Digital Logic

D.K. Sharma, D. Chakraborty, K. Chatterjee, B.G. Fernandes,
J. John, P.C. Pandey, N.S. Shiradkar, K.R. Tuckley

EE Department
IIT Bombay, Mumbai

Book References

"Digital Fundamentals" by Thomas L. Floyd, Pearson Education.
"Digital Design" by M. Morris Mano, Pearson Education.

January 17, 2024

What is Digital Logic?

- The term “digital” is derived from digit, a symbol (like 0 to 9) in a number system – (and ultimately, from fingers which are used for counting).
- Thus, digital representation is a natural fit for things which are discrete and countable.
  For example – natural numbers are discrete, so natural numbers can be represented by digital symbols.
- We are used to representation of numbers using ten symbols – this is the familiar decimal number system. However, circuit implementation of a digital system with just two symbols – a *binary* digital system – is much more robust and reliable.
- In a circuit, we can associate one of the symbols, say ‘1’ with the highest voltage (the supply voltage) and the other, say ‘0’ with the lowest voltage (ground). Now the two digits can be reliably represented and will not be confused with each other even in the presence of very high noise.

What is Digital Logic?

- The term **binary digit** (a '1' or a '0') is contracted to a **bit**.
- The binary digital system is also a natural fit for logic, in which statements can be either 'True' or 'False'.
  Thus we can use a bit to represent the truth or otherwise of a logical statement.
- This has led to the widely used terminology – **Digital Logic**.
- Circuits and systems which implement Digital Logic are used for computer design, digital communications, industrial controls and many other applications.
- Binary representation is not universal – there are systems which divide the span between the supply voltage and ground in multiple (say 4) regions. However, binary representation is used in an overwhelming majority of digital systems.

Representing Natural Numbers

- To represent bigger numbers, we use multiple bits with implied place values of powers of 2, just as we do for decimal numbers. If we use enough bits, we can represent a wide range of numbers.
- For example, we can represent the decimal number 10 as 1010 ($= 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 8 + 0 + 2 + 0 = 10$).
- The number whose powers are taken as place values is called the **base** for a number representation.
- Thus the familiar decimal numbers are base 10 numbers and binary representation is base 2.
- We shall discuss the procedure for converting numbers from decimal to binary in a subsequent lecture.

Representing Natural Numbers

- The number of bits (base 2) required to represent a number is much larger than the decimal digits required in a base 10 representation.
- So the representation of somewhat larger numbers can become awkward. For example, decimal 1000 will be represented as 1111101000.
- It is conventional to group the bits 4 by 4 from the least significant side using single symbols.
- These symbols should represent numbers from 0 to 15. We use the conventional decimal symbols for 0 to 9 and use A, B, C, D, E and F for 10, 11, 12, 13, 14 and 15 respectively. This system is called **Hexadecimal** or base 16.
- Thus 1000 (decimal) will be represented as (0011 | 1110 | 1000) =3E8 in hexadecimal. ($3 \times 16^2 + 14 \times 16^1 + 8 \times 16^0 = 768 + 224 + 8 = 1000$)

What about things which are not discrete?

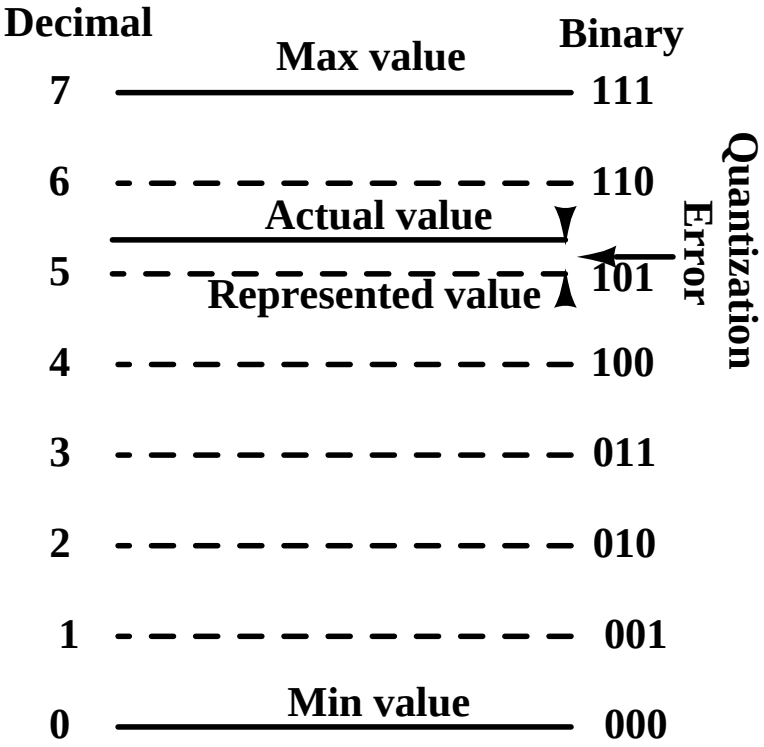
If we use enough bits, we can represent a wide range of numbers using binary digits or bits.

