

Copyright © 2011, 1997. Copyright renewed by Enoch Hwang, Ph.D. and Global Specialties ®

All rights reserved.

Printed in Taiwan.

No part of this publication may be reproduced, stored in a retrieval system or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior written permission of the author.

Table of Contents	Page
Chapter 1 Combinational Logic Design Trainer, Model DL-010.....	1
Chapter 2 Microprocessors.....	5
Section 2.1 Introduction to Microprocessors	5
Section 2.2 Combinational and Sequential Circuit Analogy	7
Chapter 3 Digital Logic Circuits	9
Section 3.1 Basic Logic Gates	9
Section 3.2 Digital Circuits	12
Section 3.3 Identifying Combinational Circuits	14
Section 3.4 Analysis of Combinational Circuits.....	14
Section 3.5 Boolean Algebra	16
Section 3.6 Simplifying Combinational Circuits.....	21
Section 3.7 Synthesis of Combinational Circuits.....	28
Chapter 4 Labs	31
Section 4.1 Lab 1: Basic Gates, Lights and Action!	31
Section 4.2 Lab 2: NAND, NOR, XOR and XNOR Gates.....	37
Section 4.3 Lab 3: Designing Combinational Circuits.....	41
Section 4.4 Lab 4: Multiplexers.....	45
Section 4.5 Lab 5: Decoders	47
Section 4.6 Lab 6: Comparators	49
Section 4.7 Lab 7: Full Adder	51
Section 4.8 Lab 8: 4-bit Adder.....	55
Section 4.9 Lab 9: 4-bit Adder / Subtractor	57
Section 4.10 Lab 10: 2-bit Arithmetic and Logic Unit (ALU)	63
Section 4.11 Lab 11: BCD to 7-Segment LED Decoder	69

Chapter 1

Combinational Logic Design Trainer

The Combinational Logic Design Trainer that you have contains all of the necessary tools for you to easily implement many combinational digital logic circuits. Combinational logic circuits are one major type of digital circuits found inside microprocessors. The layout of the trainer is shown in Figure 1.

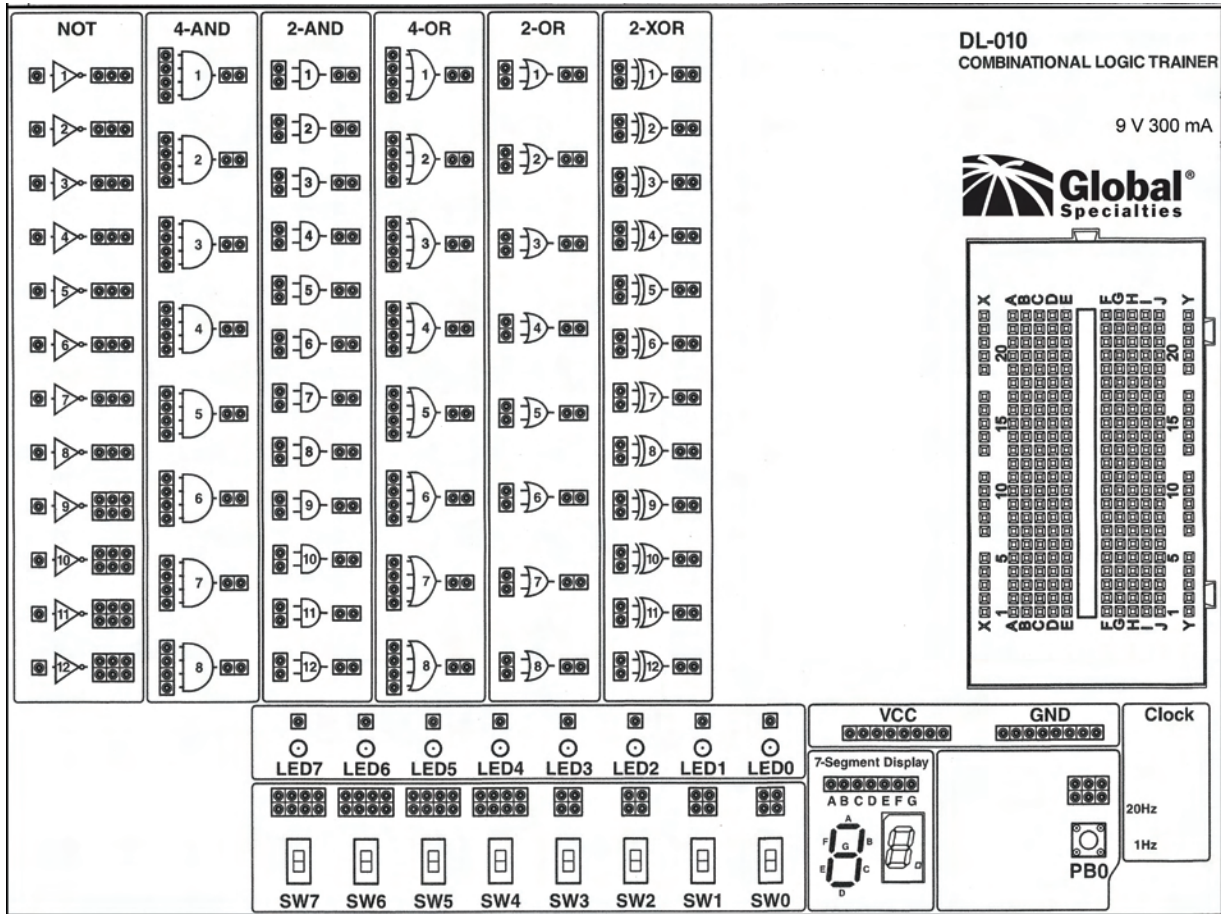


Figure 1: Combinational Logic Design Trainer layout. All of the logic gates and I / O's are pre-mounted with wire connection points connected to them.

The following is a list of all of the components on the trainer:

- > Twelve NOT gates
- > Eight 4-input AND gates
- > Twelve 2-input AND gates
- > Eight 4-input OR gates
- > Eight 2-input OR gates
- > Twelve 2-input XOR gates
- > Eight red LEDs

Combinational Logic Design

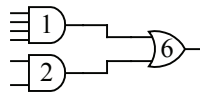
- > Two 7-segment LED displays
- > Eight toggle switches
- > One push button switch
- > VCC and GND connection points
- > General bread board area with 270 tie points
- > Hook-up wires of various lengths

The eight LEDs and the LED segments in the 7-segment displays are active high, which means that a logic 1 signal will turn the light on, and a logic 0 signal will turn the light off. The push button PB0 is also active high, so pressing the button will produce a logic 1 signal. All of the eight switches are configured so that when the switch is in the up position the output is a logic 1, and when the switch is in the down position the output is a logic 0.

You can also connect a wire to one of the VCC connection points to directly get a logic 1 signal. Similarly, connecting a wire to one of the GND connection points will get a logic 0 signal.

All of the logic gates and I/O's are pre-mounted for easy wiring of a circuit. All logic gate inputs are connected to one wire connection point, and all logic gate outputs have multiple wire connection points. To connect from the output of a logic gate to the input of another logic gate, simply use a hook-up wire to connect between the two wire connection points. For example, push button PB0 has six common wire connection points, so to use PB0 you can connect a hook-up wire to any one of these six connection points. Connect the other end of the hook-up wire to a connection point for LED0. When you press the push button, the LED should turn on. Try this simple connection now to see that it works.

The logic gates on the trainer are also numbered for easy reference for when connecting a circuit up. For instance, the eight 4-input AND gates are numbered from 1 to 8. There are also eight 4-input OR gates, and they are also numbered from 1 to 8. So be careful when a circuit diagram says gate number 1 that you know which type of gate it is referring to, i.e., whether it is the 4-input AND gate, the 4-input OR gate or even one of the other gates. For example, the following circuit diagram uses the number-1 4-input AND gate, the number-2 2-input AND gate and the number-6 2-input OR gate. In this courseware, we will use the notation 4-AND#1, 2-AND#2 and 2-OR#6 to refer to these three gates respectively.



The general breadboard area allows you to connect other components that are not available on the trainer together with your circuit. The breadboard consists of many holes for you to connect hook-up wires and integrated circuit (IC) chips. All of the holes are already connected together in groups. This way, you can connect two wires together (or connect a wire to an IC pin) simply by plugging the two wires into two holes that are already connected together. The layout of the breadboard is shown in Figure 2A.

Ch. 1 Combinational Logic Design Trainer

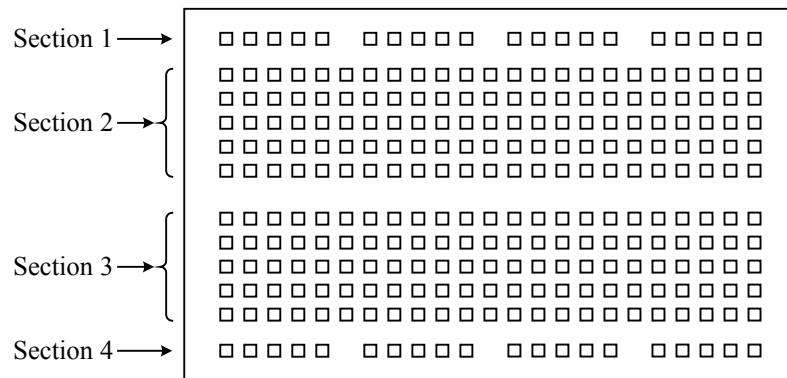
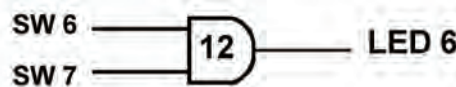


Figure 2A: Breadboard layout. The holes in section 1 are connected horizontally. The holes in section 2 are connected vertically. The holes in section 3 are connected vertically. The holes in section 4 are connected horizontally. Holes in any two different sections are not connected.

There are four general sections on the breadboard. All of the holes in section 1 are connected in common horizontally. The holes in this section are usually connected to VCC to provide power or the logic 1 signal to your circuit on the breadboard. Like section 1, all of the holes in section 4 are also connected in common horizontally. The holes in this section are usually connected to GND to provide a common ground or the logic 0 signal to your circuit. The holes in section 2 are connected vertically, so the five holes in each column are connected in common, but the vertical columns are not connected together. The holes in section 3 are also connected vertically like those in section 2, so the five holes in each column are connected in common, but the vertical columns between section 2 and section 3 are not connected together. Finally, holes in any two different sections are not connected together.

In the case where you might need more connection points for a component on the trainer, you can use the breadboard to give you extra connection points. Typically, you use a breadboard to connect wires to an IC chip. A standard dual-in-line (DIP) IC chip would be plugged into the breadboard with one row of pins in section 2 and the second row of pins in section 3.

Let us now test out the trainer. The three thick lines in Figure 2B show three wires connected from the two switches, SW6 and SW7, to the inputs of the number-12. a 2-input AND gate, and the output of this AND gate is connected to LED6. In a schematic circuit drawing, this circuit would be shown as follows:



Combinational Logic Design

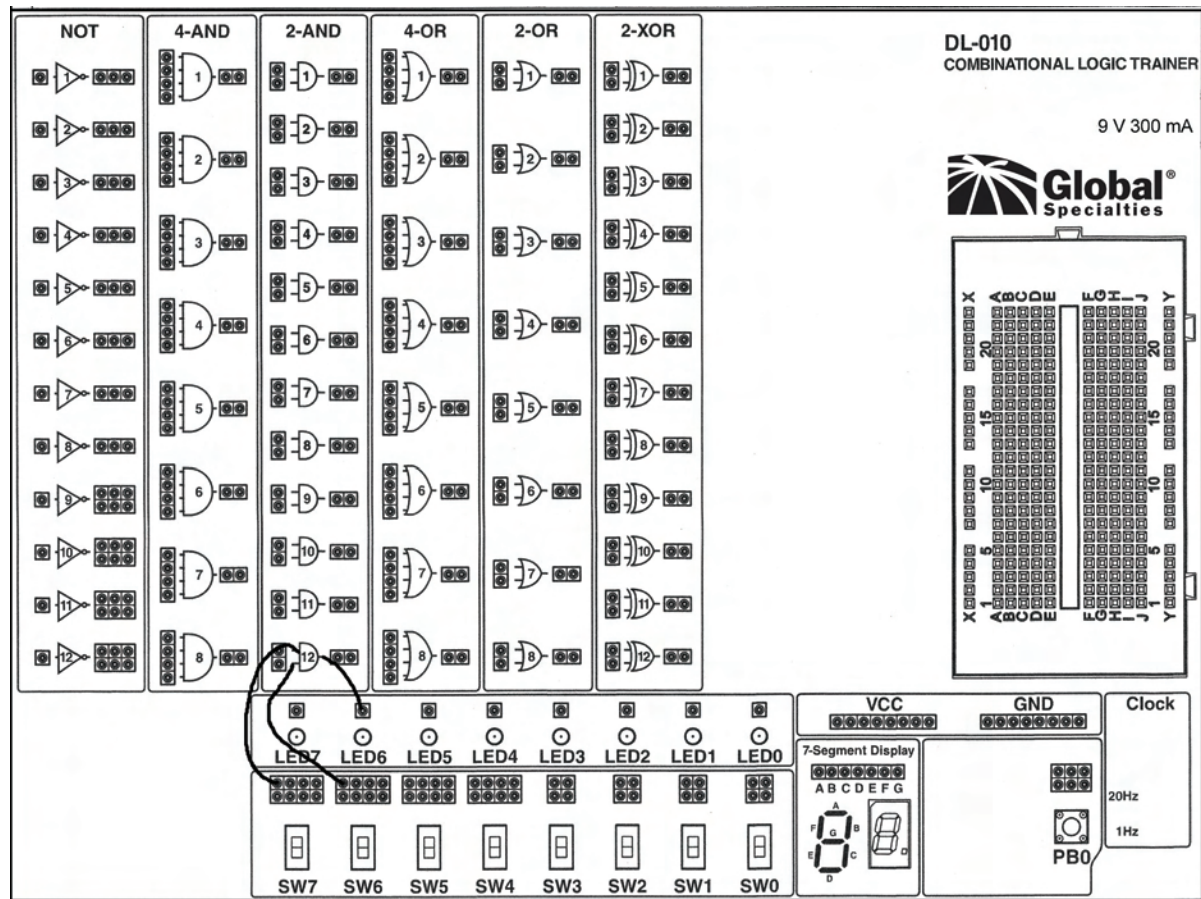


Figure 2B: Combinational Logic Design Trainer with three jumper wires connecting AND gate to Switches and LED.

Using three pieces of hook-up wires, make these same connections now on your trainer. Slide the two switches up and down and see how the LED turns on and off. At this point you may not understand why the LED turns on and off the way it does. Just keep reading and you will find out very quickly, and you will be well on your way to designing your very own microprocessors.

Chapter 2

Microprocessors

2.1. Introduction to Microprocessors

Whether you like it or not, microprocessors (also known as microcontrollers) control many aspects of our lives today – either directly or indirectly. In the morning, a microcontroller inside your alarm clock wakes you up, and another microcontroller adjusts the temperature in your coffee pot and alerts you when your coffee is ready. When you turn on the TV for the morning news, it is a microcontroller that controls the operation of the TV such as adjusting the volume and changing the channel. A microcontroller opens your garage door, and another inside your car releases your anti-lock break when you drive your car out. At the traffic light, a microcontroller senses the flow of traffic and turns on (hopefully) the green light for you when you reach the intersection. You stop by a gas station and a microcontroller reads and accepts your credit card, and let you pump your gas. When you walk up to your office building, a sensor senses your presence and informs a microcontroller to open the glass door for you. You press button eight inside the elevator, and a microprocessor controls the elevator to take you up to the 8th floor. During lunch break, you stop by a gift shop to buy a musical birthday card for a friend and find out that the birthday song is being generated by a microprocessor that looks like a dried-up pressed-down piece of gum inside the card. I can continue on with this list of things that are controlled by microprocessors, but I think you got the idea. Oh, one last example, do you know that it is also a microprocessor that is at the heart of your personal computer, whether it is a PC or a Mac? That's right the Intel Duo Core[®] CPU inside a PC is a general-purpose microprocessor.

So you see, microprocessors are at the heart of all “smart” devices, whether they be electronic devices or otherwise, and their smartness comes as a direct result of the decisions and controls that the microprocessors make. In this three part award-winning series on microprocessor design training kits, you will learn how to design and actually implement real working custom microprocessors. Designing and building microprocessors may sound very complicated, but don't let that scare you, because it is not really all that difficult to understand the basic principles of how microprocessors are designed. After you have learned the materials presented in these labs, you will have the basic knowledge of how microprocessors are designed, and be able to design and implement your very own custom microprocessors.

There are generally two types of microprocessors: general-purpose microprocessors and dedicated microprocessors. General-purpose microprocessors, such as the Intel Pentium[®] CPU, can perform different tasks under the control of software instructions. General-purpose microprocessors are used in all personal computers.

Dedicated microprocessors, also known as microcontrollers, on the other hand, are designed to perform just one specific task. So for example, inside your cell phone, there is a dedicated microprocessor that controls its entire operation. The embedded microprocessor inside the cell phone does nothing else but controls the operation of the phone. Dedicated microprocessors are therefore usually much smaller, and not as complex as general-purpose microprocessors. Although the small dedi-

Combinational Logic Design

cated microprocessors are not as powerful as the general-purpose microprocessors, they are being sold and used in a lot more places than the powerful general-purpose microprocessors that are used inside personal computers.

The electronic circuitry inside a microprocessor is called a digital logic circuit or just digital circuit, as opposed to an analog circuit. Digital circuits deal with just two discrete values, usually represented by either a 0 or a 1, whereas analog circuits deal with a continuous range of values. The main components in an analog circuit usually consist of discrete resistors, capacitors, inductors, and transistors, whereas the main components in a digital circuit consist of the AND, OR and NOT logic gates. From these three basic types of logic gates, the most powerful computer can be made. Logic gates are built using transistors—the fundamental active component for all digital logic circuits. Transistors are just electronic binary switches that can be turned on and off. The two binary values, 1 and 0, are used to represent the on and off states of a transistor. So instead of having to deal with different voltages and currents as in analog circuits, digital circuits only deal with the two abstract values of 0 and 1. Hence, it is usually easier to design digital circuits than analog circuits.

Figure 3 (a) is a picture of a discrete transistor. Above the transistor is a shiny piece of raw silicon which is the main ingredient for making transistors. As you can see in the picture, the transistor has three connections: one for the signal input, one for the signal output, and one for turning on and off the transistor. Figure 3 (b) is a picture of hundreds of transistors inside an integrated circuit (IC) chip as viewed through an electron microscope. The right half of the picture is a magnification of the rectangle area in the left half. Each junction is a transistor.

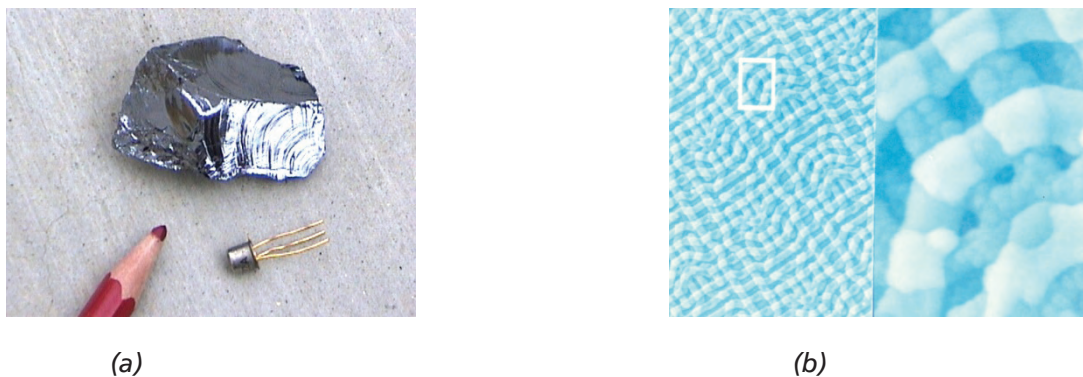


Figure 3: Pictures of transistors: (a) a discrete transistor with a piece of silicon; (b) hundreds of transistors inside an IC chip as viewed through an electron microscope. The right half of the picture is a magnification of the rectangle area in the left half.

Figure 4 is a picture with several generations of integrated circuit chips. Going clockwise from the top-left corner is a lump of silicon which can be used to make many transistors; an Intel® 8085 microprocessor with its top opened. The 8085 is an 8-bit general-purpose microprocessor with a maximum clock speed of around 10 MHz, and contains around 29,000 transistors; an Intel® 486 DX microprocessor. The 486 has a maximum clock speed of 100 MHz and contains around 1.2 million transistors; the 2732 erasable-programmable-read-only-memory (EPROM) which has a non-volatile storage capacity

of 4,096 bytes. The 2732 contains around 32,000 transistors; the tip of a pen which contains no transistor; the 7440 chip which has two 4-input NAND gates and contains 20 transistors; and finally a single discrete transistor.

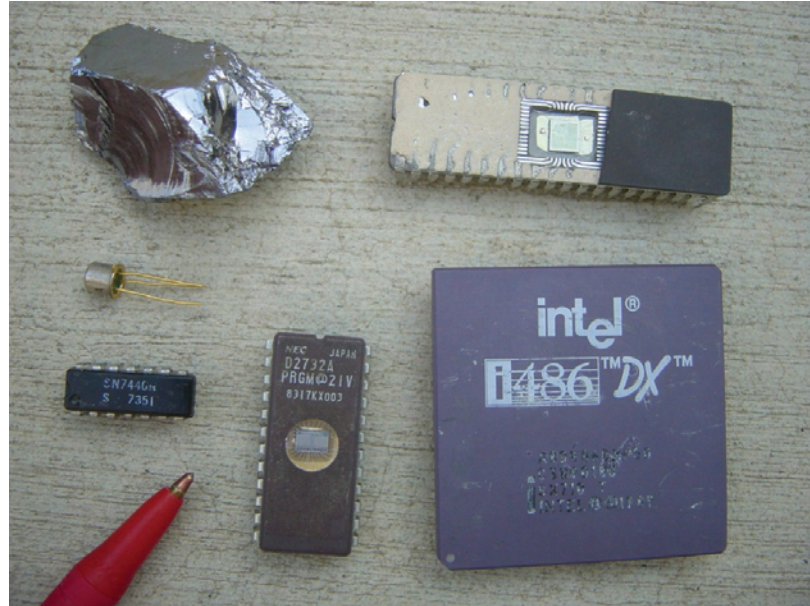


Figure 4: Picture of various integrated circuit chips. Going clockwise from the top-left corner is a lump of silicon, an eight-bit Intel® 8085 microprocessor with its top opened, an Intel® 486 DX microprocessor, the 2732 erasable-programmable-read-only-memory (EPROM) with a capacity of 4,096 bytes, the tip of a pen, the 7440 chip which contains two 4-input NAND gates, and a transistor.

Every digital circuit is categorized as either a combinational circuit or a sequential circuit. A microprocessor circuit is composed of many different combinational circuits and many different sequential circuits. In part I of this three-part series on microprocessor design training courseware you will learn how to design combinational circuits. In part II you will learn how to design sequential circuits. And finally in part III you will learn how to design microprocessors by putting these different combinational and sequential circuits together to make a real working microprocessor.

2.2. Combinational and Sequential Circuit Analogy

A simple analogy of the difference between a combinational circuit and a sequential circuit is the combination lock that we are familiar with. There are actually two different types of combination locks as shown in Figure 5. For the lock in Figure 5 (a), you just turn the three number dials in any order you like to the correct number and the lock will open. For the lock in Figure 5 (b), you also have three numbers that you need to turn to, but you need to turn to these three numbers in the correct sequence. If you turn to these three numbers in the wrong sequence the lock will not open even if you have the numbers correct. The lock in (a) is like a combinational circuit where the order in which the inputs are entered into the circuit does not matter, whereas, a sequential circuit is like the lock in (b) where the sequence of the inputs does matter.

