Using Karnaugh Maps in Software Requirements Analysis

by Anthony S. Cantone and Yawa E. Adonsou
China Lake, California

Faulty requirements leading to design deficiencies have been shown to be an avoidable root cause of many product failures. This paper is an effort to push the boundaries of system safety by proposing a novel approach for discovering faulty or missing software requirements by adapting a proven methodology heretofore used in circuit analysis. Karnaugh Mapping is employed in Application-Specific Integrated Circuit (ASIC) design to minimize power consumption, facilitate temperature control, increase functionality and minimize the number of physical logic gates. Karnaugh Maps (K-Maps) are ideally suited to impose order on logical requirements that describe the operation of electronic circuits. With the assumption that software requirements are expressible as logical statements, this paper assesses the ability of Karnaugh Mapping to effectively deconstruct and rationalize developmental requirements in the analysis of software and seeks to demonstrate that K-Maps can be used not only to minimize the number of requirements, but also to detect missing requirements. The analysis conducted in the course of developing this paper indicates that K-Maps can effectively identify faulty requirements in two examples of varying complexity, provided that sematic conventions are established and observed.

Introduction

Faulty software requirements have been implicated in safety-related incidents over the years. Reference 14 contains an example of a missing requirement for labeling the field for hours, minutes or seconds in the software used to program some SynchroMed implantable pumps. This led to two deaths and seven injuries. Medtronic recalled the software on September 24, 2004 and replaced it with new software that labels the time fields.

This paper presents K-Maps as a tool that can aid in meeting that challenge, particularly with regard to the detection of missing requirements and resolution of duplication of a set of related requirements. This paper will introduce techniques associated with K-Maps as used in other technical areas and demonstrate their applicability in software requirements analysis.

The objective of this paper is to assist the reader in using K-Maps as a requirements analysis tool. In particular, as a result of studying the techniques presented in this paper, the reader should be able to detect missing requirements and minimize a set of duplicate related requirements.

This paper describes the use of K-Maps as an analysis tool for analyzing software requirements. The *Karnaugh Map Background* subsection reviews the history of K-Maps as a tool employed in circuit design. The *Key Concepts* subsection introduces five key concepts used in this paper: Boolean Algebra, minterms, the relationship between truth tables and K-Maps, mapping from truth tables to K-Maps, and Hamming Distance. The *Requirements* subsection discusses safety aspects of missing requirements, standard forms of requirements, the concept of related requirements, the relationship between Karnaugh Map use for switching circuits and requirements, and examples of how K-Maps are used for requirements analysis. The *Applications of Karnaugh Maps in Requirements Analysis* subsection presents a list of steps that can be used to map from requirements to K-Maps, and a discussion of using K-Maps for requirement minimization. The *Other Uses of K-Maps* subsection introduces another possible use of K-Maps in requirements analysis. The *K-Maps and Complex System Requirements* subsection discusses more complex system requirements and higher order K-Maps. The *Conclusion* subsection is a summary of the paper.

Karnaugh Map Background

K-Maps provide a convenient method for simplifying Boolean Algebra expressions. In 1881, Allan Marquand built a mechanical logical machine and developed logical diagrams (also known as Marquand diagrams) [Ref. 1] associated with his machine at Princeton University. The President of Princeton decided that Marquand's approach to teaching logic was “unorthodox and uncalvinistic.” Hence, in 1883, Marquand was offered a position teaching art history, a position he held until his death [Ref. 2].

In 1952, Edward Veitch rediscovered the Marquand diagram, applying it to the problem of minimizing switching circuits. Maurice Karnaugh created what is now known as Karnaugh Maps in 1953 [Ref. 3], as a refinement of Edward Veitch's Veitch chart [Ref. 4].

A K-Map [Ref. 13] is a table of cells, each cell containing a value of a Boolean expression for a unique combination of input Boolean variable values. K-Maps, a proven methodology commonly used in switching circuit analysis, are employed in ASIC design to minimize power consumption, facilitate temperature control, increase functionality, detect possible race conditions, and minimize the number of physical logic gates. They are ideally suited to imposing order on logical requirements for electronic circuits.

Key Concepts

Boolean Algebra: A short review of Boolean algebra is presented in this subsection. Boolean Algebra [Ref. 5] is a set of symbols that take binary values, such as True and False, or 0 and 1, along with two closed binary operations, denoted by the symbols "+" and "·". The symbol "+" indicates inclusive "or" and the symbol "·" indicates "and." Juxtaposition is often used in place of "·" so that "ab" is considered equivalent to "a·b." Two symbols are in juxtaposition when they are placed side by side without a binary operation symbol between them. The binary operations satisfy the properties in Equations 1 through 6. The symbol "~", placed before a symbol, is used to denote the complement of that symbol. For example, if "a" has the Boolean value True, then "~a" has the Boolean value False.

The following identities are useful when working with K-Maps: a, b and c are variables that take on Boolean values True or False; 1 or 0; or Yes or No.

$$a + b = b + a;\ ab = ba \text{ (\textbf{Commutative laws of "+" and "·"})} \tag{1}$$

$$(a + b) + c = a + (b + c);\ (ab)c = a(bc) \text{ (\textbf{Associative laws of "+" and "·"})} \tag{2}$$

$$a(b + c) = ab + ac \text{ (\textbf{First distributive law:} "·" distributes over "+")} \tag{3}$$

$$a + bc = (a + b)(a + c) \text{ (\textbf{Second distributive law:} "+" distributes over "·")} \tag{4}$$

Note that the second distributive law permits factoring expressions such as a + bc, which cannot be factored in ordinary algebra.

The following De Morgan's Law identities are also helpful:

$$a + 0 = a;\ a + 1 = 1 \text{ (\textbf{Annihilator for "+"})} \tag{5}$$

$$a + a = a \text{ (\textbf{Idempotence of "+"})} \tag{7}$$

$$a·0 = 0;\ a·1 = a \text{ (\textbf{Annihilator for "·"})} \tag{6}$$

$$a·a = a \text{ (\textbf{Idempotence of "·"})} \tag{8}$$

Sums of Products and Minterms: Boolean expressions can be written in two different useful forms, as Sums of

$$\sim(a + b) = (\sim a) \cdot (\sim b) \tag{9}$$

$$\sim(ab) = (\sim a) + (\sim b) \tag{10}$$

Products (SOP) and as Products of Sums (POS). Each form has its unique applications. K-Map applications are most conveniently done with the expressions being in the SOP form.

Any Boolean expression can be written in the SOP form [Ref. 10]. One method for doing this is to generate a truth table for the expression and pick out the product terms for which the output is True or 1. "Or-ing" these product terms results in the SOP form. For example, if the generated truth table is as in Table 1, then the SOP form of the expression that generated this truth table is shown in Equation 11.

$$\text{Output} = \sim A \sim B + \sim AB + AB \tag{11}$$

The products are called minterms [Ref. 5]. Each term of a Boolean expression, written in the form of an SOP of Boolean variables, is a minterm.

As another example, consider the Boolean expression A(B(C+D)+CD) +BC, where A, B, C and D are Boolean variables. Using the identities in the *Boolean Algebra* Subsection, this expression can be changed to an SOP expression either as shown in Table 1, using truth tables, or using Boolean Algebra.

Table 1 — Generated Truth Table.

A	B	Output
F	F	T
F	T	T
T	F	F
T	T	T

$$\begin{aligned} A(B(C + D) + CD) + BC &= A(BC + BD + CD) + BC \text{ (Eqn. 3)} \\ &= ABC + ABD + ACD + BC \text{ (Eqn. 3)} \end{aligned} \tag{12}$$

Truth Tables and K-Maps: A K-Map is actually a type of Venn diagram [Ref. 8]. However, more important is the connection between K-Maps and truth tables. Truth tables are closely related to K-Maps. K-Maps are diagrams that contain cells, each of which correspond to a line in a truth table. There is a one-to-one correspondence [Ref. 9] between lines in a truth table and cells in a K-Map. To illustrate, consider a truth table of the Boolean expression B + ~A (Table 2).

In this table, A and B are variables that take on Boolean values, sometimes called "Boolean variables," where T designates the Boolean value "True" and F designates "False."

The corresponding K-Map of B + ~A is in Table 3.

An alternate way of drawing this K-Map, found in the literature [Ref. 3], is displayed in Table 4.

Table 3 is an array of four cells, each of which corresponds to a line in the truth table shown in Table 2. The number of variables in the truth table is the dimension of the associated K-Map. Therefore, for a truth table of four rows, the number of variables is two, and the dimension of the associated K-Map is two.

In the top row of symbols in the K-Map shown in Table 4, 0 and 1, are the possible Boolean values for A. In the left column, the symbols 0 and 1 are the possible Boolean values for B. In table 4, 0 and 1 correspond respectively to F and T in table 3. This row and column of symbols in Table 3 are called "wings." They are not really part of the K-Map. The Boolean value in each cell of the K-Maps in Table 3 and Table 4 corresponds to the value of the expression for which the K-Map is constructed. For example, the top left cell in the Table 3 K-Map contains a T, which corresponds in the truth table to the value for B + ~A on the line for which A is True and B is True.

