KFUPM - COMPUTER ENGINEERING DEPARTMENT

COE-202 – Fundamentals of Computer Engineering

Assignment # 2: Due Sunday January 4th, 2008 – in class. Solution Key

Problem 1) (20 points) Simplify the following Boolean functions to the form of sum-of-products by finding all prime implicants and essential prime implicants and applying the selection rule:

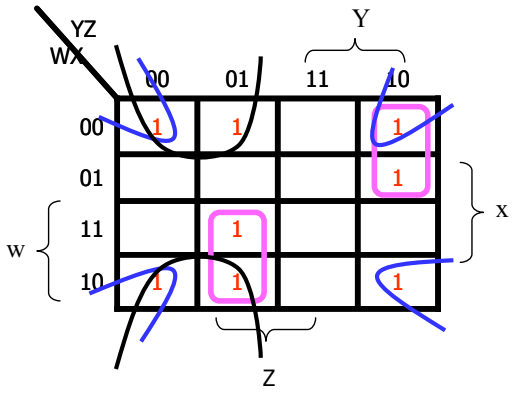
a) F(W,X,Y,Z) = Σm(0,1,2,6,8,9,10,13)

b) F(A,B,C,D) = ΠM(1,3,5,6,7,9,10,11,14)

Solution:

a) The K-map for F(W,X,Y,Z) is as shown.
List of prime implicants: X'Y', X'Z', W'YZ', and WY'Z.
List of essential prime implicants: all prime implicants are essential.

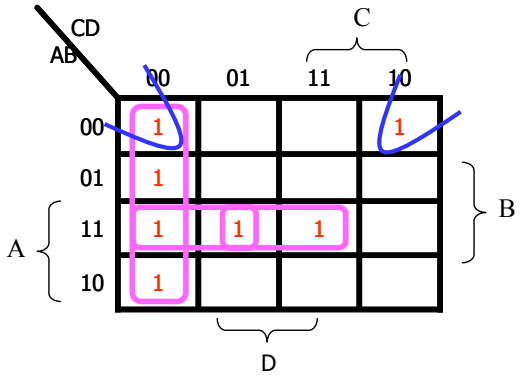
Therefore, F(W,X,Y,Z) = X'Y' + X'Z' + W'YZ' + WY'Z.



b) F(A,B,C,D) = ΠM(1,3,5,6,7,9,10,11,14)
= Σm(0,2,4,8,12,13,15).
The K-map for F(A,B,C,D) is as shown.
List of prime implicants: C'D', A'B'D', ABD, and ABC'.

List of essential prime implicants: C'D', A'B'D', and ABD.

Therefore, F(A,B,C,D) = C'D' + A'B'D' + ABD.



Problem 2) (20 points) Simplify the following expressions in (1) sum-of-products and (2) product-of-sums forms. Use the K-map method for the simplification:

a) AC' + B'D + A'CD + ABCD

b) (A' + B' + D) (A' + D') (A + B + D') (A + B' + C + D)

Solution:

a) F(A,B,C,D) = AC' + B'D + A'CD + ABCD

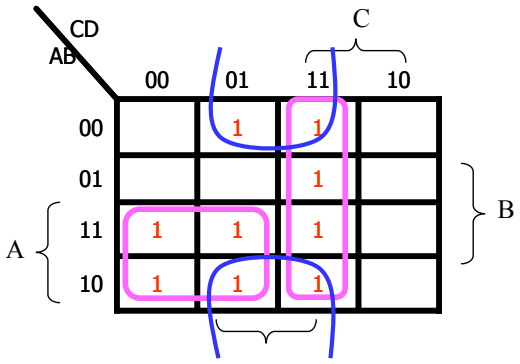
Sum of products form:

The K-map for F(A,B,C,D) is as shown.

Therefore,

F(A,B,C,D) = Σm(1,2,3,7,8,9,11,12,13,15)

= CD + AC' + B'D.