Combinational Logic Design



(a)



(b)

Figure 5: Two types of combination locks: (a) the order in which you enter the numbers does not matter; (b) the order in which you enter the numbers does matter.

So a combinational circuit is one where the output of the circuit is dependent only on the inputs to the circuit, but not dependent on the order in which these inputs are entered. One example of a combinational circuit is the adder circuit for adding two numbers. The adder takes two numbers for its inputs. With these two input numbers, it evaluates the sum of these two numbers and outputs the result. It doesn't matter which input number you enter first as long as you enter both numbers and the adder will output the sum.

Examples of combinational circuits used inside a microprocessor circuit include adders, multiplexers, decoders, arithmetic and logic unit (ALU), and comparators. In part I of this three-part series courseware you will learn about these and many other combinational circuits.

Chapter 3

Digital Logic Circuits

3.1. Basic Logic Gates

All digital circuits are implemented with logic gates. The three fundamental logic gates are the AND gate, the OR gate and the NOT gate. Using these three types of logic gates, the most complex microprocessor circuit can be built.

Logic gates are built from transistors—the fundamental active component for all digital logic circuits. Transistors are just binary switches that can be turned on and off. The two binary values, 1 and 0, are used to represent the on and off states of a transistor; usually a 1 means “on” or having voltage, and a 0 means “off” or no voltage. In operation, a transistor is like your light switch on the wall where you can either turn it on or off. When you turn on the light switch, current flows through the switch and the light comes on. Physically, transistors are of course much smaller than your light switch.

3.1.1. AND, OR and NOT Gates

An analogy for the operation of the AND gate is like connecting two binary switches together in series as shown in Figure 6 (a). In the diagram, the signal flows from the input on the left-hand side to the output on the right-hand side through the two switches *x* and *y*. In order for a signal to travel from the input to the output, both switches have to be turned on (i.e., connected). If either one or both switches are turned off, then a signal would not be able to get from the input to the output.

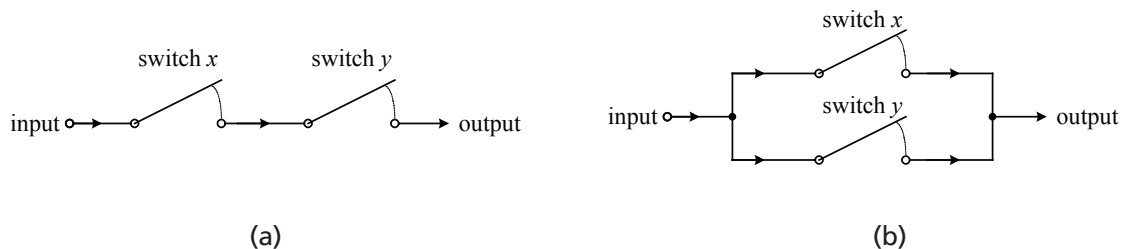


Figure 6: Two possible connections of two binary switches: (a) in series; (b) in parallel. In both diagrams, the switches are in their “off” positions.

The OR gate, on the other hand, is like connecting two binary switches together in parallel as shown in Figure 6 (b). In order for a signal to travel from the input to the output, either one or both of the switches have to be turned on. A signal is prevented from getting from the input to the output only if both switches are turned off. Note however, that when both switches are turned on, the output signal is not doubled, but is the same as having only one switch turned on.

Using the convention of a 1 meaning “on” and a 0 meaning “off”¹, and assuming that the input

¹ This familiar convention is referred to as active-high. In practice, the active-low convention is sometimes used where a 1 means “off” and a 0 means “on”.

Combinational Logic Design

has a 1, then the AND gate will output a 1 only when both switches are turned on (i.e., both switches are 1's); otherwise the output will be a 0. Since we assume that the input is always a 1, therefore, we are only interested in the status of the two switches x and y , that is, whether they are a 1 (on) or a 0 (off).

For the OR gate, and having a 1 input, the output is a 1 when either one or both of its switches are turned on (i.e., either one or both of the switches is a 1); otherwise the output will be a 0. Again, we assume that the input is always a 1, and so we are again only interested in the status of the two switches, whether they are a 1 (on) or a 0 (off).

The operation of the NOT gate simply inverts the value at its input. So if the input is a 0, then the output will be a 1, and vice versa.

3.1.2. Truth Tables

We formally describe the operation of a logic gate or any combinational digital circuit with a truth table. A truth table is a two-dimensional table that lists all possible combinations of the circuit's input values and their corresponding output values. The headings for the columns are labeled with the input and output signal names, and there is one row for each combination of input values. The values in the output column(s) are deduced by analyzing the operation of the circuit for each combination of input values.

The truth table for the AND gate is shown in Figure 7 (a). In this truth table, the two columns labeled x and y are the input signal names, and they represent the status of the two binary switches x and y . The column labeled f is the output from the AND gate. Under the x and y columns, we list all possible combinations of the status of these two binary switches, again using the convention that a 1 means that the switch is turned on and a 0 means that the switch is turned off. In the first row with both x and y being a 0, meaning that both switches are turned off, the output of the AND gate will be a 0, hence f for this row is a 0. Thus, having understood the operation of the AND gate using the binary switch analogy, we can deduce all of the output values for the AND gate. As you can see, f is a 0 for all values of x and y except for when both of them are a 1, then f is a 1.

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

(a)

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

(b)

x	f
0	1
1	0

(c)

Figure 7: Truth tables for the: (a) AND gate; (b) OR gate; (c) NOT gate. In these truth tables, x and y are the inputs and f is the output.

The truth table for the OR gate is shown in Figure 7 (b). The binary switch analogy for the OR gate shows that the output is a 0 only when both of the switches are a 0 (off). Hence, f is a 0 only when both x and y are 0's, otherwise, f is a 1.

Ch. 3 Digital Logic Circuits

The truth table for the NOT gate is shown in Figure 7 (c). Notice that the NOT gate only has one input x . When the input x is a 0 then the output f is a 1, and when x is a 1 then the output f is a 0.

3.1.3. Logic Symbols

When drawing digital circuit diagrams, also referred to as schematic diagrams, we use logic symbols to represent these gates. The logic symbols for the AND, OR and NOT gates are shown in Figure 8. x and y are the inputs to these gates, and f is the output.

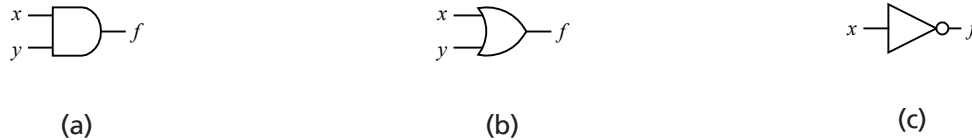


Figure 8: Logic symbols for the: (a) AND gate; (b) OR gate; and (c) NOT gate. x and y are the inputs and f is the output.

3.1.4. NAND, NOR, XOR and XNOR Gates

As mentioned before, any digital circuit, no matter how complex they may be, can be built using these three types of basic gates. However, there are several other gates that are derived from the AND, OR and NOT gates that are also used frequently in digital circuits. They are the NAND gate, NOR gate, XOR gate and XNOR gate. Their logic symbols and truth tables are shown in Figure 9.

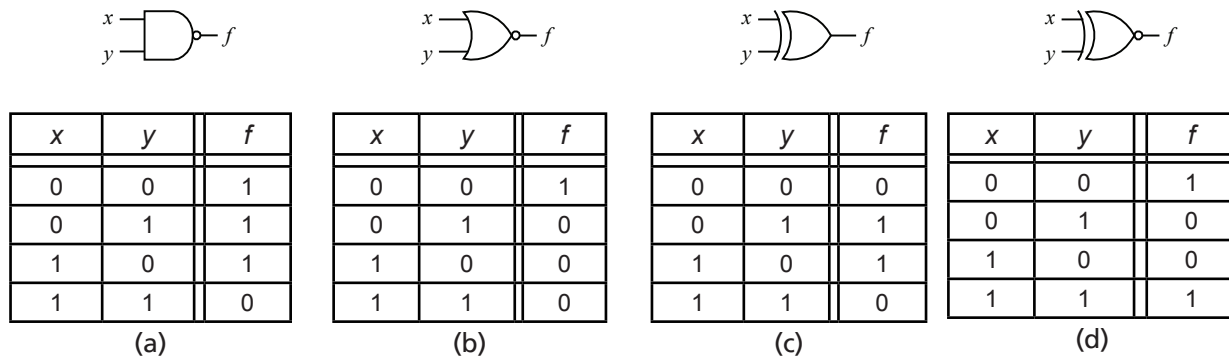


Figure 9: The logic symbols and truth tables for: (a) NAND gate; (b) NOR gate; (c) XOR gate; (d) XNOR gate. In these truth tables, x and y are the inputs and f is the output.

The NAND gate is derived from connecting a NOT gate to the output of an AND gate, hence the name "NAND" which came from the two words Not-AND. Recall that the NOT gate inverts the input value. Thus, the values in the output column of the truth table for the NAND gate are inverted from that of the AND gate. We say that the operation of the NAND gate is the inverse of the AND gate, and vice versa.

Similarly, the NOR gate is derived from connecting a NOT gate to the output of an OR gate, hence the name Not-OR or simply "NOR". Again, the values in the output column of the truth table for the NOR gate are inverted from that of the OR gate.

Combinational Logic Design

The XOR gate, also known as the eXclusive-OR gate, is similar to the OR gate but the output is a 1 only when one of the inputs is a 1. If both inputs are a 1 then the output is a 0.

The 2-input XNOR gate is the inverse of the 2-input XOR gate. However, the XNOR gate is not always the inverse of the XOR gate. It turns out that the XNOR gate is the inverse of the XOR gate only for when the number of inputs is even. When the number of inputs is odd, the XNOR gate operates exactly like the XOR gate.

In addition to having just two inputs for all of the above mentioned gates (except for the NOT gate), they can have more than two inputs. Theoretically, there is no upper limit to the maximum number of inputs to these gates. In practice, however, they have only 2-, 3-, 4-, 6- and 8- inputs. Regardless of the number of inputs they have, they always have only one output. Figure 10 shows the symbols for a 3-input NAND gate, a 4-input AND gate, and a 6-input OR gate.

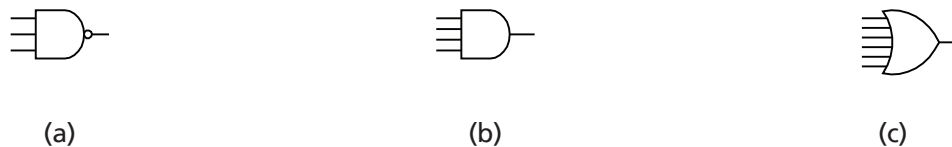


Figure 10: The logic symbols for: (a) 3-input NAND gate; (b) 4-input AND gate; (c) 6-input OR gate.

The truth tables for the 3-input NAND gate, 3-input AND gate and 3-input OR gate are shown in Figure 11.

x	y	z	f
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

(a)

x	y	z	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

(b)

x	y	z	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

(c)

Figure 11: Truth table for: (a) 3-input NAND gate; (b) 3-input AND gate; (c) 3-input OR gate.

3.2. Digital Circuits

A digital circuit is just the connections of a number of these logic gates together. There are certain rules, however, that one must follow in connecting these logic gates together to make an operationally correct circuit. (1) The output of a gate is always connected to the input of one or more gates unless it is a primary output. (2) Primary outputs are the very last outputs from the circuit and are not connected to the inputs of other gates, but instead are connected to external devices such as light emitting diodes (LEDs) and VGA monitors. (3) The outputs from two or more gates cannot be connected to the same input of the same gate. (4) Primary inputs, which are inputs from external devices such as switches and push buttons, are always connected to the inputs of various gates.

Ch. 3 Digital Logic Circuits

At this point, we will focus on the logical aspects of designing the digital circuit itself and not on how the external devices are connected to the primary inputs and primary outputs. For our implementation and testing purposes, we will connect the primary inputs to switches and buttons that will simply produce a 0 or a 1, and connect the primary outputs to simple LEDs that will be turned on when there is a 1 and turned off when there is a 0.

Figure 12 are schematic drawings of some very simple digital circuits.

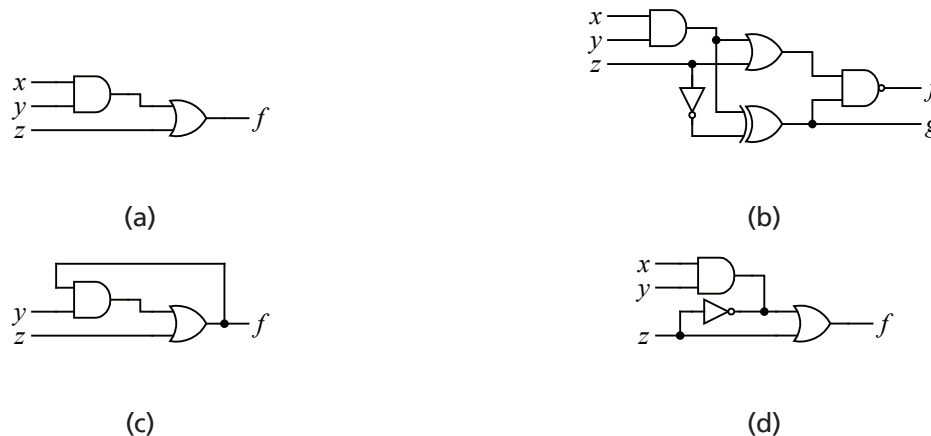


Figure 12: Sample digital circuits: (a) combinational circuit; (b) combinational circuit with one output connected to two inputs; (c) sequential circuit; (d) invalid circuit with two outputs connected to the same input.

In Figure 12 (a), the circuit has three primary inputs, x , y and z , from the external world, i.e., the user supplies either a 0 or a 1 to each of these three inputs. x and y are connected to the inputs of the AND gate, and z is connected to one input of the OR gate. The output of the AND gate is connected to the second input of the OR gate. Finally the primary output of the circuit is f which is from the output of the OR gate. Signals, either as 0's or 1's, originate from the primary inputs and travel through the various gates, finally ending at the primary output f .

The circuit in Figure 12 (b) has three inputs, x , y and z , and two outputs, f and g . This circuit also shows an example of where the output of one gate is connected to the inputs of two different gates. As you can see, the output of the AND gate is connected to the input of the OR gate and to the input of the XOR gate. Also, the primary input z is connected to both the input of the NOT gate and to the input of the OR gate. The output of the XOR gate is connected to the input of the NAND gate, and is also the primary output signal g . The output from the NAND gate is the second primary output signal f .

The circuit in Figure 12 (c) is very similar to the circuit in Figure 12 (a). The one main difference, besides having only two inputs, is that the output from the OR gate is connected back to the input of the AND gate.

Finally, the circuit in Figure 12 (d) will produce errors because both the AND gate output and the NOT gate output are connected to the same input of the OR gate. Think about what happens if the AND gate outputs a 1 and the NOT gate outputs a 0? Since a 1 is having voltage like +5 volts, and

Combinational Logic Design

0 is ground, and by connecting them together, you are creating a short circuit between power and ground which is similar to connecting the plus and minus terminals of a battery directly together with a piece of wire!

3.3. Identifying Combinational Circuits

Before learning to design combinational circuits, we need to be able to determine whether a given digital circuit is a combinational circuit or not. And if it is a combinational circuit, then we want to be able to formally describe its operation by deriving the truth table for it.

Looking back at the three correct digital circuits in Figure 12, i.e., (a), (b) and (c), two of them are combinational circuits and one is a sequential circuit. Can you figure out which two are combinational circuits? The difference is not in how many inputs or outputs the circuit has, but rather in how the gates are connected together.

Notice in Figure 12 (a) and (b), signals, as determined by the connections in the circuits, all flow in one general direction from the primary inputs on the left-hand side to the primary output on the right-hand side. However, in Figure 12 (c), not only is the output from the AND gate connected to the input of the OR gate, but the output from the OR gate is connected back to the input of the same AND gate, thus, creating a feedback loop. The difference is in the presence of this feedback loop. Another way to identify a feedback loop is that the output of a gate is connected back to one of its own input either directly or indirectly via other gates.

A circuit is a combinational circuit only if it does not have any feedback loops. Hence, the first two circuits without a feedback loop are combinational circuits. The circuit in Figure 12 (c), because of the feedback loop, makes it a sequential circuit. Be careful, however, that in larger sequential circuits, the feedback loop can go through many gates, and not just two as shown in Figure 12 (c). There might also be more than one feedback loop within a circuit, but as long as there is at least one loop, the circuit is a sequential circuit. This is the only distinction between a combinational circuit and a sequential circuit. Hence, it is very easy to tell whether a given digital circuit is a combinational circuit or a sequential circuit.

3.4. Analysis of Combinational Circuits

Analyzing a circuit means to determine its functional operation. In analyzing a combinational circuit, we are given a combinational circuit and we want to find out how it operates. The functional operation of a combinational circuit is described formally with a truth table, and you have already seen several truth tables for the basic logic gates. So in the analysis of combinational circuits, what we want to do is to derive the truth table for a given combinational circuit.

As an example, let us analyze the combinational circuit in Figure 13 (c). The first step in the analysis process is to set up the truth table for it. First, we list all of the primary inputs found in the circuit, one input per column, followed by all of the primary outputs found in the circuit, again one output per column. These columns are labeled with the names of the inputs and outputs. Since the circuit in Figure 13 (c) has three input variables, x , y and z , and one output variable, f , therefore we obtain a

table having four columns with the four variable names as shown in Figure 13 (a). In the second step, we enumerate all possible combinations of 0's and 1's for all of the input variables. For three variables, we will have $2^3 = 8$ different combinations. In general, for a circuit with n input variables, there will be 2^n combinations going from 0 to $2^n - 1$. We will insert a row for each combination into the table. Figure 13 (b) shows the eight rows with the eight sets of input values in binary counting order.

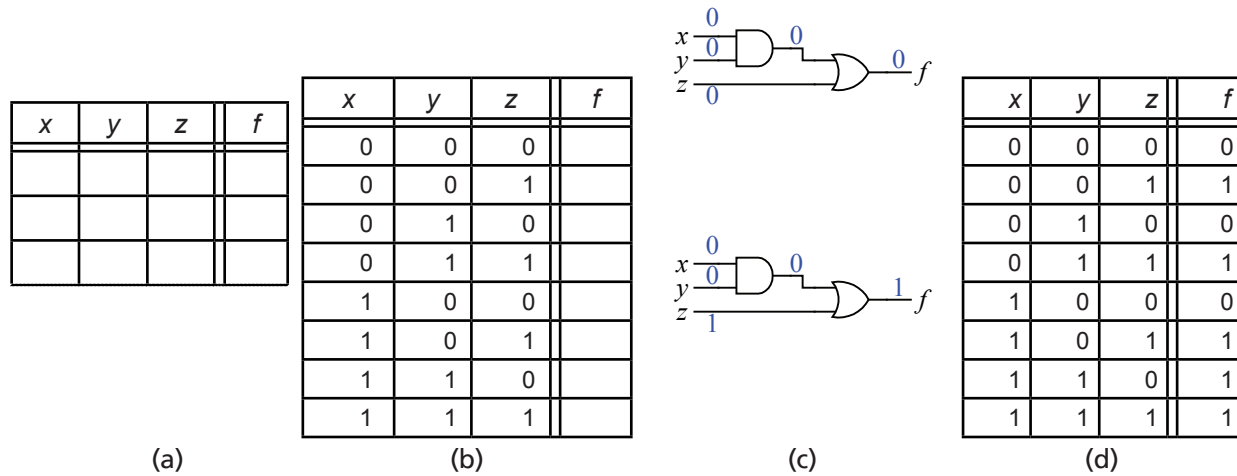


Figure 13: Deriving a truth table for a combinational circuit: (a) Step one—create a column for each input and output; (b) Step two—enumerate all possible combinations of 0's and 1's for the inputs; (c) Step three—for each set of input values, trace through the circuit to determine the output value. The top circuit is annotated with the first set of input values and the bottom circuit is annotated with the second set of input values; (d) the completed truth table.

The third and final step is to fill in the values for the output column(s). For each row in the table (that is, for each set of input values), we need to determine what the output value is. This is done by first substituting each set of input values into the circuit's primary inputs. Knowing the primary input values, we can determine for each gate in the circuit what its output ought to be by starting from the primary inputs and tracing through the circuit to the final output. For example, in the top circuit in Figure 13 (c), using $xyz = 000$ (i.e., $x = 0$, $y = 0$ and $z = 0$), the two inputs to the AND gate are both 0. Hence the output of the AND gate will be a 0, since from the AND gate truth table, 0 AND 0 is 0. This 0 from the output of the AND gate goes to one input of the OR gate, and the other input to the OR gate is also a 0 (from $z = 0$). From the OR gate truth table, we see that 0 OR 0 is 0. Hence, the output of the OR gate, which is also the primary output for the circuit at f is 0. Therefore, the value 0 for f is inserted into the truth table under the f column and in the row where $xyz = 000$ as shown in Figure 13 (d).

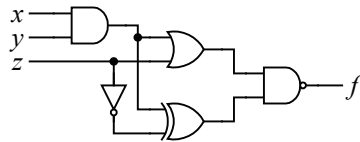
Continuing on for the next set of input values where $xyz = 001$ as shown in the bottom circuit in Figure 13 (c), the output of the AND gate is again 0. However, with z being a 1 going into the OR gate, the output of the OR gate will be a 1. Therefore, f is a 1 for this second set of inputs. This 1 is inserted under the f column and in the second row of the truth table where $xyz = 001$. This tracing process is repeated for the rest of the combinations, and at the end you will have the completed truth table for the circuit as shown in Figure 13 (d).

To help speed up the tracing process, notice that when input z is a 1, the output of the OR gate will

Combinational Logic Design

always be a 1 regardless of the other input. Therefore, when z is a 1, it doesn't matter what x and y are, f will always be a 1. By reasoning this way, you can more quickly determine many of the output values without having to actually trace through the entire circuit with each set of numbers.

As an exercise, derive the truth table for the circuit shown in Figure 14 (a). Try to derive your own truth table first, and after you have done that, then compare your results with the answer shown in Figure 14 (b).



(a)

x	y	z	f
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

(b)

Figure 14: (a) A combinational circuit and (b) the truth table for it.

3.5. Boolean Algebra

Instead of drawing combinational digital circuits using logic gate symbols as we have seen so far, we can use Boolean equations to represent these combinational circuits. Using Boolean equations to represent combinational circuits allow us to easily manipulate them by using Boolean algebra. Boolean algebra is very similar to the algebra in mathematics with which you should already be familiar. For instance, both of them have constants and variables, and operators for manipulating the constants and variables. Furthermore, both of them have axioms and theorems, and many of the theorems, such as the commutative theorem and the associative theorem, are very similar. One main difference between the two, however, is the fact that in mathematics you normally work with decimal numbers which use the digits from 0 to 9, but in Boolean algebra, we work with binary numbers which use only the two binary digits 0 and 1. Another main difference between the two is that in mathematics, there are four operations, add, subtract, multiply and divide, that you perform on the constants and variables, but in Boolean algebra, there are only three operations, AND, OR and NOT. With the reduced number of constants and operations, you can guess that Boolean algebra should be much simpler, and it is.