An example of the use of K-Maps in circuit analysis is the minimization of the components of the circuit as shown in Figure 1.

Table 2 — Truth Table of B + ~A.

A	B	~A	B+~A
F	F	T	T
F	T	T	T
T	F	F	F
T	T	F	T

Table 3 — K-Map of B + ~A.

	A	~A
B	T	T
~B	F	T

Table 4 — Another K-Map of B + ~A.

		A	
		0	1
B	0	1	0
	1	1	1

Table 5 – Truth Table for Circuit in Figure 1.

A	B	~AB+A~B+AB
0	0	0
0	1	1
1	0	1
1	1	1

Table 6 – K-Map for Circuit in Figure 1.

K-Map	A	~A	
B	1	1	A
~B	1	0	
	B		

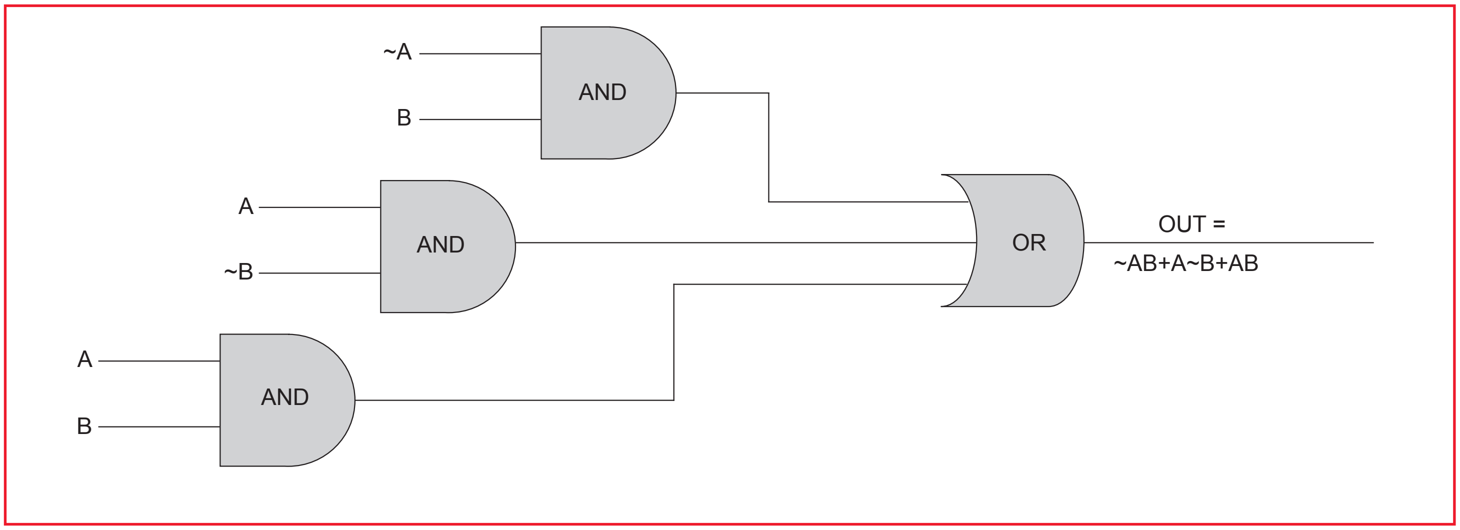

Figure 1 – Simple Circuit.

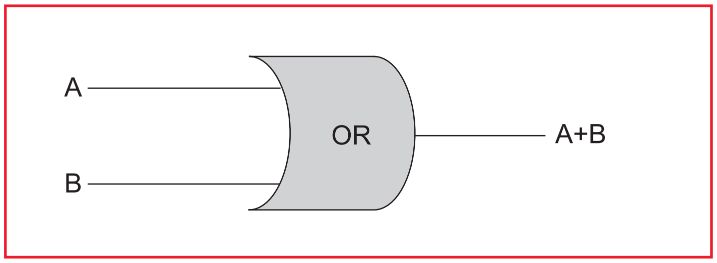

Figure 2 — Simplified Circuit from Figure 1.

Table 7 — Truth Table for ~AB + A~BC.

A	B	C	~AB	A~BC	~AB+A~BC
F	F	F	F	F	F
F	F	T	F	F	F
F	T	F	T	F	T
F	T	T	T	F	T
T	F	F	F	F	F
T	F	T	F	T	T
T	T	F	F	F	F
T	T	T	F	F	F

Table 8 — The Form of the K-Map Used in this Paper.

	A		~A	
C	T	F	T	F
~C	F	F	T	F
	~B	B		~B

Table 9 — The Form of the K-Map Common in the Literature.

		AB			
		10	11	01	00
C	1	1	0	1	0
	0	0	0	1	0

Table 5 shows the associated truth table for the circuit in Figure 1. Table 6 shows the K-Map derived from the truth table in Table 5. An analysis was performed, and it was determined that the circuit can be simplified so that the output is A + B, which is equivalent to ~AB + A~B + AB, in the sense that A + B will generate the same truth table for the same inputs A and B. The process for deriving this result will be reviewed later in this paper.

The circuit simplifies to what is shown in Figure 2.

Another example of the correspondence between the table and map is that the lower right cell in the K-Map, as shown in Table 3, corresponds to the first line in the truth table, shown in Table 2, the line for which the ~A and ~B variables are True, or A and B variables are False.

Table 7 illustrates a truth table of three variables, which will result in eight lines.

A truth table of three variables A, B and C can be illustrated with the Boolean expression ~AB + A~BC (Table 7). The variables A, B and C take on the Boolean values T (True) or F (False).

The associated K-Map (Table 8) is of dimension three, since it contains three variables and therefore eight cells.

Again, common in the literature [Ref. 3] is the alternative method of displaying the Table 8 K-Map (Table 9).

In Table 9, 00 corresponds to ~A~B, 01 to ~AB, 11 to AB, 10 to A~B, 0 to ~C, and 1 to C.

Note the order of the column indices across the top of Table 9. The binary order (in reverse) would be 11, 10, 01, 00. The order that appears in Table 9 is not the binary order. This order in Table 9 is called "Gray Code." This particular method of numbering the indices guarantees that only one variable changes from any cell to any adjacent cell. This is also true in the form of the K-Map used in this paper (Table 8). Across the top row, for example, the values for each cell are A~B, AB, ~AB and ~A~B. Only one variable changes for each step from a cell to an adjacent cell. To incorporate Boolean expressions in K-Maps, the expressions must be in the SOP form.

Using wings in K-Maps seems to be the best of the alternative methods. The wings provide room to display the conditions associated with the requirements.

It will be seen later that the Gray Coding in K-Maps permits manipulations that make possible the simplification of Boolean expressions representing the data in the truth tables. This is what makes it useful in circuit analysis (See the *Using a K-Map for Requirement Minimization* subsection later in this paper). Even though K-Maps do not offer an advantage over truth tables for discovering missing requirements, mapping the requirements to a K-Map makes it possible to analyze requirements, which may lead to requirements minimization.

Mapping from Truth Tables to K-Maps: The following steps can be used as a guide for constructing a K-Map from a truth table:

1. **Acquire Data** — Determine the number of unique variables in the Boolean.
2. **Process Data** — Place the Boolean expression in the SOP format.
3. **Determine Entries in the K-Map** — Evaluate the Boolean expression for each possible combination of variables. If there are n variables, there will be 2^n evaluations. The associated K-Map will contain 2^n cells.
4. **Populate the K-Map** — Place each evaluation in a unique cell in the K-Map.

Hamming Distance: The Hamming distance [Ref. 11] between two binary strings of the same length is the number of positions at which the binary values are different. For example, consider the two Boolean expressions, each of length four, ~ABC~D and A~BCD. In the first position, ~A and A will differ in their binary values. This will be true in the second and fourth positions, also. Therefore, these two Boolean expressions differ in three positions, and the Hamming distance between these two strings is three.

Requirements

Safety and Other Implications of Requirements Omissions: Design requirements furnish the "how," which interprets the high-level system requirements and determines how to implement them. The process of developing design requirements is probably one of the most important tasks in the system development process and also probably one of the least understood tasks. The components comprising a system are highly interrelated and complex, which means they must be well understood and defined in order to properly function when built. The requirement specification process must correctly define the system as a whole, including architectures, functions, interrelationships, constraints, etc.

This paper concentrates on requirements expressed as conditions: If a condition is satisfied, an action is taken. These are identified as the subset of software requirements known as functional requirements. Conditional requirements are typically event-driven, behavior and state-driven (see Figure 3). Examples of key words that identify conditional requirements are "when," "upon," "if/then," "while" and "where."

Requirements analysis is:

- A process of assuring that requirements are an accurate decomposition of the system requirements
- A check that the requirements are complete, unambiguous, correct, verifiable, concise, consistent, feasible, traceable and necessary
- A review to determine if there are any missing, duplicate or contradictory requirements

Missing software requirements are a common problem with writing specifications. How can one be sure all requirements essential to delivering a working solution have been included? Missing critical requirements introduce project delays, scope creep, the possibility of delivering the wrong product, and worst of all, the possibility of mishaps. K-Maps address this last safety-significant possibility by making available a tool to detect missing requirements that can arise in the design development process.

