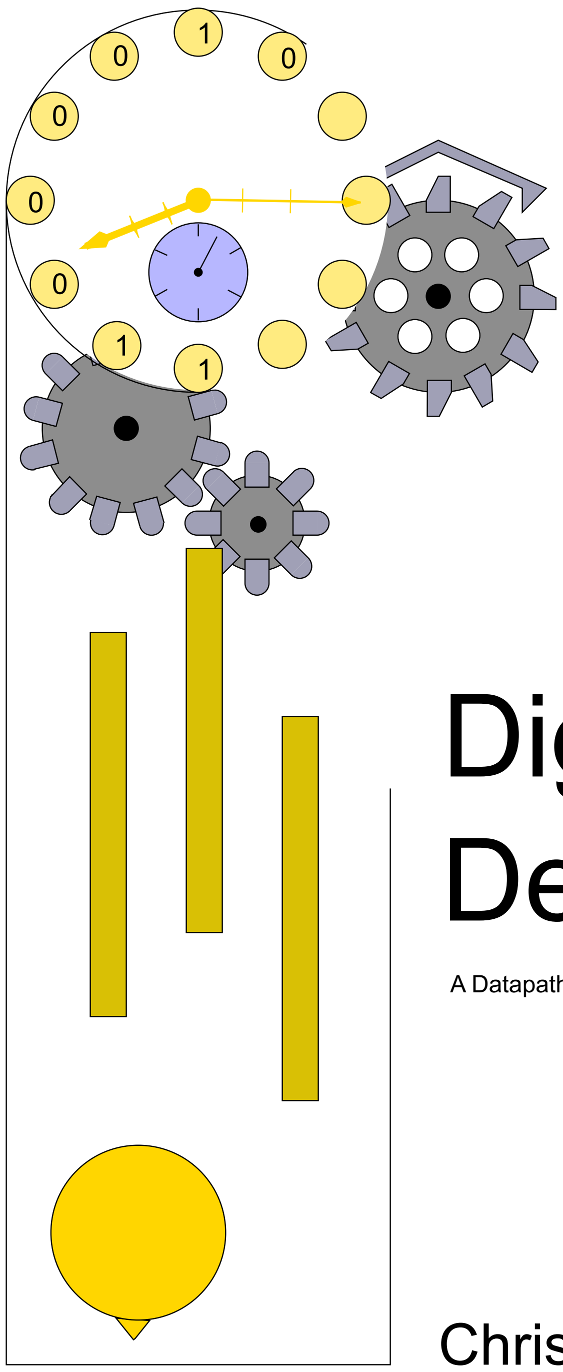


Digital Design

A Datapath and Control Approach

Chris Coulston

October 1, 2024

This document was prepared with LaTeX.

Contents

Contents **iv**

List of Processes **iv**

Digital Design Body Of Knowledge vi

1 Minimization of Logical Functions **1**

1.1 Karnaugh Maps 3
1.2 4-Variable Kmaps 6
1.3 5-Variable Kmaps 7
1.4 Multiple Output Circuits 8
1.5 "Don't Cares" 9
1.6 Minimizing to POS 11
1.7 Espresso 15
1.8 Exercises 20

A 74LS00 Data Sheets **23**

List of Processes

1.1 Solving a Kmap 3
1.2 Determine SOP_{min} given a $\sum m(\ldots)$ expression. 12
1.3 Determine SOP_{min} given a SOP expression. 12
1.4 Determine POS_{min} given a $\sum m(\ldots)$ expression. 13

1.5 Determine POS $_{\text{min}}$ given a SOP expression. 14
1.6 Determine the POS $_{\text{min}}$ given a $\prod M(\dots)$ expression. 14
1.7 Determine the POS $_{\text{min}}$ given a $\prod M(\dots)$ expression. 14
1.8 Determine SOP $_{\text{min}}$ given a POS expression. 14
1.9 Determine the POS $_{\text{min}}$ given a POS expression. 15

Digital Design Body Of Knowledge

The focus of this text is very much on the datapath and control approach. To achieve this end in a single semester, sacrafices in coverage must be made.

```
Digital Design
 |__Numbering Systems
 |   |__Positional Numbering Systems
 |   |   |__Base 10 - Decimal
 |   |   |__Base 2 - Binary
 |   |   |__Base 16 - Hexadecimal
 |   |__Conversion Between Bases
 |   |__Word Size
 |   |__2's Complement
 |__Representation of Logical Function
 |   |__Elementary Logical Functions
 |   |__Word Statement
 |   |__Truth Table
 |   |__Symbolic
 |   |__Circuit Diagram
 |   |__Hardware Description Languages
 |   |__Conversion Between Representations
 |   |__Timing Diagrams
 |__Logic Minimization
 |   |__Karnaugh Maps (Kmaps)
 |   |__Kmaps for circuits with multiple outputs
 |   |__Kmaps to find POSmin
 |   |__Logic Minimization Software
 |__Combination Logic Building Blocks
 |   |__Decoder
 |   |__Multiplexers
 |   |__Adders
 |   |__Comparators
 |   |__Three-State Buffers
 |   |__Wire Logic
 |   |__Combination
 |       |__Arithmetic Statements
 |       |__Conditional Statements
 |__Primitive Sequential Circuits
 |   |__Characteristics
 |   |__Timing
 |   |__Asynchronous set/reset
 |__Sequential Logic Building Blocks
```

Chapter 1

Minimization of Logical Functions

The intent of this chapter is to explore how to realize circuits efficiently. Efficiency can be measured in many different ways; number of gates, speed, power dissipation, and layout size, being just a few. A generally accepted meaning of minimization is to minimize the number of gates required to realize a function in SOP or POS form. It should be clear that any logic function can be realized in an SOP or an POS form. For all but the simplest digital systems, the OR gate cannot be eliminated from the SOP realization. Hence, the focus should be on eliminating as many AND gates as possible. To do this, similar minterms are combined together using a trick from Boolean Algebra.

Consider the function $F(A, B, C, D)$ defined by the truth table below.

A	B	C	$F(A, B, C)$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

The canonical SOP expression for this function is $F(A, B, C) = A'BC' + A'BC$. This digital circuit requires two NOT gates, two AND gates and one OR gate. This realization, however, is not the most minimal one for F. Consider the following Boolean Algebra manipulations:

$$\begin{array}{ll} 1 & A'BC' + A'BC = \\ 2 & A'B(C' + C) = \\ 3 & A'B(1) = \\ 4 & A'B \end{array}$$

This realization of F requires one NOT gate, one AND gate and zero OR gates. Clearly, this realization requires fewer gates then the first realization. Hence, it is a more efficient solution.

The minterms used to realize $F(A, B, C)$ are $A'BC'$ and $A'BC$, corresponding to the inputs 010 and 011 respectively. The factoring in Line 2 takes advantage of the fact that each of the minterms has two variables in common, A' and B. This factoring could also be viewed as a result of the fact that the binary inputs of the two minterms have two bits in common $A = 0$ and $B = 1$ (see the truth table for F). The input variable changing between the two minterms, C, is factored out in Steps 2 and 3 in the above derivation. This observation forms the basis of the simplification trick:

> **Simplification Trick:** Two minterms whose inputs differ by a single bit may be replaced by a single product term that contains the variables which are the same in both minterms and excludes the variable which changes.

