Experiment 291: Digital Logic

Aim

One aim of this experiment is to provide an introduction to Boolean algebra and digital logic circuits. Another is to introduce students to the capabilities of programmable devices (called FPLAs — field programmable logic arrays) which contain many thousands of logic gates and can be configured to provide the desired logical functions. The particular device used in this experiment is one from Altera. Initially, the experiment will be performed using discrete integrated circuits (ICs) and the Heathkit breadboarding system. Once this has been mastered, circuits will be implemented on the Altera design boards, the advantages of which will soon become apparent!

IMPORTANT NOTE: This experiment is **not** available to students enrolled in paper PHYSICS 219 A/B Computer Electronics.

References

There are many books on digital electronics in the Library. Two which are particularly helpful are:

1. T.L. Floyd, "Digital Fundamentals", Prentice Hall.
2. Horowitz and Hill, "The Art of Electronics", Cambridge.

Introduction

The inputs and outputs of a digital logic circuit can have either one of two states (HIGH or LOW) and are therefore *binary* variables. Physically, these two states are represented by a high voltage level and a low voltage level in the circuit but it is more convenient to represent these states using the digits 1 and 0, respectively. The fundamental building blocks of digital logic circuits are integrated circuits called logic gates.

Basic Gates

Figure 1 shows the names and circuit symbols of some basic logic gates.

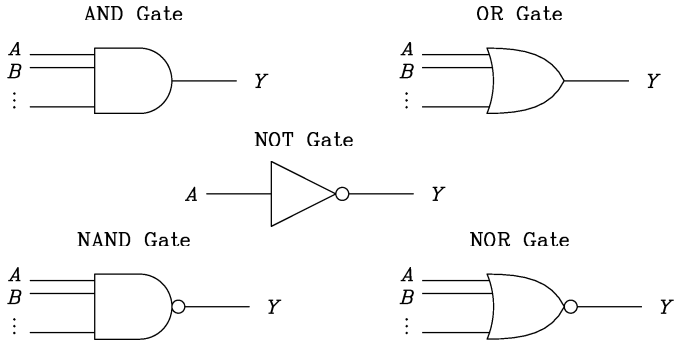

Figure 1: Circuit symbols for several common logic gates.

Notice that each of the gates shown in Figure 1 has one output terminal and one or more input terminals. The logical functions of each gate are defined as follows:

AND	The output of an AND gate is 1 if and only if all its inputs are 1.
OR	The output of an OR gate is 1 if one or more of its inputs is 1.
NOT	The output of a NOT gate is the inverse of its single input. This gate is also known as an INVERTER. A small circle at the input or output of a gate is the symbol for internal inversion.
NAND	The label NAND is an abbreviation for NOT AND. As its name suggests, a NAND gate is an AND gate with an inverted output.
NOR	Like the NAND gate, a NOR gate is an OR gate with an inverted output.

The operation of a digital logic circuit may be described in a "table of combinations" (also known as a "truth-table"). Such a table lists all the states of the output or outputs for all possible input combinations of the circuit. For a circuit with n inputs there will be 2^n input combinations. Table 1 describes the operation of two-input AND, OR, NAND and NOR basic gates.

Input		Output for function named			
A	B	AND	OR	NAND	NOR
0	0	0	0	1	1
0	1	0	1	1	0
1	0	0	1	1	0
1	1	1	1	0	0

Table 1: Truth-table for some common logic functions.

Boolean Algebra

Logic functions may be manipulated mathematically by means of an algebra named after the nineteenth century mathematician George Boole. Boolean algebra provides a *systematic* way of manipulating logic expressions, usually in order to simplify them and hence reduce the number of gates needed to construct the logic expression. In this algebra, variables (that is, the inputs and outputs of gates) are represented by *letters*, for example, A, B and Y shown in Figure 1. In producing his algebra, Boole chose to use the logical functions AND, OR and NOT although it is perfectly possible to define an algebra system using a different set of logical functions. In his algebra, the logical function AND is represented by the multiplication symbol ".", for example, $A.B$ although this is often omitted and one writes $A\,B$. The OR function is represented by the addition symbol "+", for example $A + B$ and the NOT function by a bar over the variable, for example, NOT A is written $\overline{A}$.