Common sources of missing requirements are:

- Failure to consider all phases of an operation
- Lack of specification of the sequence of operations
- Using an inappropriate Easy Approach to Requirements Specification (EARS) pattern
- Failure to include essential actors in the design language, if used
- Overlooking the inverse of a requirement

The requirements specification provides a structure for organizing the large number of requirements necessary for designing a system. This structure ideally helps to minimize or eliminate duplication of requirements and provides a means for navigating through all of the requirements.

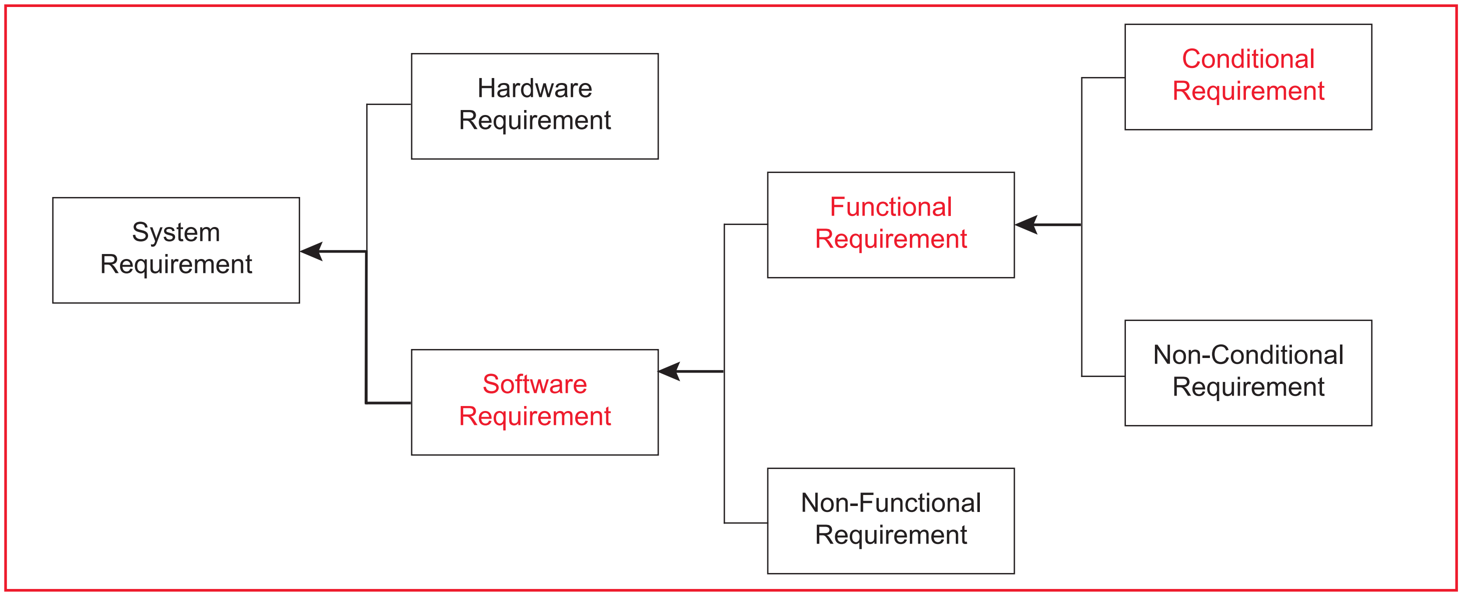

Figure 3 – Requirements Hierarchy.

Standard Forms: For the purposes of this paper, a requirement is defined as a Boolean statement and a designated action with the following properties:

- The Boolean statement is composed of conditions, each of which has a binary truth value.
- The conditions are combined by Boolean operators (and [binary operator "·"/ juxtaposition], inclusive or [binary operator "+"]).
- The Boolean statement is in the SOP form.
- The Boolean statement has a truth value, as computed from its conditions.
- The truth value of the Boolean statement determines if the designated action will be carried out or not.

The standard form of a requirement is defined as one that is written in the form of an "if/then" statement with the Boolean expression following the "if" key word, in the SOP form, and the action following the "then" key word.

Consider this requirement: "If M then X," where M can be one or more conditions connected by "or" or "and" and X is the action to perform. One must verify if M is in SOP form to ensure that this requirement is in the standard form.

- Example 1— The requirement "if A and B then Y" is in the standard form because it is written in the form of the if/then statement and the Boolean expression "A and B" is in the SOP form.
- Example 2— The requirement "if A and (B or C) then Z" is not in the standard form because the Boolean expression A and (B or C) is not in the SOP form, even though the requirement is written in the form of an "if/then" statement.

The requirement in Example 2 can be rewritten in the standard form by putting the Boolean expression A and (A or C) in the SOP form. A and (B or C) is equivalent to $A \cdot (B + C)$, which is equal to $A \cdot B + A \cdot C$ or (A and B) or (A and C). Thus the standard form of the requirement in example 2 is "if (A and B) or (A and C) then Z."

Sometimes, a requirement is written with other keywords than "if/then." One needs to convert that requirement into the form of an "if/then" statement and verify that the Boolean expression is in the SOP form. The list in Table 10 provides some of the other keywords used in requirements writing, and their conversion to "if/then" form.

Related Requirements: Requirements are related if they share one or more conditions. As a consequence, related requirements are generally requirements all of whose conditions are drawn from the same set of conditions. This indicates that the requirements are dealing with different aspects of a single subsystem or related subsystems. Related requirements are of necessity related to each other by "or" because otherwise different values of the conditions would contradict each other.

Since related requirements share conditions, a single K-Map can be constructed with these shared conditions, to which K-Map the related requirements may be mapped and analyzed. K-Maps are concerned with a limited and well-defined set of conditions. In the *Applications of K-Maps in Requirements Analysis* subsection later in this paper, we will see that one of the steps in the mapping methodology is to determine the number of unique conditions appearing in a set of requirements. This number is the dimension of the K-Map.

How to Interpret the Entries in a K-Map as a Requirement: K-Maps were designed originally as an aid in minimizing switching circuit Boolean expressions. Table

Table 10 — Other Keywords used in Requirements Writing.

Expression	Equivalent ("if/then")
B unless A	If not A then B
B only if A	If not A then not B
A provided that B, A in case B	If A then B
While A do B	If A then B
B if and only if A	If A then B and if B then A

Table 11 — Analogies Between Switching Circuit Artifacts and Related Requirements Artifacts.

Switching Circuits	Related Requirements
Input variables	Conditions
Boolean expressions or statements, with Boolean operations connecting the variables	Boolean expressions with Boolean operations connecting the conditions in these expressions
Boolean value of an expression	Indicates whether an action is performed or not

11 shows analogies between switching circuit artifacts and related requirements artifacts as seen from K-Maps.

Table 12 is a K-Map of two variables, A and B. The numbers 1, 2, 3 and 4 are inserted in the cells for ease of reference.

Table 12 — K-Map of Two Variables

	A	~A
B	1	2
~B	3	4

A and B are the Boolean variables, and each cell designates a truth value for a Boolean expression of the variables A and B. In the corresponding truth table (Table 13), T and F correspond to the truth values for A and B. The cell numbers refer to Table 12. For example, in Table 13, cell number 4 corresponds to A and B both False (F). In Table 12, the cell occupied by "4" corresponds to ~A and ~B, or to A and B both False (F).

Table 13 — Truth Table Corresponding to Table 12.

A	B	Cell Number
F	F	4
F	T	3
T	F	2
T	T	1

There is an association between these variables in a K-Map and the conditions in the Boolean statement found in a requirement. Table 14 shows this association. The True/False value of the variable A corresponds exactly to the True/False value of a condition. Similarly, the concept of union in a K-Map corresponds to the concept of "or" or "+" in the Boolean statement in a requirement, and the concept of intersection in a K-Map corresponds to the concept of "and" or "·" in the Boolean statement in a requirement.

This association between K-Maps and requirements is the key to the use of K-Maps for analyzing requirements.

Consider the following requirements with M and N as conditions:

If M is True and N is True, then <action 1>
If M is True and N is False, then <action 2>
If M is False, then <action 3>

For the requirement, "If M is True and N is True, then <action 1>," associate condition M with the Boolean variable A, and condition N with the Boolean variable B. This association is reasonable because conditions have Boolean values just as Boolean variables do. For two conditions, the corresponding K-Map is said to be of dimension 2, as it would be for two Boolean variables. Note that cell 1 (numbered as in Table 12) satisfies the condition "M is True and N is True" simultaneously; therefore, Table 15 is the K-Map for the first requirement.

Table 15 — K-Map of the Requirement "If M is True and N is True, then <action 1>."

	M	~M
N	<action 1>	
~N		

Now consider the requirement "If M is True and N is False, then <action 2>." Associating M with A and N with B, and identifying cell 2 as the only cell for which M is True and N is False simultaneously, the K-Map for the requirement is in Table 16.