The simplification trick is used to obtain a simplified form for the function G defined by the truth table below.

A	B	C	$G(A, B, C)$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

G has four minterms; the main task is to identify which pairs differ by a single bit. Inputs 001 and 101 differ by a single bit, A. Thus, with the simplification trick, the product term derived for this pair of minterms is: $B'C$. Likewise, the inputs 010 and 110 differ in the A bits, hence their product term is BC'. Since the product terms are ORed together in the SOP realization, the reduced product terms generated by the simplification trick are ORed together, hence $G(A, B, C) = B'C + BC'$. Trying to use Boolean Algebra on this expression will not produce a more minimal SOP form.

This simplification trick works fine for small examples. However, when the number of inputs to the function increases by 1, the number of rows in the truth table doubles. Identifying pairs of inputs which differ by a single bit would quickly become tedious and error-prone. A more efficient technique requires rearranging the truth table to make executing the simplification trick easier.

In order to accomplish this goal, the truth table is arranged so that rows of the truth table whose inputs differ by a single bit are adjacent to one another. With this accomplished, the simplification trick can be invoked by looking for adjacent 1s. Such a rearranged truth table is called a *Karnaugh-map* or Kmap for short. A Kmap for a function with three input variables A, B, C is shown in the leftmost Kmap.

Each of the cells in the Kmap corresponds to a row of the truth table and consequently has a unique combination of A, B, C variables. The A, B, C value of a cell is determined by reading off the A bit from the row index on the left and reading the B, C bits from the column index at the top. The Kmap to the right has the decimal representation of ABC placed in each cell. This numbering of the cells make the placement of the 1s of a function easier when specified in the $\sum m$ form.

Most importantly, notice that, given any cell, the cells to the left, right, up, and down all differ by a single bit. For example, the neighbors of the cell for the value 5 ("cell 5") are 1, 4, 7, each differing from 5 in the A, C, B variable, respectively. Cells 0 and 3 are not considered

Figure 1.1: A) The empty shell for a 3-variable kmap. Note it has 8 empty cells for the output of the function for the corresponding input. B) The 8 cells of the kmap filled in with the decimal equivalent of the 3-bit input corresponding to that cell. Note the bit order is A as the MSB and C as the LSB.

$A \backslash BC$	00	01	11	10
0				
1				

A

$A \backslash BC$	00	01	11	10
0	0	1	3	2
1	4	5	7	6

B

neighbors of cell 5 because they differ by two bits. Finally, notice that cell 4 and cell 6 differ by a single bit B, so they should be placed adjacent to one another, but are separated on the Kmap. To do this, imagine a 3-variable Kmap residing on the surface of a torus (doughnut) . The process of manipulating a 3-variable Kmap onto the surface of a torus is shown in Figure 1.2.

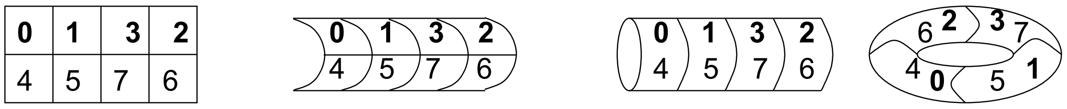


Figure 1.2: The folding and stretching required to transform a 3-variable Kmap onto the surface of a torus. The shaded numbers are on the back side of the surface.

When a Kmap is solved correctly the resulting SOP expression uses the minimum number of gates to realize the circuit in SOP form. Such a minimal SOP expression is referred to as a SOP_{min} expression.

1.1 Karnaugh Maps

Using a Kmap to determine a SOP_{min} realization of a function is a 4-step process.

Process 1.1: Solving a Kmap

This process is examined by determining the SOP_{min} expression for the function $G(A, B, C) = \sum m(1, 2, 5, 6)$. Note, this expression is the same function minimized earlier.

Step 1: Draw an empty kmap The first step is to draw an empty Kmap using the input variables of the function A, B, C. Since there are three input variables, this will be a three-variable kmap and look exactly like that in Table ??. Note that you should draw all 3-variable kmaps using this rectangular tabular format with the variables o the function in question in the upper left corner of the table.

$A \backslash BC$	00	01	11	10
0				
1				

Step 2: Place 1s in the Kmap for those inputs for which the function is to equal 1. Typically, the 0s of the function are omitted from the Kmap. It is understood that if you see a blank space in a Kmap, the function equals 0 for that input. The Kmap below shows the Kmap for G.

$A\backslash BC$	00	01	11	10
0		1		1
1		1		1

Step 3: Identify and circle pairs of adjacent 1s in the Kmap By adjacent, we mean cells whose inputs differ by a single bit. Since these neighbors lie up, down, let and right, we sometimes call these Manhattan neighbors, because in the epynomiously named city, streets are laid out in a clean grid and blocks consider neighboring when they share a street. Once you have found a pair of adjacent cells, circle them. Let's circle the pair of 1's in cell 1 and 5 as well as the pair of 1's in cell 2 and 6.

$A\backslash BC$	00	01	11	10
0		1		1
1		1		1

Step 4: Write the Boolean expression for the circled cells. The Boolean expression for a pair of adjacent 1 is formed by applying the simplification trick. Remember that this technique asks us to write down the input variables which do not change and discard the input variable which does change.

For the 1's in the red circle, $B = 0$ and $C = 1$ for both cells and the A variable changes. Hence, the Boolean expression for this grouping is $B'C$.

For the 1's in the blue circle, $B = 1$ and $C = 0$ for both cells and the A variable changes. Hence, the Boolean expression for this grouping is BC'.

Step5: OR together the circled Boolean expressions. Since we OR together the minterms when forming a canonical SOP expression, it makes sense that we should OR together the product terms found using the simplification trick. In our example, this step yields the SOP $_{\text{min}}$ expression $G(A, B, C) = B'C + BC'$.

To explore more fully how to solve Kmaps, consider the following truth table which lists seven functions $F \dots M$ each having three inputs. The SOP $_{\text{min}}$ expression is derived for each of these functions, along the way illustrating many of the properties needed to solve Kmaps.

A	B	C	F	G	H	I	J	K	L	M
0	0	0	0	0	0	1	1	1	1	0
0	0	1	0	0	1	1	0	1	1	0
0	1	0	0	1	0	0	1	1	0	0
0	1	1	1	1	1	0	0	0	1	0
1	0	0	1	0	0	0	1	0	1	0
1	0	1	1	0	1	1	1	1	1	0
1	1	0	0	0	0	0	0	1	0	0
1	1	1	0	1	1	1	0	1	1	0

$A\backslash BC$	00	01	11	10
0			1	
1	1	1		

Kmap for F

$F = AB' + A'BC$ The grouping of cell 4 and cell 5 yields the product term AB'. Cell 5 and cell 3 cannot be combined because they differ in two bits. What is to do be done with the single cell 3? Since this cell cannot be combined with another cell, it is just a minterm by itself. Sad perhaps, but perfectly legal. Hence, $F(A,B,C) = AB' + A'BC$.