Let's suppose that the AND gate shown in Figure 1 has three inputs which we shall label A, B and C. Then the output Y may be written as

$$Y = A\,B\,C$$

Similarly, if the NAND gate also has three inputs A, B, C the output Y is written

$$Y = \overline{A\,B\,C}$$

In order to see why the bar is over all three variable, remember that the NAND function is nothing other than a NOT AND.

Question 1. Write down the Boolean expression for an OR gate with three inputs A, B and C and output Y. Now do the same for the NOR gate.

Question 2. Using the gates shown in Figure 1, draw a circuit that implements the logical function

$$Y = A\,B + \overline{A}\,\overline{B}$$

It is also useful to be able to derive the truth table from the Boolean expression. As an example, we will derive the truth table for the Boolean expression given in Question 2. We note that it contains two input variables

only (A and B) and a single output variable Y. Since the truth table must list all possible combinations of input variables, it is most convenient to let the input variables "count" from 00 to 11 as shown in Table 2. In evaluating the output variable Y, it is often helpful to first evaluate the intermediate products $A\,B$ and $\overline{A}\,\overline{B}$.

A	B	$A\,B$	$\overline{A}\,\overline{B}$	$Y = A\,B + \overline{A}\,\overline{B}$
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

Question 3. Write down the truth table for the circuit shown in Figure 2 below.

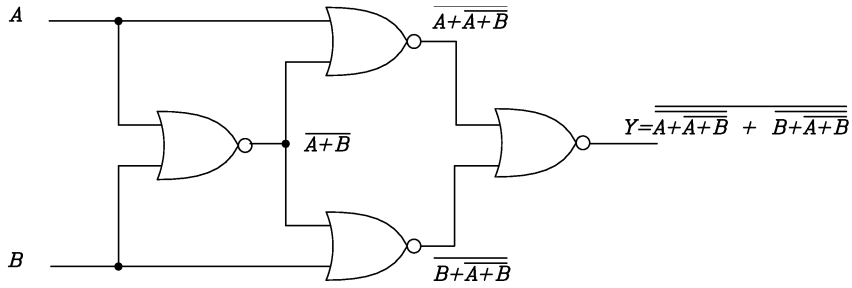

Figure 2: Logic circuit for Question 3.

Laws of Boolean Algebra

The basic rules of Boolean algebra are given below:

Defining Relations:	$A.0 = 0, \quad A.1 = A$	$A + 0 = A, \quad A + 1 = 1$
and as a consequence:	$A.A = A, \quad A.\overline{A} = 0$	$A + A = A, \quad A + \overline{A} = 1$
Commutativity	$A.B = B.A$	$A + B = B + A$
Associativity	$(A.B)\,C = A.\,(BC)$	$(A + B) + C = A + (B + C)$
Distibutive laws	$A.\,(B + C) = A.B + A.C$	$A + BC = (A + B)\,.\,(A + C)$
de Morgan's Laws	$\overline{AB} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A}.\overline{B}$

Some of these theorems are self-evident. Others may be proved by writing down the truth-tables since this tests the relationship for all possible combinations of values of the input variables. De Morgan's theorems are particularly useful when Boolean expressions are to be implemented with all NAND or all NOR gates. If both the LHS and the RHS of de Morgan's two theorems are complemented, we may deduce that:

> an AND gate may be replaced by an OR gate with inverted inputs and outputs, and
> an OR gate may be replaced by an AND gate with inverted inputs and outputs.

This is sometimes known as the "bubble rule" and is illustrated in Figure 3.

Question 4. Show how the OR function may be constructed using only NAND gates.

It is a fact that *any* logic function can be built using just NAND and NOR gates and for this reason, these gates are known as *universal gates*. However, it is often convenient to have other kinds of gates, for example, the AND and OR gates, at our disposal when designing logic circuits.

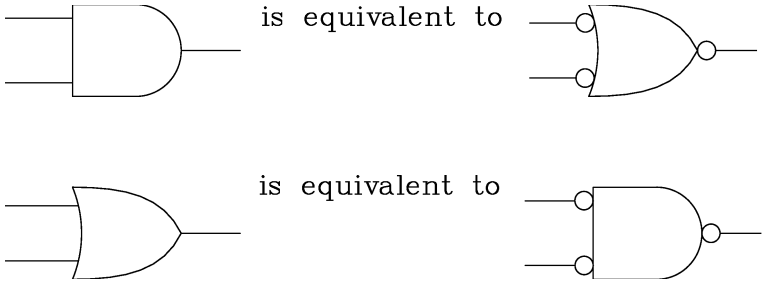

Figure 3: The "bubble" rule.