But how can we represent continuous things – like voltage, current or light intensity?

- Continuous quantities cannot be represented exactly by digital values. However, if we are willing to use an adequate number of bits, we can make a close approximation to continuous values using digital representation.
- We divide the full range of values into a number of intervals separated by discrete levels. Now we can approximate the continuous value by the number associated with the discrete level closest to the actual value.
- The inaccuracy of representation of a continuous value by a digital representation is called “quantization error”.

Digital representation of Continuous values

We approximate the actual value by the discrete level closest to it.



- The difference between the actual value and the represented value is the quantization error.
- In the worst case, this error can be equal to half the interval width.
- If we divide the range of values into a larger number of intervals, each interval will be narrower and the quantization error will be smaller.

However, the number of bits required for labeling the intervals will be more if we divide the range into a larger number of intervals.

Notice that close but unequal values may be represented by the same digital number.

Truth tables

- Binary digits ‘1’ and ‘0’ may be used to represent ‘True’ and ‘False’ values.
- There can be a logical relationship between different logical statements. For example:
  - The rear right light of a car should blink if we want to turn right.
  - Both rear lights should blink if we are in emergency mode.

So when should the blink control of the rear right light be turned on?

Clearly, when either or both of these events (intention to turn right or being in an emergency situation) is ‘True’.

We can express this logical relationship using an exhaustive enumeration of all possible combinations of causing events, since the number of combinations of such events is limited.

Tables containing such enumeration are called “Truth tables”.

Truth tables and Logic Functions

Consider the truth table for the blinking of the rear right light.

Right Turn?	Emergency?	Blink rear right light?
False	False	False
False	True	True
True	False	True
True	True	True

This table exhausts all possible logic value combinations for the causing events “Right Turn?” and “Emergency?”. The table represents a unique logic function, which in this case is termed as “OR”.

If we represent ‘True’ by 1 and ‘False’ by 0, the truth table for the OR function of two logical variables A and B can be written as shown on the right.

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

Truth tables and Logic Functions

How many distinct logic functions are possible for n input variables?

- The truth table must enumerate all possible combinations of input variables. So it will have 2^n rows.
- The function value can be either 0 or 1 in each row. Thus there are 2 independent choices per row.
- So the number of distinct truth tables is $= 2^{\text{no. of rows}} = 2^{2^n}$.
- Each distinct truth table represents a different logic function. (Though some of these functions could be trivial).
- Rules of manipulation of binary valued objects and logic functions which operate on them constitute an algebra. This algebra is known as "Boolean Algebra", named after the English mathematician George Boole.

George Boole, "Mathematical Analysis of Logic", 1847, and "An Investigation of the Laws of Thought on Which are Founded the Mathematical Theories of Logic and Probabilities", 1854

Truth tables and Logic Functions

Consider functions of a single variable – there can be $2^{2^1} = 4$ tables. Labeling input as a Boolean variable A, these are:

A	Out
0	0
1	0

A	Out
0	1
1	0

A	Out
0	0
1	1

A	Out
0	1
1	1

- The output in the first table is always 0. This is thus a trivial function which is always 0 independent of the input.
- In the second truth table, the output is always opposite to the input. This function is called an invert function or a **NOT** logical function. Given a binary variable A, NOT A is represented as $\overline{A}$.
- In the third truth table, the output is the same as the input. This is a buffer function, where Out $= A$.
- In the last truth table, the output is always 1. This is also a trivial function which is always 1, independent of the input.