$G = A'B + BC$ The natural question arising from this Kmap is, "Can cell 3 be reused in two different groups?" The answer is "Yes," and the reason can be demonstrated by performing some Boolean Algebra on the canonical SOP expression for G.

$A\backslash BC$	00	01	11	10
0			1	1
1			1	

Kmap for G

$$\begin{aligned} G(A,B,C) = \quad & A'BC' + A'BC + ABC = \\ & A'BC' + A'BC + A'BC + ABC = \\ & A'B(C' + C) + BC(A' + A) = \\ & A'B(1) + BC(1) = \\ & A'B + BC \end{aligned}$$

In the second line, the minterm $A'BC$ was duplicated using Law 3 of Boolean Algebra. Note, this is the same minterm covered twice in the Kmap. Consequently, if a cell of the Kmap is covered by three different groupings then it would have to be replicated three times in the Boolean Algebra simplification.

The grouping of cell 2 and cell 3 yields the product term $A'B$. The grouping of cell 3 and cell 7 yield the product term BC. Consequently, $G(A,B,C) = A'B + BC$. It is now possible to relate what happens in the symbolic expression when $(A,B,C) = (0,1,1)$ to the method used to solve the Kmap.

$H = C$ Grouping can be of size four. This can be explained by performing Boolean Algebra on the canonical SOP expression for H.

$A\backslash BC$	00	01	11	10
0		1	1	
1		1	1	

Kmap for H

$$\begin{aligned} H(A,B,C) = \quad & A'B'C + A'BC + AB'C + ABC = \\ & (B + B')A'C + (B' + B)AC = \\ & A'C + AC = \\ & (A' + A)C = \\ & C \end{aligned}$$

In general, grouping can be of size $2^i \times 2^j$ for integer i and j. It is a common mistake of students to make a grouping of size 3x2 – 3 is not a power of 2!

$I = A'B' + AC$ One does not have to form every possible grouping. It is a common error for students to include the term $B'C$ in the expression for I. The expression $B'C'$ does not cause the function to output an incorrect value. Rather, this expression is not necessary for the circuit to function properly. Hence, including the expression $B'C'$ would make the circuit larger by one more AND gate than necessary.

$A\backslash BC$	00	01	11	10
0	1	1		
1		1	1	

Kmap for I

$J = A'C' + AB'$ Grouping can be made over the edge of the Kmap using the doughnut (torus) property illustrated in Figure 1.2. Notice, the minterms in the grouping on the upper row differ in the B bit.

$A\backslash BC$	00	01	11	10
0	1			1
1	1	1		

Kmap for J

$K = A'B' + AC + BC'$ **or** $K = A'C' + B'C + AC$ A Kmap may have more than one correct solution.

$A\backslash BC$	00	01	11	10
0	1	1		1
1		1	1	1

Kmap for K

$L = B' + C$ Avoid the temptation to make a grouping of size 3×2. Instead, make two groupings of size 2×2 which overlap in two cells. Yes, overlapping big groupings is legal, too.

$A\backslash BC$	00	01	11	10
0	1	1	1	
1	1	1	1	

Kmap for L

$A\backslash BC$	00	01	11	10
0				
1				

Kmap for M

$M = 0$ This trivial function can be confusing when first encountered. Notice that regardless of the input, the output of the function is always 0. Hence, $M = 0$.

The process of "solving" a Kmap correctly leads to a minimal 2-level SOP expression (abbreviated SOP $_{\text{min}}$). SOP expressions are composed of two main levels of gates. A set of AND gates (level 1) leading into an OR gate (level 2). The NOT gates are ignored because they are both small and fast compared to AND and OR gates. The term "minimal" refers to the fact that the realization of the function requires the fewest possible gates among any 2-level SOP realizations.

In order to understand how to most efficiently solve a Kmap, some notation needs to be defined. An *implicant* is a legal grouping of 1s in a Kmap. An implicant which is not contained in any other implicant is called a *prime implicant* . An *essential prime implicant* is a prime implicant which covers a minterm not covered by any other prime implicant. For example, ABC is an implicant in the function $L(A, B, C)$, but it is not a prime implicant because it is contained in the essential prime implicant C.

When solving a Kmap, look for essential prime implicants and remove them from consideration. Removing the 1s from consideration does not mean removing the 1s from the problem. If needed, the 1s in an essential prime implicant can be used to form other groupings. After the essential prime implicants have been removed, look for *secondary essential prime implicants*, the essential largest groupings covering the remaining 1s. This identification of essential prime implicants goes on until either all the 1s are covered or a situation like the K function results. The K function has no essential prime implicants. At this point, resort to intuition (or brute force search) to minimize the number of groupings to cover the remaining 1s in the Kmap. Kmaps can be used to minimize functions in a variety of ways. One such use is discussed below.

$A\backslash BC$	00	01	11	10
0	1			1
1	1		1	1

Determine the SOP $_{\text{min}}$ expression for $F(A, B, C) = B'C' + BC' + ABC$. The term SOP $_{\text{min}}$ implies that a Kmap is required to solve the problem. In this case, figure out a way determine which region of the Kmap is described by each product term. For example, the product term $B'C'$ is equal to 1 when $(B, C) = (0, 0)$. Thus, 1s are placed in cell 0 and cell 4 of the Kmap. The product term BC' results in 1s being placed in cell 2 and cell 6. The product term ABC requires a 1 to be placed in cell 7. The resulting Kmap is shown in the margin. This Kmap can now be solved to determine the SOP $_{\text{min}}$ expression; $F(A, B, C) = C' + AB$. Incidentally, the grouping of cells 0, 2, 4, 6 in the Kmap above is affectionately referred to as a *Texas doughnut* because it has a big 2x2 grouping straddling the ends of the Kmap.

1.2 4-Variable Kmaps

The Kmap method can be adapted to work with functions of more than three variables. These larger Kmaps must be constructed so that adjacent cells differ by a single bit in order for the simplification trick to work. For example, a 4-variable Kmap is shown below with the decimal equivalent of the binary inputs shown in each cell.

$AB\backslash CD$	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

It is easy to verify that adjacent cells differ by a single bit. In addition, notice that adjacencies run across the top/bottom and left/right margins of the Kmap. In order to better understand this new structure, determine the SOP_{min} expressions for the following functions.

- $F(A, B, C, D) = \sum m(0, 1, 4, 5, 8, 9)$
- $G(A, B, C, D) = \sum m(0, 5, 7, 10, 11, 14, 15)$
- $H(A, B, C, D) = \sum m(0, 2, 3, 5, 6, 7, 8, 10, 11, 14, 15)$

$F = A'C' + B'C'$ Resist the temptation to make a grouping of size 3×2; 3 is not a power of 2. Notice, one of the 2×2 groupings, $B'C'$ spans the edge of the Kmap (Texas doughnut style).

