DIGITAL DESIGN PROBLEM

Problem 1 [0.5 points]
Implement a magnitude comparator circuit which compares two-bit binary numbers $A=A_1A_0$ and $B=B_1B_0$, and provides 3 outputs (A>B, A=B, A<B), using three 8-to-1 multiplexers and no additional gates.

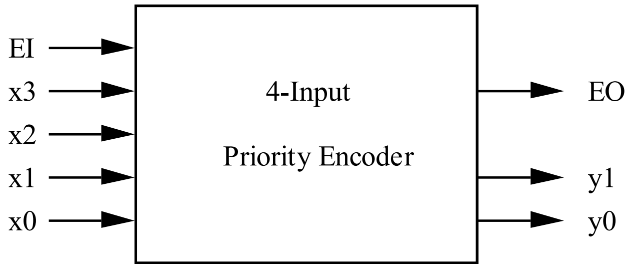
Solution:
There could be many different solutions to this problem. Suppose A_1, A_0, and B_1 are the select inputs to all three 8-to-1 multiplexers, we can have the following true table.

A_1	A_0	B_1	A = B	A > B	A < B
0	0	0	B_0'	0	B_0
0	0	1	0	0	1
0	1	0	B_0	B_0'	0
0	1	1	0	0	1
1	0	0	0	1	0
1	0	1	B_0'	0	B_0
1	1	0	0	1	0
1	1	1	B_0	B_0'	0

Implementation of this truth table is straightforward. The columns under A=B, A>B, A<B represent the inputs to each 8-to-1 mux starting from the 000 input to the 111 input. For example, A=B is the output of the mux with $I0=B_0'$, $I1=0$, $I2=B_0$, $I3=0$, $I4=0$, $I5=B_0'$, $I6=0$, $I7=B_0$.

Problem 2 [1.5 points]
A 4-input priority encoder with enable is shown below. When EI is 0, all outputs are 0. When EI is 1,

i) and all 4 xi inputs are 0, EO is 1 and y1=y0=0
ii) and not all 4 xi inputs are 0, EO is 0 and y1 and y0 are the binary encoding for the highest priority xi input that is 1 (For example, x3 is highest priority whose binary encoding is 11. x1 is lowest priority with binary encoding 01 and so on).



(a) [0.4 points] Draw a two-level NAND-gate implementation of this priority encoder. Assume both true and complementary inputs are available.

(b) [0.4 points] Draw a two-level NOR-gate implementation of this priority encoder. Assume both true and complementary inputs are available.

(c) [0.7 points] Draw a diagram of an 8-input priority encoder of the same type using two 4-input priority encoders and a few logic gates.

Solutions:

The truth table for this encoder is

EI	x3	x2	x1	x0	EO	y1	y0
0	-	-	-	-	0	0	0
1	0	0	0	0	1	0	0
1	1	-	-	-	0	1	1
1	0	1	-	-	0	1	0
1	0	0	1	-	0	0	1
1	0	0	0	1	0	0	0

(a) EO = EI x3’ x2’ x1’ x0’
y1 = EI x3 + EI x2
y0 = EI x3 + EI x2’x1
Two 2-input NANDs, one 3-input NAND, one 5-input NAND, and one inverter

(b) EO = (EI’ + x3 + x2 + x1 + x0)’
y1 = EI (x3 + x2)
y0 = EI (x3 + x2’)(x3 + x1)
Four 2-input NORs, one 3-input NORs, one 5-input NOR, and one inverter

(c) This requires two of the 4-input priority encoders, with EO of the higher priority encoder connected to EI of the lower priority encoder, and two 2 input OR gates to OR the y1 and y0 outputs of the two priority encoders.

Problem 3 [2.0 points]

(a) [0.5 points] A basic lawn sprinkler system waters the lawn at 8am, once every 3 days. Its operation can be modeled as a Moore-style finite state machine, where a clock period is one day, and a binary output W indicates whether the sprinkler is activated (W=1) or not (W=0) on that day. Show the transition diagram for this sprinkler control system.

(b) [0.5 points] An advanced sprinkler system has a rain sensor. The presence of rain during a particular day is indicated by the binary input R (with values 0/1 indicating No/Yes_rain). This sprinkler waters the lawn once every 3 days, assuming there has been no rain during previous 2 days. Show the transition diagram (Moore-style) for this advanced sprinkler system.

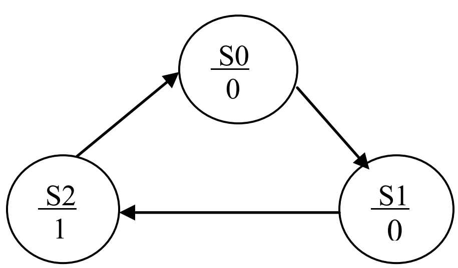
(c) [1.0 points] Implement the transition diagram of a sprinkler system in part (b) using a 4-bit counter with parallel load capability and external gates. The counter is specified by its operation table shown below. Use the most appropriate state assignment for this counter implementation. (*Implementation* means showing the block diagram with clearly labeled inputs/outputs, and derivation of the Boolean equations for control/data inputs of the counter).

Counter operation table:
(All functions are synchronous with a clock)

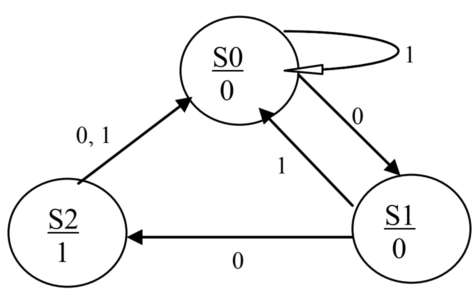
Clear	***Load***	***Count***	***Function***
0	X	X	Clear
1	0	0	No change
1	1	X	Load
1	0	1	Count up

Solutions:

(a)



(b)



(c) Use two least significant bits of the counter q_1q_2 to encode 3 states. From the transition diagram: transitions S0→S1 and S1→S2 can be implemented via counting function. All other transitions (to state S0 ~ 00) are implemented using CLEAR function. So, we use this state assignment S0~00, S1~01, S2~10. Count up function requires Load=0, Count=1.Clear input should be zero whenever input R=1 or when the system is in state S2(q_1=1).

Hence, $\text{Clear}=(q_1+R)'=q_1'R'$

Count = 1, Load = 0, parallel Load inputs are don't cares.

Alternatively, you can use a state transition table.

q_1	q_2	R	q_1^+	q_2^+		***Clear***	***Load***	***Count***
0	0	0	0	1	count	1	0	1
0	0	1	0	0	clear	0	x	x
0	1	0	1	0	count	1	0	1
0	1	1	0	0	clear	0	x	x
1	0	0	0	0	clear	0	x	x
1	0	1	0	0	clear	0	x	x
1	1	0	x	x	x	x	x	x
1	1	1	x	x	x	x	x	x

Clear = $q_1'R'$ or $(q_1+R)'$

R \ q_1q_2	**00**	**01**	**11**	**10**
0	1	1	x	0
1	0	0	x	0

Load=0

R \ q_1q_2	**00**	**01**	**11**	**10**
0	0	0	x	x
1	x	x	x	x

Count= 1 (clearly seen from the truth table).