1. Practice Problems by Homework

The following exercises represent practice problems you can use to for exam preparation.

HW2 - Number Systems and Conversions

1. **Perform the following number system conversions**
    a. $1110110.010111_2 = ?_8 = ?_{16} = ?_{10}$
    b. $15B.35_{16} = ?_8 = ?_2 = ?_{10}$

2. **Convert the following decimal number into binary:**
    a. $923.625_{10} = ?_2$

3. **Convert the following Octal number into binary and hexadecimal**
    a. $4530.152_8 = ?_2 = ?_{16}$

4. **Convert the following Hexadecimal number into the specified base:**
    a. $DABB.AD00_{16} = ?_2 = ?_8$
    b. $BAD.A_{16} = ?_{10}$

5. **Convert the following number from decimal:**
    a. $2143.64_{10} = ?_5$

HW3 - Boolean Algebra, Logic Functions, and Canonical Representation, 2-Level Implementations

1. **Prove theorem T5 using whatever method you deem fit.**

2. **Use the theorems of switching algebra to simplify each of the following logic functions:**
   a. F = WXYZ(WXYZ' + WX'YZ + W'XYZ +WXY'Z)
   b. F = AB + ABC'D + ABDE' + ABC'E + C'D

3. **Write the truth table for each of the following logic functions:**
   a. F = X'Y + X'Y'Z'
   b. F = W' + X'(Y' + Z)
   c. F = (A' + B' + C.D)(B + C' + D')
   d. F = (((A' + B)' + C')' + D)'

4. **Write the canonical sum and product for each of the following logic functions (algebraically):**
   a. F = $\Sigma_{X,Y,Z}$ (0,2,3,)
   b. F = $\prod_{A,B,C}$ (1,2,4,6)
   c. F = $\Sigma_{A,B,C,D}$ (1,2,5,7) [Hint: Are all the variables necessary?]
   d. F = X' + YZ' +YZ'
   e. F = A'B +B'C +AC'

5. **Prove T10' using Venn diagrams. Show intermediate Venn diagrams.**

6. **Prove the T11 (Consensus theorem) by using switching algebra**

7. **Convert the following to POS.**
   a. F = X'Z' + (Y(X'+Z))'
   b. G = XY + Y'Z'
   c. H = AB•(CD)' + (A+D)

8. **Simplify the following to SOP and then to minterms and the canonical sum.**
   a. Z = AB + (C' + A'B')' + A'(AB + AC'D')

9. **For the following SOP or POS functions, convert to the equivalent POS or SOP function (change SOP to POS and vice versa)**
   a. F = x'y' + xy'z + z'
   b. G = (x'+y')(y)(w'+y+z)

HW4 - Circuit Design w/ Karnaugh Maps

1. **Using a Karnaugh map, find a minimal SOP solution for each of the following functions.**
    a. $F = \Sigma_{ABCD}(1,2,4,5,9,10,12,13)$
    b. $F = \prod_{MNOP}(0,1,2,8,9)$
    c. $F = \Sigma_{ABCD}(0,1,2,5,8,10,11,15)$
    d. $F = \prod_{WXYZ}(3,6,7,12,14)$

2. **Use a Karnaugh map to find the minimal product of sums (POS) expression for each of the following logic functions.**
    a. F = X'Y' + X'Z' + W'X + XYZ
    b. The set of 4-bit prime numbers (let's define input 1 as a don't care).
    c. The set of 4-bit numbers that are not perfect squares or cubes.
    d. The set of 4-bit numbers divisible by 3 OR 5.

3. **Implement the function, $F = \Sigma_{ABCD}(1,2,4,5,9,10,12,13)$ by first finding a minimal expression using a Karnaugh map and then implement it using the following 2 level-logic implementations. Assume both the normal and complemented inputs are available (i.e. you need not use inverters to produce the complement of a variable).**
    a. AND – OR (AND gates feeding a single OR gate)
    b. OR – AND
    c. NAND
    d. NOR

4. **Use a Karnaugh map to find the minimal sum of products expression for each of the following logic functions: (Note: 'd' denotes the don't care set.)**
    a. All 4-bit values with an odd number of 1s, and you don't care about numbers divisible by 4 or 5.
    b. F = B'C'D' + BCD' + ABC'D, d = A'BC'D + A'B'CD'