Minterm and Maxterm Logical Expressions

So far we have seen how to go from a logic diagram to a Boolean expression and *vice versa*, and from a Boolean expression to a truth table. Now we examine how to go from a truth table representation to a Boolean expression. The key to this process lies in identifying combinations of input variables called *minterms* and *maxterms*. As an example, we'll consider the truth table for a simple logical function, the EXCLUSIVE-OR function. Table 2 defines the truth table for the EXCLUSIVE-OR (XOR) function.

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

Table 2: Truth-table for the XOR function.

This circuit behaves like an OR gate except when both its inputs are 1 in which case the output is 0. Although this circuit is not a fundamental building block, it is commonly classified as a gate. From the table we can deduce the following functional relationship between the output Y and the inputs A and B:

$$Y = A\overline{B} + \overline{A}B \tag{1}$$

The terms $A\overline{B}$ and $\overline{A}B$ are called "minterms" and they refer to the two input combinations which give a 1 at the output. These terms are also described as "product" terms. The functional relationship is called a minterm or "sum-of-products" expression. Alternatively, by considering the input combinations which give a 0 at the output, we can deduce the following equivalent relationship:

$$Y = (A + B)\left(\overline{A} + \overline{B}\right) \tag{2}$$

The terms $(A + B)$ and $\left(\overline{A} + \overline{B}\right)$ are called "maxterms" and they refer to the input combinations which give a 0 at the output. These terms are also described as "sum" terms. The functional relationship is called a maxterm or product-of-sums expression[1].

It should be clear by now that, given a particular truth table, an expression in either the minterm (sum-of-products) form or the maxterm (product-of-sums) form can be written. We obtain the minterm expression for the output variable Y of Table 2 in the following way. For *each* row for which the output variable $Y = 1$, write down a subexpression in which the input variables are ANDed together. Note that if the value of a particular input variable in that row is 0, then we must write down its complement. Thus, for the first row of Table 2 we have $\overline{A}B$ and for the second, $A\overline{B}$. For the minterm expression, each of these subexpression is ORed together, yielding equation (1) above.

Conversely, to obtain the maxterm expression, we look at each row of Table 2 for which $Y = 0$ and write down subexpressions in which the input variables are ORed together. Note that if the value of an input

[1] A full description of SOPs (Sum of Products) and POSs (Product of Sums) can be found in Reference 1.

variable in a given row is 0, then we *do not complement* that variable! The various subexpressions are then ANDed together to give the maxterm expression, as shown in equation (2).

Question 5. Why can the maxterm expression be obtained in this way? Hint: write down a minterm expression for $\overline{Y}$ and simplify using Boolean algebra.

Circuit implementation

Once we have obtained a Boolean expression for the output variable (or variables) from its truth table, we may implement the expression with basic logic gates in a number of ways. The circuits in Figures 4 to 7 show four ways in which the XOR function may be implemented, starting from either the minterm or the maxterm expressions.

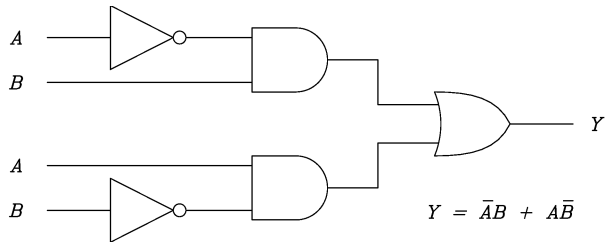

Figure 4: Direct implementation from the minterm expression.

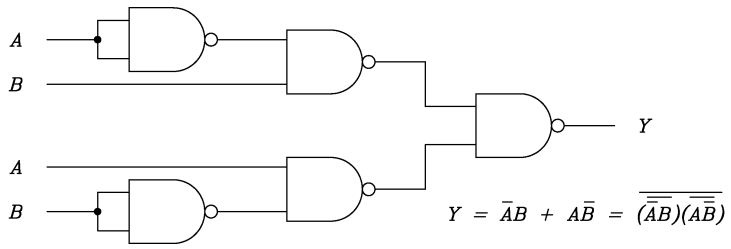

Figure 5: From the minterm expression using only NAND gates.