Map for F(A,B,C,D) in 2(a)

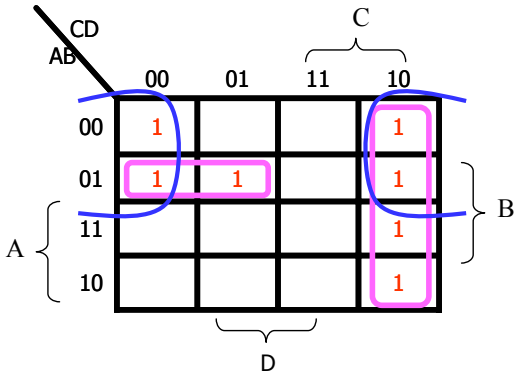
Product of sums form:

F'(A,B,C,D) = Σm(0,4,5,6,10,14)

= CD' + A'D' + A'BC'

Therefore,

F(A,B,C,D) = (C'+D)(A+D)(A+B'+C)



Map for F'(A,B,C,D) in 2(a)

b) F(A,B,C,D) = (A' + B' + D) (A' + D') (A + B + D') (A + B' + C + D)

Product of sums form:

F(A,B,C,D) = (A' + B' + D) (A' + D') (A + B + D') (A + B' + C + D)

Hence,

F'(A,B,C,D) = ABD' + AD + A'B'D + A'BC'D'.

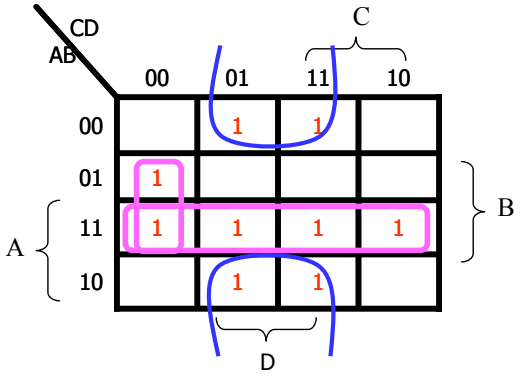
The K-map for F' is as shown

Then,

F'(A,B,C,D) = Σm(1,3, 4,9, 11, 12, 13,14,15)

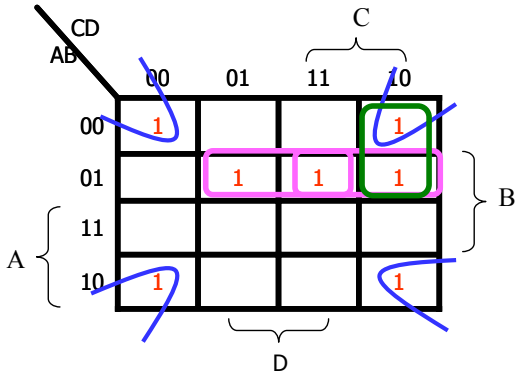
= AB + B'D + BC'D'

Therefore, F = (A'+B')(B+D')(B'+C+D).



Map for F'(A,B,C,D) in 2(b).

Sum of Products:
The K-map for F(A,B,C,D) is as shown.
F(A,B,C,D) = Σm(0,2,5,6,7, 8, 10)
= B'D' + A'BD +A'BC, OR
= B'D' + A'BD + A'CD'



Map for F(A,B,C,D) in 2(b).

Problem 3) (10 points) Implement the result for (1.a) using NAND gates. Use LogiSim to implement the circuit. Submit screen shot for your implementation with your name and ID added to the circuit.

Solution:

Problem 4) (10 points) Implement the result for (1.b) using NOR gates. Use LogiSim to implement the circuit. Submit screen shot for your implementation with your name and ID added to the circuit.

