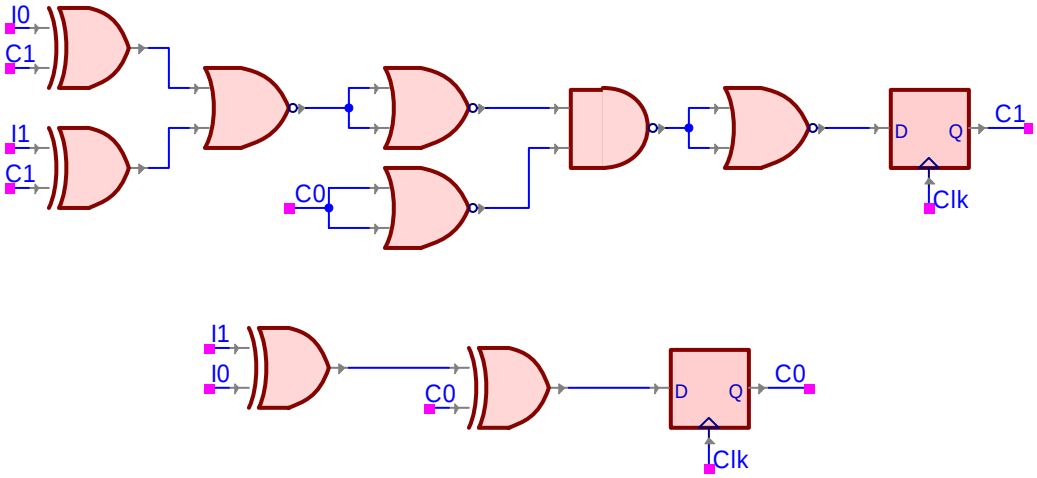
$C_0 = I_1I_0'C_0' + I_1'I_0C_0' + I_1I_0C_0 + I_1'I_0'C_0$



(c) NAND/NOR/XOR implementation:

$C_1 = [(I_0 \text{ xor } C_1) + (I_1 \text{ xor } C_1)]C_0'$

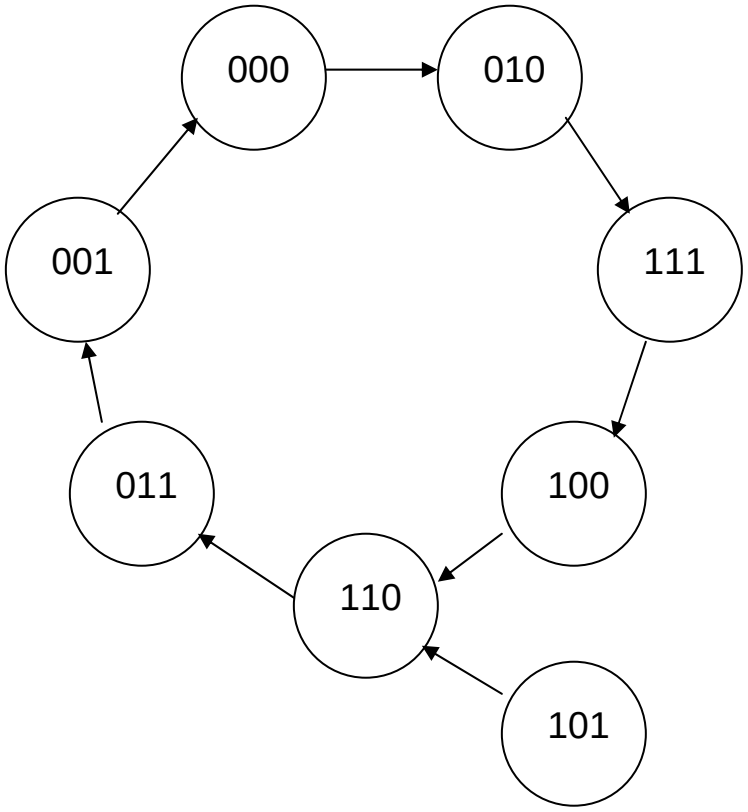
$C_0 = (I_1 \text{ xor } I_0) \text{ xor } C_0$



Exercise 7.2

We begin designing the three bit counter by first drawing the state transition graph, then the state transition table. Since state 101 is not part of the sequence, we need to make sure that it has a transition out to a state that is part of the sequence.

State diagram:



State transition table:

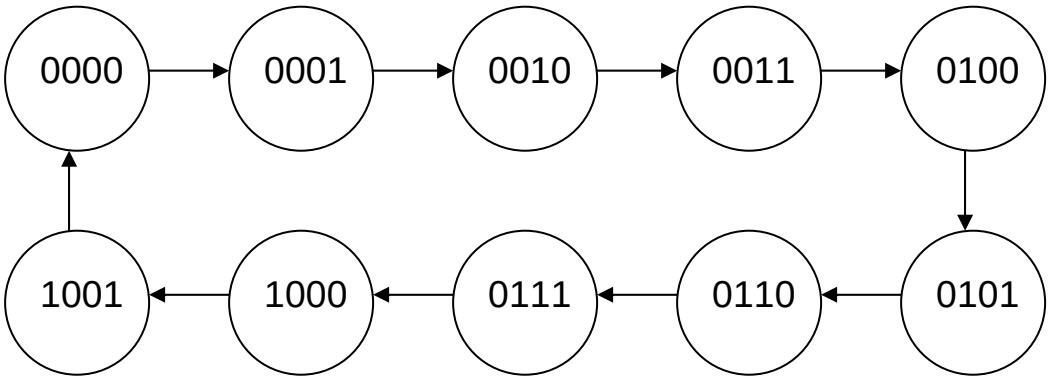
C	B	A	C+	B+	A+
0	0	0	0	1	0
0	0	1	0	0	0
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	1	1	0
1	1	0	0	1	1
1	1	1	1	0	0

Exercise 7.3

(a) 3-bit counter:

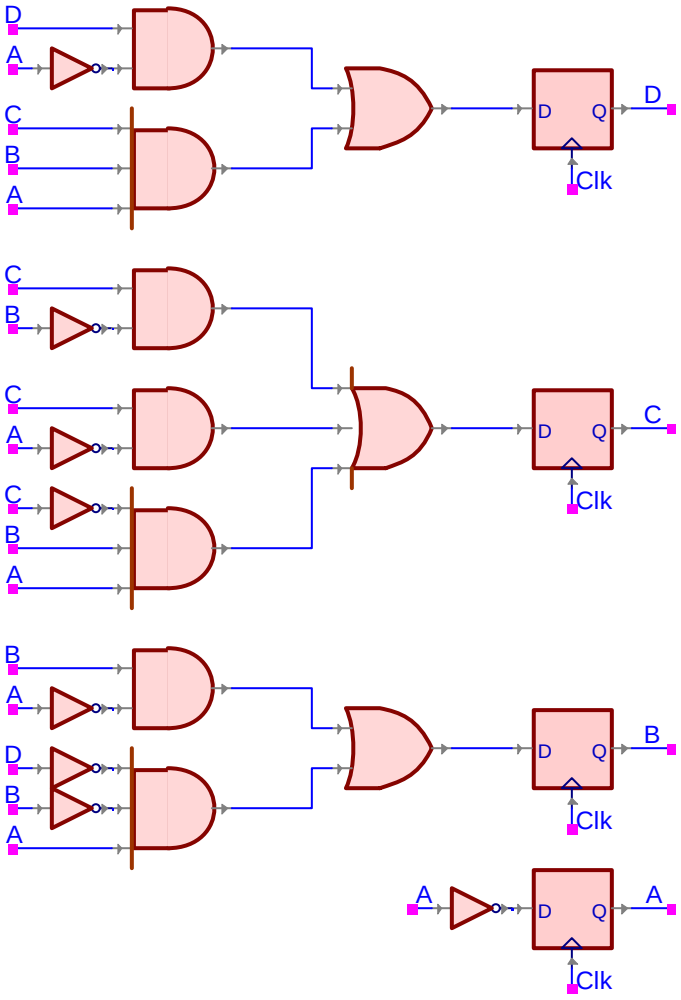
D	C	B	A	D+	C+	B+	A+
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	0	1	1
0	0	1	1	0	1	0	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	1	1
0	1	1	1	1	0	0	0
1	0	0	0	1	0	0	1
1	0	0	1	0	0	0	0

State transition table



State diagram

(b) Circuit schematic:



(c) Expressions for a self-starting counter:

D+ = DA’ + DCB’ + DC’B + D’CBA

C+ = CB’ + CA’ + C’BA

B+ = BA’ D’B’A + CB’A

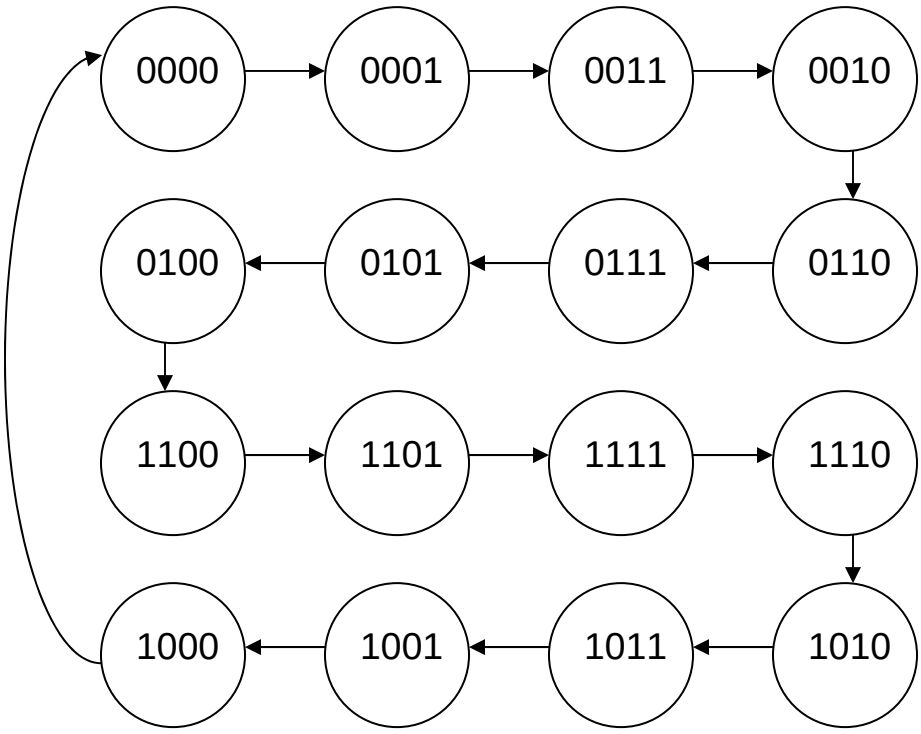
A+ = A’