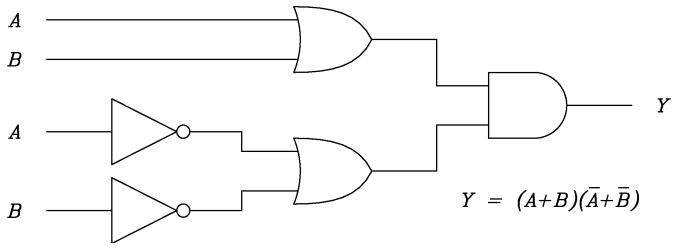

Figure 6: Direct implementation from the maxterm expression.

The table below gives the truth table for a full adder (that is, a circuit that will take two binary digits A and B plus a carry in C_{in} and produce sum S and carry out C_0 outputs).

Question 6. Write down a minterm expression for C_0 for the full adder and, using Boolean algebra, show that it can be simplified to

$$C_0 = AB + BC_{\text{in}} + AC_{\text{in}}$$

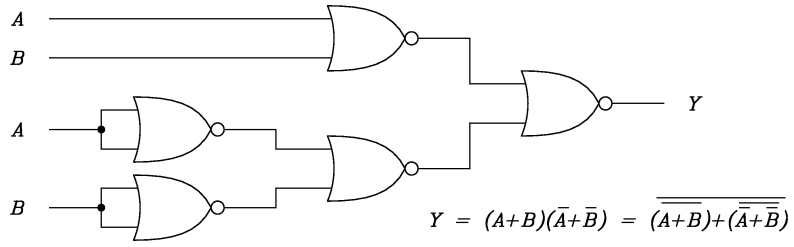

Figure 7: From the maxterm expression using only NOR gates.

A	B	C_{in}	S	C_0
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table 3: Truth table for the full adder.

Note that we can *always* find either an SOP or a POS solution for the output variables. For an SOP, our goals are to:

- minimise the number of terms
- minimise the number of factors (that is, variables) in each term.

In this way, we arrive at a circuit implementation that requires the fewest number of gates and is hence the most economical. Karnaugh maps are a graphical way in which one can be certain to obtain the minimum minterm or maxterm expressions from a truth table. A description of Karnaugh maps is contained in the Appendix and is also fully explained in Reference 1.

Procedure

This part of the experiment is to be performed on the Heathkit breadboarding system

(1) Familiarize yourself with the pin connections for the 7400 IC (quad 2-input NAND gates). Manufacturer's data sheets are available in the laboratory and should be consulted. Note that *all* ICs require electrical power! Identify the ground (gnd) and the positive supply pins (V_{cc}) for the IC. V_{cc} is +5 volts.

IMPORTANT: DO NOT connect a voltage larger than +5 volts or smaller than 0 volts to *any pin* on the IC — it will cause irrepairable damage!

Apply suitable voltage levels to the inputs of one of the gates on the 7400 IC and by observing the output with an LED logic indicator, confirm the logical function of the gates.

(2) Build the circuit you designed in answer to Question 4. Show that the NAND gate implementation of the OR gate works as expected by trying *every* combination of inputs.

A	B	S	C_0
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Table 4: Truth table for the half-adder.

(3) Construct the XOR function using just the four NAND gates of the 7400 IC. Hint: You may like to consider as your starting point the expression

$$Y = \overline{A}\,B + A\,\overline{B} + A\,\overline{A} + B\,\overline{B}$$

Why is this expression equivalent to $\overline{A}\,B + A\,\overline{B}$?

The rest of this experiment is performed on the Altera design boards. There is a tutorial pamphlet, "Introduction to the Altera Boards" that you are strongly encouraged to read. In the meantime, *ask a demonstrator* how to use the Altera software so that you can continue with the experiment.

(4) The truth table for the sum S and carry C_0 outputs of a half-adder (which may be easily derived using the rules of binary addition) are shown in Table 4.

(5) Write down minterm expressions for S and C_0 and hence obtain a circuit for the half-adder. Set up your circuit on the Altera board and verify that it is operating correctly. (By this we mean: try every combination of inputs A and B and confirm that the outputs S and C_0 follow the truth table). Use switches SW0 and SW1 to supply input A and B and use LEDs L0 and L1 to display S and C_0 respectively. Print out a copy of your circuit for inclusion in your write-up.