3.5.1. Expressions and Equations

When writing Boolean expressions or Boolean equations, we use the symbol \bullet or no symbol at all for the AND operation, the symbol $+$ for the OR operation, and the symbol $'$ or $\bar{}$ for the NOT operation. Here are two simple examples of Boolean equations: $0 \bullet x = 0$ reads "0 AND x is equal to 0," and

$0' = 1$ or $\bar{0} = 1$ reads "NOT 0 is equal to 1." Another example is

$$f = x + yz'$$

which reads "f is equal to x OR y AND NOT z." First of all, notice that these Boolean equations are written just like mathematical equations, except that the + symbol is not for addition but for the OR operation, and the • symbol is not for multiplication but for the AND operation. Second, notice that the • symbol for the AND operation can be removed (just like the multiplication in math), so $y \bullet x$ can be written as yx . Third, the expression is normally evaluated from left to right, but just like in math where the multiplication and division operations are performed before the addition and subtraction operations, so in Boolean equations, the NOT operation has the highest precedence, followed by the AND operation, and lastly the OR operation. The ordering in which these operations are performed can be changed by the use of parenthesis. So in the equation

$$f = x + yz'$$

the NOT z is performed first, followed by the y AND z', and lastly the x OR yz'. If we want to perform the x + y before the yz, we would have to add in parenthesis like this

$$f = (x + y)z'$$

If we want to negate (NOT) the entire expression $(x + y)z$, we would have to write either

$$f = ((x + y)z)' \quad \text{or} \quad f = \overline{(x + y)z}$$

Be careful when you use a variable name that has more than one letter such as *sum*, and not interpret it as $s \bullet u \bullet m$.

There are no special symbols for the NAND and NOR operations. The expression for the NAND operation is just $(xy)'$ and the expression for the NOR operation is just $(x + y)'$.

Two other symbols are also used in Boolean expressions and equations. They are \oplus for the XOR operation and \odot for the XNOR operation. From the definition of the XOR operation, we have the equality

$$x \oplus y = x' y + xy'$$

and from the definition of the XNOR operation, we have the equality

$$x \odot y = x' y' + xy$$

See Experiments 4 and 6 of Lab 2 for proving the correctness of these two equalities.

3.5.2. Axioms and Theorems

The axioms and theorems for Boolean algebra are listed in Figure 15. Figure 15 (a) lists the axioms, (b) lists the single-variable theorems, and (c) lists the two- and three-variable theorems. All of them come in pairs. For example, axiom 1a and axiom 1b is a pair. The pairing basically comes from the fact

Combinational Logic Design

that one set uses the AND operation (the “a” set), and the second set uses the OR operation (the “b” set).

The first four pairs of axioms listed in Figure 15 (a) are basically just a summary of what we already know about the AND, OR and NOT operations from their truth tables. 1a, 2a and 3a are directly from the AND truth table; 1b, 2b and 3b are directly from the OR truth table; and 4a and 4b are directly from the NOT truth table.

1a.	$0 \cdot 0 = 0$	1b.	$1 + 1 = 1$
2a.	$1 \cdot 1 = 1$	2b.	$0 + 0 = 0$
3a.	$0 \cdot 1 = 1 \cdot 0 = 0$	3b.	$1 + 0 = 0 + 1 = 1$
4a.	$0' = 1$	4b.	$1' = 0$

(a)

5a.	$0 \cdot x = 0$	5b.	$1 + x = 1$
6a.	$1 \cdot x = x$	6b.	$0 + x = x$
7a.	$x \cdot x = x$	7b.	$x + x = x$
8a.	$(x')' = x$		
9a.	$x \cdot x' = 0$	9b.	$x + x' = 1$

(b)

10a.	$x \cdot y = y \cdot x$	10b.	$x + y = y + x$
11a.	$(x \cdot y) \cdot z = x \cdot (y \cdot z)$	11b.	$(x + y) + z = x + (y + z)$
12a.	$x \cdot (y + z) = (x \cdot y) + (x \cdot z)$	12b.	$x + (y \cdot z) = (x + y) \cdot (x + z)$
13a.	$(x \cdot y)' = (x' + y')$	13b.	$(x + y)' = (x' \cdot y')$

(c)

Figure 15: Boolean algebra axioms and theorems: (a) axioms; (b) single-variable theorems; (c) two- and three- variable theorems.

The single-variable theorems listed in Figure 15 (b) basically show what happens when you combine a single Boolean variable with a constant. And again, these are a direct result from the definitions of the AND, OR and NOT operations. For example, theorem 5a says that when you AND a 0 with another value (in this case the variable x), it doesn't matter what this other value is the result will always be a 0. Similarly, theorem 5b says that when you OR a 1 with another value (in this case the variable x), it doesn't matter what this other value is the result will always be a 1. Theorems 6a and 6b say that if you AND a 1 with x or OR a 0 with x , the result will always be x . Again, theorems 5 and 6 are a direct result from the definitions of the AND and OR operations. Theorem 7a is obtained from axioms 1a and 2a, and theorem 7b is obtained from axioms 1b and 2b. Theorem 8 simply says that if you double negate a value, you get back the original value. For theorem 9a, you AND x with x' . But since there are only two possible constant values (0 and 1), therefore, if x is a 1 then x' has to be a 0, and vice versa. Therefore, one of the two values that you AND with will always be a 0, and so the result of the AND will always be a 0 (from theorem 5a).

For the two- and three-variable theorems in Figure 15 (c), theorem 10 is just the commutative property in mathematics that you know; theorem 11 is the associative property; and theorem 12 is

the distributive property. Note that for theorem 12a and b, you can have the AND distributing over the OR, and the OR distributing over the AND. This is not true for the math version of the distributive property, but true in Boolean algebra.

Theorem 13 is very special and so it has a name; it is called DeMorgan's theorem's theorem. Theorem 13a says that if you take the inverse of $(x \text{ AND } y)$, you get (the inverse of x) OR (the inverse of y). Similarly for theorem 13b, if you take the inverse of $(x \text{ OR } y)$, you get (the inverse of x) AND (the inverse of y). An easy way to remember this is to change all the $+$ to \bullet and vice versa, and change all the prime to no prime and vice versa. See Section 3.7, and Lab 1, Experiments 14 and 15 for proving the correctness of this theorem.

3.5.3. Manipulating Boolean Equations

Given a Boolean equation, we can modify it, yet without changing its functional equivalency, by applying any of the Boolean axioms and theorems to it. For example, if we start with the following equation

$$f = y + xy$$

we can transform it to any of the following intermediate forms and they will all be functionally equivalent

$$f = y + xy$$

$$= 1 \bullet y + xy \text{ using theorem 6a}$$

$$= (x + x') \bullet y + xy \text{ using theorem 9b}$$

$$= xy + x' y + xy \text{ using distributive theorem 12a}$$

$$= xy + xy + x' y \text{ using commutative theorem 10b}$$

$$= xy + x' y \text{ using theorem 7b}$$

$$= (x + x')y \text{ using theorem distributive theorem 12a in reverse}$$

$$= 1 \bullet y \text{ using theorem 9b}$$

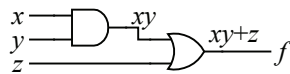
$$= y \text{ using theorem 6a}$$

3.5.4. Relationship between Boolean Equations and Combinational Circuits

It turns out that every output signal in a combinational circuit can be expressed as a Boolean equation, and every Boolean equation has a corresponding combinational circuit. Figure 16 shows some sample combinational digital circuits with their corresponding Boolean equation. In Figure 16 (g) and (h), the Boolean equations for the XOR and XNOR gates are not so obvious, because they are actually derived from their corresponding truth tables. Experiments 4 and 6 of Lab 2 show that these two gates are equivalent in operation to the two circuits in Figure 16 (e) and (f) respectively, hence the equations are identical. To derive the Boolean equation for larger combinational circuits, it is easier

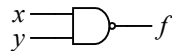
Combinational Logic Design

to annotate the circuit at the output of each gate with their intermediate expression until you reach the final output. Two examples are shown in Figure 16 (i) and (j).



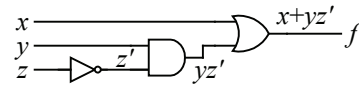
$$f = xy + z$$

(a)



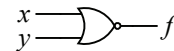
$$f = (xy)'$$

(c)



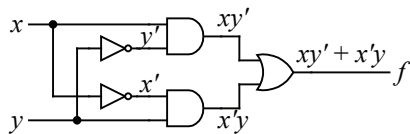
$$f = x + yz'$$

(b)



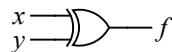
$$f = (x+y)'$$

(d)



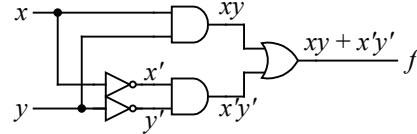
$$f = xy' + x'y$$

(e)



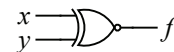
$$f = xy' + x'y$$

(g)



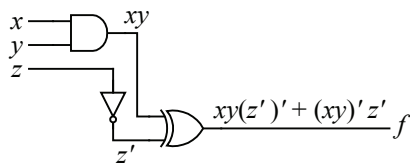
$$f = xy + x'y'$$

(f)



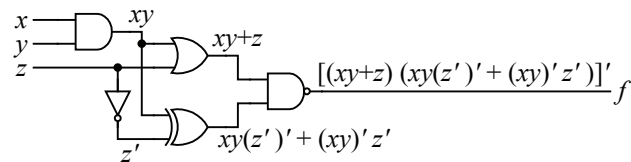
$$f = xy + x'y'$$

(h)



$$f = xy(z')' + (xy)'z'$$

(i)



$$f = [(xy+z)z' + xy]'$$

(j)

Figure 16: Sample combinational digital circuits and their corresponding Boolean equation.

Since there is a direct relationship between a combinational circuit diagram and its corresponding Boolean equation, therefore we can also easily derive the truth table from the equation instead of from the circuit diagram. The procedure would be identical. The only difference is that instead of substituting the input values into the diagram, you would substitute them into the equation.

3.6. Simplifying Combinational Circuits

Notice that the truth table in Figure 14 (b) is exactly the same as the truth table for a 3-input NAND gate shown in Figure 11 (a). What this means is that functionally, the 3-input NAND gate operates exactly the same as the circuit in Figure 14 (a), but obviously the 3-input NAND gate is much smaller in size. Hence, if we want just the functionality of the circuit in Figure 14 (a), we should implement it by using a 3-input NAND gate. This way we will have a much smaller circuit but having the same functionality.

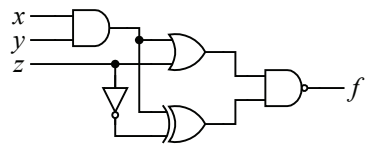
In section 3.5.3, we also saw how we use the Boolean axioms and theorems to transform the equation $f = y + xy$ to a much simpler but functionally equivalent equation $f = y$.

There are different formal methods for simplifying combinational circuits. These include the use of Boolean algebra, Karnaugh-maps (or K-maps for short), and tabulation methods. Using Boolean Theorems and Boolean algebra is the theoretical method, whereas using K-maps is a visual method. K-maps are easier to understand but only works for very few inputs (maybe five at most). In practice, there will be much more than five input variables. Tabulation methods are best suited for programming the computer and they work for any number of inputs. In this courseware, we will discuss the use of Boolean algebra and K-maps.

3.6.1. Using Boolean Algebra

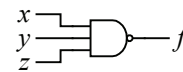
You have already seen in section 3.5.3 how a Boolean equation can be modified, but without changing its functional operation, by applying any of the Boolean axioms and theorems to it. So using the Boolean algebra method to simplify combinational circuits simply means that we start with a Boolean equation of the circuit and then use the axioms and theorems to transform the equation to a more reduced but functionally equivalent equation. The smaller the equation, the smaller the circuit will be.

We have noted that the two circuits shown in Figure 17 are equal in operation. Their corresponding equations are also shown.



$$f = [(xy+z)(xy(z')') + (xy)' z']']'$$

(a)



$$f = (xyz)'$$

(b)

Figure 17: Two functionally equivalent combinational circuits.

We will now show how the equation $f = [(xy+z)(xy(z')') + (xy)' z']']'$ can be transformed to $f = (xyz)'$ using the Boolean axioms and theorems. There are no set rules as to which axiom or theorem is to be used first and when to use it. So there might be other ways to perform this transformation.

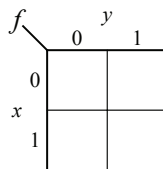
Combinational Logic Design

$$\begin{aligned}
 f &= [(xy+z)(xy(z')' + (xy)'z')]' \\
 &= [(xy+z)(xyz + (xy)'z')]' \text{ using theorem 8a on } (z')' \\
 &= [(xy+z)(xyz + (x' + y')z')]' \text{ using theorem 13a on } (xy)' \\
 &= [(xy+z)(xyz + x'z' + y'z')]' \text{ using theorem 12a on } (x' + y')z' \\
 &= [xyxz + xyx'z' + xyy'z' + zxyz + zx'z' + zy'z']' \text{ using theorem 12a two times, one for } xy \text{ and} \\
 &\quad \text{one for } z \\
 &= [xyz + xyx'z' + xyy'z' + xyz + zx'z' + zy'z']' \text{ using theorem 7a two times, one for } xy \text{ and one for} \\
 &\quad z \\
 &= [xyz + xyx'z' + xyy'z' + zx'z' + zy'z']' \text{ using theorem 7b on } xyz + xyz \\
 &= [xyz + 0 + 0 + 0 + 0]' \text{ using theorem 9a and 5a four times} \\
 &= (xyz)' \text{ using theorem 6b}
 \end{aligned}$$

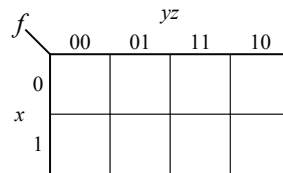
Hence, we have again shown that the two circuits are functionally equivalent.

3.6.2. Using K-Maps

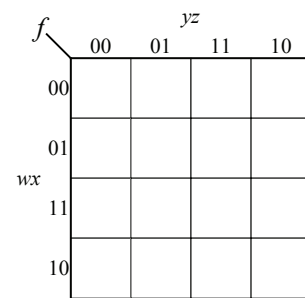
The K-map method is a visual technique for simplifying Boolean equations. The K-map itself is just another way of representing the truth table, and like the truth table, it is also a 2-dimensional table. The main difference is in the labeling of the columns and rows. The columns and rows in a K-map are labeled with the input variable names and their two possible constant values. Since each variable can have either a 0 or a 1, therefore, two columns or two rows are needed for each variable. Figure 18 shows the setup of a K-map for two-, three- and four variables. For the two-variable K-map in Figure 18 (a), we have placed the variable x in the two rows and the variable y in the two columns. The intersection of each row and column gives us the unique value for these two variables hence there are the four intersection boxes that represent the unique combination of the two input variables xy having the values 00, 01, 10 and 11.



(a)



(b)



(c)

Figure 18: K-map setup for: (a) two variables; (b) three variables; and (c) four variables.

Regardless of how many variables the equation has, the K-map for it is still going to be a 2-dimensional table. Hence, for a three-variable K-map, we need to double up two of the variables as shown in Figure 18 (b) for the two variables y and z . (It does not matter whether you put it in the columns or the rows.) Now, each column will have two unique values for yz —00, 01, 10 and 11. Notice, however, that we reversed the label ordering for the third and fourth columns. The reason is that in order for the K-map to work, the values for every adjacent column or row must differ in only one bit. So with this new ordering, 00, 01, 11 and 10, this condition is satisfied. Notice that this condition is also satisfied between the first and last columns, 00 and 10. Hence, you need to visualize that the first and last

columns are also adjacent to each other.

For a four-variable K-map, we will have two variables with four combinations for both the columns and rows as shown in Figure 18 (c). Again, the value labeling for both the third and fourth columns and rows are reversed.

As you can quickly see, it is almost impossible to visualize a five-variable K-map, although it is sometimes done. The fifth variable, v , will be in a third dimension. There will be two four-variable K-maps, one on top of the other, with one for when v is a 0, and the other for when v is a 1.

What we put into the intersection boxes in a K-map are the 1 output values in the equation or truth table. For example, if we want to minimize the equation in Figure 19 (a), the corresponding truth table and K-map for this equation is shown in (b) and (c). We do not need to include the 0 outputs from the truth table in the K-map because we are only interested in when the equation produces a 1 output.

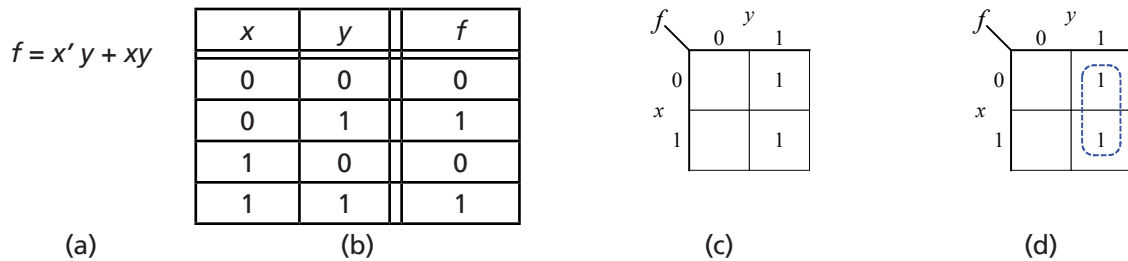


Figure 19: (a) Boolean equation; (b) corresponding truth table; (c) corresponding K-map; and (d) K-map with subcube formed.

Having set up the K-map and added all of the 1 outputs from the truth table into the K-map, we are ready to minimize the equation using the K-map by forming subcubes. We form subcubes by circling adjacent boxes with 1's in them. The following rules must be observed when forming the subcubes.

1. All of the 1-boxes must be physically adjacent to each other except for the two ends. For the two end boxes (such as those in a three- and four-variable K-maps), visualize them as also being adjacent to each other because they also differ in only one bit (from 00 to 10).
2. The size of the subcube (i.e., the number of 1-boxes inside the subcube) must be a power of two. So you can only have 1, 2, 4, 8, etc. number of 1-boxes inside a subcube.
3. The shape of a subcube must be a rectangle either horizontally or vertically.
4. All of the 1-boxes in a K-map must be inside a subcube, but the same 1-box can be inside one or more subcubes.
5. The size of each subcube should be made as large as possible.

Combinational Logic Design

Forming subcubes is like trying to figure out a puzzle where you want to have as few subcubes as possible, and each subcube to be as large as possible². Figure 19 (d) shows the subcube of size 2 containing the two adjacent 1's.

Figure 20 shows some valid subcubes of various sizes. Figure 20 (a) is a rectangular subcube of size 2. Figure 20 (b) is a square subcube of size 4. Figure 20 (c) is a square subcube of size 4. Notice that the four 1's are adjacent to each other because column 10 wraps around and is adjacent with column 00. Figure 20 (d) and (e) both have two subcubes each, one of size 8 and one of size 4. Two of the 1-boxes are in both subcubes. Figure 20 (f) have two subcubes, one of size 4 and one of size 2. For the size-4 subcube, the four cornered 1's are all adjacent because they all wrap around.

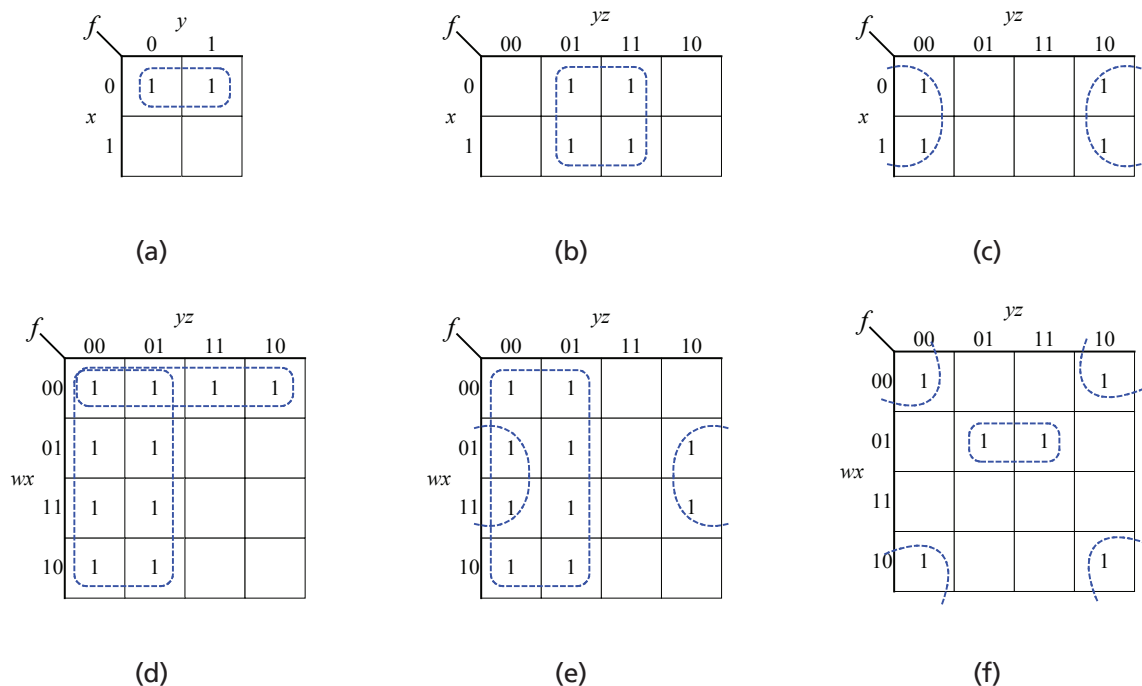


Figure 20: Examples of valid subcubes: (a) subcube of size 2; (b) subcube of size 4; (c) wrap-around subcube of size 4; (d) two subcubes, one of size 8 and one of size 4. Notice that two of the 1's are in both subcubes; (e) two subcubes, one of size 8 and one of size 4; and (f) two subcubes, one of size 2 and one of size 4.

Figure 21 shows some invalid subcubes. In Figure 21 (a), the two size-1 subcubes can be made larger by combining them together into one size-2 subcube. In Figure 21 (b), the subcube size (with three 1's) is not a power of 2. In Figure 21 (c), the subcube shape is not a rectangle. In Figure 21 (d), the size-2 subcube can be made larger into a subcube of size 4. In Figure 21 (e), the subcube with size 6 is not a power of 2, and the size-2 subcube is not horizontal or vertical. In Figure 21 (f), the four 1's are not adjacent.

² Refer to the book "Digital Logic and Microprocessor Design with VHDL" by E. Hwang for a detailed discussion on how to use K-maps to simplify combinational circuits.

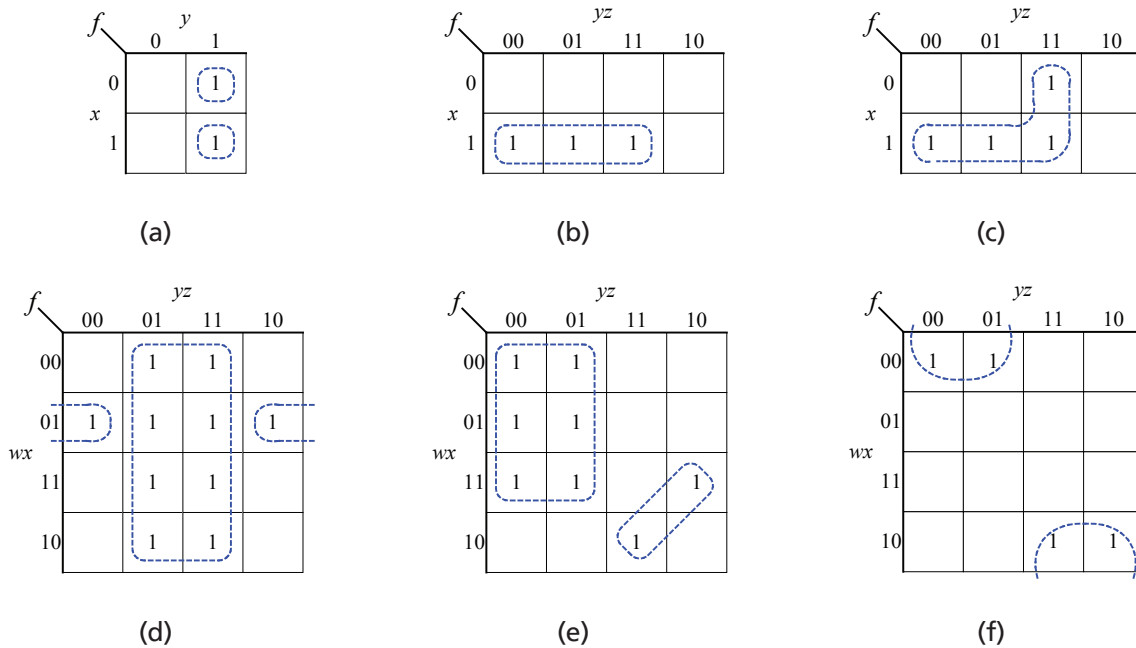
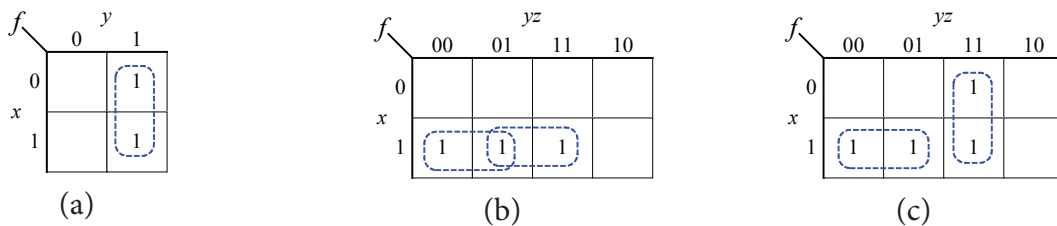


Figure 21: Examples of invalid subcubes: (a) size of both subcubes can be made larger; (b) size of subcube is not a power of 2; (c) shape of subcube is not rectangle; (d) size of the 2 subcube can be made bigger; (e) size of the larger subcube with six 1's is not a power of 2, and the smaller subcube is not horizontal or vertical; (f) the 1-boxes are not adjacent.