Table 16 — K-Map of the Requirement "If M is True and N is False, Then <action 2>."

	M	~M
N		
~N	<action 2>	

Table 14 — Association Between K-Maps and Requirements

K-Map	Boolean Statement in a Requirement
Boolean variable	Condition
Boolean expression	Boolean combination of conditions, referred to as Boolean expression or Boolean statement
Union	Or
Intersection	And
SOP expression	Set of related requirements
Products in an SOP expression	Requirement

Consider also the requirement "If M is False, then <action 3>." Associating M with A and N with B, and identifying cells 2 and 4 as the cells for which M is true, the K-Map for the requirement is in Table 17.

Table 17 — K-Map of the Requirement "If M is False, Then <action 3>."

	M	~M
N		<action 3>
~N		<action 3>

Combining the above three K-Maps, a two-dimensional K-Map is formed. See Table 18.

Table 18 — K-Map of the Requirements

	M	~M
N	<action 1>	<action 3>
~N	<action 2>	<action 3>

Let's illustrate the above with actual requirements. Consider a K-Map containing requirements describing possible International Space Station (ISS) responses to safety threats from collision with earth debris and solar debris. The "Pizza Box" is an imaginary rectangular parallelepiped drawn around the ISS defining a space that, when a threat of penetration arises, raises a hazard concern (Figure 4).

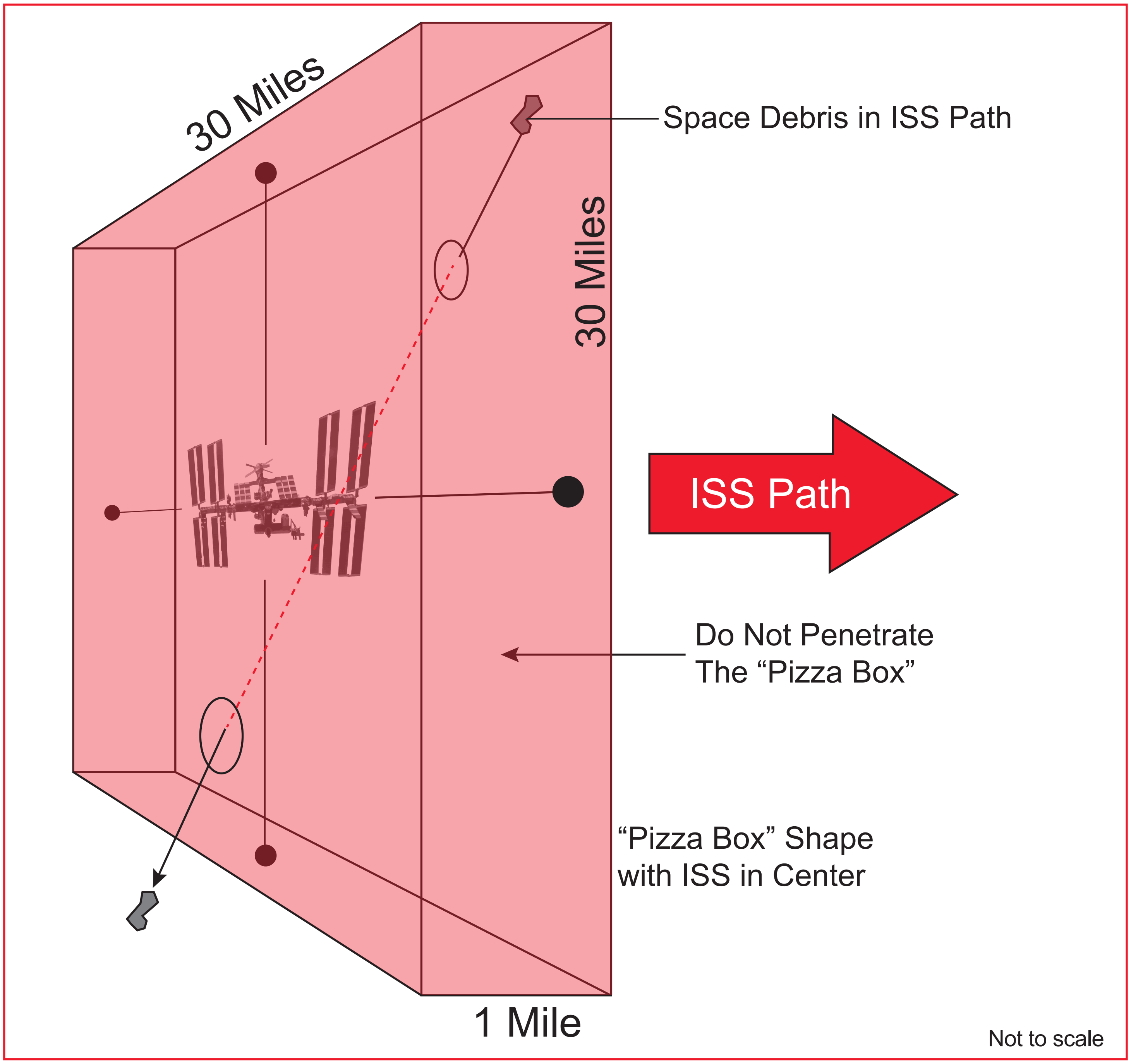
30 Miles
30 Miles
1 Mile
Space Debris in ISS Path
ISS Path
Do Not Penetrate The "Pizza Box"
"Pizza Box" Shape with ISS in Center
Not to scale

Figure 4 — International Space Station (ISS) Collision Threats.

Consider the following set of related requirements:

- **S_1000** — If Detected Object is in Earth Orbit and Relative Velocity is greater than Threshold Velocity, then Calculate Elliptical Trajectory
- **S_1001** — If Detected Object is in Earth Orbit and Relative Velocity is less than or equal to Threshold Velocity, then Take no Action
- **S_1002** — If Detected Object is not in Earth Orbit, then Calculate Hyperbolic Trajectory

These requirements are assumed to be connected by "or," which represents a union in a K-Map.

Requirement S_1000 is mapped to the upper left cell in the same way as the requirement "if M is True and N is True, then <action 1>" was mapped to the K-Map in Table 15. Wherever the conditions are connected with an "and," this represents an intersection in the K-Map. Requirement S_1000 (detailed above) is in this form. The two conditions "Detected Object is in Earth Orbit" and "Relative Velocity is greater than Threshold Velocity" are both true, so this represents an intersection in the K-Map. The first column and the first row intersect in the cell at the upper left corner of the K-Map. Therefore, the action "Calculate Elliptical Trajectory" is placed there. Table 19 shows the placement of this action.

Table 19 — Mapping Requirement S_1000.

	Detected Object is in Earth Orbit	Detected Object is not in Earth Orbit
Relative Velocity is greater than Threshold Velocity	Calculate Elliptical Trajectory S_1000	
Relative Velocity is less than or equal to Threshold Velocity		

Requirement S_1001 is mapped to the lower left cell in the same way as the requirement "if M is True and N is False, then <action 2>" was mapped to the K-Map in Table 16. See Table 20.

Table 20 — Mapping Requirement S_1001.

	Detected Object is in Earth Orbit	Detected Object is not in Earth Orbit
Relative Velocity is greater than Threshold Velocity		
Relative Velocity is less than or equal to Threshold Velocity	Take No Action S_1001	

Requirement S_1002 is mapped to the cells in the right-hand column in the same way as the requirement "if M is False, then <action 3>" was mapped to the K-Map in Table 17. See Table 21.

Table 21 — Mapping Requirement S_1002.

	Detected Object is in Earth Orbit	Detected Object is not in Earth Orbit
Relative Velocity is greater than Threshold Velocity		Calculate Hyperbolic Trajectory S_1002
Relative Velocity is less than or equal to Threshold Velocity		Calculate Hyperbolic Trajectory S_1002

Combining these mappings, we get Table 22, the final K-Map with these three requirements.

Table 22 — K-Map of Two Conditions

	Detected Object is in Earth Orbit	Detected Object is not in Earth Orbit
Relative Velocity is greater than Threshold Velocity	Calculate Elliptical Trajectory S_1000	Calculate Hyperbolic Trajectory S_1002
Relative Velocity is less than or equal to Threshold Velocity	Take No Action S_1001	Calculate Hyperbolic Trajectory S_1002

Example of Three-Variable and Four-Variable K-Maps with Requirements — Consider the related requirements:

if AB is True then X_1
if A~B is True then X_2
if ~AC is True, then X_3

where A, B and C are conditions and the X_1, X_2 and X_3 are actions. See Table 23 for the three-variable K-Map.

Sets of requirements appearing in requirements documents are generally assumed to be connected by "or."

Table 23 — Three-Variable K-Map.

	A		~A	
C	X_2	X_1	X_3	X_3
~C	X_2	X_1		
	~B	B		~B

Note that in Table 23, it is necessary to stagger one of the variables (in the bottom wing) so that the eight cells representing all possible combinations of truth values for A, B and C appear in the map. That is, each cell in the three-variable K-Map should represent a unique combination of truth values for the three variables A, B and C. The variable B was the one staggered, although A or C could have been chosen.