5. **Simplify each of the following functions using your method of choice and implement them with NAND gates:**
    a. F = A'C + ACE + CAB' + B'A'DE + A' D' E'
    b. F = (AB + A'B') (C'D' + CD)
    c. F = A'B'C'D' + A'B'CD' + A'BC'D' + A'BC'D + AB'CD'
    d. The set of all 4-bit numbers that start with the letters 't' or 'f' when written as a word (zero, one, **t**wo, **t**hree, **f**our, …)

6. **Repeat the same with NOR gates.**

7. **Design a circuit that takes in two 2-bit signed (2's complement) numbers, A (A_1A_0) and B (B_1B_0), and outputs F = A + B. F should be a signed 2's complement number.**
    a. Draw the block diagram for the circuit (remember the output should be a signed 2's complement number)
    b. Write out a truth table
    c. Use Karnaugh Maps to produce minimal equations for each output bit
    d. Show the implementation in AND-OR and NAND-NAND logic

8. **Design a circuit that accepts a 3-bit unsigned number X ($X_2X_1X_0$) and generates an output binary number Y = 3X + 1**
    a. Draw the block diagram for the circuit
    b. Write out a truth table
    c. Find the minimal POS equations for each output bit
    d. Show the implementation in OR-AND and NOR-NOR logic
    e. Now implement the same design using a single 4-bit adder and no other gates

9. **Design a circuit that takes in a 3-bit signed (2's complement) number X and produces an output $Z = X^2 + 2X + 1$.**

10. **Design a 1-bit comparator that takes in a bit X and a bit Y and outputs X<Y, X>Y, X=Y. Use a single 2-to-4 decoder and 1 single OR gate. Recall that a decoder is nothing more than the minterms of the inputs.**

HW5 – Signed Representations and Arithmetic Logic Circuits (Adders and Comparators)

1. **Perform the following binary arithmetic (assuming all number are unsigned)? When performing subtraction you may use the 2's complement method or borrow method.**
   a. $10100_2 + 11101_2 = ?$
   b. $11100_2 - 10100_2 = ?$

2. **Perform the following binary arithmetic (assuming all number are unsigned)? When performing subtraction use the borrow method.**
   a. $101.011_2 + 11.01_2 = ?$

3. **Consider the following subtraction of 2 <u>unsigned</u> numbers. Can you perform the operation?**
   a. $0110110_2 - 1000100_2 = ?$

4. **How many bits are required to represent the decimal number 283 in:**
   a. Unsigned binary
   b. 2's complement

5. **Consider the following decimal numbers +21, +55, +121, -32, -99, -128**
   a. What are the corresponding *8-bit signed-magnitude representation*?.
   b. What are the corresponding *8-bit 2's complement representation*?.
   c. Which of the above numbers can be represented in 6-bit signed-magnitude, 6-bit 1's complement, 6-bit 2's complement representations, explain ?

6. **What are the corresponding decimal representations for the following binary numbers: 01011011 , 11010010, if**
   a. The binary numbers are in *8-bit signed-magnitude format*?
   b. The binary numbers are in *8-bit 2's complement format*?

7. **Perform the following addition problems for the following 2's complement numbers. <u>State whether overflow does or does not occur for each problem</u>. Justify your answer for why overflow does or does not occur. Check your work by converting each number to decimal.**

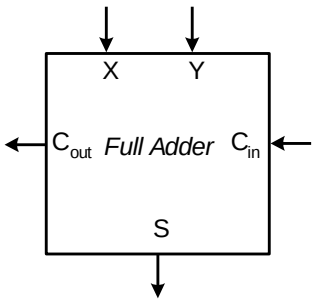
a.) `1010 0111` + `1110 0100`

b.) `1001 0110` + `1011 0011`

c.) `0101 1100` + `1011 0101`

d.) `0101 1101` + `0110 1001`

8. **This is an exercise to help you remember what gates are used to create a full adder. Using a full adder as shown below and no other gates, can you produce the function Z = A•B (Hint: Write out the logical equations for S and Cout and see if you can hook up the inputs such that one of the outputs is equivalent to Z).**