Figure 22 shows how to form valid subcubes for the K-maps shown in Figure 21. In Figure 22 (a), the two 1's are grouped together to form one subcube. In Figure 22 (b), the three 1's are separated into two subcubes of size 2 each. In Figure 22 (c), the four 1's are separated into two subcubes of size 2 each. In Figure 22 (d), the size-2 subcube is made into a subcube of size 4. In Figure 22 (e), the six 1's are separated into two subcubes of size 4 each. The 1-box in column 10 is combined with the 1-box in column 00 of the same row to make a subcube of size 2. The 1-box in column 11 is not adjacent to any other 1's, so it is a subcube all by itself. In Figure 22 (f), the two 1's at the top are not adjacent to the two 1's at the bottom, so they must be separated into two subcubes of size 2 each.



Combinational Logic Design

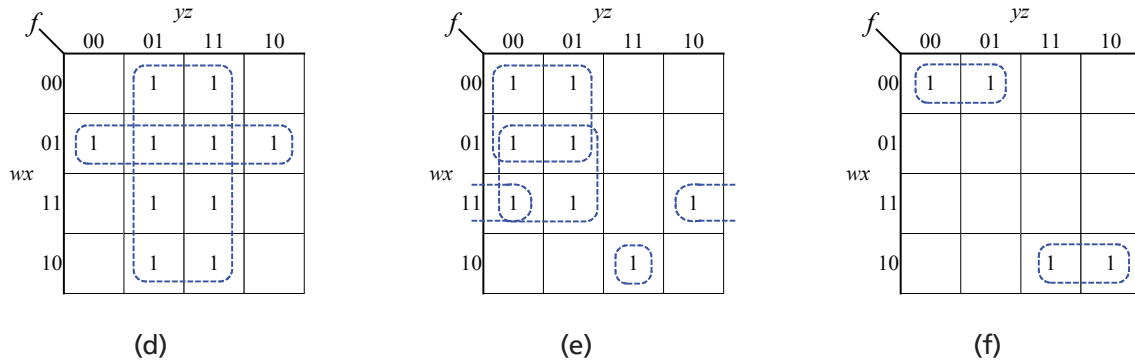
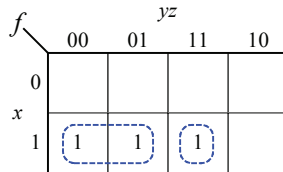


Figure 22: Examples of valid subcubes from the K-maps in Figure 21: (a) size of both subcubes can be made larger; (b) size of subcube is not a power of 2; (c) shape of subcube is not rectangle; (d) size of the 2 subcube can be made bigger; (e) size of the larger subcube with six 1's is not a power of 2, and the smaller subcube is not horizontal or vertical; (f) the 1-boxes are not adjacent.

You may have wondered why for Figure 22 (b) that we did not make the following subcubes instead.



The answer should be obvious because rule 5 says that we need to make each subcube as large as possible, and the subcube of size 1 can be made into a subcube of size 2. Furthermore, rule 4 says that a 1-box can be inside one or more subcubes.

Having formed the subcubes for covering all of the 1's in the K-map, the final step is to write up the reduced equation. Each subcube becomes one AND term in the equation, and all of the AND terms will be ORed together to produce the final simplified equation. For each subcube, write down the variable(s) having the same value for all of the 1-boxes in that subcube. If the value is a 0 then negate the variable, and if the value is a 1 then just leave the variable as is. All of the variables obtained from the same subcube are ANDed together to form one AND term.

Figure 23 shows the simplified equations as obtained from the K-maps in Figure 20. The K-maps are redrawn here for convenience. In Figure 23 (a) the value of x for both of the 1-boxes is 0, but the value of y is a 0 for the left 1-box, and a 1 for the right 1-box. Hence, there is only one variable x that has the same value (a 0) for all of the 1-boxes, therefore this AND term will consist of just one variable x' . The final reduced equation from this K-map is $f = x'$. In Figure 23 (b), the value of x is a 0 for the top two 1-boxes and a 1 for the bottom two 1-boxes. The value of y is a 0 for the left two 1-boxes and a 1 for the right two 1-boxes. Only the value of z for all four 1-boxes is a 1. Hence, the simplified equation from the K-map is $f = z$. In Figure 23 (c), again only z has the same value for all of the four 1-boxes, but this time the common value is a 0 so the equation is $f = z'$. In Figure 23 (d), there are two subcubes. For the size-8 subcube, only y has a 0 for all eight of the 1-boxes, so the AND term

Ch. 3 Digital Logic Circuits

from this subcube is y' . For the size-4 subcube, w has the value 0 and x also has the value 0 for all four of the 1-boxes. The other two variables have different values, so the AND term from this subcube is $w'x'$. The final simplified equation from this K-map will ORed the two AND terms together to give $f = y' + w'x'$. The ordering of the AND terms in the equation does not matter because of the commutative property, but it is preferred to list the larger subcubes (fewer variables in the AND terms) first. In Figure 23 (e), there are also two subcubes. The size-8 subcube is the same as the previous one so the AND term for this subcube is also y' . For the size-4 subcube, x has a common value of 1 and z has a common value of 0, so the AND term for this subcube is xz' . The simplified equation is $f = y' + xz'$. In Figure 23 (f), the size-4 subcube AND term is $x'z'$ and the size-2 subcube AND term is $w'xz$. The simplified equation is $f = x'z' + w'xz$.

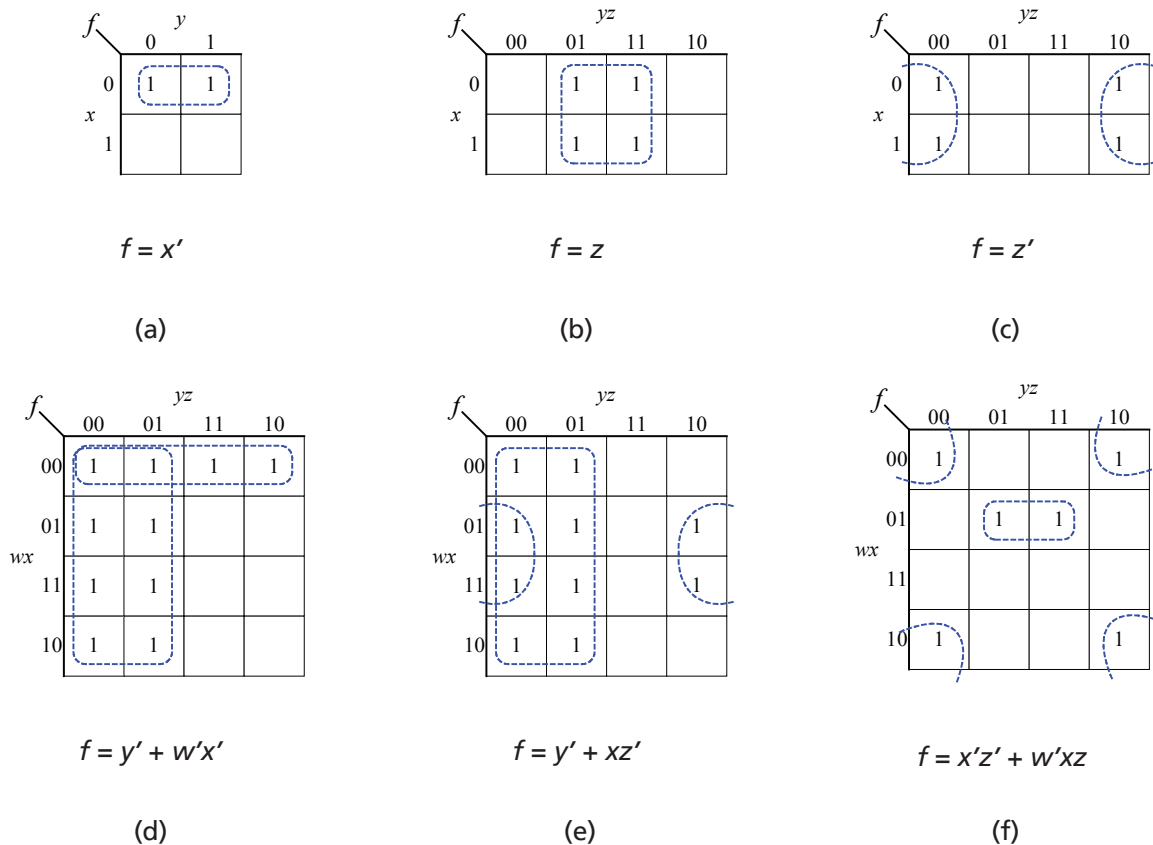


Figure 23: Simplified equations from the K-maps in Figure 20.

As an exercise, you may want to find the simplified equations from the K-maps shown in Figure 22. These equations are: (a) $f = y$; (b) $f = xy' + xz$; (c) $f = xy' + yz$; (d) $f = z + w'x$; (e) $f = w'y' + xy' + wxz' + wx'yz$; (f) $f = w'x'y + wx'y$.

The simplified equation obtained from a K-map may not necessary be the smallest possible equation. After having obtained the simplified equation from a K-map, you may be able to reduce it even further using Boolean algebra.

Combinational Logic Design

3.7. Synthesis of Combinational Circuits

In the synthesis of combinational circuits, we are first given either an informal description³ of the circuit's operation or a formal description with a truth table. If we start with an informal description, then we need to first construct the formal truth table for it. From the truth table, we can derive the Boolean equation for it. Next, we can optionally simplify the equation. And finally from the (simplified) equation we can derive and implement the circuit.

Deriving the equation from a given truth table is actually quite straight forward. There will be one equation created for each output signal in the circuit. The steps to create an equation for one output signal are as follows:

1. Create an AND term for each row in the truth table where the output signal is a 1. Each AND term is created as follows. For that one row of input values:
 - a. If the value for an input variable is a 1 then use that variable as is.
 - b. If the value for an input variable is a 0 then negate (NOT) the variable.
 - c. All of the variables from a. and b. above for that one row are ANDed together to form one AND term.
2. All of the AND terms created in step 1 above are ORed together to produce the final equation.

Once we have the equation, we can easily derive the circuit for it as shown in Section 3.5.4. Thus, given any truth table, we can produce a circuit for it by simply ANDing the inputs of a row for which the output of that row is a 1, and then ORing the outputs of all the AND gates together. If the truth table has two or more output variables then you repeat the same process creating one equation for each output variable.

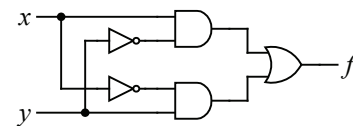
Let us now synthesize the circuit for the truth table shown in Figure 24 (a). It has two inputs, x and y , and one output f .

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

(a)

$$f = x' y + x y'$$

(b)



(c)

Figure 24: Synthesis example: (a) truth table; (b) equation; and (c) synthesized circuit.

We see that there are two rows where f has a 1, so we will have two AND terms, one for each row.

³ As in when your supervisor gives you a verbal imprecise description of a circuit that he or she wants.

Ch. 3 Digital Logic Circuits

In the first row where f has a 1, the value for x is 0 so we negate the variable x to get x' . In the same row, the value for y is 1 so we don't need to change it. The AND term for this first row is therefore $x'y$. Continuing on in the same manner, in the second row where f has a 1, x is a 1 and y is a 0, so the AND term for this second row is xy' . Finally, we ORed these two AND terms together to give the resulting equation for the circuit shown in Figure 24 (b). The circuit as obtained from this equation is shown in Figure 24 (c). You probably recognize the equation in Figure 24 (b) as the equation for the XOR gate. It has to be because the truth table that we started out with is for the XOR gate.

As another example, let us synthesize the circuit for the truth table shown in Figure 25 (a).

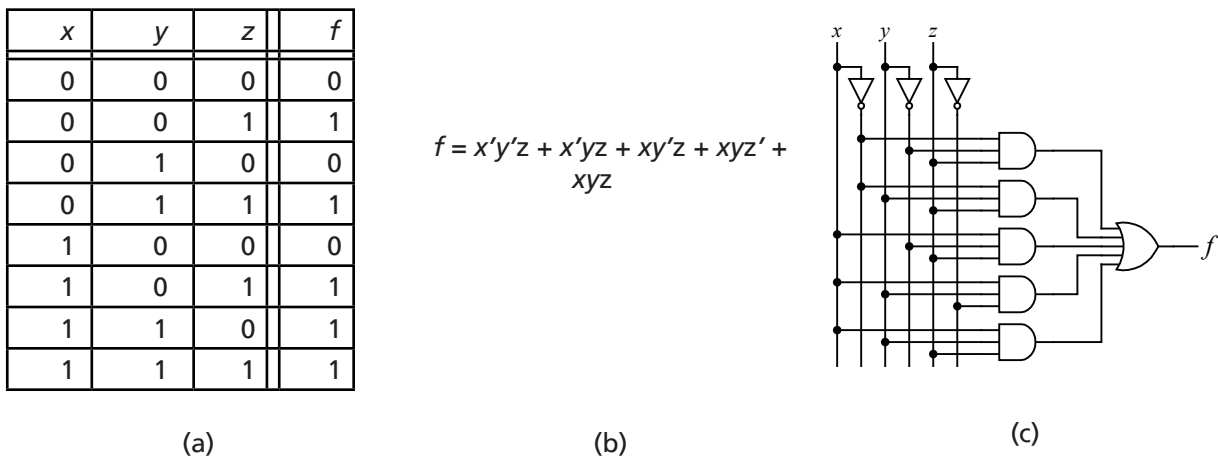


Figure 25: Synthesis example: (a) truth table; (b) equation; and (c) synthesized circuit.

There are five 1's in the output f column, so we will have five AND terms in the final equation. These five AND terms in order are: $x'y'z$, $x'yz$, $xy'z$, xyz' and xyz . ORing these five AND terms together give us the equation for the circuit shown in Figure 25 (b). The circuit as derived from this equation is shown in Figure 25 (c).

You may have recognized that this truth table in Figure 25 (a) is the same truth table from Figure 13 (d). However, the circuit obtained from this truth table certainly is much larger in size than the circuit shown in Figure 13 (c)! The conclusion is that if you start out with a circuit, derive the truth table for it, and then derive the circuit from the truth table, you most likely will not get the same circuit that you started out with. The equation that you obtain directly from a truth table without any simplification will always produce the largest circuit for that truth table. You should, therefore, always try to simplify it to get a smaller circuit.

If the truth table has more than one output, then you simply create one circuit for each output. However, it is possible that some AND terms are the same and so you can remove the duplicate ones. For example, given the truth table with two outputs f and g shown in Figure 26 (a), the two separate circuits for f and g are shown in Figure 26 (b). Notice, however, that both outputs have a 1 for the row $xyz = 110$. The AND term for this row is needed in both circuits, but the total circuit can be made smaller by using only one AND gate for both of these two AND terms as shown in Figure 26 (c).

Chapter 4

Labs

The following labs will teach you how to design and implement combinational circuits. Many of these circuits are standard components used in microprocessor circuits. You will need to understand these circuits in order for you to use them in our Microprocessor Design Trainer where you will actually design and implement your very own custom real working microprocessor!

4.1. Lab 1: Basic Gates, Lights, Action!

Purpose

In this lab you will first learn how to implement combinational logic circuits using the Combinational Logic Trainer by connecting the basic logic gates and I/Os correctly for a given circuit. You will use the trainer to confirm the operations of the basic gates. Finally, you will learn how to derive the truth table for any combinational circuits.

Introduction

The AND, OR and NOT gates are the basic building blocks for building any digital logic circuits. Any digital logic circuit, no matter how large or complex it is, can be constructed using these three types of basic gates. Therefore, in order to design digital logic circuits, you need to know how these gates operate.

Experiments

1. The three thick lines in Figure 2B show three wires connected from the two switches, SW6 and SW7, to the inputs of the 2 input AND#12 gate ⁴, and the output of this AND ⁵ gate is connected to LED6. Using three pieces of hook-up wires, make these same connections now on your trainer. Slide the two switches up and down and record the output on LED6 for all combinations of the input switches in the blank truth table provided at the end of this lab. Remember that when the switch is in the up position, it is a logic 1, and when it is in the down position, it is a logic 0. Verify that your results match the AND logic truth table shown in Figure 7 (a) (Page 10).
2. After you have confirmed that the AND gate does work according to the truth table, repeat the experiment with the 2-input OR gate. Connect the two switches to the inputs of a 2-input OR gate, and connect the output from the OR gate to the LED. Slide the two switches up and down and record the LED output for all combinations of the input switches in the blank truth table provided. Verify that your results match the truth table in Figure 7 (b).

⁴ Remember that we will be using notations such as 2-AND#12 to refer to specific gates on the trainer for ease of debugging.

⁵ Note that it doesn't have to be that specific 2-input AND gate. Any 2-input AND gate would work also.

Combinational Logic Design

3. Repeat the experiment with the NOT gate. Connect one switch to the input of a NOT gate, and connect the output from the NOT gate to a LED. Record your results in the blank truth table provided. Verify that your results match the truth table in Figure 7 (c).
4. Repeat the experiment with the 4-input AND gate and the 4-input OR gate. Record your results in the blank truth tables provided.
5. If you need to use a 3-input AND gate, you would normally get a 3-input AND gate. However, you can also use a 4-input AND gate instead by connecting the unused 4th input to a logic 1. Remember that connecting to VCC will always give you a logic 1. On the trainer, the VCC connection points are located next to the GND connection points at the bottom right corner.

For this experiment, connect three switches to three inputs of a 4-input AND gate, connect the 4th input of the AND gate to VCC, and connect the output from the AND gate to a LED. Slide the three switches up and down and record the LED output for all combinations of the input switches in the blank truth table provided. Verify that your results match the truth table in Figure 11 (b) (page 12). What happens if you connect the extra input of the 4-input AND gate to GND instead of VCC?

For ease of use, all of the 4-input AND gates in the trainer have pull-up resistors connected to all of its inputs. What this means is that you do not have to actually connect any unused inputs of the 4-input AND gates to VCC and it will still work. Verify this by disconnecting the wire that connects from the 4th input to VCC, and see that it still works as before. Keep in mind that this is only true for this trainer, and if you do this elsewhere or on a breadboard, you will still have to connect the unused inputs to VCC.

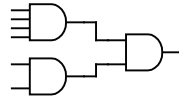
6. If you need to use a 3-input OR gate, you would normally get a 3-input OR gate. However, you can also use a 4-input OR gate instead by connecting the unused 4th input to a logic 0. Remember that connecting to GND will always give you a logic 0.

For this experiment, connect three switches to three inputs of a 4-input OR gate, connect the 4th input of the OR gate to GND, and connect the output from the OR gate to a LED. Slide the three switches up and down and record the LED output for all combinations of the input switches in the blank truth table provided. Verify that your results match the truth table in Figure 11 (c). What happens if you connect the extra input of the 4-input OR gate to VCC instead of GND?

For ease of use, all of the 4-input OR gates in the trainer have pull-down resistors connected to all of its inputs. What this means is that you do not have to actually connect any unused inputs of the 4-input OR gates to GND and it will still work. Verify this by disconnecting the wire that connects from the 4th input to GND, and see that it still works as before. Keep in mind that this is only true for this trainer, and if you do this elsewhere or on a breadboard, you will still have to connect the unused inputs to GND.

7. If you need to use a 6-input AND gate, you would normally get a 6-input AND gate. However, you can make a 6-input AND gate by combining a 4-input AND gate with a 2-input AND gate. Connect the two outputs from these two AND gates to another 2-input AND gate. The circuit for

connecting these three AND gates is shown next:



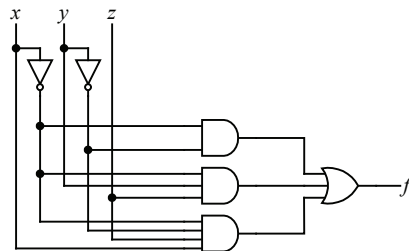
Connect six switches to this “6-input” AND gate, and connect the output from the second 2-input AND gate to a LED. Confirm its operation by recording the LED output for all combinations of the input switches.

8. Similarly, you can make a 6-input OR gate by combining a 4-input OR gate with a 2-input OR gate. Connect the two outputs from these two OR gates to another 2-input OR gate. Connect six switches to this “6-input” OR gate, and connect the output from the second 2-input OR gate to a LED. Confirm its operation.

9. Implement the circuit in Figure 12 (a). Connect the three inputs x , y and z , to three switches, and connect the output f to a LED. Record the LED output for all combinations of input switches in the blank truth table provided. Verify that it operates according to the truth table in Figure 13 (d).

10. Implement the circuit in Figure 12 (b). Connect the three inputs x , y and z , to three switches, and connect the two outputs, f and g , to two LEDs. Since the trainer does not have a NAND gate, you will need to use an AND gate and a NOT gate to get the same functionality as the NAND gate. This is accomplished by connecting the output of an AND gate to the input of a NOT gate. Determine its operation by deriving its truth table. Record LED outputs in the blank truth table provided.

11. Implement the following circuit and determine its operation by filling out the truth table provided. Recall from experiments 5 and 6 on how to get a 3-input gate from a 4-input gate. Connect the three inputs, x , y and z , to three switches, and connect the output f to an LED. Determine its operation by deriving its truth table. Also derive the Boolean equation for the circuit.



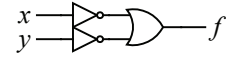
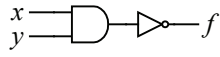
12. Implement the circuit in Figure 12 (c) and determine its operation by deriving the truth table and recording it. Notice that for the input combination $y = 1$ and $z = 0$, sometimes the output is a 0 and sometimes the output is a 1. This is because of the loop back that is causing a memory effect. You will learn more about this in sequential circuits.

13. Implement the circuit in Figure 12 (d). Connect a wire from the output of the AND gate to the input of the OR gate, and connect another wire from the output of the AND gate to the output of

Combinational Logic Design

the NOT gate. Determine its operation by deriving the truth table for it. Is there anything wrong with the results?

14. DeMorgan's theorem 13a states that $(x \cdot y)' = (x' + y')$. The circuits for the left and right side of the equality are respectively shown next.



Separately implement these two circuits and derive their truth tables. Because of the equality, the two truth tables should be the same.

15. DeMorgan's theorem 13b states that $(x + y)' = (x' \cdot y')$. Implement these two circuits and derive their truth tables similar to Experiment 14 above. Verify that these two truth tables are the same.

Lab 1: Truth Tables for Reporting Results

<table border="1" style="margin: auto;"> <thead> <tr><th>x</th><th>y</th><th>f</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td></td></tr> <tr><td>0</td><td>1</td><td></td></tr> <tr><td>1</td><td>0</td><td></td></tr> <tr><td>1</td><td>1</td><td></td></tr> </tbody> </table> <p>2-input AND gate</p>	x	y	f	0	0		0	1		1	0		1	1		<table border="1" style="margin: auto;"> <thead> <tr><th>x</th><th>y</th><th>f</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td></td></tr> <tr><td>0</td><td>1</td><td></td></tr> <tr><td>1</td><td>0</td><td></td></tr> <tr><td>1</td><td>1</td><td></td></tr> </tbody> </table> <p>2-input OR gate</p>	x	y	f	0	0		0	1		1	0		1	1		<table border="1" style="margin: auto;"> <thead> <tr><th>x</th><th>f</th></tr> </thead> <tbody> <tr><td>0</td><td></td></tr> <tr><td>1</td><td></td></tr> </tbody> </table> <p>NOT gate</p>	x	f	0		1																																																																																																																																																																																																																
x	y	f																																																																																																																																																																																																																																																			
0	0																																																																																																																																																																																																																																																				
0	1																																																																																																																																																																																																																																																				
1	0																																																																																																																																																																																																																																																				
1	1																																																																																																																																																																																																																																																				
x	y	f																																																																																																																																																																																																																																																			
0	0																																																																																																																																																																																																																																																				
0	1																																																																																																																																																																																																																																																				
1	0																																																																																																																																																																																																																																																				
1	1																																																																																																																																																																																																																																																				
x	f																																																																																																																																																																																																																																																				
0																																																																																																																																																																																																																																																					
1																																																																																																																																																																																																																																																					
Experiment 1	Experiment 2	Experiment 3																																																																																																																																																																																																																																																			
<table border="1" style="margin: auto;"> <thead> <tr><th>w</th><th>x</th><th>y</th><th>z</th><th>f</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td></td></tr> <tr><td>0</td><td>0</td><td>0</td><td>1</td><td></td></tr> <tr><td>0</td><td>0</td><td>1</td><td>0</td><td></td></tr> <tr><td>0</td><td>0</td><td>1</td><td>1</td><td></td></tr> <tr><td>0</td><td>1</td><td>0</td><td>0</td><td></td></tr> <tr><td>0</td><td>1</td><td>0</td><td>1</td><td></td></tr> <tr><td>0</td><td>1</td><td>1</td><td>0</td><td></td></tr> <tr><td>0</td><td>1</td><td>1</td><td>1</td><td></td></tr> <tr><td>1</td><td>0</td><td>0</td><td>0</td><td></td></tr> <tr><td>1</td><td>0</td><td>0</td><td>1</td><td></td></tr> <tr><td>1</td><td>0</td><td>1</td><td>0</td><td></td></tr> <tr><td>1</td><td>0</td><td>1</td><td>1</td><td></td></tr> <tr><td>1</td><td>1</td><td>0</td><td>0</td><td></td></tr> <tr><td>1</td><td>1</td><td>0</td><td>1</td><td></td></tr> <tr><td>1</td><td>1</td><td>1</td><td>0</td><td></td></tr> <tr><td>1</td><td>1</td><td>1</td><td>1</td><td></td></tr> </tbody> </table> <p>4-input AND gate</p>	w	x	y	z	f	0	0	0	0		0	0	0	1		0	0	1	0		0	0	1	1		0	1	0	0		0	1	0	1		0	1	1	0		0	1	1	1		1	0	0	0		1	0	0	1		1	0	1	0		1	0	1	1		1	1	0	0		1	1	0	1		1	1	1	0		1	1	1	1		<table border="1" style="margin: auto;"> <thead> <tr><th>w</th><th>x</th><th>y</th><th>z</th><th>f</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td></td></tr> <tr><td>0</td><td>0</td><td>0</td><td>1</td><td></td></tr> <tr><td>0</td><td>0</td><td>1</td><td>0</td><td></td></tr> <tr><td>0</td><td>0</td><td>1</td><td>1</td><td></td></tr> <tr><td>0</td><td>1</td><td>0</td><td>0</td><td></td></tr> <tr><td>0</td><td>1</td><td>0</td><td>1</td><td></td></tr> <tr><td>0</td><td>1</td><td>1</td><td>0</td><td></td></tr> <tr><td>0</td><td>1</td><td>1</td><td>1</td><td></td></tr> <tr><td>1</td><td>0</td><td>0</td><td>0</td><td></td></tr> <tr><td>1</td><td>0</td><td>0</td><td>1</td><td></td></tr> <tr><td>1</td><td>0</td><td>1</td><td>0</td><td></td></tr> <tr><td>1</td><td>0</td><td>1</td><td>1</td><td></td></tr> <tr><td>1</td><td>1</td><td>0</td><td>0</td><td></td></tr> <tr><td>1</td><td>1</td><td>0</td><td>1</td><td></td></tr> <tr><td>1</td><td>1</td><td>1</td><td>0</td><td></td></tr> <tr><td>1</td><td>1</td><td>1</td><td>1</td><td></td></tr> </tbody> </table> <p>4-input OR gate</p>	w	x	y	z	f	0	0	0	0		0	0	0	1		0	0	1	0		0	0	1	1		0	1	0	0		0	1	0	1		0	1	1	0		0	1	1	1		1	0	0	0		1	0	0	1		1	0	1	0		1	0	1	1		1	1	0	0		1	1	0	1		1	1	1	0		1	1	1	1		<table border="1" style="margin: auto;"> <thead> <tr><th>x</th><th>y</th><th>z</th><th>f</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td><td></td></tr> <tr><td>0</td><td>0</td><td>1</td><td></td></tr> <tr><td>0</td><td>1</td><td>0</td><td></td></tr> <tr><td>0</td><td>1</td><td>1</td><td></td></tr> <tr><td>1</td><td>0</td><td>0</td><td></td></tr> <tr><td>1</td><td>0</td><td>1</td><td></td></tr> <tr><td>1</td><td>1</td><td>0</td><td></td></tr> <tr><td>1</td><td>1</td><td>1</td><td></td></tr> </tbody> </table> <p>3-input AND gate</p>	x	y	z	f	0	0	0		0	0	1		0	1	0		0	1	1		1	0	0		1	0	1		1	1	0		1	1	1		<table border="1" style="margin: auto;"> <thead> <tr><th>x</th><th>y</th><th>z</th><th>f</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td><td></td></tr> <tr><td>0</td><td>0</td><td>1</td><td></td></tr> <tr><td>0</td><td>1</td><td>0</td><td></td></tr> <tr><td>0</td><td>1</td><td>1</td><td></td></tr> <tr><td>1</td><td>0</td><td>0</td><td></td></tr> <tr><td>1</td><td>0</td><td>1</td><td></td></tr> <tr><td>1</td><td>1</td><td>0</td><td></td></tr> <tr><td>1</td><td>1</td><td>1</td><td></td></tr> </tbody> </table> <p>3-input OR gate</p>	x	y	z	f	0	0	0		0	0	1		0	1	0		0	1	1		1	0	0		1	0	1		1	1	0		1	1	1	
w	x	y	z	f																																																																																																																																																																																																																																																	
0	0	0	0																																																																																																																																																																																																																																																		
0	0	0	1																																																																																																																																																																																																																																																		
0	0	1	0																																																																																																																																																																																																																																																		
0	0	1	1																																																																																																																																																																																																																																																		
0	1	0	0																																																																																																																																																																																																																																																		
0	1	0	1																																																																																																																																																																																																																																																		
0	1	1	0																																																																																																																																																																																																																																																		
0	1	1	1																																																																																																																																																																																																																																																		
1	0	0	0																																																																																																																																																																																																																																																		
1	0	0	1																																																																																																																																																																																																																																																		
1	0	1	0																																																																																																																																																																																																																																																		
1	0	1	1																																																																																																																																																																																																																																																		
1	1	0	0																																																																																																																																																																																																																																																		
1	1	0	1																																																																																																																																																																																																																																																		
1	1	1	0																																																																																																																																																																																																																																																		
1	1	1	1																																																																																																																																																																																																																																																		
w	x	y	z	f																																																																																																																																																																																																																																																	
0	0	0	0																																																																																																																																																																																																																																																		
0	0	0	1																																																																																																																																																																																																																																																		
0	0	1	0																																																																																																																																																																																																																																																		
0	0	1	1																																																																																																																																																																																																																																																		
0	1	0	0																																																																																																																																																																																																																																																		
0	1	0	1																																																																																																																																																																																																																																																		
0	1	1	0																																																																																																																																																																																																																																																		
0	1	1	1																																																																																																																																																																																																																																																		
1	0	0	0																																																																																																																																																																																																																																																		
1	0	0	1																																																																																																																																																																																																																																																		
1	0	1	0																																																																																																																																																																																																																																																		
1	0	1	1																																																																																																																																																																																																																																																		
1	1	0	0																																																																																																																																																																																																																																																		
1	1	0	1																																																																																																																																																																																																																																																		
1	1	1	0																																																																																																																																																																																																																																																		
1	1	1	1																																																																																																																																																																																																																																																		
x	y	z	f																																																																																																																																																																																																																																																		
0	0	0																																																																																																																																																																																																																																																			
0	0	1																																																																																																																																																																																																																																																			
0	1	0																																																																																																																																																																																																																																																			
0	1	1																																																																																																																																																																																																																																																			
1	0	0																																																																																																																																																																																																																																																			
1	0	1																																																																																																																																																																																																																																																			
1	1	0																																																																																																																																																																																																																																																			
1	1	1																																																																																																																																																																																																																																																			
x	y	z	f																																																																																																																																																																																																																																																		
0	0	0																																																																																																																																																																																																																																																			
0	0	1																																																																																																																																																																																																																																																			
0	1	0																																																																																																																																																																																																																																																			
0	1	1																																																																																																																																																																																																																																																			
1	0	0																																																																																																																																																																																																																																																			
1	0	1																																																																																																																																																																																																																																																			
1	1	0																																																																																																																																																																																																																																																			
1	1	1																																																																																																																																																																																																																																																			
Experiment 4	Experiment 5	Experiment 6																																																																																																																																																																																																																																																			
<table border="1" style="margin: auto;"> <thead> <tr><th>x</th><th>y</th><th>z</th><th>f</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td><td></td></tr> <tr><td>0</td><td>0</td><td>1</td><td></td></tr> <tr><td>0</td><td>1</td><td>0</td><td></td></tr> <tr><td>0</td><td>1</td><td>1</td><td></td></tr> <tr><td>1</td><td>0</td><td>0</td><td></td></tr> <tr><td>1</td><td>0</td><td>1</td><td></td></tr> <tr><td>1</td><td>1</td><td>0</td><td></td></tr> <tr><td>1</td><td>1</td><td>1</td><td></td></tr> </tbody> </table>	x	y	z	f	0	0	0		0	0	1		0	1	0		0	1	1		1	0	0		1	0	1		1	1	0		1	1	1		<table border="1" style="margin: auto;"> <thead> <tr><th>x</th><th>y</th><th>z</th><th>f</th><th>g</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td><td></td><td></td></tr> <tr><td>0</td><td>0</td><td>1</td><td></td><td></td></tr> <tr><td>0</td><td>1</td><td>0</td><td></td><td></td></tr> <tr><td>0</td><td>1</td><td>1</td><td></td><td></td></tr> <tr><td>1</td><td>0</td><td>0</td><td></td><td></td></tr> <tr><td>1</td><td>0</td><td>1</td><td></td><td></td></tr> <tr><td>1</td><td>1</td><td>0</td><td></td><td></td></tr> <tr><td>1</td><td>1</td><td>1</td><td></td><td></td></tr> </tbody> </table>	x	y	z	f	g	0	0	0			0	0	1			0	1	0			0	1	1			1	0	0			1	0	1			1	1	0			1	1	1			<table border="1" style="margin: auto;"> <thead> <tr><th>x</th><th>y</th><th>z</th><th>f</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td><td></td></tr> <tr><td>0</td><td>0</td><td>1</td><td></td></tr> <tr><td>0</td><td>1</td><td>0</td><td></td></tr> <tr><td>0</td><td>1</td><td>1</td><td></td></tr> <tr><td>1</td><td>0</td><td>0</td><td></td></tr> <tr><td>1</td><td>0</td><td>1</td><td></td></tr> <tr><td>1</td><td>1</td><td>0</td><td></td></tr> <tr><td>1</td><td>1</td><td>1</td><td></td></tr> </tbody> </table>	x	y	z	f	0	0	0		0	0	1		0	1	0		0	1	1		1	0	0		1	0	1		1	1	0		1	1	1																																																																																																																															
x	y	z	f																																																																																																																																																																																																																																																		
0	0	0																																																																																																																																																																																																																																																			
0	0	1																																																																																																																																																																																																																																																			
0	1	0																																																																																																																																																																																																																																																			
0	1	1																																																																																																																																																																																																																																																			
1	0	0																																																																																																																																																																																																																																																			
1	0	1																																																																																																																																																																																																																																																			
1	1	0																																																																																																																																																																																																																																																			
1	1	1																																																																																																																																																																																																																																																			
x	y	z	f	g																																																																																																																																																																																																																																																	
0	0	0																																																																																																																																																																																																																																																			
0	0	1																																																																																																																																																																																																																																																			
0	1	0																																																																																																																																																																																																																																																			
0	1	1																																																																																																																																																																																																																																																			
1	0	0																																																																																																																																																																																																																																																			
1	0	1																																																																																																																																																																																																																																																			
1	1	0																																																																																																																																																																																																																																																			
1	1	1																																																																																																																																																																																																																																																			
x	y	z	f																																																																																																																																																																																																																																																		
0	0	0																																																																																																																																																																																																																																																			
0	0	1																																																																																																																																																																																																																																																			
0	1	0																																																																																																																																																																																																																																																			
0	1	1																																																																																																																																																																																																																																																			
1	0	0																																																																																																																																																																																																																																																			
1	0	1																																																																																																																																																																																																																																																			
1	1	0																																																																																																																																																																																																																																																			
1	1	1																																																																																																																																																																																																																																																			
Experiment 9	Experiment 10	Experiment 11																																																																																																																																																																																																																																																			

Combinational Logic Design

		<table border="1"> <thead> <tr> <th><i>y</i></th> <th><i>z</i></th> <th><i>f</i></th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td></td> </tr> <tr> <td>0</td> <td>1</td> <td></td> </tr> <tr> <td>1</td> <td>0</td> <td></td> </tr> <tr> <td>1</td> <td>1</td> <td></td> </tr> </tbody> </table>		<i>y</i>	<i>z</i>	<i>f</i>	0	0		0	1		1	0		1	1																
<i>y</i>	<i>z</i>	<i>f</i>																															
0	0																																
0	1																																
1	0																																
1	1																																
Experiment 11, Boolean Equation		Experiment 12																															
<table border="1"> <thead> <tr> <th><i>x</i></th> <th><i>y</i></th> <th><i>f</i></th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td></td> </tr> <tr> <td>0</td> <td>1</td> <td></td> </tr> <tr> <td>1</td> <td>0</td> <td></td> </tr> <tr> <td>1</td> <td>1</td> <td></td> </tr> </tbody> </table>		<i>x</i>	<i>y</i>	<i>f</i>	0	0		0	1		1	0		1	1		<table border="1"> <thead> <tr> <th><i>x</i></th> <th><i>y</i></th> <th><i>f</i></th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td></td> </tr> <tr> <td>0</td> <td>1</td> <td></td> </tr> <tr> <td>1</td> <td>0</td> <td></td> </tr> <tr> <td>1</td> <td>1</td> <td></td> </tr> </tbody> </table>		<i>x</i>	<i>y</i>	<i>f</i>	0	0		0	1		1	0		1	1	
<i>x</i>	<i>y</i>	<i>f</i>																															
0	0																																
0	1																																
1	0																																
1	1																																
<i>x</i>	<i>y</i>	<i>f</i>																															
0	0																																
0	1																																
1	0																																
1	1																																
Experiment 14		Experiment 15																															

4.2. Lab 2: NAND, NOR, XOR and XNOR Gates

Purpose

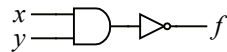
In this lab you will experiment with the NAND, NOR, XOR and XNOR gates. You will verify their operations by deriving the truth tables for them.

Introduction

In addition to the AND, OR and NOT gates, the NAND, NOR, XOR and XNOR gates are also considered as basic building blocks for digital circuits. In practice, the NAND gate is actually used more than the AND gate because it requires only four transistors, whereas the AND gate requires six transistors to implement. Furthermore, it turns out that all digital logic circuits can be built using only the NAND gate!

Experiments

1. As you recall, the operation of the NAND gate is the inverse of the AND gate. If you need to use a 2-input NAND gate, you would normally get a 2-input NAND gate, but since our trainer does not have any NAND gates, therefore we need to implement it by connecting a 2-input AND gate to a NOT gate as shown next:



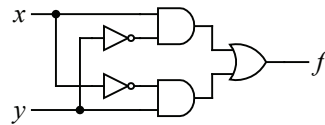
Connect the two inputs of the AND gate to two switches, the output of the AND gate to the input of the NOT gate, and finally the output of the NOT gate to an LED. Determine its operation by recording the LED output for all combinations of the two input switches in the blank truth table provided. Verify that it matches the truth table in Figure 9 (a) Page 11 .

2. The NOR gate is the inverse of the OR gate. Again, we can implement a 2-input NOR gate by using a 2-input OR gate followed by a NOT gate. Connect the two inputs of the OR gate to two switches, the output of the OR gate to the input of the NOT gate, and finally the output of the NOT gate to an LED. Determine its operation by recording the LED output for all combinations of the two input switches in the blank truth table provided. Verify that it matches the truth table in Figure 9 (b).

3. Verify the operation of the 2-input XOR gate by connecting the two inputs of a 2-input XOR gate to two switches and the output to an LED. Determine its operation by recording the LED output for all combinations of the two input switches in the blank truth table provided. Verify that it matches the truth table in Figure 9 (c).

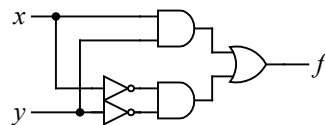
4. Implement the following circuit and determine its operation by deriving its truth table. Verify that this truth table is the same as that for the XOR gate. Also derive the Boolean equation for the circuit.

Combinational Logic Design

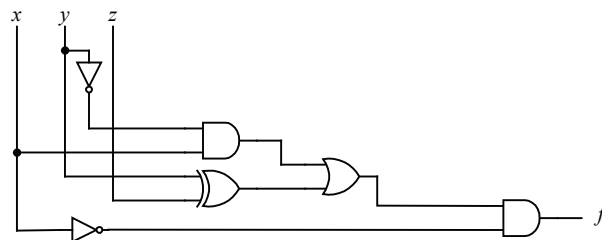


5. The 2-input XNOR gate is the inverse of the 2-input XOR gate. Implement a 2-input XNOR gate by connecting the output of a 2-input XOR gate to a NOT gate. Connect the two inputs of the XOR gate to two switches, the output of the XOR gate to the input of the NOT gate, and finally the output of the NOT gate to an LED. Determine its operation by recording the LED output for all combinations of the two input switches in the blank truth table provided. Verify that it matches the truth table in Figure 9 (d).

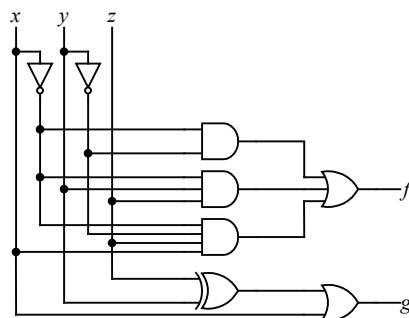
6. Implement the following circuit and determine its operation by deriving its truth table. Verify that this truth table is the same as that for the XNOR gate. Also derive the Boolean equation for the circuit.



7. Implement the following circuit and determine its operation by deriving its truth table. Also derive the Boolean equation for the circuit.

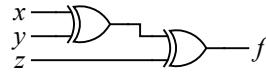


8. Implement the following circuit and determine its operation by deriving its truth table. Also derive the Boolean equation for the circuit.

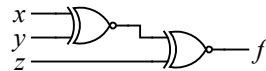


9. Implement the following 3-input XOR gate circuit and determine its operation by deriving its

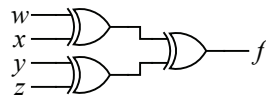
truth tabletruth table for it.



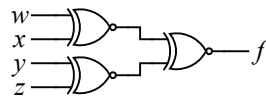
10. Implement the following 3-input XNOR gate circuit and determine its operation by deriving its truth table.



11. Implement the following 4-input XOR gate circuit and determine its operation by deriving its truth table.



12. Implement the following 4-input XNOR gate circuit and determine its operation by deriving its truth table.



Combinational Logic Design

Lab 2: Truth Tables for Reporting Results

<table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <thead> <tr><th>x</th><th>y</th><th>f</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td></td></tr> <tr><td>0</td><td>1</td><td></td></tr> <tr><td>1</td><td>0</td><td></td></tr> <tr><td>1</td><td>1</td><td></td></tr> </tbody> </table> <p>Experiment 1</p>	x	y	f	0	0		0	1		1	0		1	1		<table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <thead> <tr><th>x</th><th>y</th><th>f</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td></td></tr> <tr><td>0</td><td>1</td><td></td></tr> <tr><td>1</td><td>0</td><td></td></tr> <tr><td>1</td><td>1</td><td></td></tr> </tbody> </table> <p>Experiment 2</p>	x	y	f	0	0		0	1		1	0		1	1		<table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <thead> <tr><th>x</th><th>y</th><th>f</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td></td></tr> <tr><td>0</td><td>1</td><td></td></tr> <tr><td>1</td><td>0</td><td></td></tr> <tr><td>1</td><td>1</td><td></td></tr> </tbody> </table> <p>Experiment 3</p>	x	y	f	0	0		0	1		1	0		1	1		<table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <thead> <tr><th>x</th><th>y</th><th>f</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td></td></tr> <tr><td>0</td><td>1</td><td></td></tr> <tr><td>1</td><td>0</td><td></td></tr> <tr><td>1</td><td>1</td><td></td></tr> </tbody> </table> <p>Experiment 4</p>	x	y	f	0	0		0	1		1	0		1	1	
x	y	f																																																													
0	0																																																														
0	1																																																														
1	0																																																														
1	1																																																														
x	y	f																																																													
0	0																																																														
0	1																																																														
1	0																																																														
1	1																																																														
x	y	f																																																													
0	0																																																														
0	1																																																														
1	0																																																														
1	1																																																														
x	y	f																																																													
0	0																																																														
0	1																																																														
1	0																																																														
1	1																																																														

<table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <thead> <tr><th>x</th><th>y</th><th>f</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td></td></tr> <tr><td>0</td><td>1</td><td></td></tr> <tr><td>1</td><td>0</td><td></td></tr> <tr><td>1</td><td>1</td><td></td></tr> </tbody> </table> <p>Experiment 5</p>	x	y	f	0	0		0	1		1	0		1	1		<table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <thead> <tr><th>x</th><th>y</th><th>f</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td></td></tr> <tr><td>0</td><td>1</td><td></td></tr> <tr><td>1</td><td>0</td><td></td></tr> <tr><td>1</td><td>1</td><td></td></tr> </tbody> </table> <p>Experiment 6</p>	x	y	f	0	0		0	1		1	0		1	1		<table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <thead> <tr><th>x</th><th>y</th><th>z</th><th>f</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td><td></td></tr> <tr><td>0</td><td>0</td><td>1</td><td></td></tr> <tr><td>0</td><td>1</td><td>0</td><td></td></tr> <tr><td>0</td><td>1</td><td>1</td><td></td></tr> <tr><td>1</td><td>0</td><td>0</td><td></td></tr> <tr><td>1</td><td>0</td><td>1</td><td></td></tr> <tr><td>1</td><td>1</td><td>0</td><td></td></tr> <tr><td>1</td><td>1</td><td>1</td><td></td></tr> </tbody> </table> <p>Experiment 7</p>	x	y	z	f	0	0	0		0	0	1		0	1	0		0	1	1		1	0	0		1	0	1		1	1	0		1	1	1		<table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <thead> <tr><th>x</th><th>y</th><th>z</th><th>f</th><th>g</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td><td></td><td></td></tr> <tr><td>0</td><td>0</td><td>1</td><td></td><td></td></tr> <tr><td>0</td><td>1</td><td>0</td><td></td><td></td></tr> <tr><td>0</td><td>1</td><td>1</td><td></td><td></td></tr> <tr><td>1</td><td>0</td><td>0</td><td></td><td></td></tr> <tr><td>1</td><td>0</td><td>1</td><td></td><td></td></tr> <tr><td>1</td><td>1</td><td>0</td><td></td><td></td></tr> <tr><td>1</td><td>1</td><td>1</td><td></td><td></td></tr> </tbody> </table> <p>Experiment 8</p>	x	y	z	f	g	0	0	0			0	0	1			0	1	0			0	1	1			1	0	0			1	0	1			1	1	0			1	1	1		
x	y	f																																																																																																																
0	0																																																																																																																	
0	1																																																																																																																	
1	0																																																																																																																	
1	1																																																																																																																	
x	y	f																																																																																																																
0	0																																																																																																																	
0	1																																																																																																																	
1	0																																																																																																																	
1	1																																																																																																																	
x	y	z	f																																																																																																															
0	0	0																																																																																																																
0	0	1																																																																																																																
0	1	0																																																																																																																
0	1	1																																																																																																																
1	0	0																																																																																																																
1	0	1																																																																																																																
1	1	0																																																																																																																
1	1	1																																																																																																																
x	y	z	f	g																																																																																																														
0	0	0																																																																																																																
0	0	1																																																																																																																
0	1	0																																																																																																																
0	1	1																																																																																																																
1	0	0																																																																																																																
1	0	1																																																																																																																
1	1	0																																																																																																																
1	1	1																																																																																																																

<table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <thead> <tr><th>x</th><th>y</th><th>z</th><th>f</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td><td></td></tr> <tr><td>0</td><td>0</td><td>1</td><td></td></tr> <tr><td>0</td><td>1</td><td>0</td><td></td></tr> <tr><td>0</td><td>1</td><td>1</td><td></td></tr> <tr><td>1</td><td>0</td><td>0</td><td></td></tr> <tr><td>1</td><td>0</td><td>1</td><td></td></tr> <tr><td>1</td><td>1</td><td>0</td><td></td></tr> <tr><td>1</td><td>1</td><td>1</td><td></td></tr> </tbody> </table> <p>3-input XOR Experiment 9</p>	x	y	z	f	0	0	0		0	0	1		0	1	0		0	1	1		1	0	0		1	0	1		1	1	0		1	1	1		<table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <thead> <tr><th>x</th><th>y</th><th>z</th><th>f</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td><td></td></tr> <tr><td>0</td><td>0</td><td>1</td><td></td></tr> <tr><td>0</td><td>1</td><td>0</td><td></td></tr> <tr><td>0</td><td>1</td><td>1</td><td></td></tr> <tr><td>1</td><td>0</td><td>0</td><td></td></tr> <tr><td>1</td><td>0</td><td>1</td><td></td></tr> <tr><td>1</td><td>1</td><td>0</td><td></td></tr> <tr><td>1</td><td>1</td><td>1</td><td></td></tr> </tbody> </table> <p>3-input XNOR Experiment 10</p>	x	y	z	f	0	0	0		0	0	1		0	1	0		0	1	1		1	0	0		1	0	1		1	1	0		1	1	1		<table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <thead> <tr><th>w</th><th>x</th><th>y</th><th>z</th><th>f</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td></td></tr> <tr><td>0</td><td>0</td><td>0</td><td>1</td><td></td></tr> <tr><td>0</td><td>0</td><td>1</td><td>0</td><td></td></tr> <tr><td>0</td><td>0</td><td>1</td><td>1</td><td></td></tr> <tr><td>0</td><td>1</td><td>0</td><td>0</td><td></td></tr> <tr><td>0</td><td>1</td><td>0</td><td>1</td><td></td></tr> <tr><td>0</td><td>1</td><td>1</td><td>0</td><td></td></tr> <tr><td>0</td><td>1</td><td>1</td><td>1</td><td></td></tr> <tr><td>1</td><td>0</td><td>0</td><td>0</td><td></td></tr> <tr><td>1</td><td>0</td><td>0</td><td>1</td><td></td></tr> <tr><td>1</td><td>0</td><td>1</td><td>0</td><td></td></tr> <tr><td>1</td><td>0</td><td>1</td><td>1</td><td></td></tr> <tr><td>1</td><td>1</td><td>0</td><td>0</td><td></td></tr> <tr><td>1</td><td>1</td><td>0</td><td>1</td><td></td></tr> <tr><td>1</td><td>1</td><td>1</td><td>0</td><td></td></tr> <tr><td>1</td><td>1</td><td>1</td><td>1</td><td></td></tr> </tbody> </table> <p>4-input XOR Experiment 11</p>	w	x	y	z	f	0	0	0	0		0	0	0	1		0	0	1	0		0	0	1	1		0	1	0	0		0	1	0	1		0	1	1	0		0	1	1	1		1	0	0	0		1	0	0	1		1	0	1	0		1	0	1	1		1	1	0	0		1	1	0	1		1	1	1	0		1	1	1	1		<table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <thead> <tr><th>w</th><th>x</th><th>y</th><th>z</th><th>f</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td></td></tr> <tr><td>0</td><td>0</td><td>0</td><td>1</td><td></td></tr> <tr><td>0</td><td>0</td><td>1</td><td>0</td><td></td></tr> <tr><td>0</td><td>0</td><td>1</td><td>1</td><td></td></tr> <tr><td>0</td><td>1</td><td>0</td><td>0</td><td></td></tr> <tr><td>0</td><td>1</td><td>0</td><td>1</td><td></td></tr> <tr><td>0</td><td>1</td><td>1</td><td>0</td><td></td></tr> <tr><td>0</td><td>1</td><td>1</td><td>1</td><td></td></tr> <tr><td>1</td><td>0</td><td>0</td><td>0</td><td></td></tr> <tr><td>1</td><td>0</td><td>0</td><td>1</td><td></td></tr> <tr><td>1</td><td>0</td><td>1</td><td>0</td><td></td></tr> <tr><td>1</td><td>0</td><td>1</td><td>1</td><td></td></tr> <tr><td>1</td><td>1</td><td>0</td><td>0</td><td></td></tr> <tr><td>1</td><td>1</td><td>0</td><td>1</td><td></td></tr> <tr><td>1</td><td>1</td><td>1</td><td>0</td><td></td></tr> <tr><td>1</td><td>1</td><td>1</td><td>1</td><td></td></tr> </tbody> </table> <p>4-input XNOR Experiment 12</p>	w	x	y	z	f	0	0	0	0		0	0	0	1		0	0	1	0		0	0	1	1		0	1	0	0		0	1	0	1		0	1	1	0		0	1	1	1		1	0	0	0		1	0	0	1		1	0	1	0		1	0	1	1		1	1	0	0		1	1	0	1		1	1	1	0		1	1	1	1	
x	y	z	f																																																																																																																																																																																																																																																		
0	0	0																																																																																																																																																																																																																																																			
0	0	1																																																																																																																																																																																																																																																			
0	1	0																																																																																																																																																																																																																																																			
0	1	1																																																																																																																																																																																																																																																			
1	0	0																																																																																																																																																																																																																																																			
1	0	1																																																																																																																																																																																																																																																			
1	1	0																																																																																																																																																																																																																																																			
1	1	1																																																																																																																																																																																																																																																			
x	y	z	f																																																																																																																																																																																																																																																		
0	0	0																																																																																																																																																																																																																																																			
0	0	1																																																																																																																																																																																																																																																			
0	1	0																																																																																																																																																																																																																																																			
0	1	1																																																																																																																																																																																																																																																			
1	0	0																																																																																																																																																																																																																																																			
1	0	1																																																																																																																																																																																																																																																			
1	1	0																																																																																																																																																																																																																																																			
1	1	1																																																																																																																																																																																																																																																			
w	x	y	z	f																																																																																																																																																																																																																																																	
0	0	0	0																																																																																																																																																																																																																																																		
0	0	0	1																																																																																																																																																																																																																																																		
0	0	1	0																																																																																																																																																																																																																																																		
0	0	1	1																																																																																																																																																																																																																																																		
0	1	0	0																																																																																																																																																																																																																																																		
0	1	0	1																																																																																																																																																																																																																																																		
0	1	1	0																																																																																																																																																																																																																																																		
0	1	1	1																																																																																																																																																																																																																																																		
1	0	0	0																																																																																																																																																																																																																																																		
1	0	0	1																																																																																																																																																																																																																																																		
1	0	1	0																																																																																																																																																																																																																																																		
1	0	1	1																																																																																																																																																																																																																																																		
1	1	0	0																																																																																																																																																																																																																																																		
1	1	0	1																																																																																																																																																																																																																																																		
1	1	1	0																																																																																																																																																																																																																																																		
1	1	1	1																																																																																																																																																																																																																																																		
w	x	y	z	f																																																																																																																																																																																																																																																	
0	0	0	0																																																																																																																																																																																																																																																		
0	0	0	1																																																																																																																																																																																																																																																		
0	0	1	0																																																																																																																																																																																																																																																		
0	0	1	1																																																																																																																																																																																																																																																		
0	1	0	0																																																																																																																																																																																																																																																		
0	1	0	1																																																																																																																																																																																																																																																		
0	1	1	0																																																																																																																																																																																																																																																		
0	1	1	1																																																																																																																																																																																																																																																		
1	0	0	0																																																																																																																																																																																																																																																		
1	0	0	1																																																																																																																																																																																																																																																		
1	0	1	0																																																																																																																																																																																																																																																		
1	0	1	1																																																																																																																																																																																																																																																		
1	1	0	0																																																																																																																																																																																																																																																		
1	1	0	1																																																																																																																																																																																																																																																		
1	1	1	0																																																																																																																																																																																																																																																		
1	1	1	1																																																																																																																																																																																																																																																		

4.3. Lab 3: Designing Combinational Circuits

Purpose

In this lab you will learn how to design combinational circuits from any given truth table and implement them on the trainer.

Introduction

As you saw in Lab 1, the operation of any combinational circuit can be described formally by a truth table. Moreover, any given truth table can be implemented with a digital circuit. In fact, any truth table can be implemented with one or more different but functionally equivalent digital circuit. A digital circuit that implements a truth table is always a combinational circuit. The design process begins with an informal description of the circuit that you want. This informal description is translated into a precise and formal description of the circuit in the form of a truth table. Given a truth table, you can easily construct a combinational circuit for it. Hence, the circuit will operate according to the specifications given in the truth table.

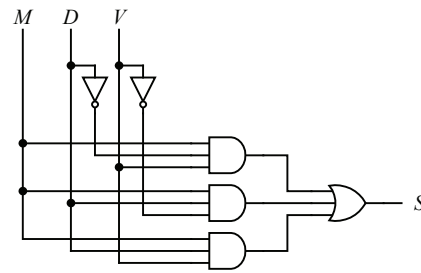
As an example, we begin with an informal description of a car security system that we would like to implement. This simple car security system consists of a master switch (M) for turning on and off the system, a door switch (D) for detecting whether the car door is open or close, a vibration sensor (V) for detecting movements of the car, and a siren (S) for sounding the alarm. Given these input and output parameters, I assume that you are at least slightly acquainted with the functional operations of a security system so that you can precisely describe its operation using a truth table.

First, you layout your truth table with the column labels being the input and output signals. Next, using binary numbers, you enumerate all possible input values; creating one row per value. You should have the table shown in Figure 27 (a) but without the values in the S column. Interpreting a 1 being "on" and a 0 being "off", determine for each row (i.e., each combination of input values) what the corresponding output ought to be (i.e., whether the siren S should be turned on or off). For example, in the first row where $M=0$, $D=0$, and $V=0$, this means that the system is off, so regardless of the state of the door switch or the vibration sensor, the siren should be off. Hence the output S is 0 for this row. In the row where $M=1$, $D=0$, and $V=0$, this means that the system is on, but since the door is closed ($D=0$) and there is no vibration ($V=0$), the siren should also be off. Hence the output S for this row is also 0. Continuing with this reasoning, you should be able to complete the table and obtain the complete truth table as shown in Figure 27 (a).

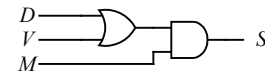
Combinational Logic Design

M	D	V	S
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

(a)



(b)



(c)

Figure 27: Car security system: (a) truth table; (b) circuit diagram derived from the truth table; (c) simplified circuit diagram.

Obtaining a circuit from a truth table is fairly straight forward. For each row where the output (S) is a 1, you use the AND operation to AND the input values together. Where the input value is a 1, you connect that input directly to an input of the AND gate. Where the input value is a 0, you connect the negated input (through the NOT gate) to an input of the AND gate. Finally, the outputs of all the AND gates are connected to the inputs of an OR gate to produce the primary output signal. Hence from the truth table in Figure 27 (a), you should obtain the circuit in Figure 27 (b). This circuit will operate exactly according to the truth table.

In practice, we usually want to make the circuit as small as possible but still operates exactly the same. Using logical reasoning and looking at the truth table in Figure 27 (a) more carefully, you might notice that the siren should be turned on ($S=1$) only when the master switch is on ($M=1$), and either the door is opened ($D=1$) or there is vibration ($V=1$). In other words, you want S to be 1 only when $M=1$ and either $D=1$ or $V=1$ (or both D and V are 1's). Writing this out as a Boolean equation, we get

$$S = M \text{ AND } (D \text{ OR } V)$$

In textbooks⁶, you might see the equation above written as

$$S = M \bullet (D + V)$$

where the \bullet symbol denotes the AND operation, and the $+$ symbol denotes the OR operation, or even without the \bullet symbol for the AND operation like this

$$S = M (D + V)$$

This equation gives rise to the simplified circuit shown in Figure 27 (c). The two circuits in Figure 27 are functionally equivalent, i.e., they satisfy the same truth table.

⁶ For an in-depth discussion on Boolean equations, refer to the book "Digital Logic and Microprocessor Design with VHDL" by E. Hwang.

Experiments

1. Implement the two circuits in Figure 27 (b) and (c) separately, and verify that both of them operate according to the truth table in Figure 27 (a). Connect the master switch M to SW0. Connect the door switch D and the vibration switch V to the two push buttons PB0 and PB1. Connect the siren output S to LED0. Verify that the circuit operates as described.

2. Design and implement the circuit for the following truth table where x , y , and z are the inputs, and f is the output. See Experiment 8 in Lab 1 on how to construct a 6-input OR gate.

x	y	z	f
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

3. Design and implement the circuit for the following truth table where x , y , and z are the inputs, and f is the output.

x	y	z	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

4. Notice that the outputs for Experiment 3 are just the inverse of those from Experiment 2, i.e., all the 0's and 1's are flipped. In other words, if you implement a circuit by selecting the rows where the output f is a 0 instead of a 1, you will end up with the inverse of f . Also notice that the circuit for Experiment 3 is much smaller than the circuit for Experiment 2. So a better way to implement the truth table for Experiment 2 is to start with the circuit for Experiment 3, and simply add a NOT gate to the output to invert the result. Implement this new circuit and verify that it operates exactly the same as the circuit from Experiment 2, i.e., both circuits produce the same truth table.

Combinational Logic Design

5. Design and implement the circuit for the following truth table where x , y , and b_{in} are the inputs, and b_{out} and d are the outputs.

x	y	b_{in}	b_{out}	d
0	0	0		0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

4.4. Lab 4: Multiplexers

Purpose

In this lab you will learn about multiplexers. A multiplexer is used to channel data from multiple sources to one destination. You will design multiplexer circuits and implement them on the trainer.

Introduction

Multiplexer, also known as a mux, is a frequently used component in a digital circuit. They are used to pass data from multiple sources to one destination. An analogy of its operation is like a railroad switch where two rail tracks are merged into one track. Depending on the switch setting, trains from either one of the two tracks are directed onto the one track. The logic symbol for a 2-to-1 mux is shown in Figure 28 (a). It has two inputs labeled d_0 and d_1 , and one output labeled y . Instead of passing trains, data is passed from either input d_0 or d_1 to the output y depending on the select line s . If the value of s is 0 then the data from input d_0 is passed to the output, and if the value of s is 1 then the data from input d_1 is passed to the output. The simplified circuit for the 2-to-1 mux is shown in Figure 28 (b). The truth table and the equation⁷ are shown in Figure 28 (c) and (d).

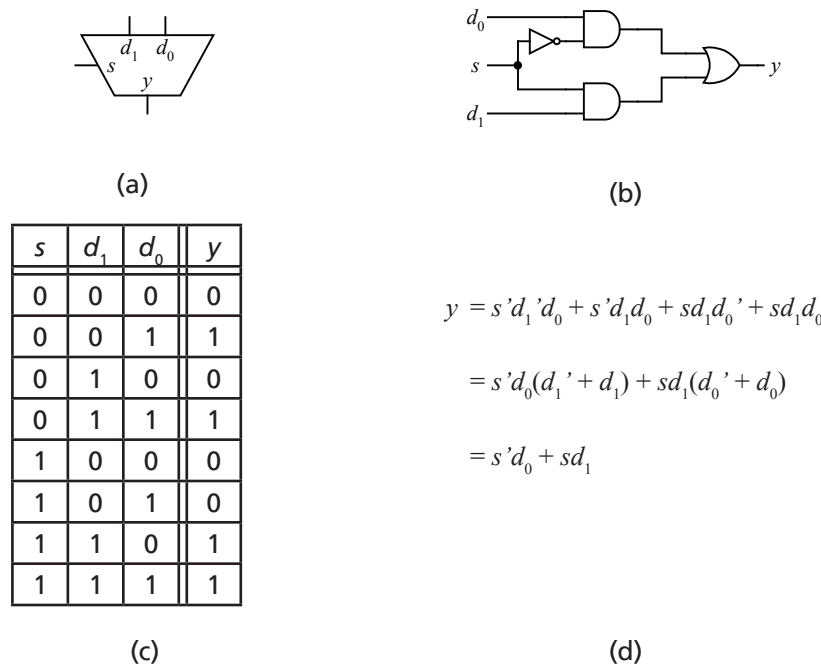


Figure 28: A 2-to-1 multiplexer: (a) logic symbol; (b) circuit; (c) truth table; (d) equation.

Experiments

1. Implement the simplified 2-to-1 mux circuit as shown in Figure 28 (b), and confirm that it operates according to the truth table shown in Figure 28 (c). Connect the three inputs, s , d_1 , and d_0 , to three switches, and connect the output y to an LED.

⁷ For an in-depth discussion on how to simplify Boolean equations and circuits, refer to the book "Digital Logic and Microprocessor Design with VHDL" by E. Hwang.

Combinational Logic Design

2. Design the 2-to-1 mux circuit based on the truth table shown in Figure 28 (c) and without any simplifications. Implement your circuit and confirm that it operates according to the truth table.

3. A larger size mux is a 4-to-1 mux where there are four data inputs, d_3 , d_2 , d_1 and d_0 . In order to select one of the four data inputs, two select lines, s_1 and s_0 , are needed. When $s_1s_0 = 00$, data from d_0 is passed to the output; when $s_1s_0 = 01$, data from d_1 is passed to the output; when $s_1s_0 = 10$, data from d_2 is passed to the output; and When $s_1s_0 = 11$, data from d_3 is passed to the output. Design a 4-to-1 mux circuit by first deriving the truth table and equation.

With six inputs, s_1 , s_0 , d_3 , d_2 , d_1 and d_0 , the truth table will have $2^6 = 64$ rows! I am sure you don't want to write out this entire table. An easier way to reduce the size of the truth table is to not have the d input columns, but instead write out the d 's as entries under the output column. Using this method, the 2-to-1 mux truth table shown in Figure 28 (c) reduces to:

s	y
0	d_0
1	d_1

From this simplified truth table, we can immediately obtain the simplified equation $y = s'd_0 + sd_1$. Applying this method for the 4-to-1 mux, we get the following simplified truth table:

S_1	S_0	y
0	0	d_0
1	1	d_1
1	0	d_2
1	1	d_3

and the equation will be $y = s_1's_0'd_0 + s_1's_0d_1 + s_1s_0'd_2 + s_1s_0d_3$.

Draw out the circuit from this equation, then implement your circuit and confirm that it works correctly.

4. Design and implement an 8-to-1 multiplexer. There should be 8 data input lines, 3 select lines and 1 output line. Since there are only 8 switches on the trainer, you can connect the three select lines directly to either VCC (for 1) or GND (for 0).

4.5. Lab 5: Decoders

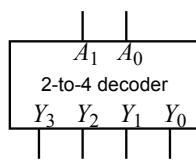
Purpose

In this lab you will learn about decoders. A decoder is used to select one among several devices or memory locations. You will design decoder and encoder circuits and implement them on the trainer.

Introduction

A decoder is another frequently used component in a digital circuit. The function of the decoder is to select one thing among several things. For example, if you have an array of 16 memory locations and you want to read from one particular location, a decoder will be used to select which one of the 16 memory locations that you want to access. The logic symbol for a 2-to-4 decoder is shown in Figure 29 (a).

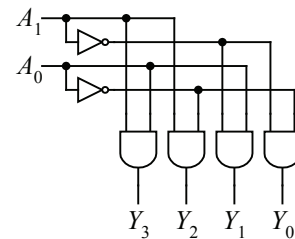
As implied by the name, 2-to-4 decoder, this component has two input lines, A_1, A_0 , which are the two address select lines for selecting one of the four output lines, Y_3, Y_2, Y_1, Y_0 . When the address lines $A_1, A_0 = 00$, the output line Y_0 will be selected; when the address lines $A_1, A_0 = 01$, the output line Y_1 will be selected; when the address lines $A_1, A_0 = 10$, the output line Y_2 will be selected; and when the address lines $A_1, A_0 = 11$, the output line Y_3 will be selected. The selected output line will have a 1 value, while all of the remaining output lines will have 0's. Given this description, you should now be able to derive the truth table and the circuit for a 2-to-4 decoder. The truth table and the circuit are shown in Figure 29 (b) and (c).



(a)

A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

(b)

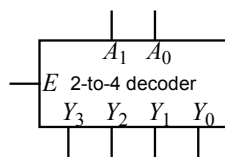


(c)

Figure 29: A 2-to-4 decoder: (a) logic symbol; (b) truth table; (c) circuit.

A variation of the 2-to-4 decoder has an extra enable (E) input line. When E is a 1, this decoder works exactly like the original. However, when the circuit is disabled with E set to 0, then it doesn't matter what the address input lines are, all of the output lines will be a 0. The logic symbol, truth table, and the circuit for the 2-to-4 decoder with enable are shown in Figure 30.

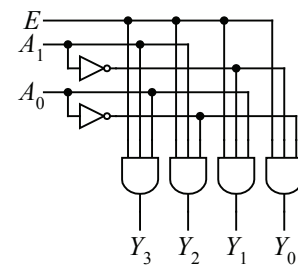
Combinational Logic Design



(a)

E	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

(b)



(c)

Figure 30: A 2-to-4 decoder with enable: (a) logic symbol; (b) truth table; (c) circuit.

In addition to the 2-to-4 decoder, there are other sizes of decoders, namely, the 1-to-2 decoder and the 3-to-8 decoder. Both of these can have the variation of either having or not having the enable input line. The 3-to-8 decoder will have three address lines and eight output lines.

Experiments

1. Implement the 2-to-4 decoder circuit as shown in Figure 29 (c), and verify that it works according to the truth table shown in Figure 29 (b). Connect the two inputs, A_1 , and A_0 , to two switches, and connect the outputs Y_3 , Y_2 , Y_1 , and Y_0 , to four LEDs.
2. Implement the 2-to-4 decoder with enable circuit as shown in Figure 30 (c), and verify that it works according to the truth table shown in Figure 30 (b).
3. Design and implement a 3-to-8 decoder.
4. Design and implement a 3-to-8 decoder with enable.

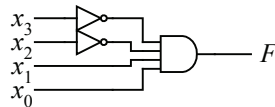
4.6. Lab 6: Comparators

Purpose

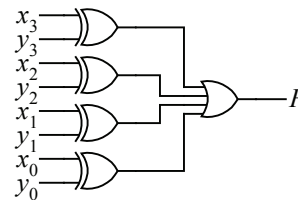
In this lab you will learn about comparator circuits. Comparators are for comparing between two numbers to see if one number is less than, equal to, or greater than a second number. You will implement different comparator circuits and verify their operations.

Introduction

A comparator will output a 1 if the logical condition that it is testing for is true, and outputs a 0 if the condition is false. The simplest comparator is to compare whether a value is equal or not equal to a constant. The use of an AND gate is all that is needed for the equality comparator with a constant. For example, the circuit in Figure 31 (a) tests whether a 4-bit variable x is equal to the constant 3 or not. Since 3 in binary is 0011, therefore, with x_3 and x_2 inverted, the AND gate will output a 1 when x is equal to 0011. For all numbers other than 3, it will output a 0.

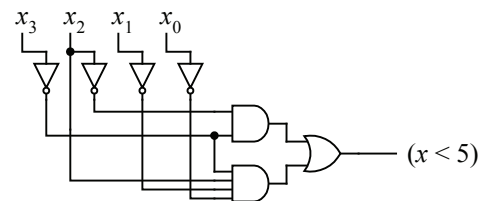


(a)



(b)

x_3	x_2	x_1	x_0	$(x < 5)$
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	x	x	x	0



$$(x < 5) = x_3'x_2' + x_3'x_2x_1'x_0'$$

(c)

Figure 31: Simple 4-bit comparators for: (a) $x = 3$; (b) $x \neq y$; (c) $x < 5$.

The XOR gate can be used for comparing inequality between two variables. Recall that the XOR gate outputs a 1 when its two input values are different. Hence, we can use one XOR gate for comparing each bit pair of the two operands. A 4-bit inequality comparator is shown in Figure 31 (b). Four XOR gates are used, with each one comparing the same position bit from the two operands. The outputs of the XOR gates are ORed together so that if any bit pair is different then the two operands

Combinational Logic Design

are different, and the resulting output is a 1.

For the greater-than or less-than relationships, we can construct a truth table and build the circuit from it. For example, to compare whether a 4-bit value x is less than five, we get the truth table shown in Figure 31 (c). The first five rows have a 1 output since their decimal values are equal to 0 to 4 respectively. The remaining rows from 5 to 15 will all have a 0 output. The resulting simplified circuit⁸ is shown in Figure 31 (c).

Experiments

1. Design and implement a 4-bit comparator circuit that tests for the condition $x = 7$. Verify that it operates correctly.
2. Design and implement a 4-bit comparator circuit that tests for the condition $x \neq 7$. Verify that it operates correctly.
3. Design and implement a 4-bit comparator circuit that tests for the condition $x = y$. Verify that it operates correctly.
4. Design and implement a 4-bit comparator circuit that tests for the condition $(x = 7) \text{ AND } (x = 12)$. Verify that it operates correctly.
5. Design and implement a 4-bit comparator circuit that tests for the condition $(x = 7) \text{ OR } (x = 12)$. Verify that it operates correctly.
6. Design and implement a 4-bit comparator circuit that tests for the condition $x \leq 7$. Verify that it operates correctly.
7. Design and implement a 4-bit comparator circuit that tests for the condition $x \geq 7$. Verify that it operates correctly.
8. Design and implement a 4-bit comparator circuit that tests for the condition $(x = 3) \text{ AND } (x \geq 7)$. Verify that it operates correctly.

⁸ For an in-depth discussion on how to simplify Boolean equations and circuits, refer to the book "Digital Logic and Microprocessor Design with VHDL" by E. Hwang.

4.7. Lab 7: Full Addder

Purpose

In this lab you will learn about the full adder (FA) circuit. The full adder circuit is for adding two 1-bit binary numbers with carry-in and carry-out. You will implement the circuit and verify its operations.

Introduction

Binary Numbers

Before we discuss the construction of the full adder, we need to talk about binary numbers. Binary numbers are positional numbers just like decimal numbers that you are very familiar with. In other words, the position of a digit or the position of a bit (in the case of binary numbers) denotes a specific value. For example, in the decimal number 38, the 3 actually has a higher value than the 8 because it is in the tens position whereas the 8 is in the ones position. For binary numbers we only use 0's and 1's instead of the ten digits for decimal numbers, and the base that we use is based-2 instead of based-10.

The value for the decimal number 238 is calculated as follows:

$$\begin{aligned} 238 &= (2 \cdot 10^2) + (3 \cdot 10^1) + (8 \cdot 10^0) \\ &= 200 + 30 + 8 \\ &= 238 \end{aligned}$$

Similarly, the value for the binary number 10011_2 (the subscript 2 denotes that the number is a binary number) is calculated as follows:

$$\begin{aligned} 10011_2 &= (1 \cdot 2^4) + (0 \cdot 2^3) + (0 \cdot 2^2) + (1 \cdot 2^1) + (1 \cdot 2^0) \\ &= 16 + 0 + 0 + 2 + 1 \\ &= 19 \text{ in decimal} \end{aligned}$$

When adding and subtracting binary numbers, we carry and borrow a two instead of a ten. So when we add $1 + 1$ in binary, we get a 10_2 instead of a 2.

The full adder circuit adds a 1-bit binary number with a second 1-bit number to produce a sum and a carry-out bit. The circuit also has a carry-in bit that allows it to be connected in series with other FA circuits. Consider the following addition of two 4-bit binary numbers

$$\begin{array}{r} 1001 \\ + 0011 \\ \hline 1100 \end{array}$$

c_{out} c_{in}

The full adder circuit is designed to only add one bit slice of the 4-bit number, for example, the bit slice that is outlined in the rectangle. In adding this bit slice, in addition to the two input operands,

Combinational Logic Design

there is also a carry-in bit (c_{in}) from the previous bit slice. The result of the addition produces a sum bit and a carry-out bit (c_{out}) for a carry to the next bit slice. The carry-in and carry-out bits allow the FA to pass bits from one bit position to the next bit position when several FAs are connected together in series. So in total, there are three input bits, first input operand (x), second input operand (y) and the carry-in (c_{in}), and two output bits, carry-out (c_{out}) and the sum (s). The logic symbol, truth table and the simplified circuit are shown in Figure 32 (a), (b) and (c) respectively.

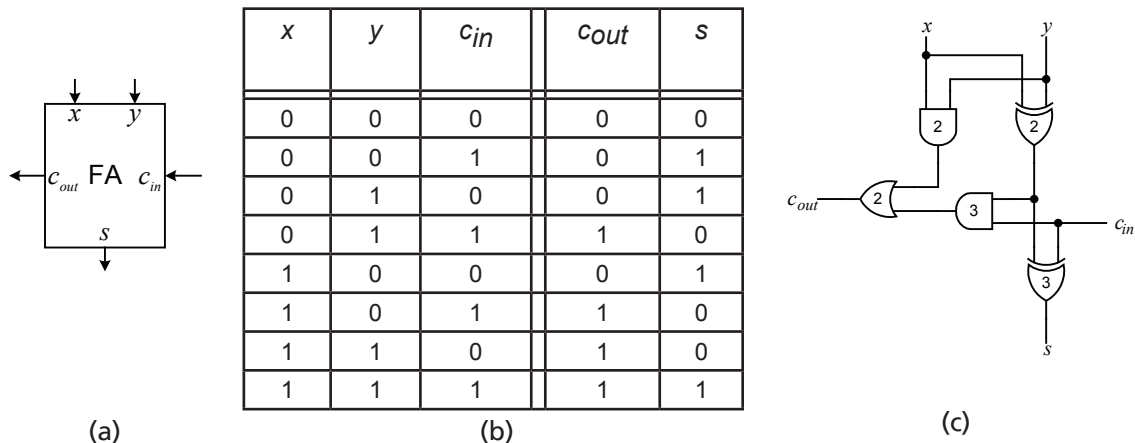


Figure 32: The full adder (FA): (a) logic symbol; (b) truth table; (c) simplified circuit.

Experiments

1. Implement the full adder circuit shown in Figure 32 (c) and verify that it operates according to the truth table shown in Figure 32 (b).
2. Notice that the full adder circuit shown in Figure 32 (c) is much simplified. If you construct the circuit by picking out the 1's rows in the output columns of the truth table, you will end up with a much larger circuit. Derive this larger circuit, then implement and verify that it operates according to the truth table.
3. What should the value for c_{in} be if all you want is to add the values of the two operands x and y ?
4. Derive the truth table for a full subtractor circuit following the same way that the full adder was constructed. You will have three inputs, x , y and b_{in} (for borrow in), and two outputs, b_{out} (for borrow out) and d (for difference). The circuit performs the subtraction $x - y$ to give d . A 1 for b_{in} denotes that there is a borrow from the bit on the right side, and a 1 for b_{out} denotes that this current bit needs to borrow from the bit on the left. So in essence, you are doing $x - y - b_{in}$ to give d , and if you need to borrow then set b_{out} to a 1, otherwise b_{out} is a 0. Keep in mind that you are working with binary numbers, so when you borrow, you get a 2 and not a 10. The first two rows of the truth table are shown next.

For the second row, you have $0 - 0 - 1$, and x being a 0 is not enough, so you borrow by setting b_{out} to a 1. When you borrow, you get a 2, so x now has a 2; and $2 - 0 - 1 = 1$, so d is also a 1. Try to complete the truth table on your own before looking for the answers. The complete truth table is found in Experiment 5 of Lab 3.

5. Implement the circuit from the truth table in Experiment 4, and verify that it operates correctly.

x	y	b_{in}	b_{out}	d
0	0	0	0	0
0	0	1	1	1
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Combinational Logic Design

4.8. Lab 8: 4-bit Adder

Purpose

In this lab you will learn about the 4-bit adder circuit. This 4-bit adder circuit is for adding two 4-bit binary numbers producing a 4-bit sum and a carry-out. You will implement the circuit and verify its operations.

Introduction

There are different ways to design a 4-bit adder circuit, but one very easy to understand way is to simply connect four full adder circuits together in series as shown in Figure 33 (b). Each box with the label FA is a full adder circuit from Figure 32 (c), and repeated here in Figure 33 (c) for easy reference. The carry-out (c_{out}) signal from each FA is connected to the carry-in (c_{in}) signal of the next FA on the left side. The first carry-in signal (c_0) is connected to a logic 0, and the final carry-out (c_{out}) signal is the overflow signal for the complete 4-bit adder circuit. The two 4-bit operands are x and y , and the resulting 4-bit sum from the adder is s . The subscripts for x , y , and s denote the bit position of the 4-bit number, e.g., x_0 is bit zero or the first bit of operand x , and s_3 is bit three or the fourth bit of s . The complete detailed schematic diagram is shown in Figure 34. This adder circuit is called a ripple-carry adder⁹ because of how the carry signal ripples through the chain of FAs.

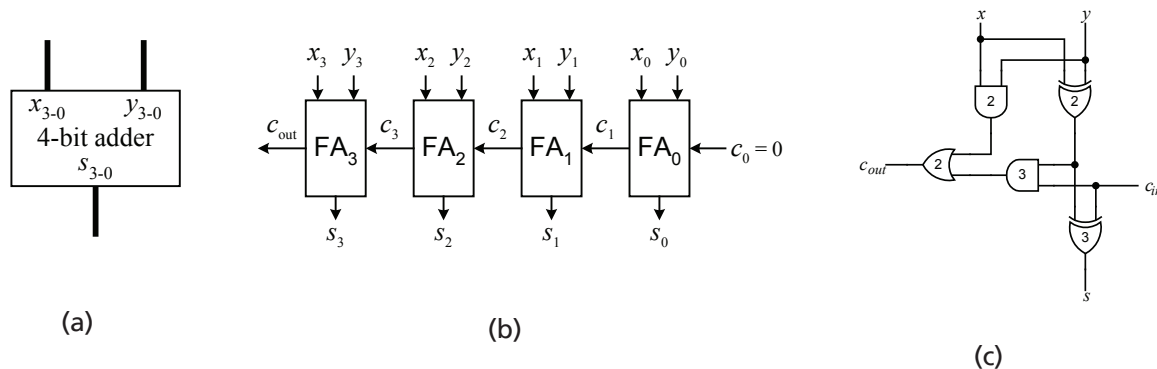


Figure 33: The 4-bit adder: (a) logic symbol; (b) circuit; (c) the FA circuit from Figure 32 (c).

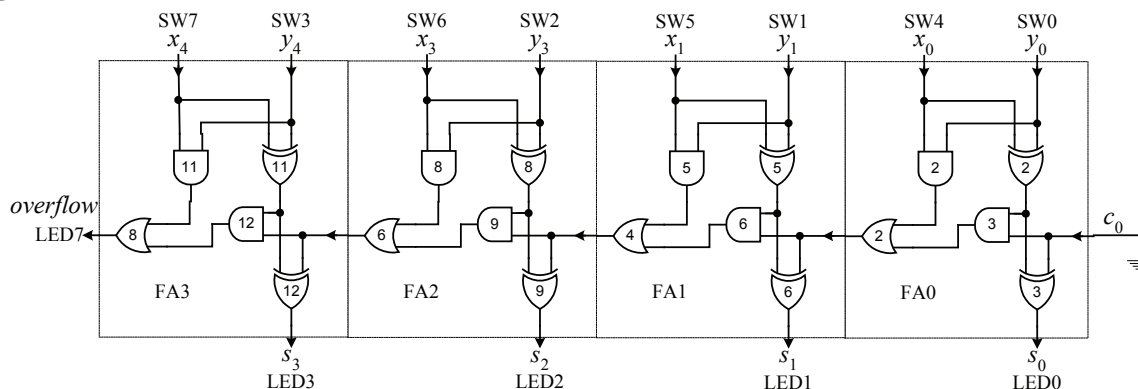


Figure 34: The complete detailed 4-bit adder circuit.

⁹ The operation of the ripple-carry adder is relatively slow. A faster adder circuit is called the carry-lookahead adder.

Combinational Logic Design

A 4-bit adder will produce the correct sum only if the result is in the range from 0 (binary 0000) to 15 (binary 1111). If the sum of the two numbers is greater than 15 then the overflow bit will be asserted, and the resulting sum bits will be incorrect. For example, $7 + 12 = 19$, but since 19 is 10011 in binary, therefore, the four sum bits will be 0011. In order to get the correct result for $7 + 12$, we need to use a 5-bit adder instead.

Experiments

1. Implement the circuit shown in Figure 34. You will need to have four FA circuits with their c_{in} 's and c_{out} 's connected together. Connect the 4-bit operand x_3, x_2, x_1 and x_0 to switches SW7 to SW4. Similarly connect the 4-bit operand y_3, y_2, y_1 and y_0 to switches SW3 to SW0. Connect the 4-bit sum s_3, s_2, s_1 and s_0 to LED3 to LED0. Connect the carry-in signal (c_{in}) to GND. Connect the carry-out signal (c_{out}) to LED7. Verify that your circuit does add two 4-bit binary numbers correctly. The following table lists some sample numbers that you might want to try. The numbers in parenthesis are the corresponding decimal numbers. Note that to be absolutely certain that your circuit works correctly, you need to test all possible input combinations.

Input		Output		Observed Result
Operand x	Operand y	Carry-out c_{out}	Sum s	
0010 (2)	0011 (3)	0	0101 (5)	
0101 (5)	0100 (4)	0	1001 (9)	
0110 (6)	1001 (9)	0	1111 (15)	
0111 (7)	1100 (12)	1	0011 (16+3=19)	
1001 (9)	1101 (13)	1	0110 (16+6=22)	
1111 (15)	1111 (15)	1	1110 (16+14=30)	

2. What happens if the initial carry-in signal c_0 is connected to a 1 instead of a 0?
3. How would you expand this 4-bit adder circuit to be an 8-bit adder circuit?
4. When you add 7 (0111) plus 12 (1100), you get a 1 for c_{out} and the sum s is actually a 3 (0011). What do you need to do in order to actually get the correct binary value in the *Sums* output when you add $7 + 12$?
5. How many bits do you need for an adder in order to produce the correct result for adding $16 + 16$ and?

4.9. Lab 9: 4-bit Adder/Subtractor

Purpose

In this lab you will learn about a 4-bit adder/subtractor circuit. This 4-bit adder/subtractor circuit is for both adding and subtracting two 4-bit binary numbers producing either a 4-bit sum or difference. You will implement the circuit and verify its operations.

Introduction

It turns out that with very minimal modifications to the 4-bit adder circuit from Figure 33 (b), you can get it to subtract numbers as well. This simple and elegant solution is a direct result of how negative numbers are represented inside computers.

Negative Binary Numbers

Binary numbers can be interpreted as either signed or unsigned. Unsigned numbers include only positive numbers and zero, whereas signed numbers include positive, negative, and zero. Given a binary number such as 01101001_2 , the computer does not know whether it is a signed or unsigned number. It is up to you, the designer, to decide how you want to interpret it. If you say that this binary number represents an unsigned number, then the decimal value of this number would be

$$\begin{aligned} & 01101001_2 \\ &= (0 \times 2^7) + (1 \times 2^6) + (1 \times 2^5) + (0 \times 2^4) + (1 \times 2^3) + (0 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) \\ &= (1 \times 2^6) + (1 \times 2^5) + (1 \times 2^3) + (1 \times 2^0) \\ &= 64 + 32 + 8 + 1 \\ &= 105 \text{ in decimal} \end{aligned}$$

(Notice that the power x used in 2^x denotes the position of the bit in the binary number. The first bit of the binary number starting from the right hand side is position 0; the second bit to the left is position 1; etc.)

If you say that this same binary number 01101001_2 represents a signed number, then the decimal value of this number would be evaluated differently. For signed numbers, the most significant bit (MSB) which is the left-most bit, tells whether the number is positive or negative. If the most significant bit is a 0, then the number is positive, and the value of this positive signed number is obtained exactly as for unsigned numbers. So if we interpret 01101001_2 as a signed number, we would get the same decimal value 105.

However, if the most significant bit of a signed number is a 1, then the number is negative, and we use what is referred to as the *two's complement* method to determine its value. The 2's complement¹⁰ is a method for representing negative or signed numbers, and it involves three steps to determine the value. In step 1, you flip all the 1 bits in the binary number to 0's and all the 0 bits to 1's. In step 2, you add a 1 to the result obtained from step 1. Finally, for step 3, interpret the binary number obtained in step 2 as an unsigned number, and determine its value. The negative of this resulting value is the value of the original negative signed number.

¹⁰ Sometimes two's complement is also written as 2's complement.

Combinational Logic Design

For example, if we say that the binary number 11101001_2 represents a signed number, then we would have to do the following to determine its value. First we note that the MSB is a 1 so we know that it is a negative number. To determine the value of a negative number, we perform the three steps for the 2's complement process.

Starting with this number:	11101001
Step 1, flip the bits:	00010110
Step 2, add a one to the number from step 1:	00010111
Step 3, determine the value of the number from step 2: $(1 \times 2^4) + (1 \times 2^2) + (1 \times 2^1) + (1 \times 2^0)$	
	= $16 + 4 + 2 + 1$
	= 23

Therefore, the value for the signed number 11101001_2 is -23 .

To find the 2's complement binary representation of a negative number, we start with the positive binary representation and then perform only the first two steps in the process. For example, to find the 2's complement binary representation for -35 , we start with the binary representation for $+35$.

Starting with $+35$:	00100011
Step 1, flip the bits:	11011100
Step 2, add a one to the number:	11011101

Therefore, 11011101_2 is the 2's complement representation for -35 . You can verify that this is indeed correct by performing the three step process again.

Subtractor Circuit

Now that we understand how negative numbers are represented, we are ready to design the subtractor circuit. We know from algebra that subtracting a positive number is the same as adding the negative of the number. So using this fact, we can use the 4-bit adder to do subtraction simply by changing the second number to its negative equivalent. In other words, we simply perform the first two steps ("flip the bits" and "add a 1") in the 2's complement process to convert the second number to its negative value, and then pass this negative number to the adder circuit. The interesting trick is that these two steps can be easily done at two different places in the circuit. First, we can flip the bits by using a NOT gate for every bit of the second number. Second, remember that setting c_0 to a 1 in the FA adds an extra one to the sum, and since the addition of a 1 can occur anytime during the addition process, therefore we can add the 1 simply by setting c_0 to a 1 instead of the original 0. Hence, we obtain the subtractor circuit shown in Figure 35.

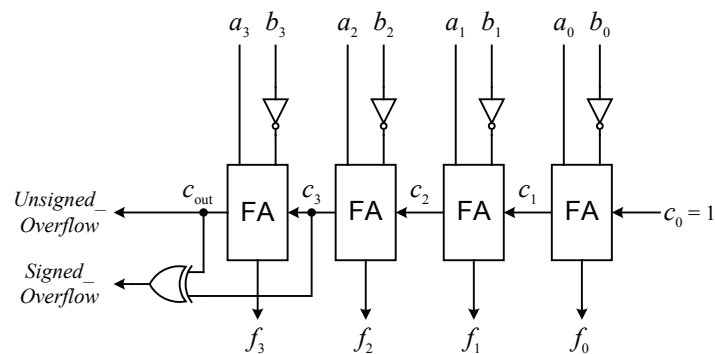


Figure 35: The 4-bit subtractor circuit.

Notice that there is an extra XOR gate whose inputs are connected to c_3 and c_{out} . The output from this XOR gate indicates whether there is an overflow or not for signed numbers. The c_{out} signal alone only indicates an overflow for unsigned numbers. Whether you want to consider the status of the unsigned overflow bit or the signed overflow bit depends on how you want to interpret the input numbers. If you want to interpret your input numbers as unsigned numbers then you need to only look at the unsigned overflow bit to determine whether there is an overflow or not. On the other hand, if you interpret your input numbers as signed numbers then you need to only look at the signed overflow bit.

The complete detailed schematic diagram for the 4-bit subtractor circuit is shown in Figure 36.

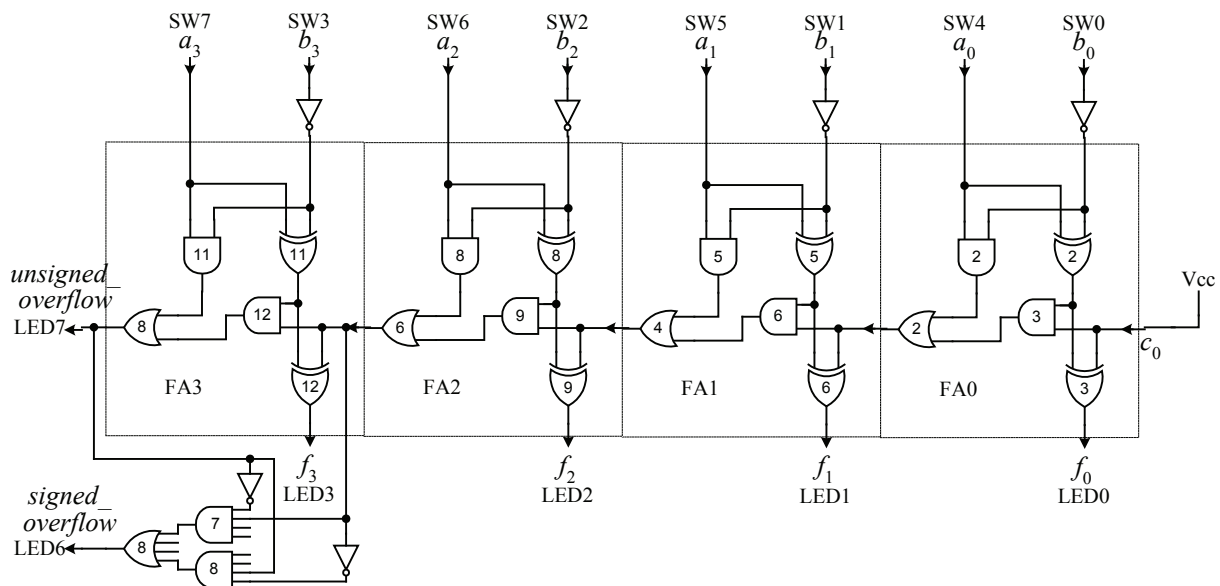


Figure 36: The complete detailed 4-bit subtractor circuit.

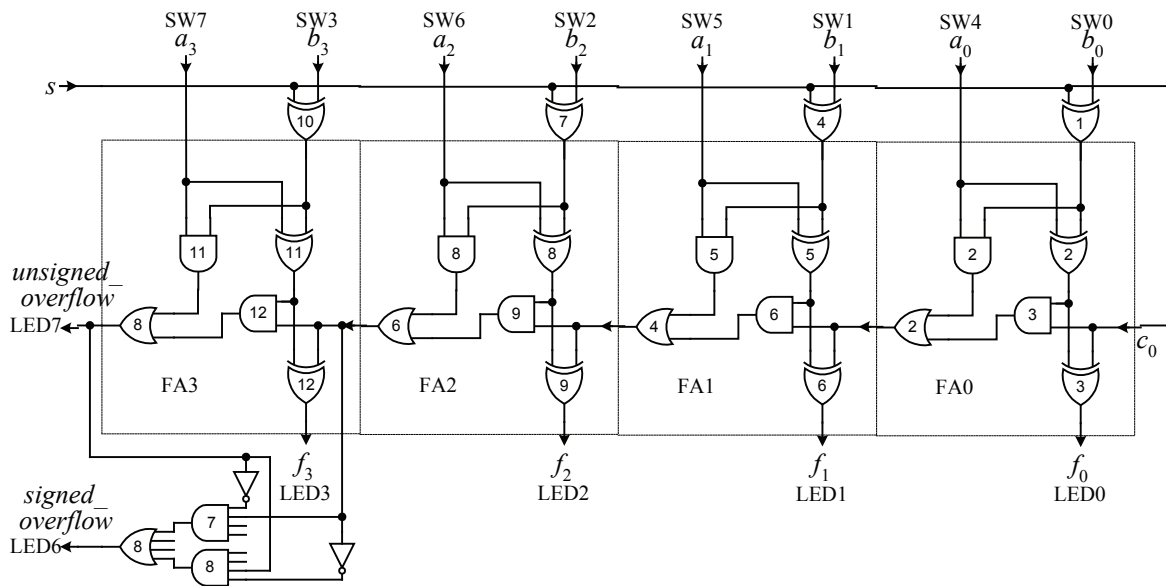


Figure 38: The complete detailed 4-bit adder/subtractor circuit.

Experiments

1. Implement the circuit shown in Figure 36. You will need to have four FA circuits with their c_{in} 's and c_{out} 's connected together. Connect the 4-bit operand a (a_3 to a_0) to switches SW7 to SW4. Connect the 4-bit operand b (b_3 to b_0) to switches SW3 to SW0. Connect the 4-bit result f (f_3 to f_0) to LED3 to LED0. Connect the carry-out signal (c_{out}) to LED7. Verify that your circuit does subtract two 4-bit binary numbers correctly. The following table lists some sample numbers that you might want to try. The numbers in parenthesis are the corresponding decimal numbers.

Input		Output			Observed Result
Operand x	Operand y	unsigned-overflow	signed-overflow	Difference f	
0011 (3)	0010 (2)	1	0	0001 (1)	
0011 (3)	0011 (3)	1	0	0000 (0)	
0010 (2)	0011 (3)	0	0	1111 (-1)	
0111 (7)	0100 (4)	1	0	0011 (3)	
1001 (-7)	1101 (-3)	0	0	1100 (-4)	
1111 (-1)	1111 (-1)	1	0	0000 (0)	
1000 (-8)	0011 (3)	1	1	0101 (5)	
1000 (-8)	0111 (7)	1	1	0001 (1)	

2. In the first row of the table for Experiment 1 where you performed $3 - 2$, the unsigned overflow bit is a 1 because you are actually doing $3 + (-2)$ which is $0011 + 1110$. If you interpret these as unsigned numbers, then you are really adding $3 + 14 = 17$, and 17 cannot be represented as a 4-bit unsigned value!

Combinational Logic Design

3. Implement the circuit shown in Figure 38. Verify that your circuit does either add or subtract two 4-bit binary numbers correctly. The following table lists some sample numbers that you might want to try. The numbers in parenthesis are the corresponding decimal numbers.

<i>Input</i>			<i>Observed Output</i>			<i>Observed Result</i>
<i>s</i>	<i>Operand x</i>	<i>Operand y</i>	<i>unsigned-overflow</i>	<i>signed-overflow</i>	<i>Sum/Difference f</i>	
0	0000 (0)	0000 (0)	0	0	0000 (0)	
0	0011 (3)	0010 (2)	0	0	0101 (5)	
0	0011 (3)	0011 (3)	0	0	0110 (6)	
0	0010 (2)	0011 (3)	0	0	0101 (5)	
0	0101 (5)	0100 (4)	0	1	1001 (-7)	
0	0101 (5)	1001 (-7)	0	0	1110 (-2)	
0	0111 (7)	1001 (-7)	1	0	0000 (0)	
0	1001 (-7)	1001 (-7)	1	1	0010 (2)	
0	1111 (-1)	1111 (-1)	1	0	1110 (-2)	
0	1000 (-8)	1101 (-3)	1	1	0101 (5)	
1	0000 (0)	0000 (0)	1	0	0000 (0)	
1	0011 (3)	0010 (2)	1	0	0001 (1)	
1	0011 (3)	0011 (3)	1	0	0000 (0)	
1	0010 (2)	0011 (3)	0	0	1111 (-1)	
1	0101 (5)	0100 (4)	1	0	0001 (1)	
1	0101 (5)	1001 (-7)	0	0	1110 (-2)	
1	1001 (-7)	1001 (-7)	1	0	0000 (0)	
1	0111 (7)	1001 (-7)	0	1	1110 (-2)	
1	1111 (-1)	1111 (-1)	1	0	0000 (0)	
1	1000 (-8)	1101 (-3)	0	0	1011 (-5)	

4.10. Lab 10: 2-bit Arithmetic and Logic Unit (ALU)

Purpose

In this lab you will learn about the Arithmetic and Logic Unit (ALU). This is the main component inside the microprocessor for performing simple logical and arithmetic operations. The logical operations are like those of the basic logic gates such as the AND, OR and NOT. The arithmetic operations are just simple additions and subtractions. More complex arithmetic operations such as multiply and divide are done in other dedicated components. You will design a 2-bit ALU, implement the circuit, and verify its operations.

Introduction

Like with most circuits, there are many ways of designing the ALU circuit. One method is to do something similar to what we did with the adder/subtractor circuit in Lab 8 by starting out with the basic layout of connecting several full adders (FA) together in series. And just like for the adder/subtractor circuit, we will modify the two operand inputs to the FA appropriately so that the FAs will produce the correct results.

For doing additions and subtractions, we will modify the second (y) operand to the FAs doing something similar to the XOR gate that was used in the adder/subtractor circuit, but with some slight modifications. We will label this sub-circuit black box the AE for Arithmetic Extender.

For doing logical operations, we need to do something a little more extensive. After all, you remember that the FAs can only add numbers, and you saw in Lab 8 how you could use the FAs to both add and subtract numbers. However, the FAs cannot do logical operations, so what we will need to do is to have another sub-circuit to perform the actual logical operations and then pass the result of the logical operations through the FAs to the output. So for logical operations, we do not want the FAs to modify the number that is being passed through its first (x) operand, and the way to do this is to have the FAs add a 0 to the number. We will label this sub-circuit black box the LE for Logic Extender.

Furthermore, depending on the operation that we want to perform, the initial carry-in signal, c_0 , has to be set appropriately. In the design, this is done by the Carry Extender, CE, black box.

Select lines are needed to tell the ALU which one of several operations to perform. If we want the ALU to be able to perform four operations, we will need two select lines, s_1 and s_0 , because two bits will give four different combinations (00, 01, 10 and 11). Since the outputs of the LE, AE and CE are dependent on which operation we want to perform, therefore, the select lines are also inputs to these three boxes.

s_1	s_0	Operation Name	Operation	x_j (LE)	y_j (AE)	c_0 (CE)
0	0	Addition	A + B	a_j	b_j	0
0	1	Subtraction	A - B	a_j	b_j'	1
1	0	Logical AND	A AND B	a_j AND b_j	0	0
1	1	Logical OR	A OR B	a_j OR b_j	0	0

(a)

Op	s_1	s_0	a_j	b_j	x_j (LE)
Add	0	0	0	0	0
	0	0	0	1	0
	0	0	1	0	1
	0	0	1	1	1
Subtract	0	1	0	0	0
	0	1	0	1	0
	0	1	1	0	1
	0	1	1	1	1
AND	1	0	0	0	0
	1	0	0	1	0
	1	0	1	0	0
	1	0	1	1	1
OR	1	1	0	0	0
	1	1	0	1	1
	1	1	1	0	1
	1	1	1	1	1

(b)

Op	s_1	s_0	b_j	y_j (AE)
Add	0	0	0	0
	0	0	1	1
Sub	0	1	0	1
	0	1	1	0
AND	1	0	0	0
	1	0	1	0
OR	1	1	0	0
	1	1	1	0

(c)

s_1	s_0	c_0 (CE)
0	0	0
0	1	1
1	0	0
1	1	0

(d)

Figure 40: ALU operations: (a) function table; (b) LE truth table; (c) AE truth table; (d) CE truth table.

For the two logical operations, the actual operation will be performed in the LE, so the LE will output the result of the respective logical operation. So for the combination $s_1s_0 = 10$ where the ALU will perform the logical AND operation, the LE will output the result of a_j AND b_j . Similarly, for the combination $s_1s_0 = 11$ where the ALU will perform the logical OR operation, the LE will output the result of a_j OR b_j . For all logical operations, we do not want the FAs to add anything, so the AE, which outputs to the second operand of the FA, should output a 0. Similarly, the CE should output a 0.

So from the above analysis, we are able to come up with the three truth tables for the LE, AE and CE as shown in Figure 40 (b), (c) and (d) respectively. Given these three truth tables for the LE, AE and

Experiments

1. Implement the 2-bit ALU circuit shown in Figure 42. You will need to have two FA circuits with their c_{in} 's and c_{out} 's connected together. For each x input to the FA, you will need to connect the output of the LE circuit to it. Similarly, for each y input to the FA, you will need to connect the output of the AE circuit to it. Connect the output of the CE circuit to c_0 . Connect the 2-bit operand a to switches SW3 and SW2. Connect the 2-bit operand b to switches SW1 and SW0. Connect the 2-bit result f to LED1 and LED0. Connect the *unsigned-overflow* signal to LED3, and the *signed-overflow* signal to LED4. In the circuit diagram, the numbers inside each gate denote the corresponding component number on the board to help in debugging. These numbers are for reference only. Verify that your ALU circuit does perform the four different operations as given in the table in Figure 40 (a) correctly. Record your results and compare your results with the following table. The four bits (e.g. 0-0-00) in the entries denote, from left to right, the signed-overflow bit, unsigned-overflow bit, f_1 , and f_0 .

Operand a	Operand b	$a + b$ $s = 00$	$a - b$ $s = 01$	$a \text{ AND } b$ $s = 10$	$a \text{ OR } b$ $s = 11$
00	00	0-0-00	0-1-00	0-0-00	0-0-00
	01	0-0-01	0-0-11	0-0-00	0-0-01
	10	0-0-10	1-0-10	0-0-00	0-0-10
	11	0-0-11	0-0-01	0-0-00	0-0-11
01	00	0-0-01	0-1-01	0-0-00	0-0-01
	01	1-0-10	0-1-00	0-0-01	0-0-01
	10	0-0-11	1-0-11	0-0-00	0-0-11
	11	0-1-00	1-0-10	0-0-01	0-0-11
10	00	0-0-10	0-1-10	0-0-00	0-0-10
	01	0-0-11	1-1-01	0-0-00	0-0-11
	10	1-1-00	0-1-00	0-0-10	0-0-10
	11	1-1-01	0-0-11	0-0-10	0-0-11
11	00	0-0-11	0-1-11	0-0-00	0-0-11
	01	0-1-00	0-1-10	0-0-01	0-0-11
	10	1-1-01	0-1-01	0-0-10	0-0-11
	11	0-1-10	0-1-00	0-0-11	0-0-11

2. Design and implement a 2-bit ALU having the following functions:

s_1	s_0	Operation Name	Operation
0	0	Increment	$A + 1$
0	1	Decrement	$A - 1$
1	0	Addition	$A + B$
1	1	Subtraction	$A - B$

Combinational Logic Design

3. Design and implement a 2-bit ALU having the following functions:

s_1	s_0	Operation Name	Operation
0	0	Pass A through	A
0	1	Pass B through	B
1	0	Logical NOT	NOT A
1	1	Logical NAND	A NAND B

4. Design a 4-bit ALU having the following functions:

s_1	s_0	Operation Name	Operation
0	0	Pass A through	A
0	1	Addition	A + B
1	0	Subtraction	A – B
1	1	Increment	A + 1

5. Design a 4-bit ALU having the following functions:

s_2	s_1	s_0	Operation Name	Operation
0	0	0	Pass A through	A
0	0	1	Addition	A + B
0	1	0	Subtraction	A – B
0	1	1	Increment	A + 1
1	0	0	Decrement	A – 1
1	0	1	Logical AND	A AND B
1	1	0	Logical OR	A OR B
1	1	1	Logical NOT	NOT A

4.11. Lab 11: BCD to 7-segment LED Decoder

Purpose

In this lab you will learn how to construct a Binary-Coded-Decimal (BCD) to 7-segment LED display decoder circuit. This circuit converts a 4-bit binary number to cause a 7-segment LED display to show the corresponding decimal digit. You will implement the circuit and verify its operation.

Introduction

BCD is the name given for the 4-bit binary representation of the ten decimal digits. Only the first ten combinations of the 4-bit binary number (from 0000 to 1001) are used as shown in the first two columns (Inputs and Decimal Digit) of the table in Figure 43. For each one of these ten binary combinations, the corresponding decimal digit is to be shown on the 7-segment LED display as shown in the Display column of the table in Figure 43.

Each LED segment in the 7-segment display has a letter name (from *a* to *g*) given to it as shown in the header of the last seven columns in Figure 43. Hence, to display the decimal digit 0, we want segments *a*, *b*, *c*, *d*, *e* and *f* to be turned on, while segment *g* is turned off. Similarly, to display the decimal digit 1, we want only segments *b* and *c* to be turned on, while the remaining segments are turned off. Continuing on in this fashion, we obtain the rest of the truth table for the seven individual segments as shown in the last seven columns of the table in Figure 43.

The remaining six binary combinations (from 1010 to 1111) are not used in the BCD to 7-segment decoder, therefore, it does not matter what those values are. The x symbol is used to denote the "don't care" value of a variable.


















Inputs				Decimal Digit	Display	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>	<i>g</i>
i_3	i_2	i_1	i_0									
0	0	0	0	0		1	1	1	1	1	1	0
0	0	0	1	1		0	1	1	0	0	0	0
0	0	1	0	2		1	1	0	1	1	0	1
0	0	1	1	3		1	1	1	1	0	0	1
0	1	0	0	4		0	1	1	0	0	1	1
0	1	0	1	5		1	0	1	1	0	1	1
0	1	1	0	6		1	0	1	1	1	1	1
0	1	1	1	7		1	1	1	0	0	0	0
1	0	0	0	8		1	1	1	1	1	1	1
1	0	0	1	9		1	1	1	0	0	1	1
Rest of the Combinations						x	x	x	x	x	x	x

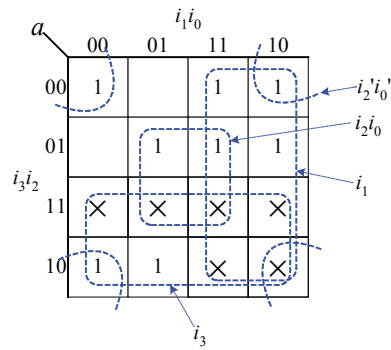
Figure 43: Design and truth table for the BCD to 7-segment LED decoder. X denotes 0 or a 1.

Combinational Logic Design

Having completed the truth table for the decoder, we can continue with the design by deriving the equations and drawing the circuits for each of the seven segments. As discussed in Lab 2, given any truth table, we can produce a circuit for it by simply ANDing the inputs of a row for which the output of that row is a 1, and then ORing the outputs of all the AND gates together. Looking at the column in the truth table for segment *a*, we note that there are eight 1's. Hence we obtain the following Boolean equation.

$$a = i_3'i_2'i_1'i_0' + i_3'i_2'i_1'i_0 + i_3'i_2'i_1'i_0 + i_3'i_2'i_1'i_0 + i_3'i_2'i_1'i_0 + i_3'i_2'i_1'i_0 + i_3'i_2'i_1'i_0 + i_3'i_2'i_1'i_0$$

To implement the circuit for segment *a* based on this equation would require four NOT gates, eight 4-input AND gates and one 8-input OR gate. However, we can get a much smaller circuit by simplifying the equation. The K-map for simplifying the Boolean equation for segment *a* is shown next.



The x's are also placed in the K-map. Each x can either be considered as a 1 if it helps to make a larger subcube, or it can be considered as a 0 and be ignored if it doesn't help to make a larger subcube. In this K-map for segment *a*, all of the x's are taken to be a 1. From evaluating the above K-map for segment *a*, we obtain the following simpler equation for segment *a*

$$a = i_3 + i_1 + i_2'i_0' + i_2i_0 = i_3 + i_1 + (i_2 \odot i_0)$$

Proceeding in a similar manner, we get the following remaining six simplified equations for segments *b*, *c*, *d*, *e*, *f* and *g*.

$$b = i_2' + (i_1 \odot i_0)$$

$$c = i_2 + i_1' + i_0$$

$$d = i_1i_0' + i_2'i_0' + i_2'i_1 + i_2i_1'i_0$$

$$e = i_1i_0' + i_2'i_0'$$

$$f = i_3 + i_2i_1' + i_2i_0' + i_1'i_0'$$

$$g = i_3 + (i_2 \text{ \AA } i_1) + i_1i_0'$$

Based on these seven simplified equations, we obtain the circuit as shown in Figure 44.

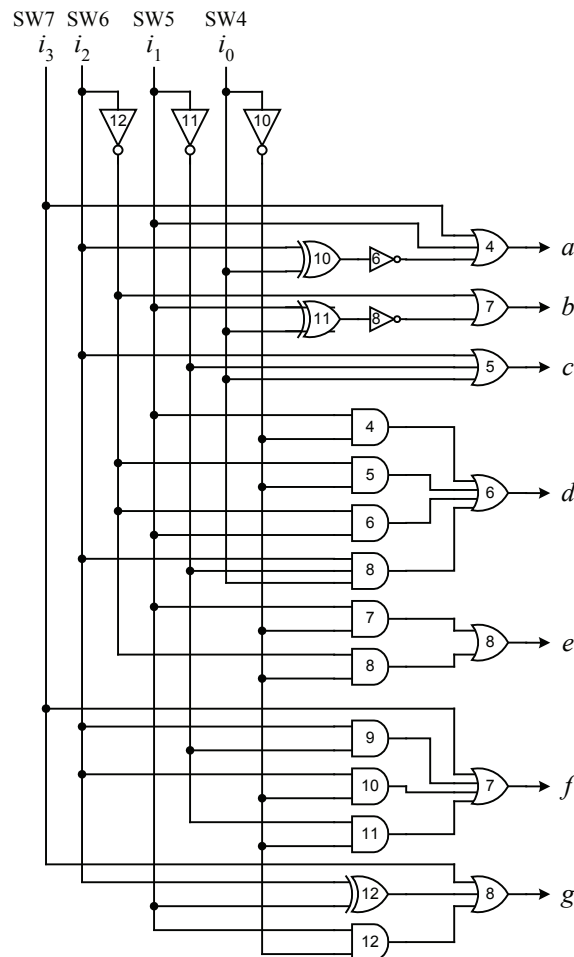


Figure 44: Reduced circuit for the BCD to 7-segment LED decoder.

Experiments

1. Implement the BCD to 7-segment LED display decoder circuit as shown in Figure 44. Connect the four inputs i_3, i_2, i_1, i_0 to the four switches SW7, SW6, SW5 and SW4 respectively, and connect the seven outputs a, b, c, d, e, f, g to the 7-segment LED display. Notice that the output of NOT gate number 10 needs to connect to seven inputs. Since there are only six wire connection points, you can make one connection to the breadboard to give you more connection points. The XNOR gate can be obtained by connecting a NOT gate to the output of the XOR gate. In the schematic diagram it is the XOR gate number 10 connected to the NOT gate number 6, and the XOR gate number 11 connected to the NOT gate number 8. Verify that it operates correctly according to the table in Figure 43.
2. Implement the BCD to 7-segment LED display decoder circuit based directly from the original truth table shown in Figure 43 and without doing any simplifications. Verify that it operates correctly according to the table in Figure 43.
3. Recall from Experiment 4 of Lab 2 that you can get the same functional circuit by selecting the

Combinational Logic Design

rows in the truth table where the output is a 0 instead of a 1 and then inverting the final output. By doing this, you might get a smaller circuit. Use this method to design the circuit for the truth table in Figure 43. Do you get a smaller circuit than the one in Figure 44?

4. Instead of the BCD to 7-segment decoder where only ten of the 16 combinations are used, we can build a complete 4-bit to 7-segment hexadecimal LED display decoder where we decode all of the 16 combinations of the 4-bit binary number. Figure 45 shows the initial design table. Complete the design of this 4-bit to 7-segment hexadecimal LED display decoder circuit and implement it.
























Inputs				Decimal Digit	Display	a	b	c	d	e	f	g
i_3	i_2	i_1	i_0									
0	0	0	0	0		1	1	1	1	1	1	0
0	0	0	1	1		0	1	1	0	0	0	0
0	0	1	0	2		1	1	0	1	1	0	1
0	0	1	1	3		1	1	1	1	0	0	1
0	1	0	0	4		0	1	1	0	0	1	1
0	1	0	1	5		1	0	1	1	0	1	1
0	1	1	0	6		1	0	1	1	1	1	1
0	1	1	1	7		1	1	1	0	0	0	0
1	0	0	0	8		1	1	1	1	1	1	1
1	0	0	1	9		1	1	1	0	0	1	1
1	0	1	0	A		1	1	1	0	1	1	1
1	0	1	1	B		0	0	1	1	1	1	1
1	1	0	0	C		1	0	0	1	1	1	0
1	1	0	1	D		0	1	1	1	1	0	1
1	1	1	0	E		1	0	0	1	1	1	1
1	1	1	1	F		1	0	0	0	1	1	1

Figure 45: Design of the 4-bit to 7-segment hexadecimal LED display decoder.