(6) The truth table for a full adder is given in Table 3. Write down minterm expressions for the sum and carry outputs S and C_0 respectively. We wish to build the full adder in such a way that it will be possible to use it in more complex circuits. To do this, we will create the full adder block as a new symbol. *Ask a demonstrator* to show you how to do this. Insert the new symbol into your graphic design window and verify its operation by connecting switches SW0 to SW2 to the inputs and L0 to L1 to the S and C_0 outputs, respectively.

(7) Using several full adder blocks, build a circuit that will add two unsigned four-bit binary numbers. Verify that your circuit works by recording the sum and carry outputs for a few different numbers that you have chosen.

(8) Optional part! Decoding the binary numbers displayed by the LEDs can be a little tedious. There is a pre-defined symbol called hexto7 that will allow the outputs of the four-bit adder to be displayed on a 7 segement display in hexadecimal. You may need some help connecting the buses of this symbol, so please ask a demonstrator. Double clicking on the hexto7 symbol will reveal its inner workings!

Question 7. What is the largest binary number that can be displayed by the four-bit adder? What happens if the sum of the two input numbers exceeds this number?

(9) Convert the four-bit adder into one that will perform subtraction by taking the twos complement of the number to be subtracted. Record the results for a few different numbers, making sure that you test numbers that produce both positive and negative answers. With some care, you can modify the circuit so that it will do twos complement "on the fly" so that your circuit can act as both a subtracter and an adder.

(10) Figure 8 shows a circuit that multiplies two 2-bit numbers together, using the half adder symbol and some additional gates. Build this circuit and verify that it works as expected.

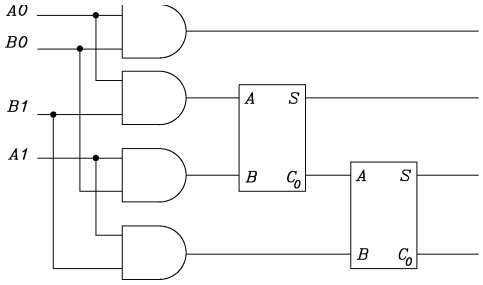

Figure 8: 2-bit multipier circuit.

Write-up

The experimental write up must include:

1. The answers to all questions.
2. Any Karnaugh maps and logic expressions used in the design of the circuits.
3. Printouts of the various Altera circuits built.

List of Equipment

1. Heathkit ET-3700 Digital Trainer
2. One Breadoard with 7400 Quad 2-input NAND mounted.
3. Personal computer with MaxPlus2 software installed.
4. Altera design boards.

L.R. Watkins
S.M. Tan
February 2004

APPENDIX

Karnaugh Maps

Logical expressions derived from truth tables generally contain redundant variables. The elimination of these variables would simplify the expressions and fewer gates would then be needed for their implementation. For minterm expressions, the basis for the elimination of redundant variables is the theorem:

$$\overline{A} + A = 1$$

If $\left(\overline{A} + A\right)$ is a common factor of two minterms, then the redundant variable A may be eliminated, for example:

$$\begin{aligned} Y &= \overline{A}BC + ABC + A\overline{B}\,\overline{C} + A\overline{B}C \\ &= BC\left(\overline{A} + A\right) + A\overline{B}\left(\overline{C} + C\right) \\ &= BC + A\overline{B} \end{aligned}$$

When all the redundant variables have been eliminated, the expression will have a minimum number of terms and each term will have a minimum number of inputs. For this reason the process of eliminating redundant variables is referred to as minimisation.

Locating redundant variables by algebraic means is not always straight-forward, especially when the number of different variables in the expression is greater than three. A graphical method based on the Venn diagram simplifies the process considerably.

Figure 11(a) shows the locations in the Venn diagram of all the minterms of 3-variable expressions. It can be seen that if an expression has two minterms which occupy adjacent areas, a common redundant variable is present. This variable may be eliminated by grouping the two minterms. If four minterms are adjacent to each other, two redundant variables are present and they can be eliminated by grouping all four adjacent minterms. Figure 11(b) shows how the Venn diagram can be rearranged giving rise to the Karnaugh map in Figure 11(c). Note that the map wraps around, i.e. top and bottom rows are adjacent and so too are the left and right columns.