Solution:

Problem 5) (50 points) It is required to design a 4-bit ripple-borrow subtractor to find the subtraction X-Y for the two unsigned numbers, X=X3-X0, and Y=Y3-Y0.
a) **(**25 points**)** Design a 1-bit full subtractor. The 1-bit full subtractor performs the following operation: D = X – bin – Y. Where D is the output, X, bin (borrow in), and Y are the input. Plot the truth table, and design the required circuit. Using a block diagram show how it can be used to construct the 4-bit ripple-borrow subtractor.
b) (25 points**)** Implement the 4-bit ripple-borrow subtractor in LogiSim and verify its operation. Provide several printouts displaying the required operation with selected examples of subtracting 4-bit numbers.

Solution:
a) Design
1-bit full subtractor:

Three inputs:
The two bits to subtract: X and Y,
The borrow in: bin

The required operation:
D = X-bin-Y
The operation may require a borrow from the next stage, bout
Two outputs: D and bout – The truth table is shown on the right

Bin	X	Y	Bout	D	Comment
0	0	0	0	0	0-0-0=0, No borrow
0	0	1	1	1	0-0-1→borrow 2, D = 2-1=1
0	1	0	0	1	1-0-0=1, no borrow
0	1	1	0	0	1-0-1=0, no borrow
1	0	0	1	1	0-1-0 → borrow 2, D = 1
1	0	1	1	0	0-1-1→ borrow 2, D = 0
1	1	0	0	0	1-1-0=0, no borrow
1	1	1	1	1	1-1-1→ borrow 2, D = 1

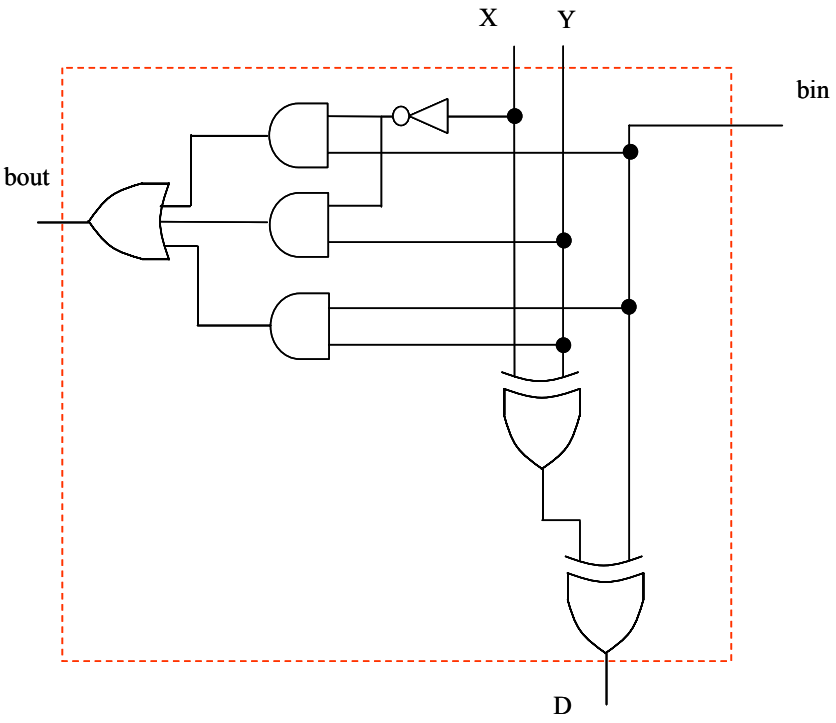
XY / bin	00	01	11	10
0	0	1	0	0
1	1	1	1	0