$AB\backslash CD$	00	01	11	10
00	1	1		
01	1	1		
11				
10	1	1		

Kmap for F

$G = A'B'C'D' + A'BD + AC$ This example demonstrates an interesting point: The size of the grouping determines the number of variables in the SOP_{min} expression. For example, the grouping for covering one cell, $A'B'C'D'$, has four variables, the grouping covering two cells, $A'BD$, has three variables and the grouping covering four cells, AC, has two variables. The larger the grouping, the fewer variables that are required to describe the grouping, and consequently requires a smaller AND gate.

$AB\backslash CD$	00	01	11	10
00	1			
01		1	1	
11			1	1
10			1	1

Kmap for G

$H = B'D' + A'BD + C$ This solution is notable because it contains the *HyperDoughnut* grouping – the cells 0,2,8,10 forming the product $B'D'$. The only other notable feature in this Kmap is the large grouping of size 8, C.

$AB\backslash CD$	00	01	11	10
00	1		1	1
01		1	1	1
11			1	1
10	1		1	1

Kmap for H

1.3 5-Variable Kmaps

A 5-variable Kmap is a rearrangement of the rows of a 5-variable truth table such that adjacent cells of the Kmap differ by a single input bit. This rearrangement is accomplished by floating two, 4-variable Kmaps one above the other. A cell in a 5-variable Kmap can be combined with the cell to its left, right, below, above, up or down! That is, the three perpendicular directions along which adjacencies lie must be checked.

Determine the SOP_{min} expression for $F(A, B, C, D, E) = \sum m(0, 1, 2, 5, 7, 8, 10, 15, 16, 18, 23, 24, 26, 28, 29, 31)$. Each of the 4-variable Kmaps is labeled with one of the two possible values of A, 0, or 1. The decimal values of the inputs can be formed by converting the binary input $ABCDE$ for each cell into decimal. The 4-variable Kmap with $A = 0$ is labeled just like an ordinary 4-variable Kmap. The 4-variable Kmap with $A = 1$ is labeled in the same order as a regular 4-variable Kmap, except the numbering starts at 16 because the MSB is 1.

$BC \backslash DE$	00	01	11	10
00	1	1		1
01		1	1	
11			1	
10	1			1

$A = 0$

$BC \backslash DE$	00	01	11	10
00	1			1
01			1	
11	1	1	1	
10	1			1

$A = 1$

This Kmap is notable because it contains the granddaddy of all the border jumping groupings, the *ultra-doughnut*. This grouping occupies the eight corners of the 5-variable Kmap. A grouping of size four jumps between the two Kmaps. Other than this aspect, the solution is fairly straightforward. $F(A,B,C,D,E) = C'E' + A'B'D'E + CDE + ABCD'$

To conclude the Kmap concept, a few trends should be noted. First, there is a relationship between the number of variables in a Kmap and the number of neighbors.

Variables	Neighbors
3	3
4	4
5	5

The relationship is linear, that is a cell in a Kmap with N variables should have N neighbors. By considering the simplification trick, this relationship should make sense. Two cells are adjacent if their binary representations differ by a single bit. An N-bit number has N positions where a single bit could be changed, thus it has N neighbors in the Kmap.

Also there is a relationship between the number of 1s in a grouping and the number of variables appearing in the grouping's product term. For example, in the $G(A,B,C,D)$ function above, the grouping of a single minterm was described by the product term $A'B'C'D'$ containing four variables. The largest grouping of four minterms was described by the product term AC containing two variables. The following table describes this relationship for a 5-variable function.

Number of 1's	Variables
32	0
16	1
8	2
4	3
2	4
1	5

This table demonstrates why making the groupings as large as possible is desirable, large groupings have smaller AND gates.

1.4 Multiple Output Circuits

In the previous chapter, digital systems with more than one output were considered. Multiple output functions are realized by realizing each output independently of the others. For example, consider a digital system with three inputs A, B, C and two outputs $F(A,B,C)$ and $G(A,B,C)$ shown in the truth table below:

A	B	C	F	G
0	0	0	0	0
0	0	1	0	0
0	1	0	0	1
0	1	1	1	1
1	0	0	1	0
1	0	1	1	0
1	1	0	0	0
1	1	1	0	1

If the functions are solved independently of one another, $F(A,B,C) = AB' + A'BC$ and $G(A,B,C) = BC + A'BC'$. The only connection the two circuits have to each other is their inputs, both share the same (A,B,C). Thus, when $(A,B,C) = (1,0,1)$ $F = 1$ and $G = 0$ just as expected according to the truth table. The circuit diagram of the F and G functions is shown in Figure 1.3.

$A\backslash BC$	00	01	11	10
0			1	
1	1	1		

$F(A,B,C) = AB' + A'BC$

$A\backslash BC$	00	01	11	10
0			1	1
1			1	

$G(A,B,C) = BC + A'BC'$

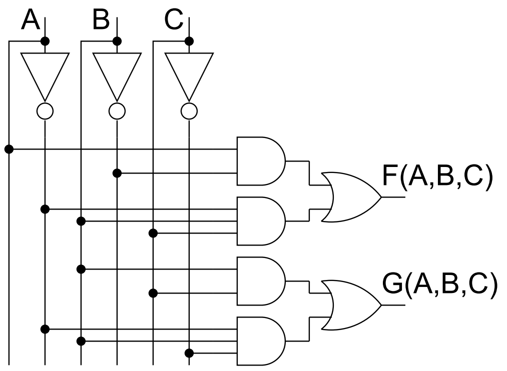


Figure 1.3: Two functions, F and G, which share inputs.

However, when you a design circuit that has multiple outputs which share the same inputs, there are efficiencies that you can extract if you share product terms. A shared product term is an AND gate that can be used in the realization of more than one function. For example, consider the two functions $H(A,B,C) = \sum m(1,4,5,6)$ and $I(A,B,C) = \sum m(1,2,3,5)$.

The Kmaps and the solutions for these two functions are shown in the margins. Notice that the grouping $B'C$ appears in both SOP $_{\text{min}}$ expressions. This AND gate need appear only once in the circuit diagram because its output can be directed to both OR gates which form the outputs for H and I. The circuit diagram for these two functions is shown in Figure 1.4.

$A\backslash BC$	00	01	11	10
0		1		
1	1	1		1

$H(A,B,C) = \sum m(1,4,5,6) = B'C$
AC'

$A\backslash BC$	00	01	11	10
0		1	1	1
1		1		

$I(A,B,C) = \sum m(1,2,3,5) = B'C$
AB

1.5 "Don't Cares"

There are situations when engineers "don't care" what the output of a digital system is for certain inputs. This situation often happens when a particular input will never occur. For

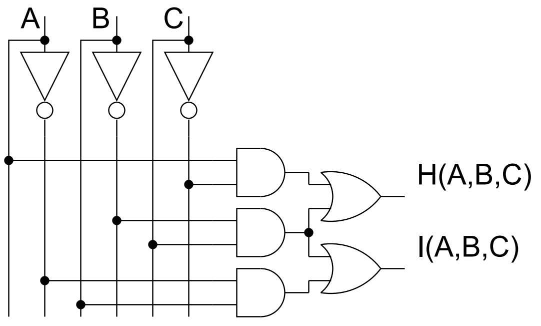


Figure 1.4: Two functions which share the product term $B'C$.

example, consider the a digital system which classifies its inputs as either even or odd. It has four bits of input $a_3a_2a_1a_0$, and one bit of output, F. The 4-bit input represents a decimal number, $0 \leq A \leq 9$, the inputs $10 \leq A \leq 15$ should never be applied. The output equals 1 when A is even (0 is considered an even number) and outputs 0 when A is odd.