Logic Functions of two variables

For 2 input variables, we can have $2^{2^2} = 16$ distinct logic functions.
A few important ones are described below:

A	B	OR	NOR	AND	NAND	XOR	XNOR
0	0	0	1	0	1	0	1
0	1	1	0	0	1	1	0
1	0	1	0	0	1	1	0
1	1	1	0	1	0	0	1

The output of OR is '1' if either or both inputs are '1'.

$|A - B|$

The output for the AND function is '1' only if both inputs are '1'.

The output for the XOR (exclusive OR) function is '1' only if exactly one of the inputs is '1'. It is like the OR function but excludes the case when both inputs are '1' – hence the name.

NOR, NAND and XNOR are inverted values of OR, AND and XOR respectively.

Boolean Algebra: Terminology

George Boole provided a mathematical foundation to logic through two monographs published in 1847 and 1854.

- Boolean algebra uses the symbol "+" for the OR logical function and the symbol "$\cdot$" for AND.
- We use a bar on top of a variable to denote the NOT function. Thus, NOT $x = \overline{x}$. Since a bar is not easily typeset, a $'$ sign is also used to indicate the NOT operation. Thus x' is an alternative to $\overline{x}$.
- NOT, OR and AND are the basic functions of Boolean algebra. The XOR function is represented as $\oplus$.
- If x is True, $\overline{x}$ must be False and if x is False, $\overline{x}$ must be True. Therefore one of x and $\overline{x}$ must always be True and one of x and $\overline{x}$ must always be False.

George Boole, "Mathematical Analysis of Logic", 1847, and "An Investigation of the Laws of Thought on Which are Founded the Mathematical Theories of Logic and Probabilities", 1854

Boolean Algebra: logical relationships

$+$ = or $\cdot$ = and

We can verify some identities for Boolean functions quite easily.
For example: $x + 0 = x, \quad x + 1 = 1, \quad x \cdot 0 = 0, \quad x \cdot 1 = x$
(Since OR is True if either input is True while AND is False if either input is False).

Some other identities for Boolean functions can be verified easily from the truth tables.

A	B	OR	AND
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

OR as well as AND outputs are 0 and 1 respectively for both inputs being 0 or 1.
Thus $x + x = x, \quad x.x = x$

$x + \overline{x} = 1, \quad x.\overline{x} = 0$ since one of x and $\overline{x}$ must be True and the other must be False.

Boolean Function Identities

Obviously, inverting a Boolean variable twice will give back the original value. So $\overline{\overline{x}} = x$.

A	B	OR	AND
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

From the truth tables, we can see that the OR and AND outputs are symmetric with respect to the two inputs.
so $x + y = y + x, \quad x.y = y.x$
(OR and AND functions are commutative).

Using these results, further identities can be proven algebraically.

What is the least number of logical relations which we can assume as obvious and from which all the other properties of Boolean functions can be derived algebraically?

This minimal set of results is called “axioms” or “postulates”.

Huntington Postulates

Huntington formulated postulates for Boolean algebra in 1904. Using these, many of the useful theorems of Boolean logic can be derived (without having to take further recourse to truth tables). These are:

Identity:

$A + 0 = A, \quad A \cdot 1 = A$

complement:

$A + \overline{A} = 1, \quad A \cdot \overline{A} = 0$

commutation:

$A + B = B + A, \quad A \cdot B = B \cdot A$

distributivity:

$A + (B \cdot C) = (A + B) \cdot (A + C),$

$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$

Most of these properties are shared with ordinary algebra.

Only the distribution over the "$\cdot$" operator – $(A + (B \cdot C) = (A + B) \cdot (A + C))$ is peculiar to Boolean algebra.

Notice the duality between $+$ and $\cdot$ operators. The dual postulate is obtained by changing $+$ to $\cdot$ and changing the identity from 0/1 to 1/0.

Some theorems of Boolean Algebra

Once the postulates have been verified (say using truth tables), we can proceed to prove several useful theorems using these postulates:

$x + x = x, \quad x \cdot x = x$

$x + 1 = 1, \quad x \cdot 0 = 0$

Associativity

$x + (y + z) = (x + y) + z,$

$x \cdot (y \cdot z) = (x \cdot y) \cdot z$

Absorption:

$x + x \cdot y = x$

x "absorbs" any **added** term of the form x · (anything).