Some cells are blank because there are no requirements to define an action for those cells. Table 24 depicts another way to place three Boolean variables on the K-Map.

Table 24 — Three-Variable K-Map With a Different Placement of Boolean Variables.

	B		~B	
C	X_3	X_1	X_2	X_3
~C		X_1	X_2	
	~A	A		~A

If the format is different in this way, the definition of each of the cells would be changed, but all possible values of the Boolean variables in the K-Map would still be represented. The resulting K-Map would in this sense be equivalent to the original. That is, the K-Map representation of a Boolean expression is invariant with respect to the placement of the variables in the wings of the K-Map.

In Table 23, there are only three possibilities for the placement of the variable B. Another possibility is if B were directly under the variable A, and ~B directly under ~A. But then it would not be possible to map the combination A~B to the K-Map. The third possibility is if ~B were directly under the variable A, and B directly under ~A. But then it would not be possible to map the combination AB to the K-Map.

The following example of a three variable K-Map contains simplified requirements involving responses to threats to the ISS from collision with orbital debris.

Consider the following set of related requirements. See Table 25 of the resulting K-Map.

- **MC_1957 —** If object is detected, and Pizza Box will be penetrated, and there is time for Collision Avoidance maneuver, then MC shall display "Maneuver."
- **MC_1958 —** If object is detected, and Pizza Box will not be penetrated, then MC shall display "Object Detected – no action."
- **MC_1959 —** If object is detected, and Pizza Box will be penetrated, and there is no time for collision Avoidance maneuver, then MC shall display "Evacuate."
- **MC_1960 —** If no object is detected, then MC shall display Debris Statistics.

Table 25 — Example of Three Variable K-Map with Requirements.

	No Object Detected		Object Detected	
Time for Collision Avoidance Maneuver	MC displays Debris Statistics MC_1960	MC displays Debris Statistics MC_1960	MC displays “Maneuver” MC_1957	MC displays “Object Detected - no action” MC_1958
No Time for Collision Avoid-ance Maneuver	MC displays Debris Statistics MC_1960	MC displays Debris Statistics MC_1960	MC displays “Evacuate” MC_1959	MC displays “Object Detected - no action” MC_1958
	Pizza Box will not be Penetrated	**Pizza Box will be Penetrated**		**Pizza Box will not be Penetrated**

By staggering the “Pizza Box will be Penetrated” condition on the bottom of the K-Map with respect to the “Object Detected” condition on the top of the K-Map, as illustrated in Table 25, each cell has a unique combination of values for the conditions associated with it, and the K-Map covers all possible combinations of values for the conditions. To draw a four-variable K-Map, the fourth variable D would also be staggered with respect to the variable C so that each of the 16 cells in the K-Map would represent a unique combination of truth values for the four variables A, B, C and D. See the example in Table 26.

- MC_1947 If Radar Mode is enabled, object is detected, Pizza Box will be penetrated, and there is time for Collision Avoidance maneuver, then MC shall display “Maneuver.”
- MC_1948 If Radar Mode is enabled, object is detected, Pizza Box will not be penetrated, then MC shall display “Object Detected – no action.”
- MC_1949 If Radar Mode is enabled, object is detected, Pizza Box will be penetrated, and there is no time for Collision Avoidance maneuver, then MC shall display “Evacuate.”
- MC_1950 If Radar Mode is enabled, no object is detected, then MC shall display Debris Statistics.
- MC_1951 If Radar Mode is disabled, MC shall display “Enable Radar Mode.”

Table 26 — Example of Four-Variable K-Map.

	Radar Mode Enabled		Radar Mode Disabled		
Time for Collision Avoidance Maneuver	MC displays Debris Statistics MC_1950	MC displays “Object Detected - no action” MC_1948	MC displays “Enable Radar Mode” MC_1951	MC displays “Enable Radar Mode” MC_1951	**Pizza Box will not be Penetrated**
	MC displays Debris Statistics MC_1950	MC displays “Maneuver” MC_1947	MC displays “Enable Radar Mode” MC_1951	MC displays “Enable Radar Mode” MC_1951	**Pizza Box will be Penetrated**
No Time for Collision Avoidance Maneuver	MC displays Debris Statistics MC_1950	MC displays “Evacuate” MC_1949	MC displays “Enable Radar Mode” MC_1951	MC displays “Enable Radar Mode” MC_1951	
	MC displays Debris Statistics MC_1950	MC displays “Object Detected - no action” MC_1948	MC displays “Enable Radar Mode” MC_1951	MC displays “Enable Radar Mode” MC_1951	**Pizza Box will not be Penetrated**
	No Object Detected	**Object Detected**		**No Object Detected**	

Note that for a four-variable K-Map there are 16 possible combinations of the variables, i.e., 16 possible actions. In general, the number of actions is

$$\text{Number of Actions} = 2^{\text{number of conditions}} \tag{13}$$

Applications of K-Maps in Requirements Analysis

Mapping from Requirements to K-Map: A method that can be used for mapping requirements to K-Maps is presented in these steps:

1. **Acquire Data (Determination of related conditional requirements):** This step is concerned with determining sets of related requirements. Requirements associated with one subsystem or part of a subsystem are usually related.
2. Process Data: This step is concerned with placing the requirements in standard form. This means writing the requirements in the form "if M then X," where M is a Boolean expression and X is an action to be performed if M has the value "True," and M is in SOP form.
3. **Identify Topography:** In this step, the number of unique conditions is determined. This will be the dimension of the K-Map. The K-Map is constructed using wings, which contain the conditions.
4. **Perform Analysis:** In this step, the Boolean expression M is mapped to the K-Map for each requirement. If M is true, the condition X is placed in the cell corresponding to the value of the conditions in M. After all the requirements are mapped, missing requirements are detected at this step. An analysis to determine if minimization is possible is also done at this step.
5. **Write a Report:** This report details the results of the analysis. It should contain combinations of conditions for which there is no action defined, and the existence of any duplicate or superfluous requirements. The report can also contain a recommendation to the reader on further actions.

Using a K-Map to detect Missing Requirements: Missing requirements are difficult to detect analytically, and can escape functional and structural tests, resulting in failures that are difficult to diagnose and that may possibly cause human injury or system damage.

Almost all accidents related to software components in the past 20 years can be traced to flaws in the requirements specifications, such as unhandled cases [Ref. 6, 12].

When there is a missing software requirement and the code does not anticipate the missing requirement, then requirement-level tests will pass with 100 percent coverage. Missing software requirements in the Production and Deployment (P&D) phase can be both dangerous and expensive.

Consider the following requirements:

- **MC_1947 —** If Radar Mode is enabled, object is detected, Pizza Box will be penetrated, and there is time for Collision Avoidance maneuver, then MC shall display "Maneuver."
- **MC_1948 —** If Radar Mode is enabled, object is detected, Pizza Box will not be penetrated, then MC shall display "Object Detected – no action."
- **MC_1950 —** If Radar Mode is enabled, no object is detected, then MC shall display Debris Statistics.
- **MC_1951 —** If Radar Mode is disabled, MC shall display "Enable Radar Mode."

Mapping the above requirements to a K-Map results in Table 27.

Table 27 — Example of Four-Variable K-Map with a Missing Requirement.

	Radar Mode Enabled		Radar Mode Disabled		
Time for Collision Avoidance Maneuver	MC displays Debris Statistics MC_1950	MC displays "Object Detected - no action" MC_1948	MC displays "Enable Radar Mode" MC_1951	MC displays "Enable Radar Mode" MC_1951	Pizza Box will not be Penetrated
	MC displays Debris Statistics MC_1950	MC displays "Maneuver" MC_1947	MC displays "Enable Radar Mode" MC_1951	MC displays "Enable Radar Mode" MC_1951	Pizza Box will be Penetrated
No Time for Collision Avoidance Maneuver	MC displays Debris Statistics MC_1950	?	MC displays "Enable Radar Mode" MC_1951	MC displays "Enable Radar Mode" MC_1951	
	MC displays Debris Statistics MC_1950	MC displays "Object Detected - no action" MC_1948	MC displays "Enable Radar Mode" MC_1951	MC displays "Enable Radar Mode" MC_1951	Pizza Box will not be Penetrated
	No Object Detected	Object Detected		No Object Detected	

The question mark (?) in Table 27 denotes the missing requirement. A system engineer or software engineer could have forgotten to write this requirement, or the action in this cell may be missing because the variable values at this cell location may represent an impossible combination of values. Only further analyses can resolve this ambiguity.

By examining the K-Map, we note that for the conditions "Radar Mode Enabled" and "Object Detected" and "No Time for Collision Avoidance maneuver" and "Pizza Box will be Penetrated," there is no action. Consequently, there is a missing requirement of the form "If Radar Mode is enabled, object is detected, Pizza Box will be penetrated, and there is no time for Collision Avoidance maneuver, then <Action required>." The system engineer needs to define an action for this combination of conditions. One example of a possible action is "MC displays 'Evacuate.'"