Exercise 7.4

(a) Gray-code counter:

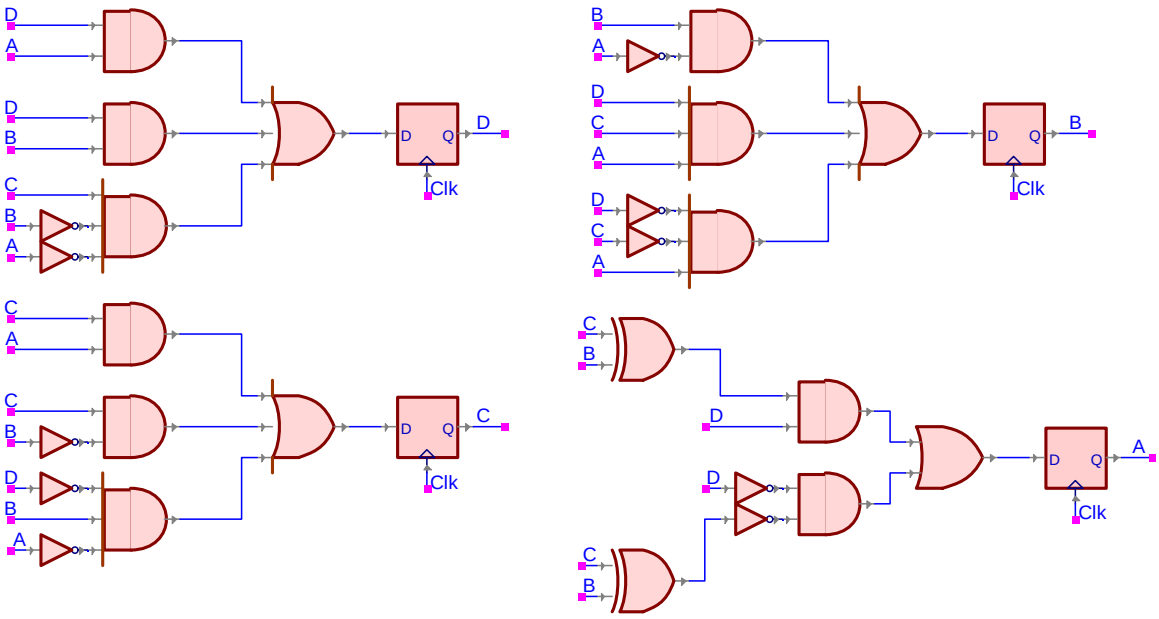
D	C	B	A	D+	C+	B+	A+
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	1
0	0	1	0	0	1	1	0
0	0	1	1	0	0	1	0
0	1	0	0	1	1	0	0
0	1	0	1	0	1	0	0
0	1	1	0	0	1	1	1
0	1	1	1	0	1	0	1
1	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0
1	0	1	0	1	0	1	1
1	0	1	1	1	0	0	1
1	1	0	0	1	1	0	1
1	1	0	1	1	1	1	1
1	1	1	0	1	0	1	0
1	1	1	1	1	1	1	0

State-transition table



State Diagram

(b) Circuit Schematic:



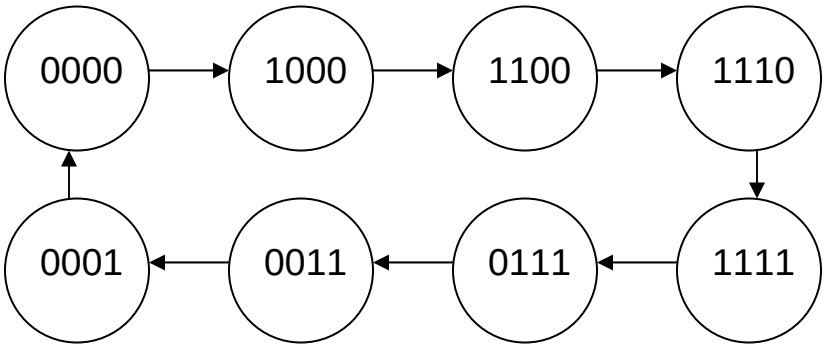
(c) We don't need to worry about the self-starting because every possible state is part of the sequence that the Gray-code counter goes through. So no matter what state the counter starts out in, it will always count correctly.

Exercise 7.5

4-bit Johnson Counter:

D	C	B	A	D+	C+	B+	A+
0	0	0	0	1	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	-	-	-	-
0	0	1	1	0	0	0	1
0	1	0	0	-	-	-	-
0	1	0	1	-	-	-	-
0	1	1	0	-	-	-	-
0	1	1	1	0	0	1	1
1	0	0	0	1	1	0	0
1	0	0	1	-	-	-	-
1	0	1	0	-	-	-	-
1	0	1	1	-	-	-	-
1	1	0	0	1	1	1	0
1	1	0	1	-	-	-	-
1	1	1	0	1	1	1	1
1	1	1	1	0	1	1	1

State Transition Table



State Diagram

If we assume that the states that are not part of the sequence will never be entered, we can use the outputs for those states as don't-cares, which will make our implementation less complex. The following are the next-state expressions:

D+ = CA' + BA'

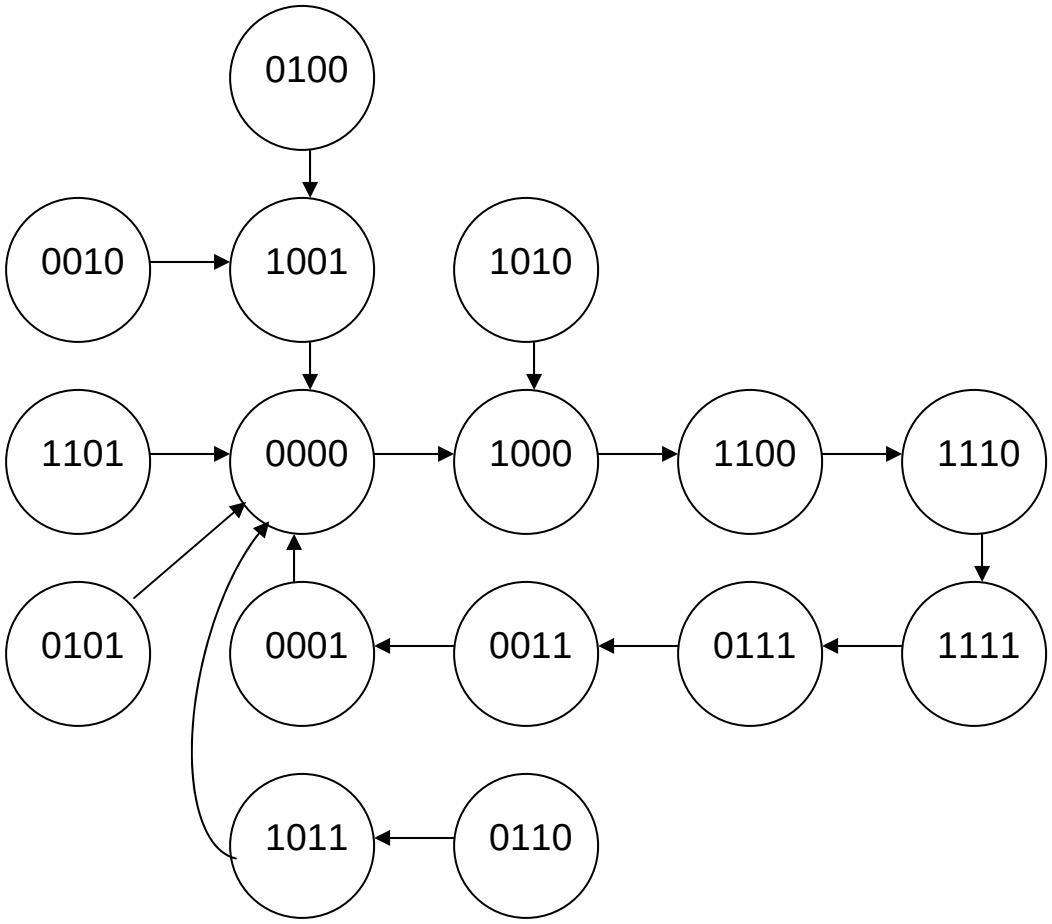
C+ = DB'A' + DCB

B+ = CB + CA'

A+ = D'B + CB

Exercise 7.6

The solution for Exercise 7.5 is self-starting. All of the states eventually lead into the counting sequence. The following state diagram illustrates the transitions from each of the 16 possible states:



State diagram for the 4-bit Johnson counter

Exercise 7.7

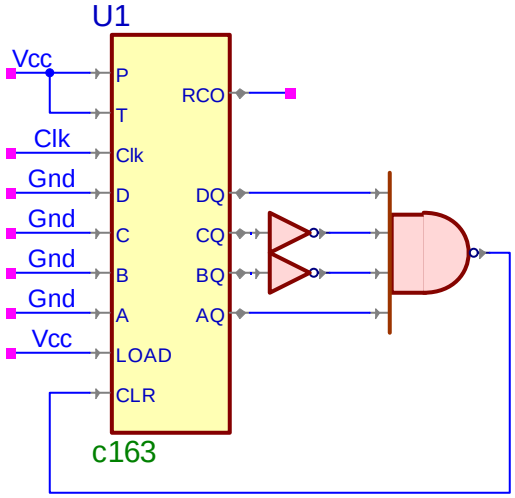
An asynchronous load or reset signal may be acceptable when we need to immediately change the contents of a register without waiting for an extra clock cycle. This could be important in communication between two FSMs or counters. Of course, we would have to make sure that asynchronous behaviors in two communicating FSMs were not inadvertently connected to create an asynchronous feedback cycle (that is, a loop not interrupted by waiting for a clock signal).