$x \cdot (x + y) = x$

Here x "absorbs" any **multiplied** term of the form (x + anything).

Notice that associativity is not a postulate, but can be proved as a theorem from the postulates described earlier.

DeMorgan's theorems

DeMorgan's theorems deal with the effect of the NOT operation on OR and AND logic functions. These state that:

$$\overline{x + y} = \overline{x} \,.\, \overline{y} \quad \text{and} \quad \overline{x.y} = \overline{x} + \overline{y}$$

Essentially, when applying the NOT operator to an expression, all variables get converted to their 'bar' versions (known as complements), all '+' operators get converted to '.' operators and all '.' operators become '+' operators.

Order of precedence:

In a an expression involving logic variables and functions, the order of precedence for evaluation is:

i) parentheses, ii) NOT, iii) AND, iv) OR operation.

From Truth Tables to Logic Expressions

Every truth table is equivalent to a logic expression.

How do we associate a given truth table with a logic expression?

- A truth table must enumerate all possible combinations of logic values for the input variables.
- Select all rows which have a '1' as the function value.
  If the input combination falls in any of these rows, the function evaluates to '1'.
- Therefore the output must be the OR of all these rows.

But how do we associate a row of the truth table with a logic expression?

From Truth Tables to Logic Expressions

Each row of the truth table may be represented by an AND function of each of the input variables or their complements.

- Consider a truth table for 3 input variables A, B and C. A particular row may contain the values 1 0 1 for these three inputs. These values uniquely identify this row of the truth table.
- We know that the AND of three Boolean variables can be '1' only if **all** of them are '1'. (If any one is '0', the product would be '0').
- So the row is characterised by the AND of all variables whose value is '1' and the complement of all variables whose value is '0' in this row.
- Thus this row would be represented as $A \cdot \overline{B} \cdot C$.

Identification of rows by the product of all input variables or their complements (as used for uniquely identifying a row of the truth table) is called a **minterm**.

From Truth Tables to Logic Expressions

The logic function is an OR of all those minterms for which the function evaluates to '1'.

Let us illustrate it with an example with 3 input variables. We show two independent logic functions F1 and F2 of these 3 variables.

A	B	C	F1	F2
0	0	0	0	1
0	0	1	1	0
0	1	0	1	1
0	1	1	1	1
1	0	0	0	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

The first function has the value '1' for rows with values of A, B, C as $001, 010, 011$ and 111.

So the function is:
$\overline{A} \cdot \overline{B} \cdot C + \overline{A} \cdot B \cdot \overline{C} + \overline{A} \cdot B \cdot C + A \cdot B \cdot C$.

The second function is '1' for ABC values of $000, 010, 011, 101, 110$ and 111.

Correspondingly it can be written as
$\overline{A} \cdot \overline{B} \cdot \overline{C} + \overline{A} \cdot B \cdot \overline{C} + \overline{A} \cdot B \cdot C + A \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C} + A \cdot B \cdot C$

From Truth Tables to Logic Expressions

This method of associating a truth table with a logical expression is general and can be used with any number of input variables.

- A logical expression which identifies a row of the truth table as a product is called a "**minterm**". Each "minterm" is the product of *all* input variables – either in their original form or in the complement form.
- The final logic function representing the truth table is then the OR (or sum) of all "minterms" for which the function has the value '1'.
- The method of expressing the logic function using minterms is a standard (or **canonical**) form which is also known as a **sum of products** form.
- A minterm expression for the *i*'th row in a truth table is represented as $\mathbf{m}_i$.

Canonical: which follows canons or well defined rules.

From Truth Tables to Logic Expressions

Recall that we had said that we can represent numbers using binary digits (bits) by allocating place values to these bits which are powers of 2, just like decimal numbers.
Now consider the functions F1 and F2 of 3 variables which we had used as examples earlier.

A	B	C	F1	F2
0	0	0	0	1
0	0	1	1	0
0	1	0	1	1
0	1	1	1	1
1	0	0	0	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

- The rows for which the output of the first function is '1' are: $001, 010, 011$ and 111.
- If we interpret the row as a decimal number derived with place values which are powers of 2, we can identify these rows as no. $1, 2, 3$ and 7.
- Using this convention, we can also write the function as $m_1 + m_2 + m_3 + m_7$, where m_i represents the i'th minterm.