As a first attempt to solve this problem, ignore the inputs corresponding to $10 \leq A \leq 15$ and leave the corresponding outputs blank. The outputs are to be defined for inputs $0 \leq A \leq 9$. The resulting Kmap and its solution is shown in the margins.

$a_3a_2\backslash a_1a_0$	00	01	11	10
00	1			1
01	1			1
11				
10	1			

$F(a_3, a_2, a_1, a_0) = a_3'a_0' + a_2'a_1'a_0'$

The solution shown in the margin will certainly work correctly, since the outputs are correctly defined for the legitimate inputs. However, the realization could have been made more efficient if the fact that the inputs $10 \leq A \leq 15$ will never be applied had been taken into consideration. Then, it does not matter what the circuit outputs when $10 \leq A \leq 15$. Notice, that in the first solution, implicitly the illegal inputs were treated as odd numbers; the circuit would output a 0 for $10 \leq A \leq 15$. Since these are illegal inputs, they are assigned any convenient value with an eye on making the final realization as efficient as possible. To denote this freedom in assigning the output of the function either value, an "X" is placed in any cell of the Kmap where the output doesn't matter. These Xs are referred to as "don't cares".

The utility of "don't cares" lies in the fact that they can be used to make groupings larger and consequently the realization of the function more efficient. If an X can be used to make a grouping larger, then it should be included in that grouping. Including "don't cares" in a group will cause the circuit to output 1 for the input corresponding to the "don't care". If, on the other hand, an X cannot be used in any grouping, then leave it uncovered. The circuit will output a 0 for the uncovered "don't care".

To better understand how "don't cares" are used, they are included into the even/odd function for inputs $10 \leq A \leq 15$ and the resulting Kmap shown in the margin is solved.

$a_3a_2\backslash a_1a_0$	00	01	11	10
00	1			1
01	1			1
11	X	X	X	X
10	1		X	X

$F(a_3, a_2, 2_1, a_0) = a_0'$

By using the "don't cares" in cells 10, 12, and 14, a grouping of size eight is formed. This grouping reduces the cost of the solution to a single inverter.

"Don't cares" are included in the abbreviated canonical SOP form by writing a "*d*" followed by a list of inputs for which the output is unimportant. For example, the even/odd function is described as $F(A,B,C,D) = \sum m(0,2,4,6,8) + \sum d(10,11,12,13,14,15)$.

"Don't cares" can be used on the input variables of a truth table in order to compress the size of the truth table. Two rows can be combined when their outputs are the same and their inputs are adjacent (in the Kmap sense) to one another. A single "don't care" on an input of a row of a truth table generates two rows; one with the "don't care" set to 1 and another with the "don't care" set to 0. For example, the following two truth tables describe the same function.

A	B	C	F(A,B)
x	0	x	0
x	1	0	1
0	1	1	0
1	1	1	1

Compressed Truth Table

A	B	C	F(A,B)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Full Truth Table

The row $(x,1,0)$ generates two rows in the full truth table, $(0,1,0)$ and $(1,1,0)$. The row $(A,B,C) = (x,0,x)$ has two "don't cares". Since the values of these "don't cares" can be selected independent of one another, this row generates four rows in the full truth table, $(0,0,0)$, $(0,0,1)$, $(1,0,0)$, and $(1,0,1)$. In general, a row with N "don't cares" generates 2^N rows in the full truth table. The motivation for using "don't cares" will become more clear in later chapters when digital systems with six or more inputs are encountered.

1.6 Minimizing to POS

So far, the chapter has focused on finding efficient SOP $_{\text{min}}$ realizations of circuits. However, Section **??** provided a function whose POS realization was less costly than the SOP realization. It is not difficult to imagine functions for which the POS $_{\text{min}}$ realization is less costly than the SOP $_{\text{min}}$ realization. In order to derive the POS $_{\text{min}}$ realization of a function F, the function is transformed – put into a Kmap, a SOP $_{\text{min}}$ realization found, and then the SOP $_{\text{min}}$ realization transformed into a POS $_{\text{min}}$ realization for F. In order for this process to work, the two transformations of the function will have to "undo" one another so they have no net effect on the function.

In order to cover all possibilities, the transform from any of the starting forms into one of the ending forms is investigated.

$\sum m$		
$\prod M$	$\longrightarrow$	SOP $_{\text{min}}$
SOP		POS $_{\text{min}}$
POS		
Starting Form		Ending Form

Eight potential transforms need to be explored. For each transformation, a plan is developed; a series of steps. The following five facts become the steps in all these plans.

1 Negating a Kmap for F negates F. If all the bits in a Kmap for a function F are flipped, then it is the same as negating the output of F.

2 Negating a SOP expression for F yields a POS expression for F'. This statement is based on Law 9 and Law 9D of Boolean Algebra. Law 9 states that $(x+y)' = x'y'$. Law 9D states that $(xy)' = x' + y'$. The transformation from SOP to POS is illustrated in the following example.

$F = A'B'C' + A'BC + AB' + C$	negate F
$F' = (A'B'C' + A'BC + AB' + C)'$	Law 9
$F' = (A'B'C')'(A'BC)'(AB')'(C)'$	Law 9D
$F' = (A+B+C)(A+B'+C')(A'+B)(C')$	

The shortcut typically used to determine the POS expression is to replace all ANDs and ORs and negate each variable.

3 Negating a POS expression for F yields a SOP expression for F'. The proof of this statement is derived by examining the previous algebraic example from bottom to top.

4 $\sum m(list) = \prod M(list')$. *list'* denotes the decimal numbers not in *list*. Since *list* describes those inputs for which the function equals 1, *list'* describes those inputs for which the function equals 0.

5 $F'' = F$. This expression is just Law 4 of Boolean Algebra.

The plans used to transform between forms consist of a sequence of steps. At each step, the function (or its complement) is represented in one of its various forms (Kmap, SOP, $\sum m$, etc..). Step 1 always describes the given form of the function. Likewise, the last step always describes the desired form of the function. The action between the steps is implicit from the beginning and ending forms.

To better understand how a plan is put together and how the transformations are performed, consider a plan for the transformation of a $\sum m$ expression into a SOP $_{\text{min}}$ realization.

Process 1.2: Determine SOP $_{\text{min}}$ given a $\sum m(\ldots)$ expression.

This should be the (by now) familiar process of drawing a kmap with the correct number of variable and putting 1's in the cells where the function equals 1 and then solving the kmap.