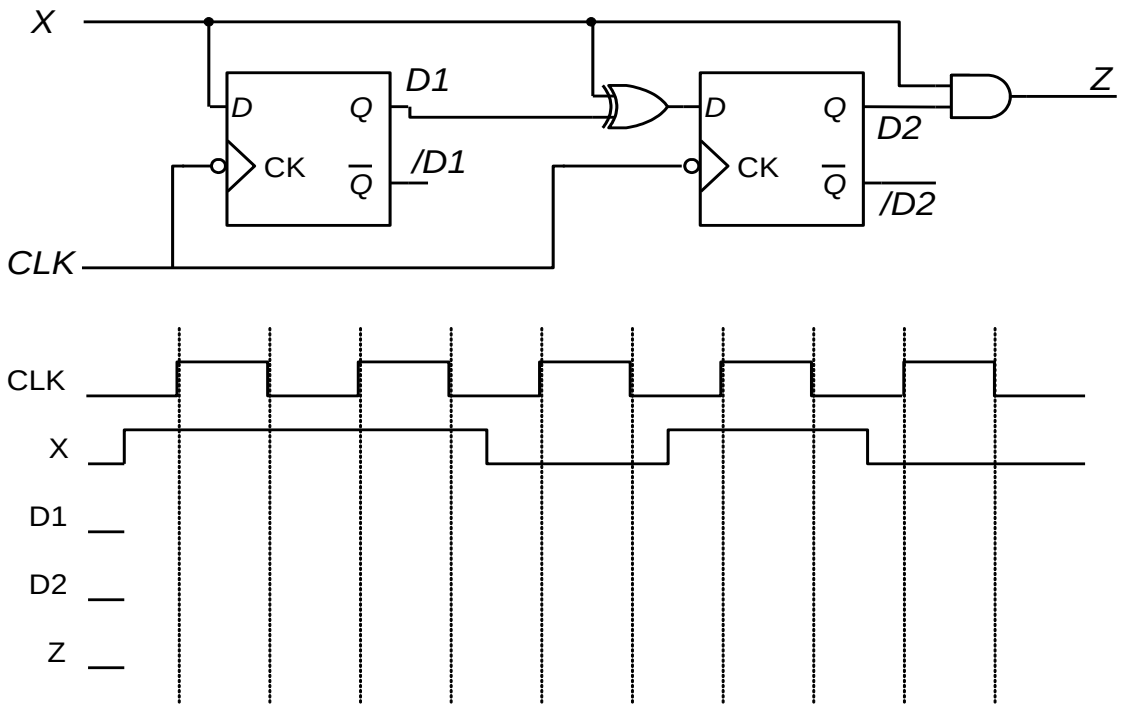
9. **Using 1 half-adder and a minimal number of 4-bit binary adders, design a circuit to calculate Y = 25*X, where X is a 4-bit inputs.**

10. **Using a minimal number of 4-bit adders, design a circuit that implements Y=20*X+107, where X is a 3-bit unsigned number.**

11. **Design a minimal circuit using AND/OR/NOT gates to implement the comparison**
    A > 10 where A is a 4-bit number.
    a. Start by writing out the logical algorithm for when A > 10 then implement it using gates
    b. Check that your work is minimal by using a K-Map.

HW 6 – Latches, and Flip-Flops

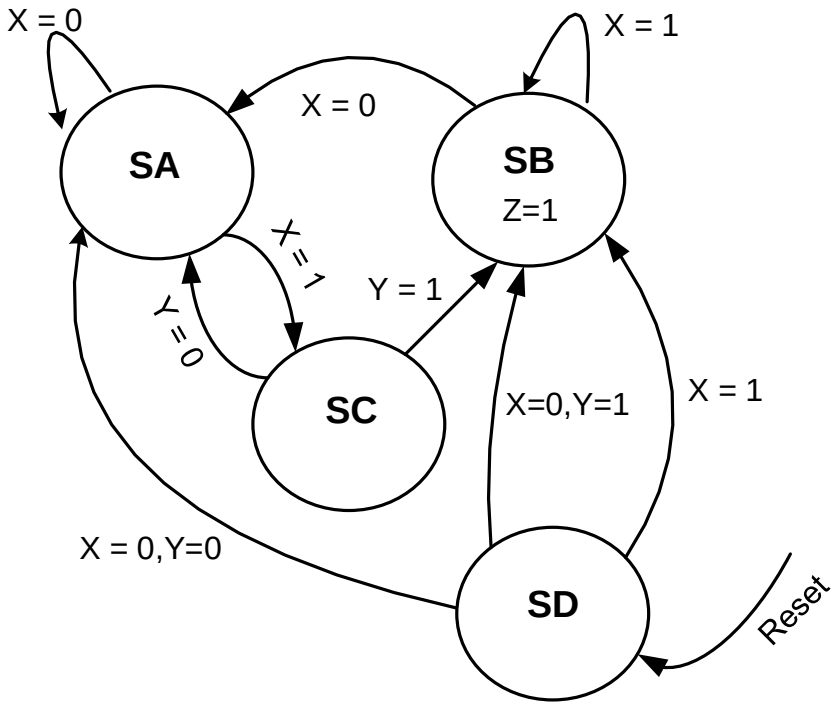
1. Complete the following waveforms for negative edge-triggered D Flip-Flops.