From Truth Tables to Logic Expressions

We had remarked on the duality of theorems in Boolean logic. That provides us with another way of associating a logic expression with a truth table.

Select all rows which have a '0' as the result. If the input combination falls in *any* of these rows, the output is '0'. The output must be the AND of all these rows.

How do we uniquely represent a row for which the function evaluates to '0'?

- Consider again a truth table for 3 input variables A, B, and C, with a row which contains the values 1 0 1 for these three inputs.
- We know that the OR of three Boolean variables can be '0' only if **all** of them are '0'. (If any one is '1', the 'sum' would be '1').
- So the row is characterised by the OR of all input variables whose value is '0' and the complement of all input variables whose value is '1' in this row.

From Truth Tables to Logic Expressions

This alternative way of finding a logical expression to represent a row of the truth table for which the function evaluates to '0' is called a **Maxterm** and represented as $\mathbf{M}_i$.

Let us use the same example as the one we had taken earlier (with a truth table for two functions F1 and F2 of three input variables).

A	B	C	F1	F2
0	0	0	0	1
0	0	1	1	0
0	1	0	1	1
0	1	1	1	1
1	0	0	0	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

The rows for which the value of the first function (F1) is '0' are: 000, 100, 101 and 110.

These are represented by Maxterms: $A + B + C$, $\overline{A} + B + C$, $\overline{A} + B + \overline{C}$, and $\overline{A} + \overline{B} + C$, respectively.

So, F1 can be represented in the alternative form as:

$$(A+B+C)\cdot(\overline{A}+B+C)(\overline{A}+B+\overline{C})\cdot(\overline{A}+\overline{B}+C)$$

From Truth Tables to Logic Expressions

For the function F2, we have:

A	B	C	F1	F2
0	0	0	0	1
0	0	1	1	0
0	1	0	1	1
0	1	1	1	1
1	0	0	0	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

The rows for which the value of the second function (F2) is '0' are: 001 and 100.

These are represented by Maxterms: $A + B + \overline{C}$ and $\overline{A} + B + C$.

So, F2 can be represented in the alternative form as:

$$(A + B + \overline{C}) \cdot (\overline{A} + B + C)$$

This is also a **canonical** form and is known as the **product of sums** form.

Logic Reduction

- Any logical combination can be expressed using truth tables. We have seen that a truth table can be reduced to **canonical** forms such as "**sum of products**" or "**product of sums**". These provide standard ways of representing any logic function.
- However, these logic expressions are not in their simplest form.
- When logic is implemented in hardware, each operation corresponds to some logic element which contributes to the cost as well as the delay in evaluating the expression.
- Thus there is a strong motivation to simplify these expressions as much as we can.
- Using Boolean algebra identities, it is often possible to reduce these expression to a much simpler form, which is then easier to implement.

Logic Reduction

As an example of logic simplification, let us take the function F1 which we had derived in the sum of products form earlier.
$F1 = \overline{A} \cdot \overline{B} \cdot C + \overline{A} \cdot B \cdot \overline{C} + \overline{A} \cdot B \cdot C + A \cdot B \cdot C$

Using the logic identity $x + x = x$, we can write the above function as:
$F1 = (\overline{A} \cdot \overline{B} \cdot C + \overline{A} \cdot B \cdot C) + (\overline{A} \cdot B \cdot \overline{C} + \overline{A} \cdot B \cdot C) + (\overline{A} \cdot B \cdot C + A \cdot B \cdot C)$
where we have replicated the third term and added it to first and second terms. This can be simplified to
$\overline{A} \cdot (\overline{B} + B) \cdot C + \overline{A} \cdot B \cdot (\overline{C} + C) + (\overline{A} + A) \cdot B \cdot C$

Using the identity $x + \overline{x} = 1$, we can write the above as
$\overline{A} \cdot C \cdot 1 + \overline{A} \cdot B \cdot 1 + 1 \cdot B \cdot C = \overline{A} \cdot C + \overline{A} \cdot B + B \cdot C = \overline{A} \cdot (B + C) + B \cdot C$

It is easy to verify that this much simpler function evaluates to '1' for ABC combinations of (001), (010), (011) and (111), which agree with the truth table from which we had derived the sum of products function.