The Plan:

Step	1	2	3
Function	F	F	F
Form	$\sum m(\ldots)$	Kmap	SOP $_{\text{min}}$

The plan consists of three steps. The function in all three steps is F. In Step 1, the function F is written in the $\sum m(\ldots)$ notation. In the second step, the function is placed into a Kmap using the process outlined in this chapter. In the third step, the SOP $_{\text{min}}$ expression is derived from the Kmap.

Another familiar transform is handled next.

Process 1.3: Determine SOP $_{\text{min}}$ given a SOP expression.

The only difference in this transformation is the beginning form. The discussion on page 6 illustrates how to put a SOP Boolean expression into a Kmap. Once in a Kmap, the remaining steps should be clear.

The Plan:

Step	1	2	3
Function	F	F	F
Form	SOP	Kmap	SOP $_{\text{min}}$

The next transformation has a worked out solution to guide you through the steps of the process.

Process 1.4: Determine POS $_{\text{min}}$ given a $\sum m(\ldots)$ expression.

The Plan:

Step	1	2	3	4	5
Function	F	F	F'	F'	F"=F
Form	$\sum m(\ldots)$	Kmap	Kmap'	SOP $_{\text{min}}$	POS $_{\text{min}}$

In Steps 1 and 2 of the plan, the function, F, is placed into a Kmap. In Step 3, the Kmap is inverted (all the 1s replaced with 0s and 0s replaced with 1s). With this inversion, the SOP $_{\text{min}}$ expression for F' determined from the Kmap can be negated, yielding a POS $_{\text{min}}$ expression for $F'' = F$.

An example shows how this plan is put into practice. Determine the POS $_{\text{min}}$ expression for $F(A,B,C) = \sum m(3,4,5)$.

Step 1 $F(A,B,C) = \sum m(3,4,5)$.

Step 2 The Kmap for F

$A\backslash BC$	00	01	11	10
0			1	
1	1	1		

Step 3 The Kmap for F'

$A\backslash BC$	00	01	11	10
0	1	1		1
1			1	1

Step 4 $F'(A,B,C) = A'B' + AB + BC'$

Step 5 Relies on helpful fact 2, negating a SOP expression for F yields a POS expression for F'. We will work through the steps here as a review.

$$\begin{aligned} F''(A,B,C) = F(A,B,C) &= (A'B' + AB + BC')' \\ &= (A'B')'(AB)'(BC')' \\ &= (A'' + B'')(A' + B')(B' + C'') \\ &= (A + B)(A' + B')(B' + C) \end{aligned}$$

The POS $_{\text{min}}$ expression for F can be checked to make sure that it generates the correct outputs, by plugging in values for A, B, C. The SOP $_{\text{min}}$ expression for F can be determined from the Kmap in Step 2 as $F(A,B,C) = AB' + A'BC$. This fact demonstrates, in general, the SOP $_{\text{min}}$ and POS $_{\text{min}}$ realizations have different costs.

Process 1.5: Determine POS $_{\text{min}}$ given a SOP expression.

The Plan:

Step	1	2	3	4	5
Function	F	F	F'	F'	F"=F
Form	SOP	Kmap	Kmap'	SOP $_{\text{min}}$	POS $_{\text{min}}$

This transformation is the same as the $\sum m(\ldots)$ to POS $_{\text{min}}$ transformation from Step 2 onward.

Process 1.6: Determine the POS $_{\text{min}}$ given a $\prod M(\ldots)$ expression.

The Plan:

Step	1	2	3	4
Function	F	F	F	F
Form	$\prod M(\ldots)$	$\sum m(\ldots)$	Kmap	SOP $_{\text{min}}$

This transformation utilizes the fourth helpful fact. The swapping of the list of 0s for a list of 1s. This step confuses many first time students so lets work an example to show how this plan is put into practice. Determine the POS $_{\text{min}}$ expression for $F(A,B,C) = \sum m(0,1,2,6,7)$.

Step 1 $F(A,B,C) = \sum M(0,1,2,6,7)$.

Step 2 $F(A,B,C) = \sum m(3,4,5)$.

Step 3 Follow steps 2 onward in process 4.

Process 1.7: Determine the POS $_{\text{min}}$ given a $\prod M(\ldots)$ expression.

The Plan:

Step	1	2	3	4	5	6
Function	F	F	F	F'	F'	F"=F
Form	$\prod M(\ldots)$	$\sum m(\ldots)$	Kmap	Kmap'	SOP $_{\text{min}}$	POS $_{\text{min}}$

In Step 2, the list of maxterms, given in Step 1, is transformed into a list of minterms. The Kmap is negated in Step 4 so that the SOP $_{\text{min}}$ description of F' can be negated, yielding a POS $_{\text{min}}$ expression for F.

Process 1.8: Determine SOP $_{\text{min}}$ given a POS expression.

The Plan:

Step	1	2	3	4	5
Function	F	F'	F'	F"=F	F
Form	POS	SOP	Kmap	Kmap'	SOP $_{\text{min}}$

This is an interesting transformation because F must be negated before being placed into a Kmap. Since F' is in the Kmap, the Kmap must be negated so that the solution to the Kmap yields a SOP $_{\text{min}}$ expression for F.

Process 1.9: Determine the POS $_{\text{min}}$ given a POS expression.

The Plan:

Step	1	2	3	4	5
Function	F	F'	F'	F'	F"=F
Form	POS	SOP	Kmap	SOP $_{\text{min}}$	POS $_{\text{min}}$

An example shows how this plan is put into practice. Determine the POS $_{\text{min}}$ expression for $F(A,B,C) = (A'+B+C')(A+C')B'$.

Step 1 $F(A,B,C) = (A'+B+C)(A+C')B'$.

Step 2 Relies on helpful fact 3, negating a POS expression for F yields a SOP expression for F'.

$$\begin{aligned} F'(A,B,C) &= ((A'+B+C)(A+C')B')' \\ &= (A'+B+C)'+(A+C')'+B'' \\ &= A''B'C'+A'C''+B \\ &= AB'C'+A'C+B \end{aligned}$$

Step 3 The Kmap for F'

$A\backslash BC$	00	01	11	10
0		1	1	1
1	1		1	1

Step 4 $F'(A,B,C) = AC'+A'C+B$

Step 5 Relies on helpful facts 5 and 2.

$$\begin{aligned} F''(A,B,C) = F(A,B,C) &= (AC'+A'C+B)' \\ &= (AC')'(A'C)'B' \\ &= (A'+C'')(A''+C')B' \\ &= (A'+C)(A+C')B' \end{aligned}$$

1.7 Espresso

The Kmap minimization method can be applied to problems with up to eight variables. Beyond eight variable, the process becomes too tedious and error-prone to be practical. Functions with 20 variables are not uncommon, but would require a truth table containing over a million rows! An algorithmic approach is needed to handle such large instances. Espresso `https://en.wikipedia.org/wiki/Espresso_heuristic_logic_minimizer` is a 2-level logic minimization tool, that while not guaranteed to find a minimal solution, does a good job on most functions. Since Espresso was written before the days of visual interfaces, it is a small program, run from a command line interface. (A command line interface is a text interface to the programs and operating services available to the user of a computer.) The operating system prompts the user by displaying a ">" character in a window. The user then types in the name of the program or service they want to access. Typing "espresso" at the command line prompt invokes the Espresso program.