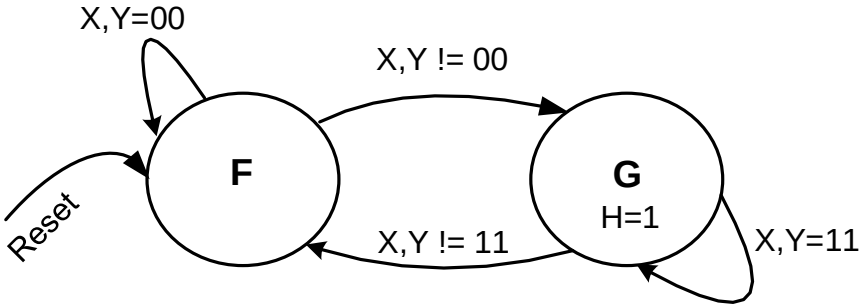
2. Design the following state machine. Use the following state assignment:

State Name	Q1,Q0
SA	00
SB	01
SC	10
SD	11

Find the Initial State as well.



3. Design a state machine to implement the following state diagram. Show all the steps of design and draw out the implementation. In this diagram, X and Y are external inputs and H is an output. Use D flip-flops and for state assignment let Q=0 represent state F, and Q=1 represent state G.



4. Design a state machine with the state/output table shown below using positive-edge triggered JK Flip-flops (w/ preset and clear inputs) Use two state variables, Q1 and Q0, with the state assignment A=00, B=01, C=10, D=11. Show all steps of your design and draw out the implementation. Use the preset and clear inputs along with the signal /RESET to initialize the state machine to the initial state A. Then complete the waveform on the attached sheet.

Current State	Next State		Output
	X=0	X=1	
S	S*	S*	Z
A (initial st.)	B	D	0
B	C	B	0
C	B	A	1
D	B	C	0

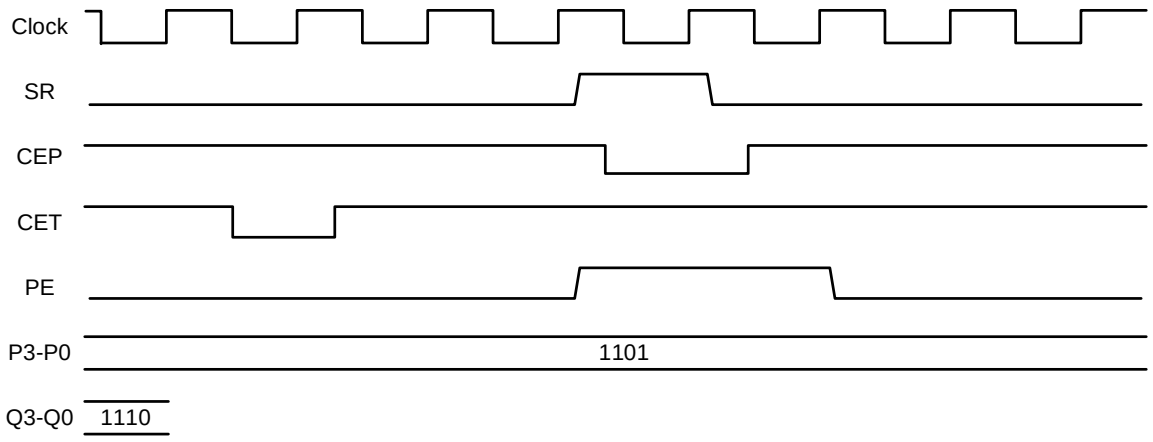
5. Design a modulo 4 Up/Down Counter (2-bit up/down counter). This counter has two inputs UP and DOWN. The circuit should not count (it should stay at it's current value) at all if UP=DOWN=1 or UP=DOWN=0. The circuit should count up if UP=1 and DOWN=0 and the circuit should count down if UP=0 and DOWN=1.

   a.) Draw a state diagram of this machine. On power on (reset) the counter should start at Q1,Q0 = 00