Logic Reduction

Simplification of canonical forms like sum of products or product of sums is not always obvious and it is not clear what is the best strategy for simplifying the expression.

A graphical solution which follows a standard procedure and works well for a small number of input variables was suggested by Maurice Karnaugh.

This method is known as the **Karnaugh Map** method. (The name is often shortened to K-map).

Karnaugh, Maurice (November 1953) [1953-04-23, 1953-03-17].
“The Map Method for Synthesis of Combinational Logic Circuits” (PDF).
Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics, 72 (5): 593-599.

doi:10.1109/TCE.1953.6371932.

Logic Reduction by Karnaugh Maps

- In this method, we translate the truth table to a 2D map, where each row and column corresponds to combinations of a small number of input variables.
- The row and columns of a Karnaugh Map are arranged such that only one variable changes from a row/column to the adjacent one. (This is known as **Gray code**).
- For example, a Karnaugh map for functions of 4 variables is shown below:

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

The function values (‘1’ or ‘0’) corresponding to respective input combination are now transferred to the cells at the crossing of AB and CD values.

Logic Reduction by Karnaugh Maps

Minterms represented by each cell

$CD\downarrow$ $AB\rightarrow$	00	01	11	10
00	$\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$	$\overline{A}\,B\,\overline{C}\,\overline{D}$	$A\,B\,\overline{C}\,\overline{D}$	$A\,\overline{B}\,\overline{C}\,\overline{D}$
01	$\overline{A}\,\overline{B}\,\overline{C}\,D$	$\overline{A}\,B\,\overline{C}\,D$	$A\,B\,\overline{C}\,D$	$A\,\overline{B}\,\overline{C}\,D$
11	$\overline{A}\,\overline{B}\,C\,D$	$\overline{A}\,B\,C\,D$	$A\,B\,C\,D$	$A\,\overline{B}\,C\,D$
10	$\overline{A}\,\overline{B}\,C\,\overline{D}$	$\overline{A}\,B\,C\,\overline{D}$	$A\,B\,C\,\overline{D}$	$A\,\overline{B}\,C\,\overline{D}$

- Notice that nearest neighbour cells differ in a single term being with or without bar.
- If the function has the value ‘1’ in any two adjoining cells, these two terms will be included in the OR expression.
- While taking the OR of these terms, we can take the identical terms as common and the differing terms will have the form $(x + \overline{x})$.
- Thus the differing terms will get eliminated.

Logic Reduction by Karnaugh Maps

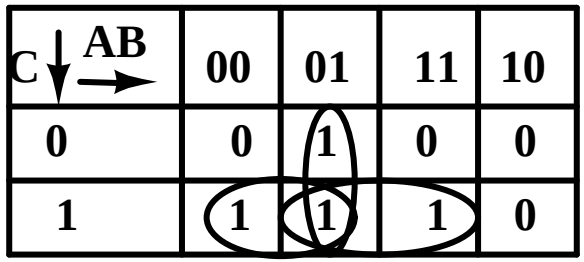
This leads to the following procedure for logic minimization of sum of product forms:

- Arrange cells as shown earlier, such that the minterms represented by the nearest row/column differ in only one input variable being with or without bar.
- Left and right extreme columns and top and bottom rows are considered as being adjacent, since these also differ in a single input variable being with or without bar.
- Circle the largest possible groups of '1' values covering 2^n rows/columns.
- Each circle will eliminate input variables for which both '0' and '1' values are included in the circle.
- Each circle gives a term in the final logic expression. Any isolated '1's will have their full minterm included.

Logic Reduction by Karnaugh Maps

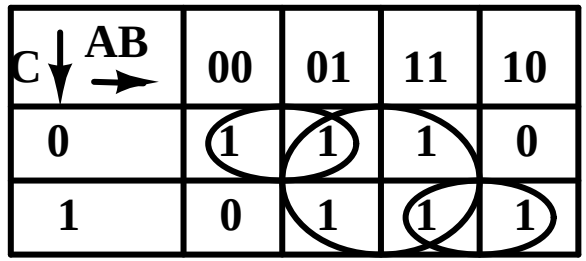
Let us illustrate the procedure by minimizing the two functions we had taken as examples earlier.