Using a K-Map for Requirement Minimization: It is easier to execute requirements-level tests if the set of requirements are smaller and contains fewer conditions. This subsection considers simplifying requirements by eliminating those that are unnecessary.

Consider the following set of related requirements:

- **MC_1947 —** If Radar Mode is enabled, object is detected, Pizza Box will be penetrated, and there is time for Collision Avoidance maneuver, then MC shall display "Maneuver."
- **MC_1948 —** If Radar Mode is enabled, object is detected, Pizza Box will not be penetrated, then MC shall display "Object Detected – no action."
- **MC_1949 —** If Radar Mode is enabled, object is detected, Pizza Box will be penetrated, and there is no time for Collision Avoidance maneuver, then MC shall display "Evacuate."
- **MC_1952 —** If Radar Mode is enabled, no object is detected, and there is time for collision Avoidance Maneuver, then MC shall display Debris Statistics.
- **MC_1953 —** If Radar Mode is enabled, no object is detected, and there is no time for collision Avoidance Maneuver, then MC shall display Debris Statistics.
- **MC_1951 —** If Radar Mode is disabled, MC shall display "Enable Radar Mode."

The four-variable K-Map associated with this set of related requirements is in Table 28.

Table 28 — Four-Variable K-Map Example with Requirements.

	Radar Mode Enabled		Radar Mode Disabled		
Time for Collision Avoidance Maneuver	MC displays Debris Statistics MC_1950	MC displays "Object Detected - no action" MC_1948	MC displays "Enable Radar Mode" MC_1951	MC displays "Enable Radar Mode" MC_1951	Pizza Box will not be Penetrated
	MC displays Debris Statistics MC_1950	MC displays "Maneuver" MC_1947	MC displays "Enable Radar Mode" MC_1951	MC displays "Enable Radar Mode" MC_1951	Pizza Box will be Penetrated
No Time for Collision Avoidance Maneuver	MC displays Debris Statistics MC_1950	MC displays "Evacuate" MC_1949	MC displays "Enable Radar Mode" MC_1951	MC displays "Enable Radar Mode" MC_1951	
	MC displays Debris Statistics MC_1950	MC displays "Object Detected - no action" MC_1948	MC displays "Enable Radar Mode" MC_1951	MC displays "Enable Radar Mode" MC_1951	Pizza Box will not be Penetrated
	No Object Detected	Object Detected		No Object Detected	

Note that in 28, all cells in the first column (bracketed) on the left have the same action. "MC displays Debris Statistics." These cells are exactly the cells for the conditions "Radar Mode Enabled" and "No Object

Detected." The requirement "If Radar Mode is enabled, and no object is detected, then MC shall display Debris Statistics" will be represented by these four cells. But these are exactly the cells that represent the requirements **MC_1952** and **MC_1953**. That is, the requirement "**MC_1950** If Radar Mode is enabled, and no object is detected, then MC shall display Debris Statistics" is a combination of **MC_1952** and **MC_1953**, and can replace these two requirements. The minimized set of requirements is:

- **MC_1947 —** If Radar Mode is enabled, object is detected, Pizza Box will be penetrated, and there is time for Collision Avoidance maneuver, then MC shall display "Maneuver."
- **MC_1948 —** If Radar Mode is enabled, object is detected, Pizza Box will not be penetrated, then MC shall display "Object Detected – no action."
- **MC_1949 —** If Radar Mode is enabled, object is detected, Pizza Box will be penetrated, and there is no time for Collision Avoidance maneuver, then MC shall display "Evacuate."
- **MC_1950 —** If Radar Mode is enabled, and no object is detected, then MC shall display Debris Statistics.
- **MC_1951 —** If Radar Mode is disabled, MC shall display "Enable Radar Mode."

This minimization is possible because of the Gray Code ordering described earlier following Table 9. In Gray Code ordering, the Hamming distance between adjacent cells is always one. Therefore, exactly one condition differs between those adjacent cells. Thus, two requirements whose actions are the same but in adjacent cells with Hamming distance 1 can be combined into one requirement, and the condition that differs can be eliminated.

The K-Map allows a few manipulations, as presented in the remainder of this section. The first and fourth columns of X_1 can be combined into a single requirement with only a single condition, instead of two requirements, each of which has two conditions. The resulting requirement is "If B is false then X_1." See Table 29.

It helps to imagine that the K-Map wraps around a vertical axis to form a cylinder so that the right edge contacts the left edge. The first step is to visualize Table 29 without the wings on the K-Map. Each cell in the map can display its own label as in Table 30. The labels in the cells of Table 30 are derived from the values of the variables in the K-Map. For example, the label ~ABC~D in the upper right cell of Table 30 corresponds to the cell on the first row, third from the left, in Table 29.

Table 29 — Combining Two Columns using Wraparound.

	A		~A		
C	X_1	X_2	X_1	X_1	~D
C	X_1	X_2	X_1	X_1	D
~C	X_1	?	X_1	X_1	D
~C	X_1	X_2	X_1	X_1	~D
	~B	B	B	~B	

Table 30 — Table 29 without Wings

X_1 A~BC~D	X_2 ABC~D	X_1 ~ABC~D	X_1 ~A~BC~D
X_1 A~BCD	X_2 ABCD	X_1 ~ABCD	X_1 ~A~BCD
X_1 A~B~CD	? AB~CD	X_1 ~AB~CD	X_1 ~A~B~CD
X_1 A~B~C~D	X_2 AB~C~D	~X_1 ~AB~C~D	~X_1 ~A~B~C~D

As a check, note that the Hamming distance between adjacent cells in Table 30 is 1. This makes possible the minimization of requirements as explained in the text following Table 28.

If we wrap Table 30 around a vertical axis to form a cylinder so that the A~BC~D cell is adjacent to the ~A~BC~D cell, we see that the Hamming distance between these two cells is 1. These cells are actually adjacent cells. The cylindrical shape shown in Figure 5 emphasizes this.

This is also true for the pairs (A~BCD, ~A~BCD), (A~B~CD, ~A~BCD) and (A~B~C~D, ~A~B~C~D). The consequence for adjacent cells having a common action is that one of the conditions can be eliminated between them. The analogous Boolean expression is (using sum of minterms):

$$A{\sim}BC{\sim}D + {\sim}A{\sim}BC{\sim}D = {\sim}BC{\sim}D(A+{\sim}A) = {\sim}BC{\sim}D(1) = {\sim}BC{\sim}D \qquad (14)$$

Note that A~B and ~A~B in Table 29 represent the expression A~B+~A~B. Using Boolean algebra, this expression can be reduced as follows:

$$A{\sim}B+{\sim}A{\sim}B = (A+{\sim}A)({\sim}B) = {\sim}B \qquad (15)$$

Thus, the unnecessary condition A and the associated unnecessary requirement are eliminated.

The following set of requirements is mapped to Table 31.

1. If A and ~B then X_1
2. If ~A and B then X_1
3. If ~A and ~B then X_1
4. If A and B and C then X_2
5. If A and B and ~C and ~D then X_2

The third and fourth columns of X_1 in Table 31 can be combined into a single requirement with only a single condition, instead of two requirements, each with two conditions (If A is false, then X_1). This new requirement replaces requirements 2 and 3, and the unnecessary condition A is eliminated.

Table 32 is a repeat of Table 31. In the second column of Table 32, the top and bottom X_2 can be combined into a single requirement with three conditions, instead of two requirements, each having four conditions (If A is true and B is true and D is false, then X_2).

It helps to imagine that the K-Map wraps around a horizontal axis so that the top edge contacts the bottom edge, as shown in Figure 6.

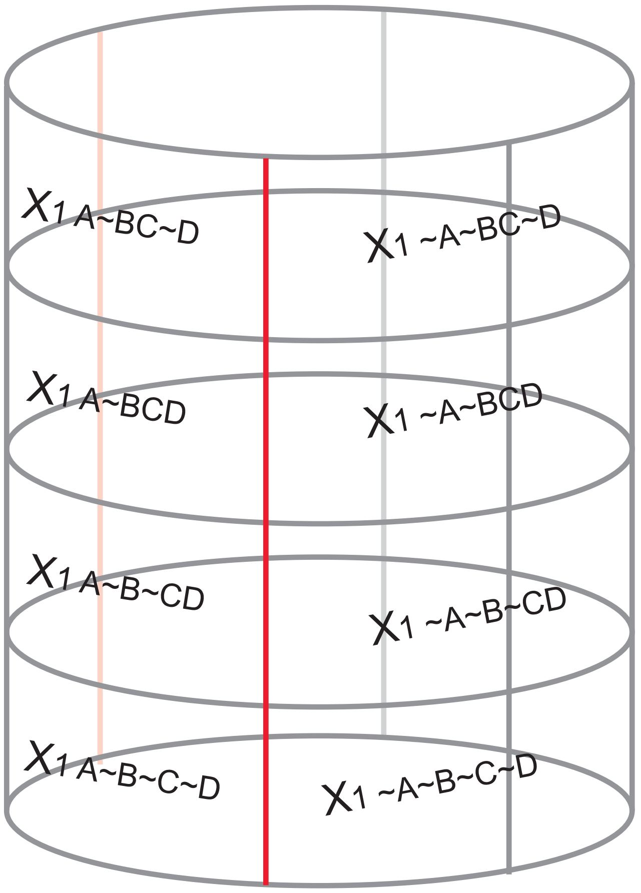

Figure 5 — Cylindrical Representation of Table 30.