Figure 12 shows an example of how a 4-variable expression may be reduced using a 4 x 4 Karnaugh map.

$$Y = \overline{A}\,\overline{B}\,\overline{C}D + \overline{A}B\overline{C}\,\overline{D} + \overline{A}B\overline{C}D + \overline{A}BC\overline{D} + AB\overline{C}\,\overline{D} + ABCD + ABC\overline{D}$$

reduces to:

$$Y = \overline{A}\,\overline{C}D + B\overline{D} + ABC$$

We can summarise the rules for forming Karnaugh maps as follows:

1. The rows and columns of the map are labelled $00, 01, 11, 10$ so that states in adjacent cells differ by one variable only. This sequence is called *Gray coding*.
2. Every cell containing a 1 must be included at least once.
3. The largest possible group of cells (powers of two only) must be formed. This ensures that the maximum number of variables is eliminated, i.e., groups formed on 2^n cells eliminate n variables.
4. The 1's must be contained in the minimum number of groups to avoid duplication.

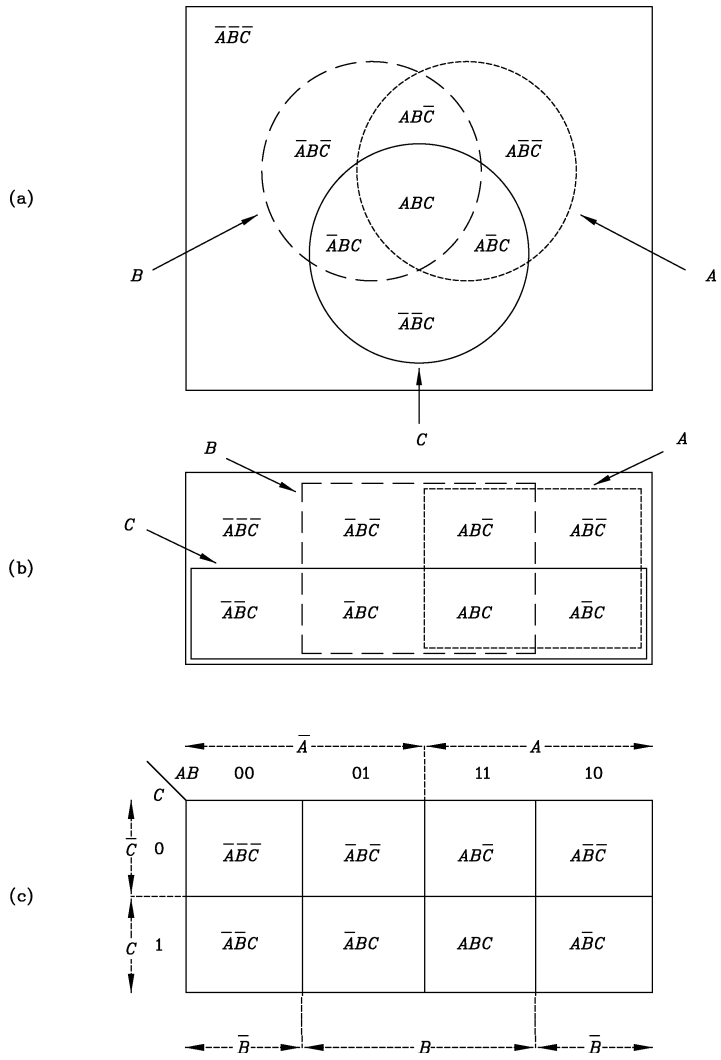

Figure 9: (a) Venn diagram. (b) Conversion of Venn diagram to Karnaugh map. (c) Karnaugh map.

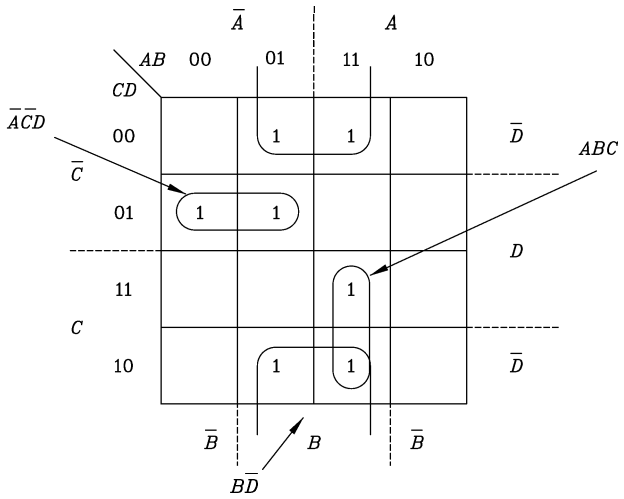

Figure 10: Karnaugh map for 4-variable logical function Y.