Espresso is simple to use, just create the truth table for a function in a text editor, then run Espresso on the file. The input file for Espresso contains:

1. **Comments.** Comments are always preceded with the # symbol. If a comment is encountered, Espresso ignores the rest of the line.

2. **The number of inputs.** The number of inputs is described by the statement `.i` followed by a number describing the number of inputs to the function. The `.i` must be included in an Espresso file.

3. **The number of outputs.** The number of outputs is described by the statement `.o` followed by a number describing the number of outputs from the function. The `.o` must be included in an Espresso file.

4. **The labels for the inputs.** The labels for the inputs are described by the statement `.ilb` followed by a list of names of the inputs separated by spaces. The `.ilb` does not have to be included in an Espresso file. If not included, then Espresso assigns its own names to the inputs.

5. **The labels for the output(s).** The labels for the outputs are described by the statement `.ob` followed by a list of names of the outputs separated by spaces. The `.ob` does not have to be included in an Espresso file. If not included, then Espresso assigns its own names to the outputs.

6. **The truth table.** The truth table is organized with the input bits on the left and the output(s) bit(s) on the right. "Don't cares" in the inputs or outputs are denoted with a minus sign `-`. The order of the rows in the truth table is unimportant.

The following is an example Espresso input file for the function $F(a,b,c) = \sum m(1,3,6,7)$.

```
# File: simple.txt
# Name: <your name>
#       <course name>
# Date: <semester>
# Desc: F(a,b,c) = sum m(1,3,6,7)
.i 3
.o 1
.ilb a b c
.ob F

000 0
001 1
010 0
011 1
100 0
101 0
110 1
111 1
```

This file must be created in text format – do not create this document in a "word processor" such as MS Word or WordPerfect. NotePad, Edit, emacs, or vi are preferable choices for creating this file. Save the example Espresso file as `simple.txt`. The command line

Running Espresso on the example file produces the following output.

```
> espresso simple.txt
# File: simple.txt
# Name: <your name>
#       <course name>
# Date: <semester>
# Desc: F(a,b,c) = minterms (1,3,6,7)
.i 3
.o 1
.ilb a b c
.ob F
.p 2
11-     1
0-1     1
.e
```

Espresso echoes back comments, the number of inputs, outputs, and labels defined in the input file, and identifies the number of product terms used in the realization. In the example, the `.p 2` statement means that the solution found by Espresso required two product terms. The two lines after the `.p 2` statement describe the realization in a programmable logic array (PLA) format. In the PLA format, each row represents a product term. The left three columns of each row represent the state of the inputs variable in the product term. The variables are in the same order as they were defined in the truth table. The state of a variable can be "1", "0", or "-".

- 1 means, include the variable in the product.
- 0 means, include the negated variable in the product.
- - means, exclude the variable from the product.

Thus, the row "11-" represents the product term AB and the row "0-1" represents the product term $A'C$. The column of 1s to the right of these bits describes which product terms to use in the realization of the function. A 1 means to include the product term in the function, 0 means exclude the product term from the function – and is seen only for two or more outputs.

Putting both of Espresso's product terms together yields the optimal solution $F = A'C + AB$. The Kmap for this function and its SOP $_{\text{min}}$ solution is shown in the margin.

$A\backslash BC$	00	01	11	10
0		1	1	
1			1	1

$F(A,B,C) = A'C + AB$

Command line parameters can be included to specify options, changing the behavior of Espresso. All options consist of a minus sign, a letter, and a command, placed between "espresso" and the name of the input file. As an example the `-o eqntott` option will force Espresso to generate a symbolic output instead of the default PLA output.

```
> espresso -o eqntott simple.txt
#comments
F = (a&b) | (!a&c);
```

The entire set of options is listed by typing `espresso -help` at the command prompt. The `-epos` option inverts the truth table, yielding a SOP solution for F'. The POS solution is generated by inverting Espresso's output using the second helpful fact on page 12. This option allows a digital designer to quickly compare the cost of a SOP and POS realization.

Generated output can be saved by Espresso by redirecting standard output to a file. This option is indicated by putting a "greater than" symbol after a command, followed by the name of the output file. Think of the "greater than" symbol as a funnel, taking the streaming output of the Espresso program and funneling it into the output file. For example, the output of the previous Espresso example can be saved into a file called `simple.out` using the following command.

```
a:\> espresso -o eqntott simple.txt > simple.out
```

Espresso can solve design problems involving multiple outputs. The following Espresso file describes the example on page 9, where $F(A, B, C) = \sum m(1, 4, 5, 6)$ and $G(A, B, C) = \sum m(1, 2, 3, 5)$.

```
# File: harder.txt
# Name: <your name>
#       <course name>
# Date: Fall 2020
# Desc: F(a,b,c) = minterms(1,4,5,6)
#       G(a,b,c) = minterms(1,2,3,5)

.i 3
.o 2
.ilb a b c
.ob F G


000 00
001 11
010 01
011 01
100 10
101 11
110 10
111 00
.e
```

The output from Espresso is:

```
a:\> espresso harder.txt
# Comments
.i 3
.o 2
.ilb a b c
.ob F G
.p 3
1-0 10
01- 01
-01 11
.e
```

Espresso used three product terms to realize the functions AC', $A'B$, and $B'C$. The two columns to the right of the product terms describe which product terms are used to realize each outputs. Symbolically, the solution found by Espresso is:

- $F(A, B, C) = AC' + B'C$
- $G(A, B, C) = A'B + B'C$

The circuit diagram of the realization for the three functions is shown in Figure 1.4. Notice, the connections – denoted by dots – between the inputs and the AND gates has a pattern which is similar to the PLA description of the product terms by Espresso. Likewise, the pattern formed by the connections of the AND gates and OR gates is similar to the rightmost three columns of the Espresso output. This similarity is no accident; the designers of Espresso used a generalized layout of the of the circuit diagram in Figure 1.4 as a template to describe the output.

1.8 Exercises

1. **(6 pts.)** Design a circuit called DECODE. DECODE has two bits of input S, D and two bit of output y_1y_0. If $S = 0$ then $y_0 = D$ and $y_1 = 0$ else if $S = 1$ then $y_0 = 0$ and $y_1 = D$.

   a) Write down the truth table for the DECODE function.