A	B	C	F1	F2
0	0	0	0	1
0	0	1	1	0
0	1	0	1	1
0	1	1	1	1
1	0	0	0	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



$F1 = \overline{A}\,B + \overline{A}\,C + B\,C$

Vertical ellipse eliminates C
Left horizontal eliminates B
Right horizontal eliminates A



$F2 = \overline{A}\,\overline{C} + B + AC$

Left horizontal eliminates B
Big ellipse eliminates A and C
Right horizontal eliminates B

Notice that F1 expression is the same as the one derived earlier using Boolean algebra.

Canonical expressions in the product of sums form can be similarly minimized by grouping 0's in the Karnaugh map.

Karnaugh Map example: Adders

The truth table for arithmetic addition of two bits is:

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Karnaugh Map: Sum

B↓ A→	0	1
0	0	1
1	1	0

Karnaugh Map: Carry

B↓ A→	0	1
0	0	0
1	0	1

No minimisation is possible using Karnaugh Maps.

$\text{sum} = A \cdot \overline{B} + B \cdot \overline{A} = A \oplus B$, $\text{carry} = A \cdot B$

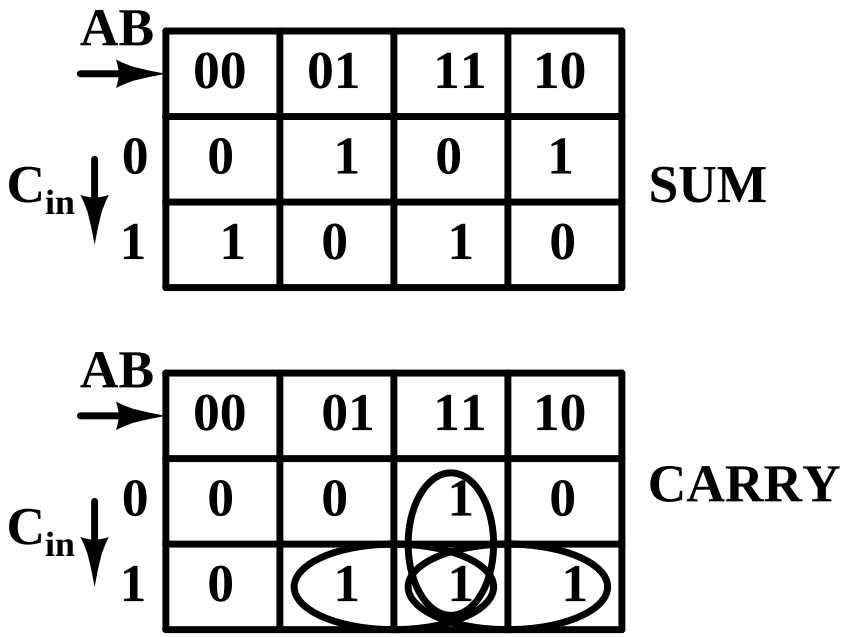
- What do we do with the carry?
- Obviously, it must be added to more significant bits.
- So we need an adder with *three* inputs.

Full Adder

Full Adder Truth Table:

A	B	C_{in}	Sum	C_{out}
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

It gives the following K maps:



$$\text{sum} = \overline{A} \cdot \overline{B} \cdot C_{in} + \overline{A} \cdot B \cdot \overline{C_{in}} + A \cdot B \cdot C_{in} + A \cdot \overline{B} \cdot \overline{C_{in}} = A \oplus B \oplus C$$

$$C_{out} = A \cdot B + B \cdot C_{in} + C_{in} \cdot A = A \cdot B + C_{in} \cdot (A + B)$$

- In this lecture, we have learnt about Boolean algebra, its axioms and theorems and their use in expressing and manipulating logic expressions.
- We have learnt about Karnaugh maps – a graphical method for logic minimization.
- In the next lecture, we shall learn:
  - how digital logic is implemented in hardware,
  - number systems using binary and hexadecimal representations, and
  - digital to analog conversion (DAC) and analog to digital conversion (ADC).

Book References

"Digital Fundamentals" by Thomas L. Floyd, Pearson Education.
"Digital Design" by M. Morris Mano, Pearson Education.