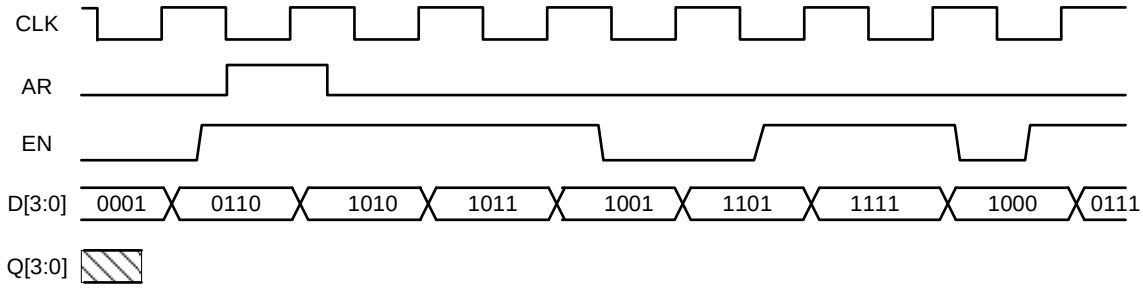
   b.) Using positive-edge triggered D-Flip-Flops implement the circuit (show all steps.) You do NOT need to draw out the circuit, just come up with equations for the flip-flop inputs (excitation variables). Use the state variables Q1 and Q0.

   c.) Complete the waveform on the attached sheet.

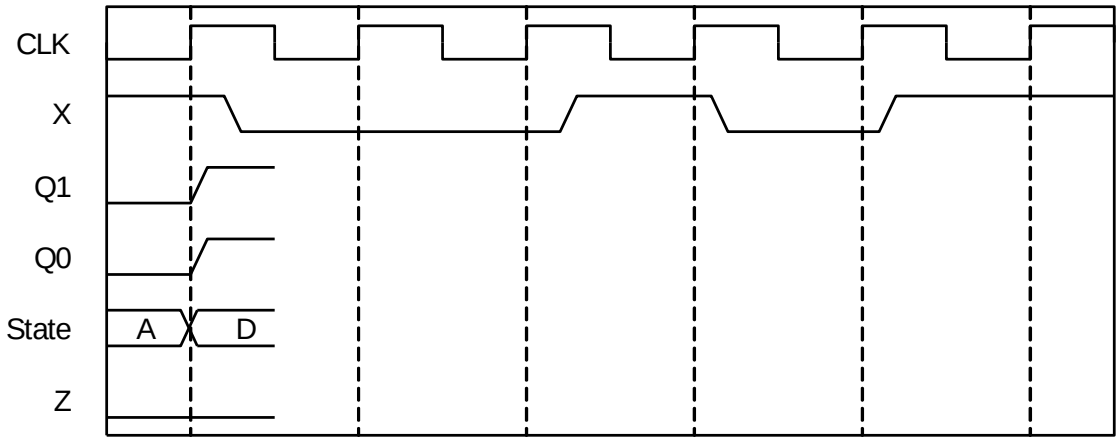
6. Complete the waveform for the 74LS163A 4-bit counter.



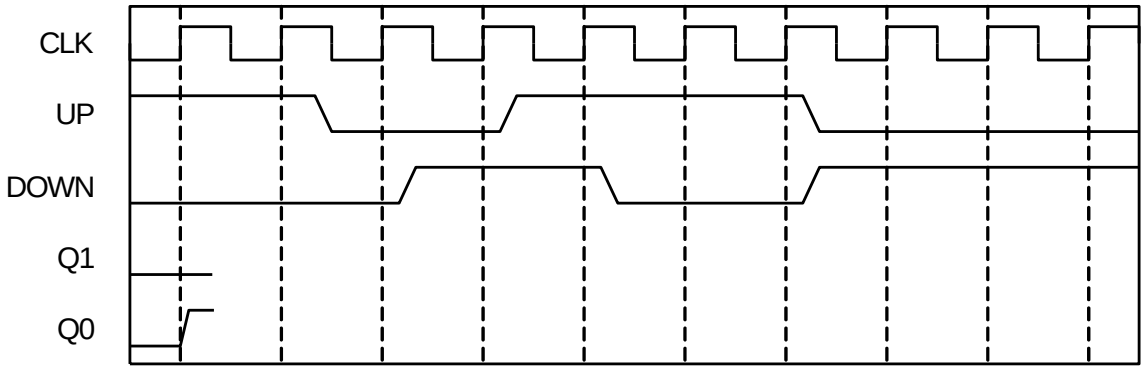
7. Complete the waveform for a 4-bit D-Register with active-hi Data (Load) Enable and active-low asynchronous reset.