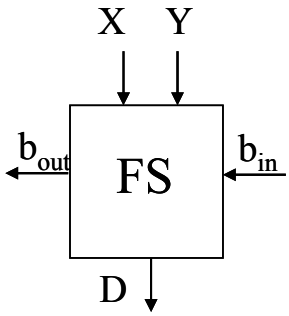
bout = X'bin + X'Y + binY

XY / bin	00	01	11	10
0	0	1	0	1
1	1		1	0

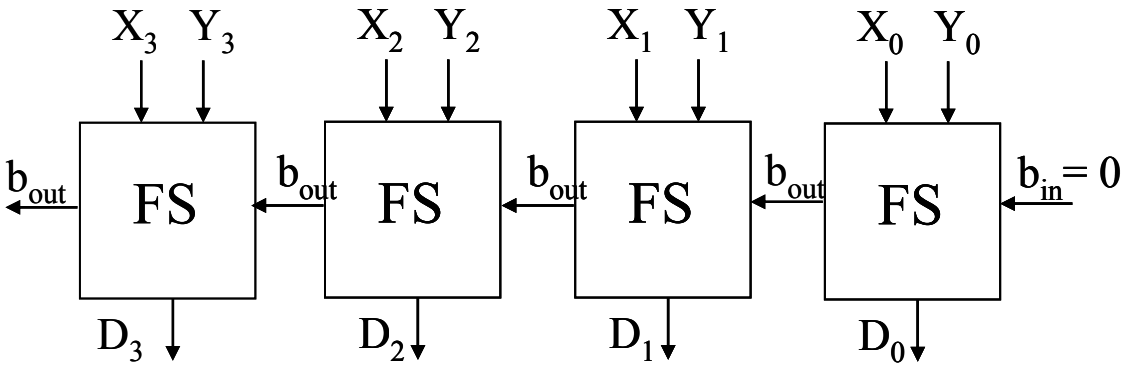
$D = X \oplus Y \oplus bin$

The circuit is as shown – the equivalent 1-bit full subtractor box is the dotted box





The 4-bit ripple borrow subtractor is formed by connecting four 1-bit full subtractor boxes of part (a) – see the figure below:



b) the LogiSim implementation:

Problem 6) (50 points) Design a combinational circuit that receives a 4-bit signed number in 2's complement representation and returns the absolute value of the number i.e., the output returned should be 3-bit. Show all steps of design and implement your design in LogiSim. Submit several printouts displaying the function of your design.

Solution:

a) the input is a four bit 2's complement number A = A3A2A1A0
the output is the absolute value of A – 3-bit number B = abs(A) = B2B1B0
The truth table and K-Maps are shown below:

	inputs				outputs			
# A	A3	A2	A1	A0	B2	B1	B0	# B = Abs(# A)
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	1	1
2	0	0	1	0	0	1	0	2
3	0	0	1	0	0	1	0	3
4	0	1	0	1	1	0	1	4
5	0	1	0	1	1	0	1	5
6	0	1	1	0	1	1	0	6
7	0	1	1	1	1	1	1	7
-8	1	0	0	0	0	0	0	0
-7	1	0	0	1	1	1	1	7
-6	1	0	1	0	1	1	0	6
-5	1	0	1	0	1	0	1	5
-4	1	1	0	1	1	0	0	4
-3	1	1	0	1	0	1	1	3
-2	1	1	1	0	0	1	0	2
-1	1	1	1	1	0	0	1	1

The corresponding K-Maps and equations are:

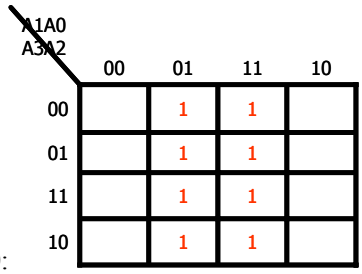
K-map for B2:

A1A0 / A3A2	00	01	11	10
00				
01	1	1	1	1
11	1			
10		1	1	1

K-map for B1:

A1A0 / A3A2	00	01	11	10
00			1	1
01			1	1
11		1		1
10		1	0	1

K-map for B0:



The formulas are:
B2 =A3'A2 + A2A1'A0' + A3A2'A1 + A3A2'A0
B1 = A1A0' + A3'A1 + A3A1'A0
B0 = A0

The circuit implementation (required – could be from LogiSim) is direct translation of the formulas above.