Table 31 — Combining Two Columns.

	A		~A		
C	X_1	X_2	X_1	X_1	~D
C	X_1	X_2	X_1	X_1	D
~C	X_1	?	X_1	X_1	D
~C	X_1	X_2	X_1	X_1	~D
	~B	B		~B	

Table 32 — Combining Two Cells using Wraparound.

	A		~A		
C	X_1	X_2	X_1	X_1	~D
C	X_1	X_2	X_1	X_1	D
~C	X_1	?	X_1	X_1	D
~C	X_1	X_2	X_1	X_1	~D
	~B	B		~B	

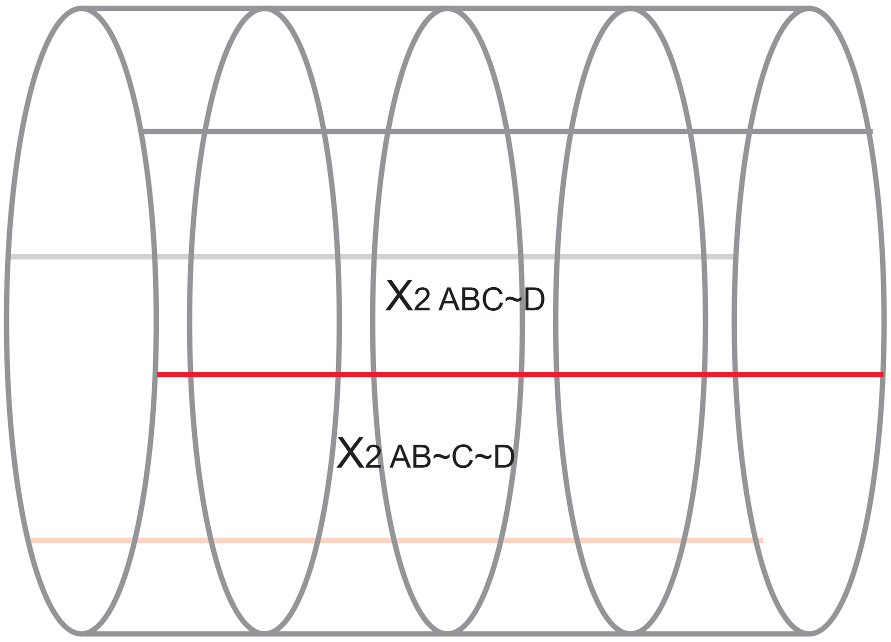

Figure 6 — Cylindrical Representation of Table 32.

Thus, ABC~D and AB~C~D are adjacent cells representing the Boolean expression ABC~D+AB~C~D.

Using Boolean algebra, it can be confirmed that this expression can be reduced as follows:

$$ABC{\sim}D+AB{\sim}C{\sim}D = AB{\sim}D(C+{\sim}C) = AB{\sim}D(1) = AB{\sim}D. \quad (16)$$

Thus, the unnecessary condition C and the associated unnecessary requirement are eliminated.

As indicated, the combinations in Table 29 and Table 32 become obvious if one imagines the alternate cylinder representation of the K-Map. Actually, the wraparounds in both Figure 5 and Figure 6 can be illustrated simultaneously by wrapping the four-condition K-Map into a torus shape. This is done by forming a cylinder as shown in Figure 5, then bringing the ends of the cylinder together. See Figure 7 [Ref. 7]. Note that the cells with the dots in the torus are each 1 Hamming distance from the adjacent cells with dots.

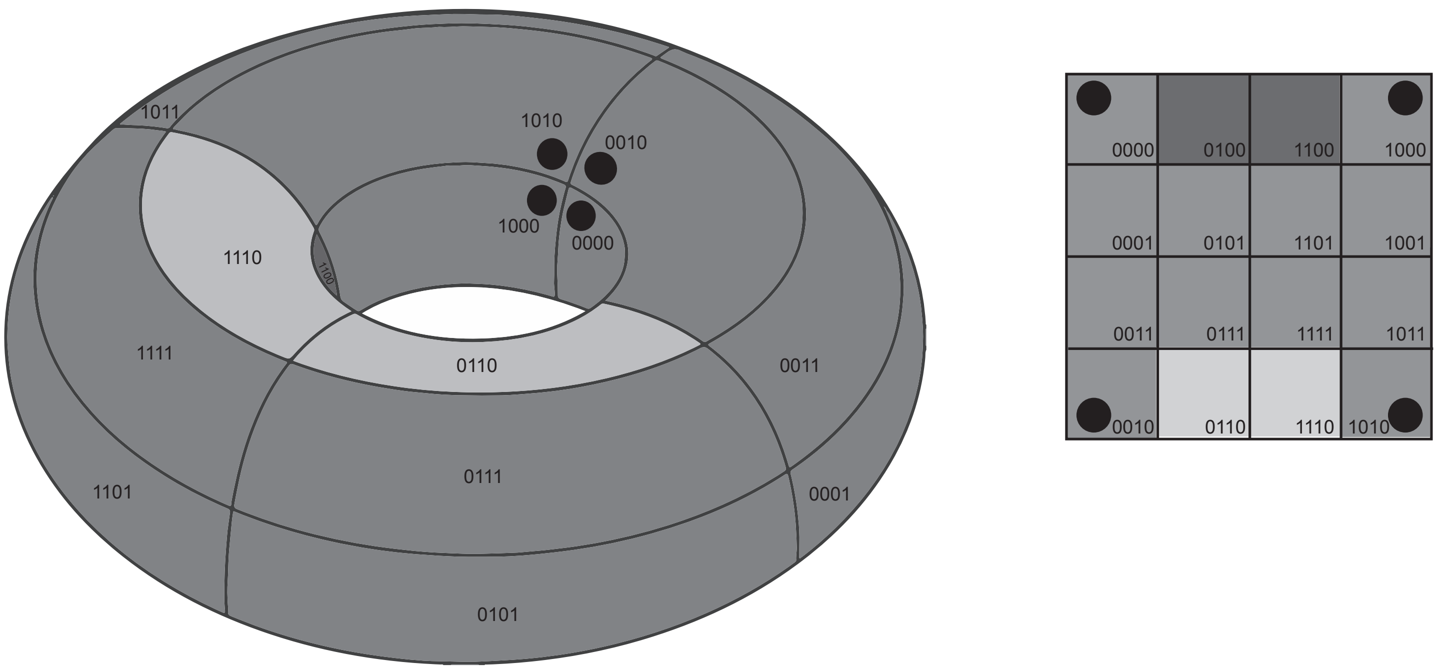

Figure 7 — Toroidal Representation of Figure 5 and Figure 6.

Other Uses of K-Maps

Although it is beyond the scope of this paper, it is worthwhile to consider that K-Maps can help with detecting conflicting requirements. These considerations will not be discussed in this paper.

K-Maps and Complex System Requirements

Requirements with more than four conditions are frequently encountered. It is possible to construct K-Maps with more than four variables that can handle such requirements, although, typically, these situations are better handled using advanced programming methods.

Figure 8 is an example of a K-Map with the five variables A, B, C, D and E. This five-variable K-Map can be drawn by combining two K-Maps, each with the four variables A, B, C, D, then defining each K-Map as an E and ~E branch of the five-variable K-Map. This K-Map can be used to both discover missing requirements and for requirement minimization.

Consider the following two sets of four requirements:

1. If A and C and E then X
2. If ~A and B and ~C then Z
3. If ~A and B and C and E then Y
4. If ~B and C and E then X

1'. If A and C and ~D and ~E then X
2'. If ~A and B and C and ~E then Y
3'. If ~A and ~B and C and D and ~E then X
4'. If A and B and C and ~E then X

Each set of requirements may be mapped to the five-condition K-Map shown in Figure 8. The mapping is done in a similar fashion as was done with a K-Map of four conditions, either in the E or in the ~E branch depending on the requirement. Examining each of the eight requirements, note that each requirement specifies either E or ~E, except for requirement 2. Because E and ~E are not specified in requirement 2 it is mapped to both branches.

