LAB 1: LOGIC MINIMIZATION

This lab will give you a chance to review the basics of Boolean algebra and implement the Karnaugh Map (K-map) and Quine-McCluskey (tabular) methods to minimize Boolean functions, and to realize said Boolean functions in a logic circuit.

When prompted to implement a circuit, feel free to use Logisim-evolution or CircuitVerse.

Part 1: Boolean Algebra Review

1.A) Equivalent Operations

Consider two functions f and g, of two inputs x and y, with x as the MSB, and y as the LSB:

$$f(x,y) = \sum m(1,2) \qquad \textbf{(1)}$$

$$g(x,y) = \prod m(0,3) \qquad \textbf{(2)}$$

a) Draw up the truth tables of f and g. Are f and g equivalent functions? If so, is there a name for this Boolean function?
b) One of these functions, as expressed, is more naturally implemented as an OR-AND circuit, and the other as an AND-OR circuit. Which is which?
c) Implement f and g, based on your response in b).

SOLUTION:

a) From the SOP and POS representations of f and g, it is clear that f is 1 when xy = 01 or 10, and so can be expressed as f = x'y + xy', and g is 0 when xy is 00 or 11 and so can be expressed as (x+y)(x'+y'). One can then see that f and g are clearly equivalent functions from the truth tables:

x	y	f	g
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

And clearly, f and g are the XOR function.

b) SOPs are more naturally implemented as AND-OR circuits, and POSs are more naturally implemented as OR-AND circuits, although this is far from the only way to do either, as we will see again and again in future labs.

c) and d) will have their solutions in the GitHub repository as <ECE_Project_1_Part1Solution>.

END OF SOLUTION

1.B) De Morgan's Theorem

Recall the De Morgan's Theorem:

$$\overline{(a_1 + a_2 + \cdots + a_n)} = (\overline{a_1} \cdot \overline{a_2} \cdot \ldots \cdot \overline{a_n}) \quad \textbf{(3)}$$

$$\overline{(a_1 \cdot a_2 \cdot \ldots \cdot a_n)} = (\overline{a_1} + \overline{a_2} + \cdots + \overline{a_n}) \quad \textbf{(4)}$$

d) Suppose you only have access to NAND and NOR gates. Implement the OR-AND and AND-OR Circuits using only NAND and NOR gates.

SOLUTION:

d) will have its solutions in the GitHub repository as <ECE_Project_1_Part1BSolution>, with "f2" being the re-implementation of f for part 1b), and "g2" being the re-implementation of g for part 1b). The general principle is that if the inputs of a NAND gate are inverted, the NAND gate becomes an OR gate, and similarly if the inputs of a NOR gate are inverted, the NOR gate becomes an AND gate. This will be somewhat important for labs later on, as we will introduce more and more constraints on the kind of components students will have access to (constraints from which they're largely free of in most of this lab), and one running theme is that they will only have access to NANDs or NORs.

END OF SOLUTION

Part 2: The K-Map Method

Consider two functions H and L of three inputs x, y, and z, with x as the MSB and z as the LSB.

$$H(x, y, z) = \sum m(1,2,4,5,6,7) \quad \textbf{(5)}$$

$L(x, y, z) = \prod m(3,5,6,7)$ **(6)**

a) Draw up a K-map for each of these functions, and minimize them to determine an SOP or POS for each of them.
b) Implement each of these as OR-AND (for SOPs) or AND-OR circuits (for POSs).

SOLUTION:

H has the K-map:

$xy \rightarrow$ $z \downarrow$	00	01	11	10
0	0	1	1	1
1	1	0	1	1

H1 (orange) = x; H2 (green) = x'yz'; H3 = (blue) = x'y'z;

Then, $H = H_1 + H_2 + H_3 = x + x'yz' + x'y'z$

And L has the K-map:

$xy \rightarrow$ $z \downarrow$	00	01	11	10
0	1	1	0 (%)	1
1	1	0(*)	0 (*,%, &)	0 (&)

L1 (denoted by %) = x' + y'; L2 (denoted by &) = x' + z'; L3 (denoted by *) = y' + z'

Then $L = L_1L_2L_3 = (x' + y')(x' + z')(y' + z')$.

And so each of them will have the implementation solutions <ECE_Project_1_H> and <ECE_Project_1_L> in the Github repository respectively.

END OF SOLUTION

Part 3: The Tabular Method

We will once again use the function H in Part 2.

a) Before employing the tabular method, attempt to implement H as an AND-OR circuit without grouping and combining any of the implicants.
b) Construct an Implicant Table for H, and determine the prime implicants of H. Keep track of the minterms covered by each implicant at each stage of the process.

c) Now, construct an Implicant Chart for H, and determine the SOP of H.
d) Compare the two implementations for H from this Part, and that of Part 2. Are they the same, or different? Why or why not would that be the case? If they are different, which is the most efficient?

SOLUTION:

a) Solution will be in Github repository as <ECE_Project_1_H_unoptimized>.
b) Given the minterms of H, we get the Implicant Table:

Group	Col 1	Group	Col 2	Group	Col 3
G1	001 (1) 010 (2) 100 (4)	G4	-01 (1,5) (*) -10 (2,6) (*) 10- (4,5) 1-0 (4,6)	G6	1-- (4,5,6,7) (*)
G2	101 (5) 110 (6)	G5	1-1 (5,7) 11- (6,7)		
G3	111 (7)				

First we group by number of 1 bits, then attempt to combine nearest-neighbor groups. Can't combine non-nearest-neighbor groups, since they will differ in number of 1 bits by more than one, and will therefore definitely differ by more than one bit.

c) After finding the prime implicants P1 = 1--, P2 = -10, and P3 = -01, then we get the Implicant Chart:

	m1	m2	m4	m5	m6	m7
P1			x	x	x	x
P2	x			x		
P3		x			x	

Students should notice that as they go along, the number of rows is being reduced as more and more implicants are merged or being marked essential (and therefore, the complexity of the problem is being reduced as the scope of possible SOPs/combinations of implicants you will have to consider is being reduced) We can begin by noticing that minterm m1 is only covered by P2, so P2 is clearly essential:

	m2	m4	m6	m7
P1		x	x	x
P3	x		x	

The same can be said for minterm m2, only covered by P3:

	m4	m7

P1	x	x

Leaving only m4 and m7 uncovered, and so we must take P2 to cover them.

So, H's prime implicants are P1, which represents x and matches with H1 in Part 2, P2, which represents yz', and P3, which represents y'z. And so we get the SOP for H:

$$H = x + yz' + y'z$$

d) We can see the two implementations "H1" (H's implementation from Part 2) and "H2" (from the SOP above that we just obtained) in the Github circuit file <ECE_Project_1_Part3Comparison>. Clearly, the two are slightly different, but in the end, they are the same function. It is certainly possible to get the same SOP for H in both Parts 2 and 3—they are the same function after all—it is possible to get different but equivalent expressions, and this comes from the fact that students can group 1s and 0s in the K-maps differently (I could have grouped in the original H2 with the upper right 1 in H1, and the lower left 1 in H1 with H3, so that H2 becomes yz' and H3 becomes y'z, so that the expressions for H match in both Parts 2 and 3).

END OF SOLUTION