The LogiSim implementation.

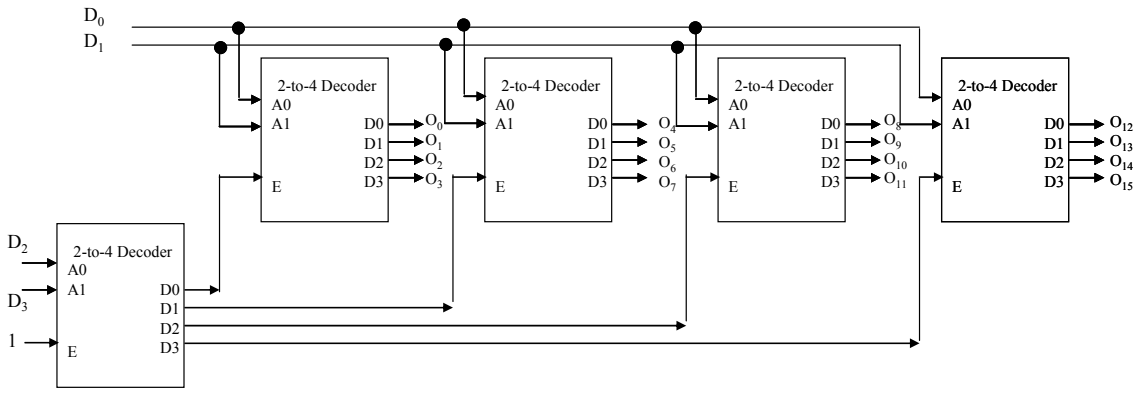
Problem 7) (20 points) Construction of a 4-to-16 line decoder with enable input using *five* 2-to-4 line decoders with enable inputs. Do not use other logic gates.

Solution:
(see the online material – solved as an example).
The truth table for the 4-to-16 line decoder is as shown:

It can be seen the A3A2, when decoded, can enable the require 1 of 4 2-to-4 decoders A1A0 always decoded into 4 outputs

Therefore, the final design as shown below.

	A_3	A_2	A_1	A_0	Output
A_3A_2 = 00	0	0	0	0	O0
	0	0	0	1	O1
	0	0	1	0	O2
	0	0	1	1	O3
A_3A_2 = 01	0	1	0	0	O4
	0	1	0	1	O5
	0	1	1	0	O6
	0	1	1	1	O7
A_3A_2 = 10	1	0	0	0	O8
	1	0	0	1	O9
	1	0	1	0	O10
	1	0	1	1	O11
A_3A_2 = 11	1	1	0	0	O12
	1	1	0	1	O13
	1	1	1	0	O14
	1	1	1	1	O15



Problem 8) (50 points) Construction of a 15-to-1 line multiplexer

a) Construct a 15-to-1 line multiplexer with two 8-to-1 line multiplexers. The corresponding figure depicts the multiplexer block and its functional table. Minimize the added logic required to have selection codes 0000 through 1110.

b) Use LogiSim to construct an 8-to-1 line multiplexer. Save the block under the name 8x1MUX. Use two of such blocks plus added logic gates to implement the final 15-to-1 line multiplexer using LogiSim. Submit both the designs for the 8-to-1 block and the 15-to-1 line multiplexer. Highlight on the submitted screen shots the operation of the 15-to-1 multiplexer

Hint: *See the supplied figures: P8a, P8b, and P8c.*

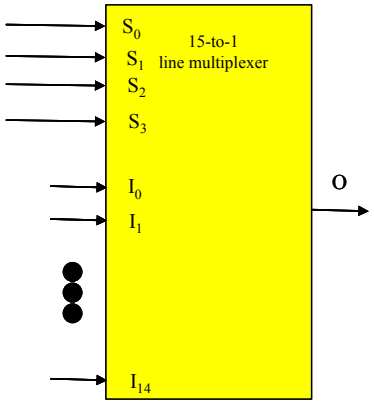


Figure P8a: 15-to-1 line multiplexer block.