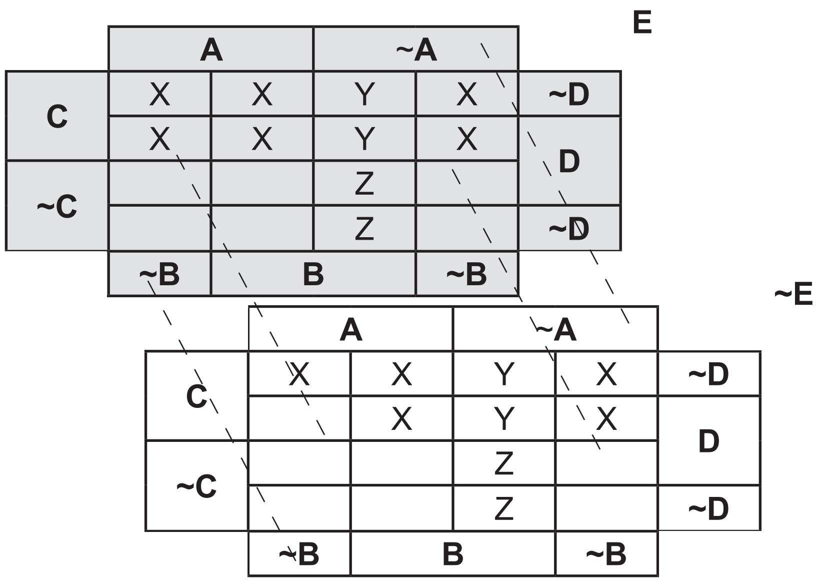

Figure 8 — Five-Variable K-Map

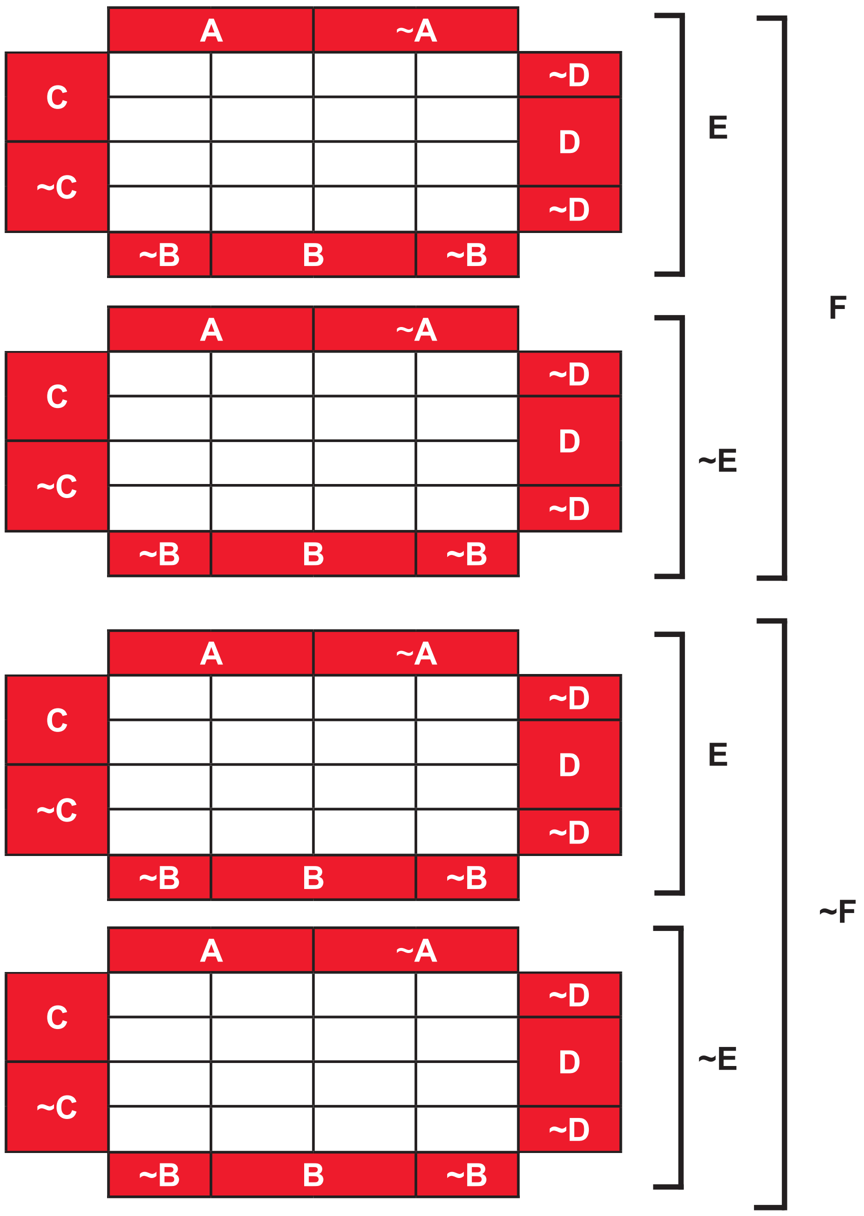

Figure 9 — Six-Variable K-Map.

It helps to visualize the E branch overlaying the ~E branch of this five-variable K-Map. Then, the cells in one branch that are a Hamming distance of one from corresponding cells in the other branch are adjacent to each other. On the K-Map the cells in the first branch that are adjacent to the cells in the second branch are directly above their adjacent cells. Requirements that map to the same cells in the upper and lower branches of the K-Map shown in Figure 8 can be combined, and the variable E can be eliminated. The following simplifications and replacements can be made:

Replace 1' with
If A and C and ~D then X (because AC~D have the same action X in E and ~E, and are adjacent in E and ~E)
Replace 3 and 2' with
If ~A and B and C then Y (because ~ABC have the same action Y in E and ~E, and are adjacent in E and ~E)
Replace 3' with
If ~A and ~B and C and D then X (because ~A~BCD have the same action X in E and ~E, and are adjacent in E and ~E)
Replace 4' with
If A and B and C then X (because ABC have the same action X in E and ~E, and are adjacent in E and ~E).

In Figure 8, blank cells are an indication of requirements not yet defined, i.e., missing. However, the example above is an exercise in minimization.

By extension, six- and seven-variable K-Maps can be constructed by employing duplication. An example of a six-variable K-Map is shown in Figure 9.

Conclusion

In this paper, the authors have demonstrated that K-Maps, a proven tool used in ASIC design, can be used to detect missing requirements and aid in the minimization of sets of requirements. Reviews of key concepts, including basic relationships in Boolean Algebra, the importance of minterms in the use of K-Maps, the relationships of truth tables to K-Maps, mapping from truth tables to K-Maps, and how Hamming Distance makes K-Maps useful, were presented. Safety implications of missing requirements were discussed. Two-, three- and four-variable K-Maps, along with several examples, were presented. Other uses of K-Maps relating to requirements were noted. Techniques borrowed from K-Maps applications to circuit analysis were shown to be useful for detecting missing requirements and for minimizing requirements.

Acknowledgment

We thank our office colleagues who provided insight on the delivery of this paper. We also thank Kenneth Chirkis, our manager, for his contribution on the safety issues related to the quality of software requirements. ⬣

References

1. Marquand, Allan. "On Logical Diagrams for n Terms," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, Vol. 5, No. 12, 1881, pp. 266-270.
2. Lavin, Marilyn Aronberg. *The Eye of the Tiger: The Founding and Development of the Department of Art and Archaeology, 1883-1923*, *Princeton University*, Princeton University, 1983.
3. Karnaugh, Maurice. "The Map Method for Synthesis of Combinational Logic Circuits," *Transactions of the American Institute of Electrical Engineers*, Vol. 1 No. 72 , 1953, pp. 593-599.
4. Veitch, Edward W. "A Chart Method for Simplifying Truth Functions," *Electronic Switching Circuits: Boolean Algebra and Mapping*, edited by Matthew Mandl, Prentice-Hall, 1969, pp. 127-133.
5. Kuphaldt, Tony R. "Boolean Algebra," *Introduction to Boolean Algebra*, All About Circuits, Accessed from: https://www.allaboutcircuits.com/textbook/digital/chpt-7/introduction-boolean-algebra.
6. Leveson, Nancy G. *Safeware: System Safety and Computers*. Addison Wesley, Reading, Massachusetts, 1995.
7. Burghardt, Jochen. "Karnaugh6.gif," *Wikipedia, the Free Encyclopedia*, Accessed from: https://en.wikipedia.org/wiki/File:Karnaugh6.gif.
8. Lewis, Clarence Irving. *A Survey of Symbolic Logic*, Berkeley: University of California Press. Southern Illinois UP, 1995, pp. 80-99.
9. Weisstein, Eric W. "Bijection," *MathWorld — A Wolfram Web Resource*, Accessed from: http://mathworld.wolfram.com/Bijection.html
10. Bender, Edward A. and S. Gill Williamson. *A Short Course in Discrete Mathematics*, Dover Publications, 2005.
11. Ayala, Jose L., et al. *Integrated Circuit and System Design*. Springer, Berlin, Heidelberg, 2012.
12. Firesmith, Donald. *Engineering Safety and Security Related Requirements for Software Intensive Systems*, p. 6. Accessed from: http://www.academia.edu/2891484/Engineering_safety_and_security_related_requirements_for_software_intensive_systems.
13. Kleitz, William. *Digital Electronics a Practical Approach with VHDL*, 9th Edition, Pearson, 2012.
14. Williams, Lippincott. "Two Deaths, Multiple Infections Linked to Sterility Failure of Sublingual Sensors," *Journal of Clinical Engineering*, Vol. 30, No. 1, January/March 2005, pp. 12-13.