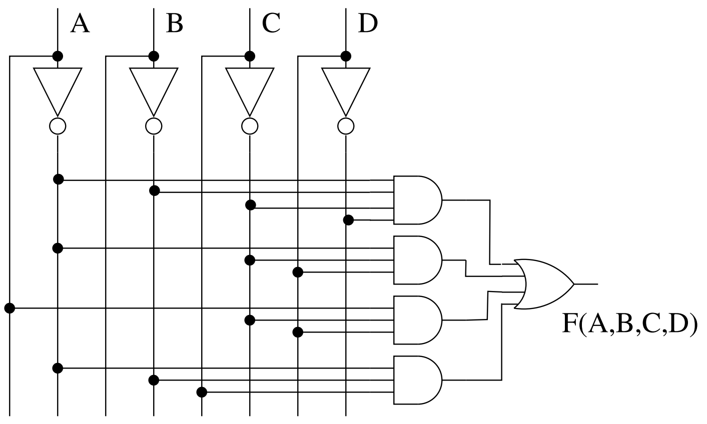
   b) Determine the SOP $_{\text{min}}$ realization for DECODE.

2. **(6 pts.)** Design a circuit called FULLADD. FULLADD has three bits of input a, b, c and two bits of output s_1s_0. The output represents the sum of the three bits.

   a) Write down the truth table for the FULLADD function.

   b) Determine the SOP $_{\text{min}}$ realization for FULLADD.

3. **(4 pts.)** Determine SOP $_{\text{min}}$ expression for the following circuit and draw the circuit using the fewest number of gates possible.



4. **(8 pts.)** Design a digital system with four bits of inputs $I_3I_2I_1I_0$ and two bits of outputs O_1O_0. At least one of the inputs is always equal to 1. The output encodes the index of the most significant 1 in the input. For example, if $I_3I_2I_1I_0 = 0101$, then the index of the most significant 1 is 2, hence $O_1O_0 = 10$. Submit:

   - The truth table.
   - SOP $_{\text{min}}$ expression for O_1 and O_0.

5. **(8 pts.)** Design a 4-input $a_1a_0b_1b_0$, 4 -output $O_3O_2O_1O_0$ digital system. $A = a_1a_0$ and $B = b_1b_0$ represent 2-bit binary numbers. The output should be the product (multiplication) of the inputs, that is $O = A * B$. In addition to determining the output, determine the number of bits of output. Submit:

   - Truth tables.
   - Minimal SOP expression for the outputs.

6. **(8 pts.)** Design a 4-bit Gray-code to binary converter. A 4-bit gray-code is a sequence of 4-bit values where successive values differ by a single bit. For this problem use the sequence: 0000, 0001, 0011, 0010, 0110, 0111, 0101, 0100, 1100, 1101, 1111, 1110, 1010, 1011, 1001, 1000. The index of the 4-bit gray code is its binary value. For example, the 4-bit gray code 0111 is at index 5, therefore when presented with 0111 on its input, the converter should output 0101. Submit:

   - A truth table for the converter.
   - Four k-maps for the converter.
   - SOP $_{\text{min}}$ expression for the outputs, no product sharing please (use the `-Dso` command line option).
   - Espresso file for the converter
   - Espresso output in PLA format
   - Compare the number of gates required in your solution versus the number of gates required by Espresso.

7. **(4 pts. each)** Determine SOP $_{\text{min}}$ expression for:

   a) $F(A,B,C) = \sum m(0,1,3,4,5)$
   b) $F(A,B,C,D) = \sum m(1,5,6,7,11,12,13,15)$
   c) $F(A,B,C,D) = \sum m(0,2,5,6,8,11,12,13,14,15)$
   d) $F(A,B,C,D,E) = \sum m(0,8,9,10,13,15,22,26,29,30,31)$
   e) $F(A,B,C,D,E) = \sum m(0,2,4,5,7,10,13,15,18,21,24,26,28,29)$

8. **(4 pts. each)** Determine SOP $_{\text{min}}$ expression for:

   a) $F(A,B,C,D) = \sum m(4,7,9,12,13,15) + \sum d(0,1,2,3,10,14)$
   b) $F(A,B,C,D) = \sum m(0,1,5,7,10,14,15) + \sum d(2,8)$
   c) $F(A,B,C,D) = \sum m(0,1,3,4,15) + \sum d(10,12)$
   d) $F(A,B,C,D,E) = \sum m(2,3,5,7,11,13,17,19,29,31) + \sum d(1,4,9,16,25)$
   e) $F(A,B,C,D,E) = \sum m(2,3,6,10,12,13,14,18,25,26,28,29) + \sum d(11,27)$

9. **(8 pts. each)** Determine SOP $_{\text{min}}$ and POS $_{\text{min}}$ expressions for:

   a) $F(A,B,C,D) = \sum m(0,1,2,5,8,10,13,15)$
   b) $F(A,B,C,D) = \prod M(0,4,6,10,11,12)$
   c) $F(A,B,C,D) = \sum m(0,5,7,10,11,14) + \sum d(3,12,15)$
   d) $F(A,B,C,D) = \prod M(2,6,7,9,15) * \prod d(4,12,13)$
   e) $F(W,X,Y,Z) = WX'Z' + X'YZ + W'Y'Z + XYZ + WXY'$
   f) $F(W,X,Y,Z) = (W + X' + Y')(W' + Z')(W + Y')$

   Hint, the negation of a "Don't care" is a "Don't care".

10. **(3 pts.)** While grading homework for a digital design class the following question/answer pair is encountered. What is the problem with the answer given?

    Question: Generate the POS $_{\text{min}}$ expression for $F(A,B,C) = \sum m(2,3,4,5)$
    Answer: $F(A,B,C) = (A + B')(A' + B)$

11. **(6 pts.)** Determine the SOP_{min} realization of the following function.

A	B	C	D	F(A,B,C,D)
x	1	1	x	0
0	x	0	1	0
x	x	0	0	x
x	0	1	x	1
1	x	0	1	1

12. **(6 pts.)** What is the worst function SOP_{min} of 3 variable that can be created? That is, define a function whose minimal SOP form has the largest possible number of product terms. What is the largest number of product terms that a 4-variable SOP_{min} expression can have? How about N variables?

13. **(16 pts.)** Sometimes a logic circuit needs to output a logic 0 in order to produce some behavior. For example, an LED can be attached to a digital circuit output so that it lights up when the circuit outputs a 0. This response is called an active low output; the output device is *active* then the digital output is *low*.

Build a digital circuit that takes as input two 2-bit numbers, A and B. The circuit has three outputs which drive three LEDs labeled G, L, and E. The G LED should be illuminated when A>B. The L LED should be illuminated when A<B. The E LED should be illuminated when A=B. The LEDs are illuminated when the circuit outputs a 0, otherwise they are turned off.

Determine SOP_{min} expression for the G, L and E outputs. Determine POS_{min} expression for the G, L and E outputs.

Appendix A

74LS00 Data Sheets

Since the device documentation for the TI chips is covered by TI's copyright it was decided to leave these pages out of this text. The documents discussed in the text can be found onilne at:
    http://focus.ti.com/lit/ds/symlink/sn74ls00.pdf

Index

Doughnut
- Hyper, 7
- Ultra, 8

doughnut, 3, 5
- Texas, 6

Implicant, 6
- Essential Prime, 6
- Prime, 6

product term: shared, 9

SOP
- SOP $_{\text{min}}$, 3

trick
- simplification, 2