The required functional table: S0, S1, S2, and S3 are the four select lines for the 15-to-1 line multiplexer, and O is the output line. I0, I1, …, I14 are the 15 input lines to select from.

S0	S1	S2	S3	O
0	0	0	0	I0
0	0	0	1	I1
0	0	1	0	I2
0	0	1	1	I3
0	1	0	0	I4
0	1	0	1	I5
0	1	1	0	I6
0	1	1	1	I7
1	0	0	0	I8
1	0	0	1	I9
1	0	1	0	I10
1	0	1	1	I11
1	1	0	0	I12
1	1	0	1	I13
1	1	1	0	I14

Figure P8b: 15-to-1 line multiplexer functional table.

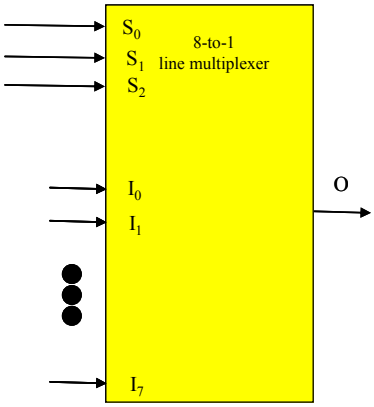


Figure P8c: 8-to-1 line multiplexer block.

Solution:

a) Construction of 15-to-1 mux:

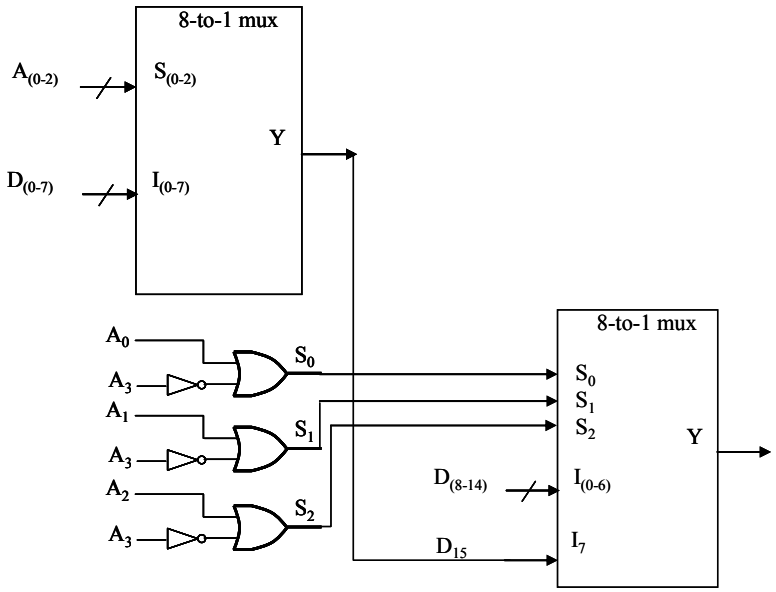
The required 15-to-1 mux has the following truth table

The table shows that two 8-to-1 muxes can be used to where A3 determines which of the two muxes (upper or lower) is to be used.

A2A1A0 are sufficient to select from the upper 8 inputs or the lower 8 inputs.

When A3 is 1 then the output of the 1st mux passed to the 2nd mux is ALWAYS selected to appear as the final output, regardless of A2A1A0.

A_3	A_2	A_1	A_0	Output
0	0	0	0	D0
0	0	0	1	D1
0	0	1	0	D2
0	0	1	1	D3
0	1	0	0	D4
0	1	0	1	D5
0	1	1	0	D6
0	1	1	1	D7
1	0	0	0	D8
1	0	0	1	D9
1	0	1	0	D10
1	0	1	1	D11
1	1	0	0	D12
1	1	0	1	D13
1	1	1	0	D14
1	1	1	1	



b) Implementation using LogiSim.