Waveform for problem 4.



Waveform for problem 5.

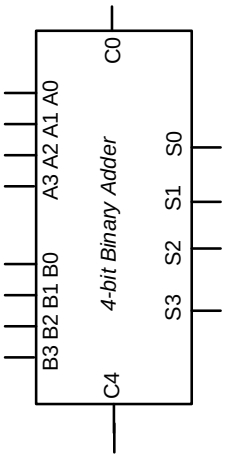


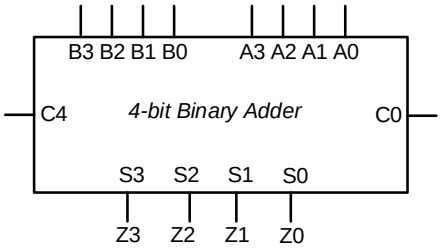
1.1. Unit 7 - Datapath Design

1. Implement a circuit that takes in a 4-bit number X[3:0] and produces a 4-bit value Z[3:0] according to the following function:

   if X < 8 then
   Z = X + 5;
   else
   Z = X – 2;

   Using the building blocks below and at most 1 inverter (no other gates), implement this function.





2. Design a circuit that takes in an 4-bit number X[3:0] and outputs a number Y, and performs the operation below. Assume these numbers are unsigned. Use a 4-bit adder and 2-to-1 muxes. Hint: Multiplying by 4 (i.e. 2^2) in binary can be done with no gates just as multiplying by 100 (i.e. 10^2) in decimal can be done simply and easily.

   if($X < 0011_2$)
   Y = 4X
   else
   Y = X

3. Design a circuit that takes in two 4-bit numbers, X[3:0] and Y[3:0] along with two function select bits, FS1 and FS0, and produces an output, Z[3:0], according to the following table:

FS1,FS0	Z
0,0	X+Y
0,1	X-Y
1,0	Y-X
1,1	don't care

   Use (1) 4-bit adder, 4-bit wide 2-to-1 muxes, and any basic gates you desire.

   Approach: Take a 4-bit adder and for each operation listed in the table above identify what should be passed to the A and B inputs of the adder. Then use muxes to perform that function. Design logic for the select bits of the muxes and the carry-in of the adder that use FS1 and FS0 as input.

4. Design a circuit that takes in two 4-bit **signed magnitude** numbers, X[3:0] and Y[3:0] and produces two 4-bit outputs, A[3:0] and B[3:0] according to the following algorithm:

   if X > Y then
   A = X and B = Y
   else
   A = Y and B = X

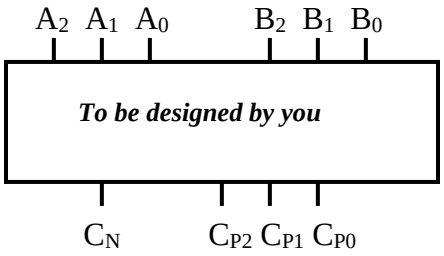
   Use 4-bit adders, 4-bit wide 2-to-1 muxes, and basic logic gates. Assume the input 1000 = -0 will never occur on X or Y.

   Approach: First think about how to compare signed magnitude numbers by using unsigned comparison techniques and how to precondition the inputs to allow the use of unsigned comparison. For example, 1001 = -1 in signed magnitude and 0100 = +4 in signed magnitude. So 1001 is really less than 0100. Use the comparison result to control the selects of the muxes.

5. Design a circuit that adds two 4-bit, unsigned numbers: X[3:0] and Y[3:0] and outputs either the sum if there is no overflow or 1111 (=15_{10}, the maximum unsigned 4-bit number) if there is overflow.

6. We would like to build an adder whose inputs are in units of pennies and produces sum in nickels and pennies. Essentially, this is a base-5 adder. The 2 inputs of this adder are A={$A_2A_1A_0$} and B={$B_2B_1B_0$}. **Assume that the maximum input number that A and B can be is 4 pennies**. Design an adder to produce the correct nickels/pennies sum, C={ [C_N] ; [$C_{P2}C_{P1}C_{P0}$] }. To implement this design you may assume you can use as many 4-bit adders as you like.

if X+Y > 4 dec. then
    CN = 1, CP[2:0] = X+Y-5
else
    CN = 0, CP[2:0] = X+Y



Block diagram for the nickel/penny adder you are going to design