Exercise 7.8

The Load signal on the counter-chip tells the counter to load the value that is on the parallel load inputs. We can see from the timing diagram that when the counter reaches 1111_2, the compliment of the RCO signal causes Load to be driven low and the counter circles back to the value at the parallel load inputs. If we take the RCO compliment and AND it with an active-low reset signal, we will be able to force the counter to start counting in its first valid state whenever the reset is driven low. Since we are using an AND gate with the reset and RCO signals, the functionality of the counter will be exactly the same but we will be able to start out in the first valid count sequence whenever we wish. One assumption that we must make about the reset signal is that it is asserted for at least one clock cycle. If it doesn't, the counter may behave unpredictably.

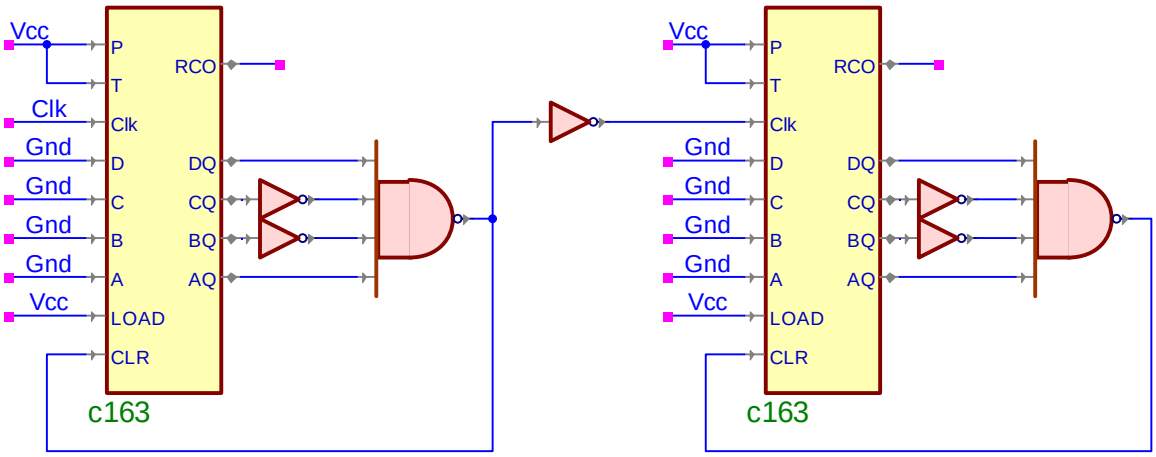
Exercise 7.9

To build a BCD counter out of a 163 counter component, we need to have the counter roll back to zero when 1001_2 is reached. We can accomplish this by checking the value of the parallel outputs and when it reaches 1001_2, drive the CLR input low, which will cause the next output to be zero. The following circuit diagram illustrates this:



1-Digit BCD Counter

We can add another bit by using two 1-digit BCD counters. The second counter will be for the most significant bit. This counter should only be incremented every time the least significant counter rolls over. We can achieve this functionality by using the same signal that we use to roll over the low order counter as a clock signal for the high order counter. We have to invert the output of it so that when the low order counter rolls over, the signal will go high for one clock cycle causing the high order counter to increment itself. This can be generalized to an n-digit BCD counter by using n BCD counters and using the CLR output from the n-1 counter as the n counter Clk. The following circuit schematic illustrates this with a 2-digit BCD counter:

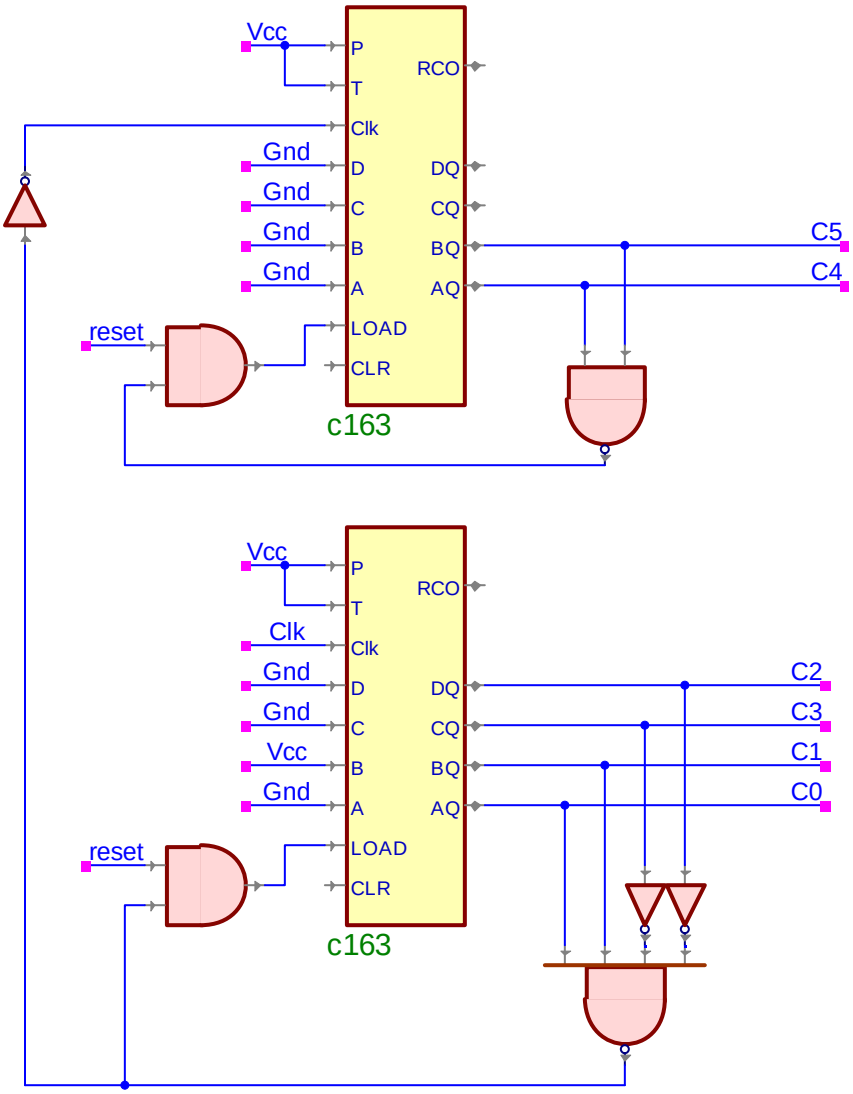


Low order BCD counter

High order BCD counter

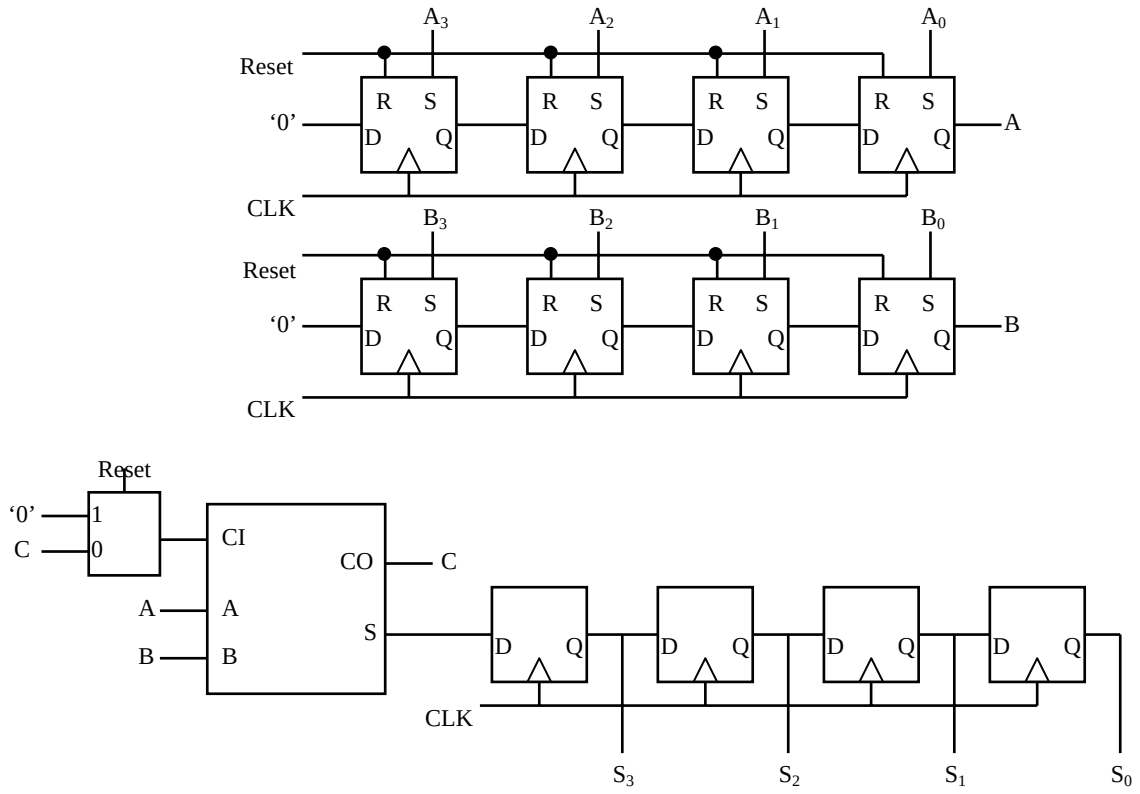
Exercise 7.10

So, essentially what we have is two cutoff/offset counters that are set to roll over at their corresponding correct values. When they roll over, they load the starting value from the parallel inputs and continue counting. C5-C0 are the bit values for the 6-bit output. The reset is also tied to the load, which will cause the counters to reset to their start value. The following circuit schematic illustrates a 6-bit BCD counter:



Exercise 7.11

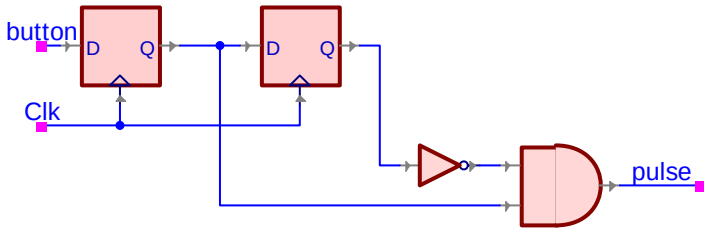
(a) Circuit schematic for the 4-bit serial adder:



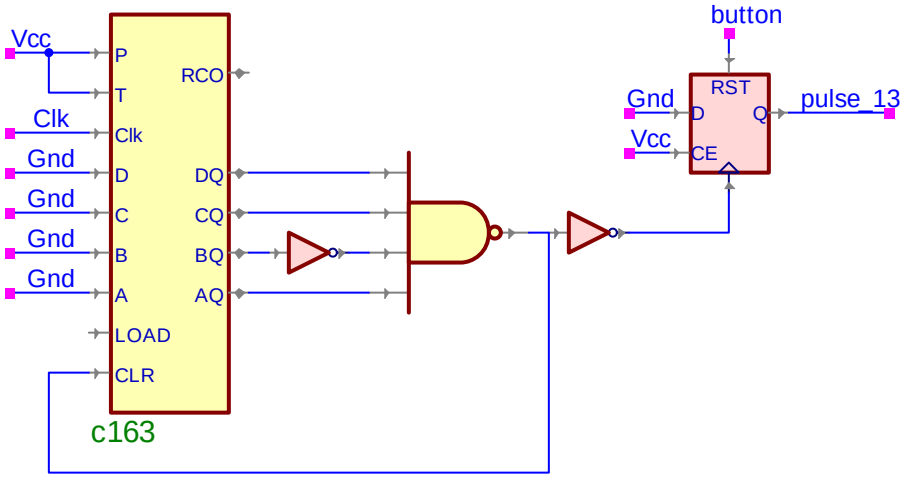
(b) The only control signal in this implementation is Reset. When the Reset signal is asserted, each bit of the two operands is loaded into the registers in parallel. When the reset signal goes low, each value is shifted to the right on every clock cycle. The Reset signal also acts as the control for a 2:1 multiplexor, which loads the first carry-in with a zero. The carry-out feeds back into the carry-in, which causes it to be used for the subsequent add. After the entire 4-bit value has been shifted through the adder, the result can be accessed on the output of each sum register.

Exercise 7.12

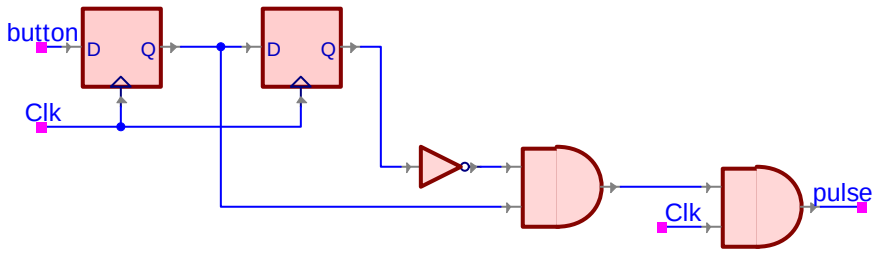
(a) The way to guarantee that the clock signal only goes high for one clock cycle is to only assert it when the input signal is changing from low to high. We can do this by using a 2-bit shift register and checking the value of each bit. When the second one is low and the first one is high, we assert the output signal.



(b) The simplest way to implement this design is to use a register which loads a one into it when it is reset. The button signal is used as the reset for this register. The clock input to this register is tied to the output signals from the 163 counter in a manner that will cause the clock to cycle once on the 13th cycle and load a zero.

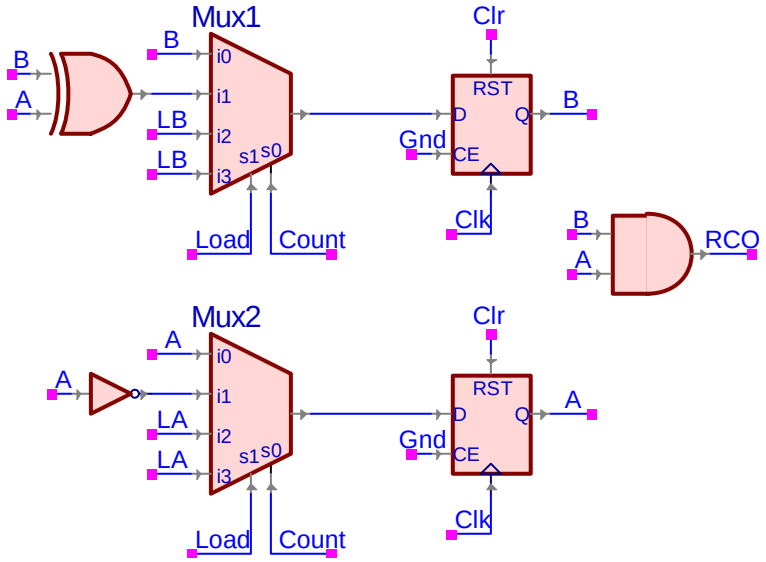


(c) A system that will provide a single clock step can be implemented with an extension of the design in part a. The portion of the design from part a guarantees that the output will be exactly one clock cycle. ANDing this output with the clock signal will cause it to cycle from low to high and back down to low again.



Exercise 7.13

The first piece that we need to design is the next state logic. 4:1 multiplexors work well to implement this functionality. We will use load and count as the control signals for these multiplexors. When load is zero, we are either counting or not counting. In the case that we are not counting, the multiplexor just needs to feed back the current value that the flip-flop has in it. If we are counting, the multiplexor for the B flip-flop is simply the sum of the A and B flip-flops, which can be implemented with an 2-input XOR. The mutliplexor for the A flip-flop is just the current output inverted when we are counting. If the load signal is high, the input to both multiplexors is simply their respective load values. To achieve the clear functionality, we tie the CLR signal to the reset input of the flip-flops and tie the CE input to the flip-flops to Gnd. This will reset both of the flip-flops to zero when the CLR signal is asserted and it will have precedence over both the count and the load signals. Lastly, the RCO signal is just the outputs of the A and B flip-flops ANDed together. The resulting circuit schematic looks as follows:



2-bit up-counter

Exercise 7.14

The biggest problem associated with using both positive and negative edge-triggered flip-flops is that they latch their inputs at different times and the output for each one is not valid at the same time as the output of the other one. This can cause serious problems if the combinational logic which computes the next state of one flip-flop relies on the output of the other flip-flop. For example, if we have flip-flop A and B, each of which is triggered on different clock edges, the combinational logic of A may be half way done computing the next state for A and the output of B changes. This would cause unpredictable behavior in the combinational logic for A.

Exercise 7.15

Using both flip-flops clocked on both clock edges changes timing constraints dramatically. Since we are assuming we are not keeping negative and positive edge-triggered flip-flops segregated, instead of having the single constraint:

$$T_{period} > T_{pd} + T_{comb} + T_{su}$$

We will now have two constraints, one for each of the two parts of the clock cycle, one for when the clock is high and one for when the clock is low:

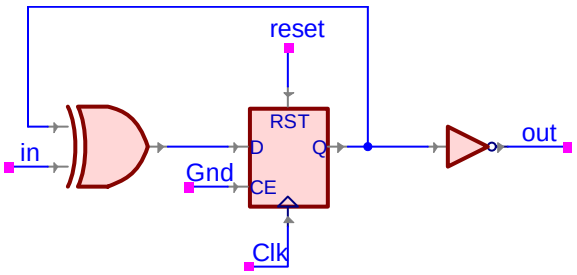
$$T_{high} > T_{pd} + T_{comb\text{-}high} + T_{su}$$
$$T_{low} > T_{pd} + T_{comb\text{-}low} + T_{su}$$

Therefore, the constraints are much tighter as T_{high} and T_{low} add up to T_{period}. If the clock's duty-cycle is 50%, then things can be made to be pretty similar to a single clock-edge system if the combinational logic delays can be evenly split between the two parts. If the clock duty-cycle is 90%, then one constraint will be very tight (10% of the clock period) while the other will be almost a full period (90% to be exact).

There is rarely an advantage to clocking a single system with both clock edges as it makes it more difficult for the designer to decide where to place combinational logic.

Exercise 7.16

Since we already have an odd parity checker, designing an even parity checker is trivial. An even parity checker is just the inverse of an odd one. So we can simply add an inverter to the output of the odd parity checker. The resulting circuit schematic looks as follows:

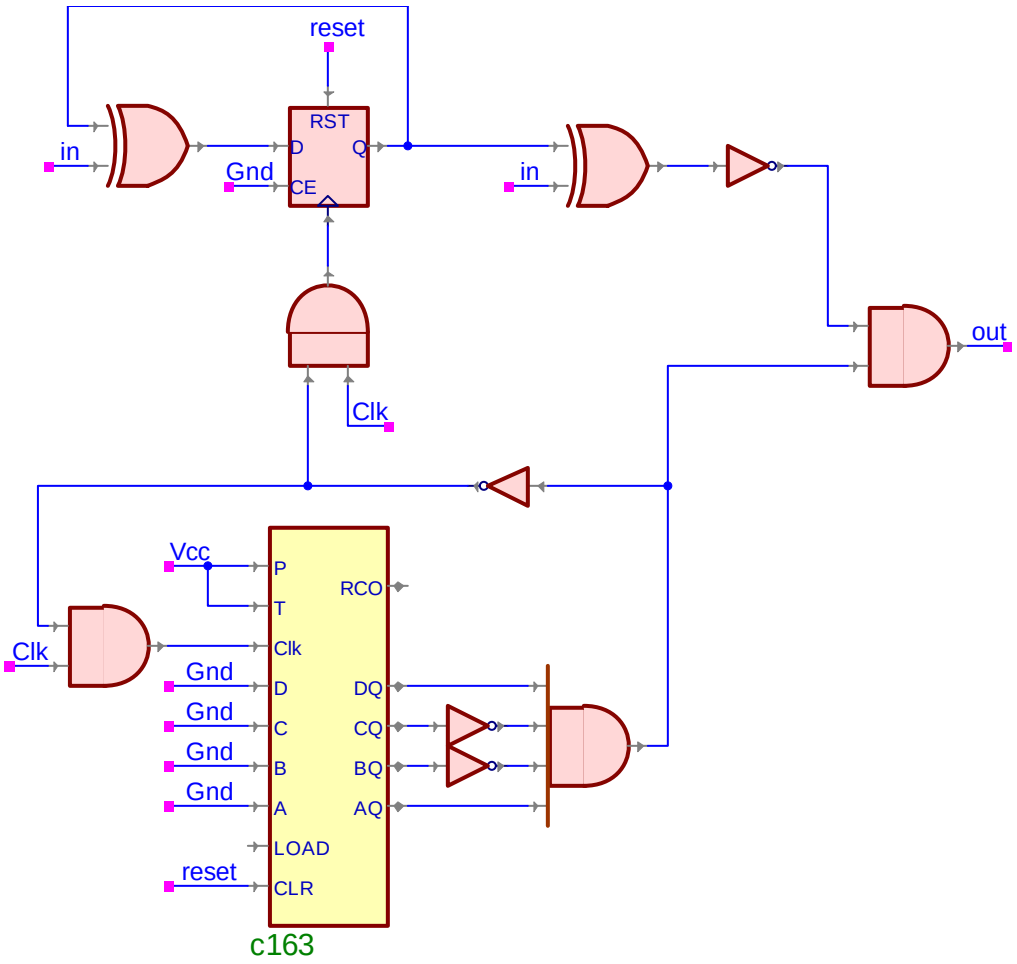


Even parity checker

Exercise 7.17

(a) It is possible to designe a state diagram with a small number of states to implement this 9th bit parity checker system. The system doesn't need to track all possible combinations of 9 bits. If it did, the system would have 2^{9th} or 512 states. In reality, the state diagram only needs to have 9 states. The first 8 states would count how many bits the input has seen and update the parity checker for each one. The last state would compare the current parity checker value with the 9th bit and assert OK or ERROR.

(b) The way that this design works is when the system is reset, the counter clears its contents out to be zero. Each clock cycle the counter counts up one and the parity checker reads in a new value. When the counter reaches 9, the clock inputs to the counter and flip-flop are disabled, the current value in the flip-flop is XORed with the 9th bit of the input and inverted. If this value is a one, it means that the parity bit and 9th input bit are the same and the output is asserted.



9-bit serial parity checker

Exercise 7.18

Let L be the number of flip-flops, J be the number of inputs, and K be the number of outputs.

(a) The maximum number of states is: $2^L = 8$

The minimum number of states is: $2^L = 8$

(b) The maximum number of transitions starting in a particular state is: $2^J = 4$

The minimum number of transitions starting in a particular state is: $2^J = 4$

(c) The maximum number of transitions that can end in a particular state is: $2^J * 2^L = 32$

The minimum number of transitions that can end in a particular state is: 0

(d) The maximum number of binary patters that can be displayed on the outputs is:

$\text{Min}(2^K, 2^J * 2^L) = 32$

The maximum number of binary patters that can be displayed on the outputs is:

1

Exercise 7.19

Let L be the number of flip-flops, J be the number of inputs, and K be the number of outputs.

(a) The maximum number of states is: $2^L = 32$

The minimum number of states is: $2^L = 32$

(b) The maximum number of transitions starting in a particular state is: $2^J = 8$

The minimum number of transitions starting in a particular state is: $2^J = 8$

(c) The maximum number of transitions that can end in a particular state is: $2^J * 2^L = 256$

The minimum number of transitions that can end in a particular state is: 0

(d) The maximum number of binary patters that can be displayed on the outputs is:

$\text{Min}(2^K, 2^L) = 32$

The maximum number of binary patters that can be displayed on the outputs is:

1

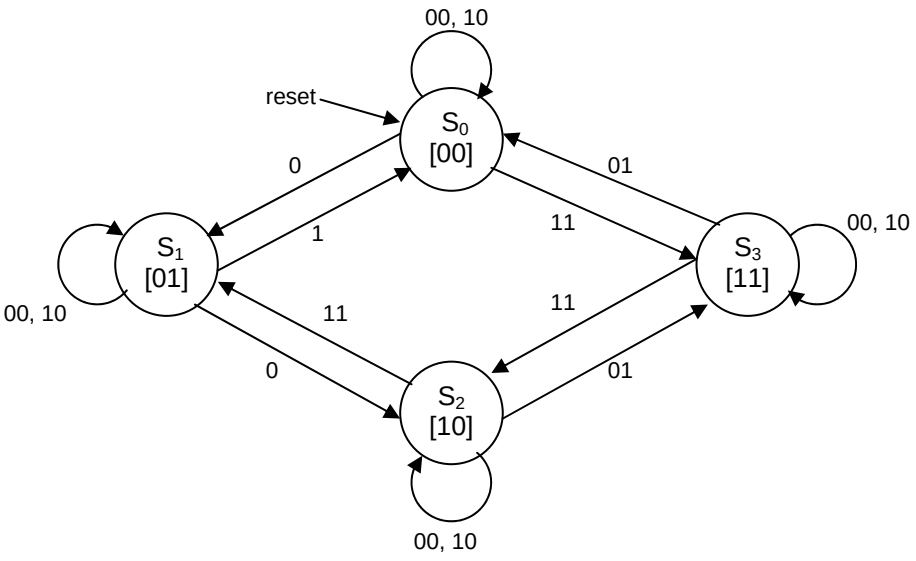
Exercise 7.20

The counter circuit in this exercise has 2 flip-flops, which tells us that the state diagram will have four states. We will assume that we don't care what the transitions are if the input C is 0. This leaves us with a total of eight entries in our truth table that we care about. To figure out these values we will trace through the circuit. Once we have the next state values, we can draw our state diagram.

D	C	S_1	S_0	S_1+	S_0+
0	0	0	0	-	-
		0	1	-	-
		1	0	-	-
		1	1	-	-
0	1	0	0	0	1
		0	1	1	0
		1	0	1	1
		1	1	0	0
1	0	0	0	-	-
		0	1	-	-
		1	0	-	-
		1	1	-	-
1	1	0	0	1	1
		0	1	0	0
		1	0	0	1
		1	1	1	0

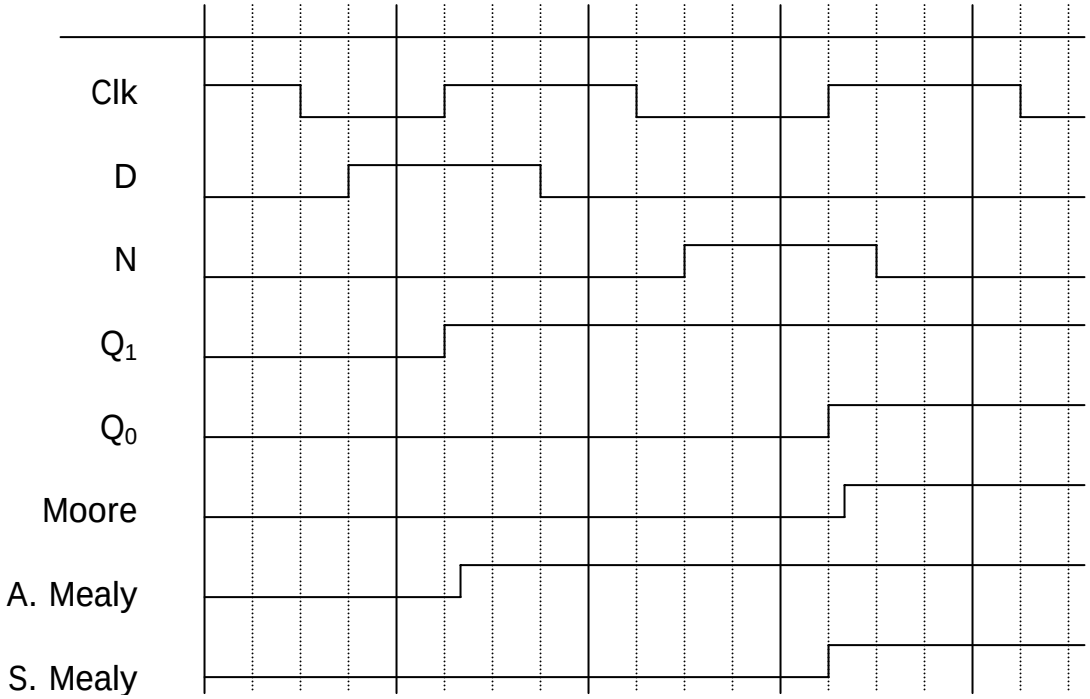
Truth table

As we can see from this truth table the forward sequence of the counter is: 00, 01, 10, 11, and the reverse sequence is: 00, 10, 01, 11. The transitions are labeled DC, where D is the count direction input and C is the count input. The state diagram for this design looks as follows:



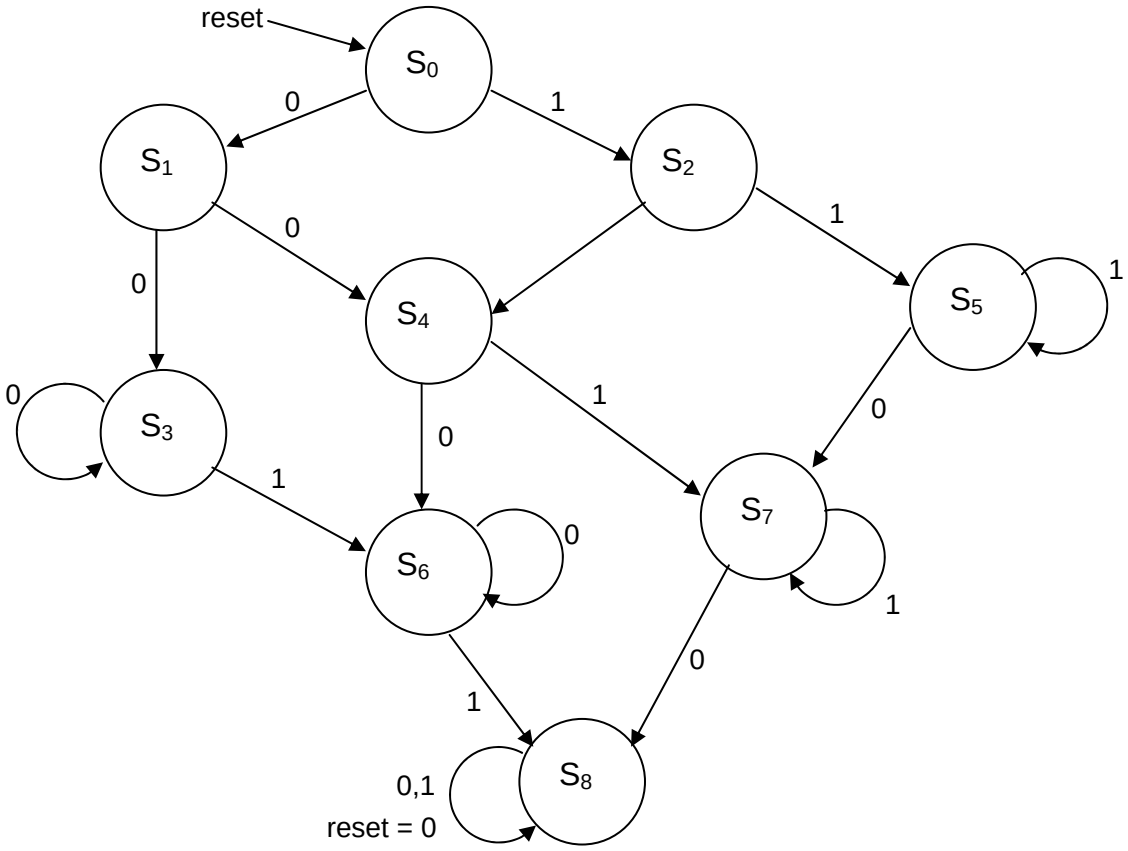
Exercise 7.21

A dime and a nickel are asynchronously inserted into the vending machine. At the first rising edge of the clock, Q_1 latches a 1. At the second rising edge of the clock, Q_0 latches a 1 from the nickel. One gate delay later the Moore machine's Open is asserted. The asynchronous Mealy machine's Open is asserted one gate delay after the nickel is inserted. The synchronous Mealy machine's Open is asserted when the Open register latches a 1. We can see that the synchronous Mealy machine is the only implementation that exhibits the behavior we are looking for. The timing diagram looks as follows:



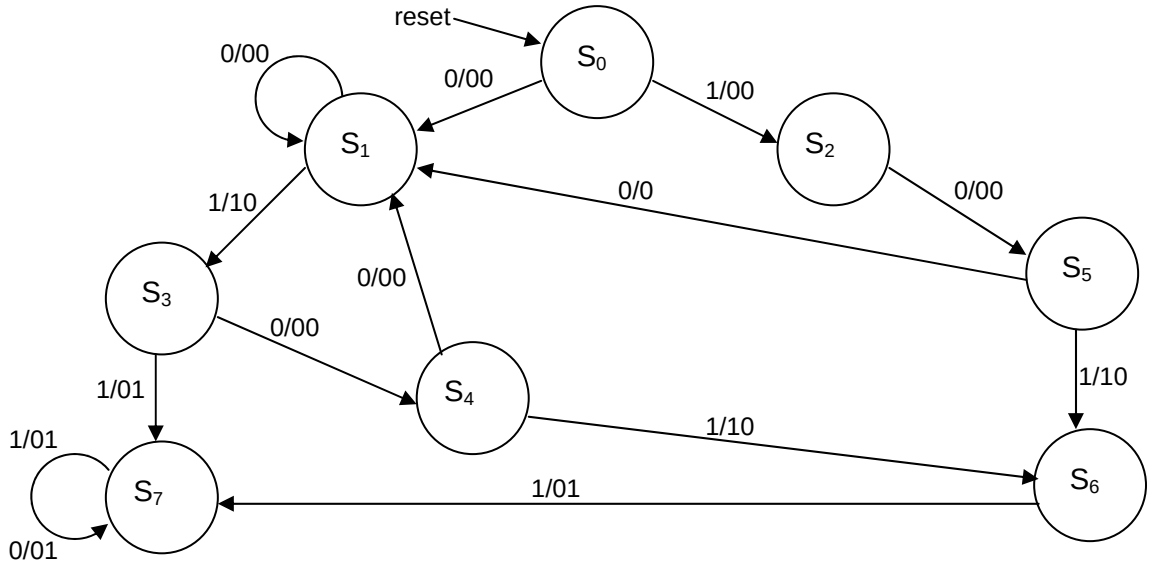
Exercise 7.22

The state machine needs to be designed so that we must see at least 2 zeros in a row and 2 ones in a row before the output is asserted. So the output is 0 in all states except the final state, S_8. The state diagram looks as follows:



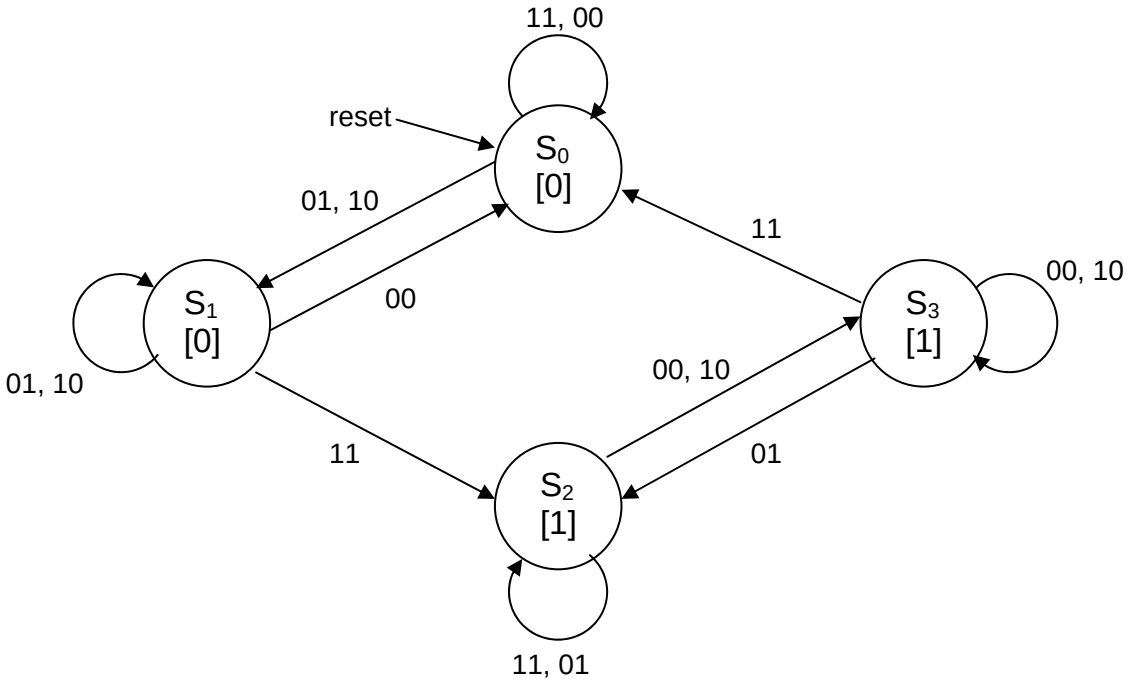
Exercise 7.23

This design can be implemented with 8 states because the system only needs to keep track of the last 3 input bits. The transitions are labeled X/Z_1Z_2, where X is the current input and Z_1 and Z_2 are the current outputs. State S_7 is the trap state that the system goes into if it sees the sequence 011. The resulting state diagram looks as follows:



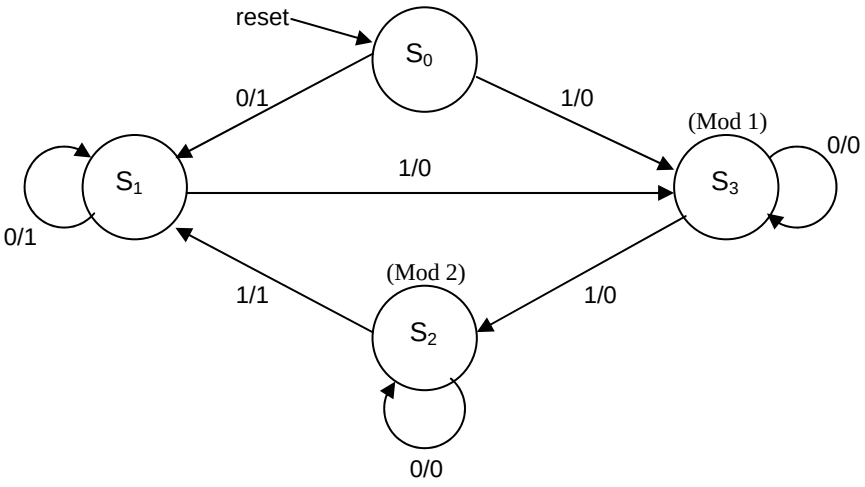
Exercise 7.24

Since this system only needs to remember 2 bits of the input sequence, the state diagram only needs four states. The input is shown next to each transition and the output is shown in brackets within each state. The resulting state diagram looks as follows:

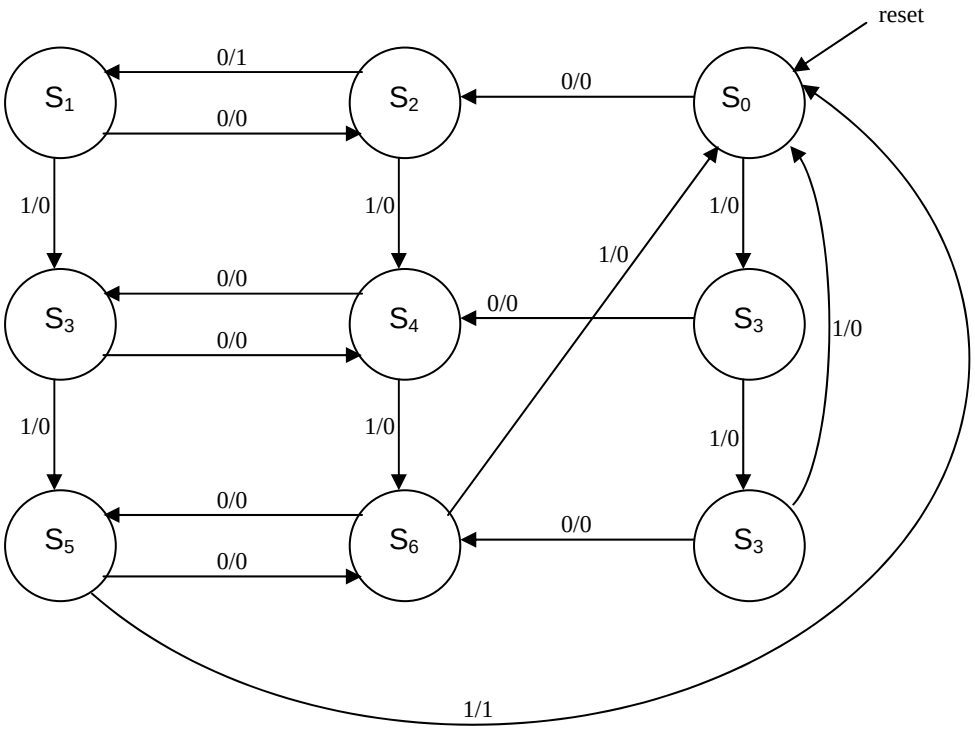


Exercise 7.25

(a) This system can be implemented with four states. Essentially, all it does is count how many ones have been received. State S_1 is the only state which has the output asserted. The transitions are labeled X/Z, where X is the input and Z is the output. The resulting state diagram looks as follows:

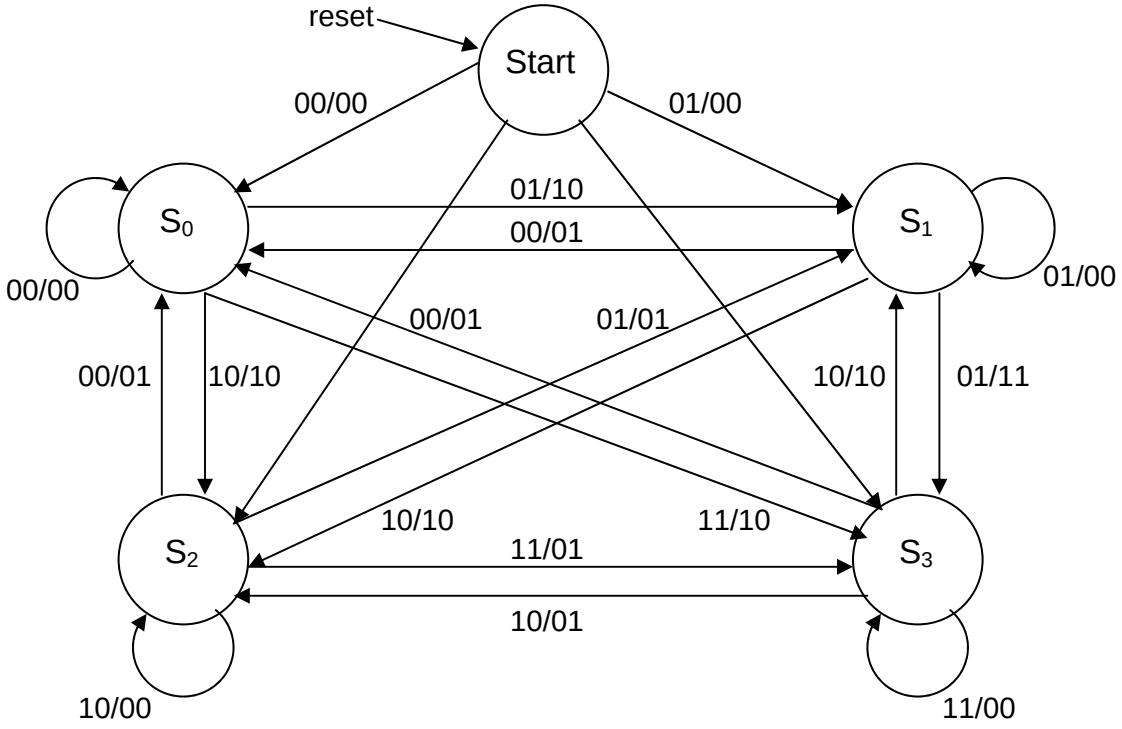


(b) This system is similar to the system in part a except that it needs to keep track of whether or not the number of zeros received is even. This basically doubles the number of states plus it needs an extra state for the start state giving a total of nine. The transitions are labeled X/Z, where X is the input and Z is the output. The resulting state diagram looks as follows:

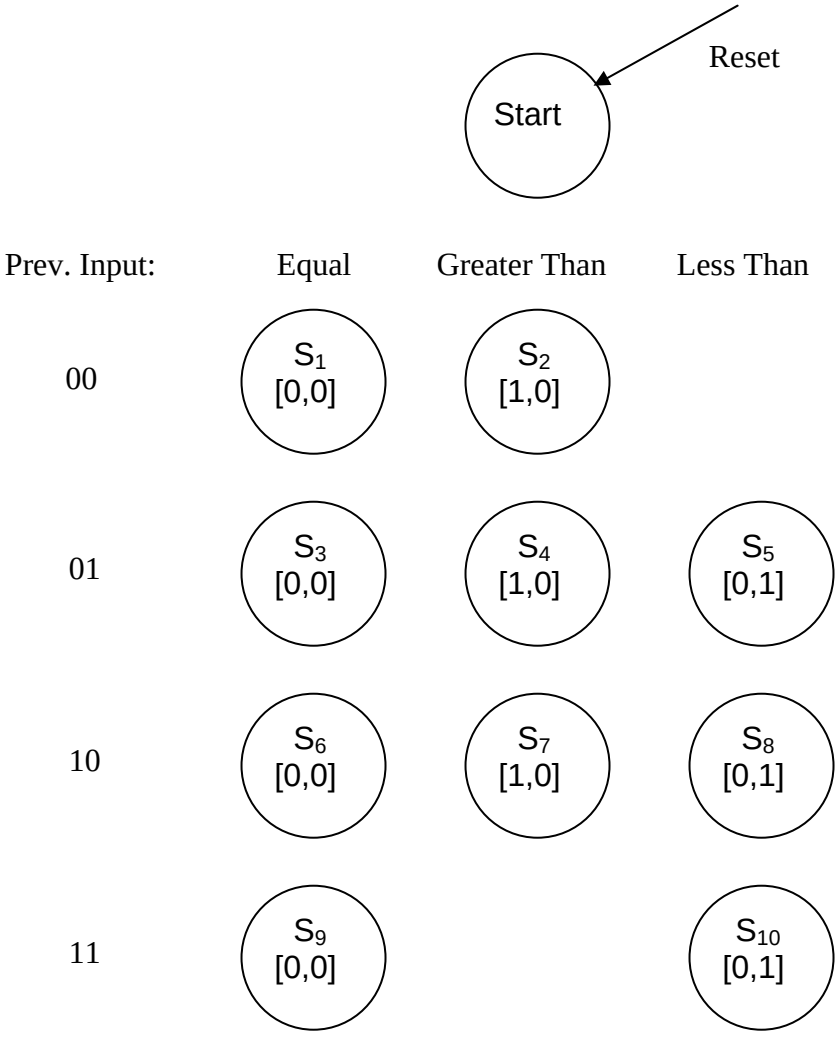


Exercise 7.26

(a) The Mealy machine for this system has four states for each of the combination of the previous input and the current input plus one for the start state. The transistions are labeled X_1X_2/Z_1Z_2, where X_1 and X_2 are the inputs and Z_1 and Z_2 the outputs. The resulting state diagram looks as follows:

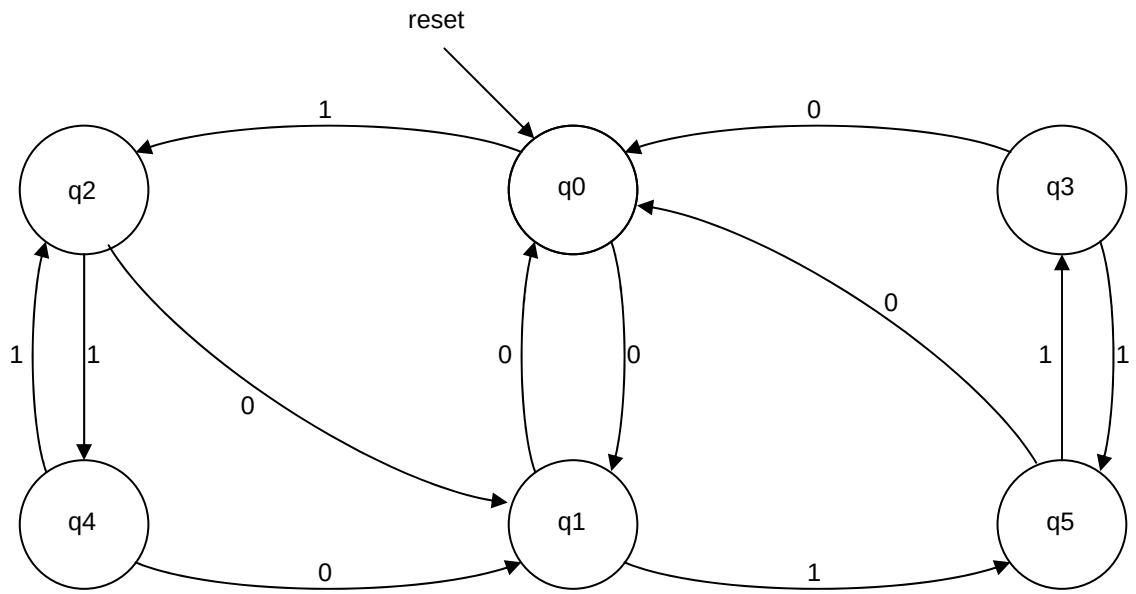


(b) Since the outputs of a Moore machine only depend on the current state, the Moore-implementation of this system is much more complicated. In order to simplify the diagram the transition arrows have been left out. Assume a transition arrow from each of the states to each state S_1 through S_{10} (this corresponds to 90 transitions, hence, the reason they were omitted). Each tier of the state diagram represents the previous value seen. The state diagram looks as follows:



Exercise 7.27

States q0, q2, and q4 represent an even number of 0's, with states q0, q1, and q2 representing an even number of zeros. The output would be one in states q1 and q3. The state diagram looks as follows:



Exercise 7.28

There are six different states that the lights can be in. The following chart illustrates one complete cycle of the traffic lights:

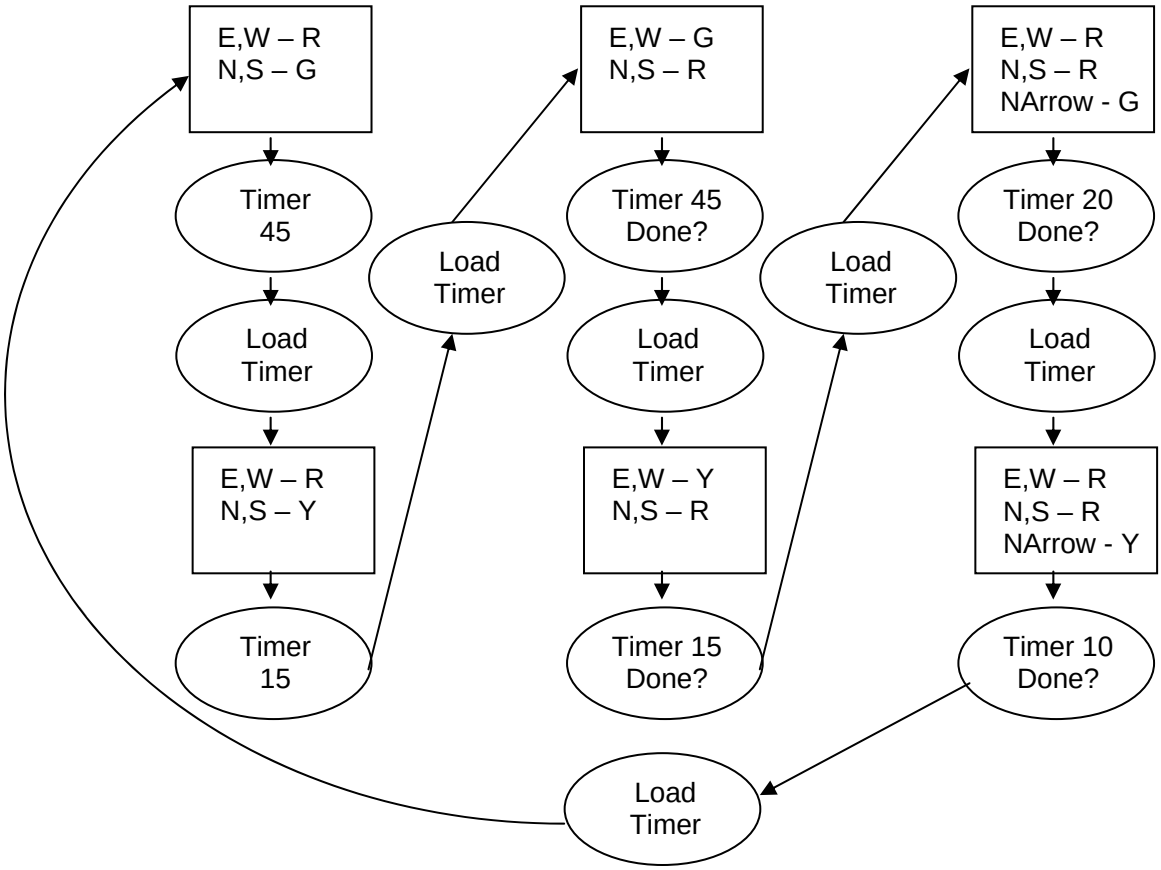
East	Red	Red	Green	Yellow	Red	Red	Red
West	Red	Red	Green	Yellow	Red	Red	Red
South	Green	Yellow	Red	Red	Red	Red	Green
North	Green	Yellow	Red	Red	G Arrow	Y Arrow	Green
	45	15	45	15	20	10	
	45	60	105	120	140	150	

The input signals to this system include:

Timer 45 done, Timer 15 done, Timer 20 done, Timer 10 done, and Reset

The output signals include:

East Red, East Yellow, East Green, West Red, West Yellow, West Green,
South Red, South Yellow, South Green, North Red, North Yellow, North Green,
N Arrow Yellow, N Arrow Green,
Load 45 Timer, Load 15 Timer, Load 20 Timer, Load 10 Timer



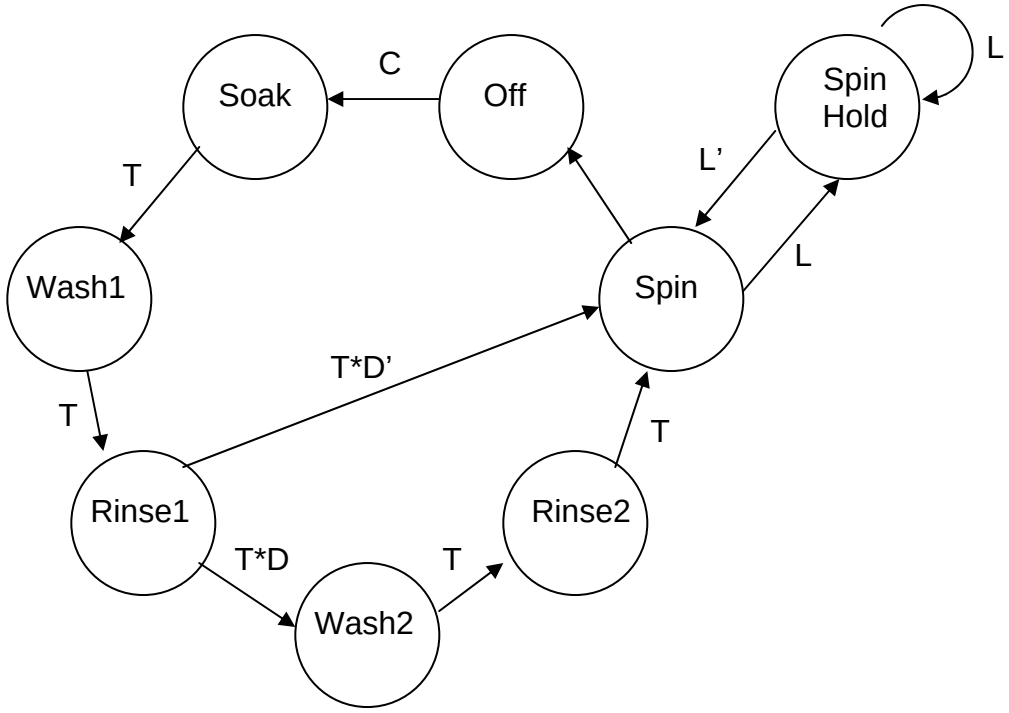
Traffic Controller State diagram

Exercise 7.29

The inputs for the washing machine design are as follows:

Coin = C, Timer = T, Double Wash = D, Lid Open = L

The state diagram looks as follows:



Exercise 7.30

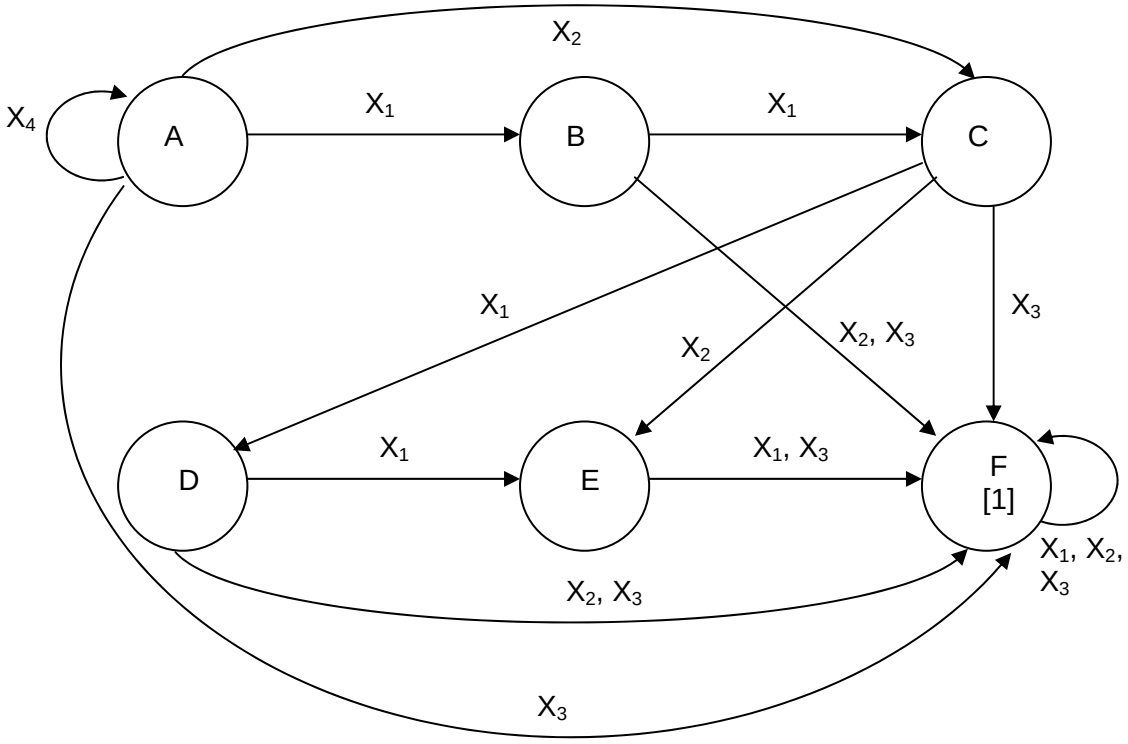
The candy vending machine has a total of 6 states. F is the only state in which the output is enabled. The states are as follows:

A = Reset, B = 5 cents, C = 10 cents, D = 15 cents, E = 20 cents, F = 25 cents or more

The inputs are as follows:

X_1 = 5 cents, X_2 = 10 cents, X_3 = 25 cents, X_4 = reset

The state diagram is as follows:



Exercise 7.31

The newspaper vending machine has a state that represents how much money has been deposited so far. Each state has three output transitions, which correspond to a nickel, dime and quarter being deposited. The transitions are labeled X/Z, where X represents the input and Z represents the output. If there is no input on a transition it is labeled X, similarly, if there is no output signals asserted it is labeled Z. The only state that a newspaper is released in is state q50. If the user deposits more than 50 cents, the system jumps to a ref state where the money deposited is refunded. The resulting state diagram is as follows: