Learning Computer Logic Architectures on Apple II System

Falah Reda Al-Saffar

Follow this and additional works at: https://scholarworks.wmich.edu/masters_theses

Recommended Citation
https://scholarworks.wmich.edu/masters_theses/1458

This Masters Thesis-Open Access is brought to you for free and open access by the Graduate College at ScholarWorks at WMU. It has been accepted for inclusion in Master's Theses by an authorized administrator of ScholarWorks at WMU. For more information, please contact maira.bundza@wmich.edu.
LEARNING COMPUTER LOGIC ARCHITECTURES  
ON APPLE II SYSTEM  

by  

Falak Reda Al-Saffar  

A Thesis  
Submitted to the  
Faculty of The Graduate College  
in partial fulfillment of the  
requirements for the  
Degree of Master of Science  
Department of Computer Science  

Western Michigan University  
Kalamazoo, Michigan  
April 1984
LEARNING COMPUTER LOGIC ARCHITECTURES
ON APPLE II SYSTEM

Falah Reda Al-Saffar, M.S.
Western Michigan University, 1984

For this thesis seven computer-assisted instruction packages have been designed to illustrate the design of electronic circuits of gates, the logic of combinational and sequential circuits, and integrated and magnetic core memories.

The programs in these packages have been written in the Pascal computer language on an Apple II microcomputer and stored on a double-sided floppy disk.
ACKNOWLEDGEMENTS

I would like to thank my thesis advisor, Dr. Dionysios Kountanis, for his advice toward the accomplishment of my thesis, and Dr. William McCabe and Dr. Dalia Motzkin-Groper for reading my thesis. Special thanks are considered to my wife, Jayne, who gave assistance in correcting the paper.

Also, I would like to thank my parents who have supported me to reach my goal.

Falah Reda Al-Saffar
AL-SAFFAR, FALAH REDA MAHDI

LEARNING COMPUTER LOGIC ARCHITECTURES ON APPLE II SYSTEM

WESTERN MICHIGAN UNIVERSITY M.S. 1984

University Microfilms International 300 N. Zeeb Road, Ann Arbor, MI 48106
PLEASE NOTE:

In all cases this material has been filmed in the best possible way from the available copy. Problems encountered with this document have been identified here with a check mark √.

1. Glossy photographs or pages ______
2. Colored illustrations, paper or print ______
3. Photographs with dark background ______
4. Illustrations are poor copy ______
5. Pages with black marks, not original copy ______
6. Print shows through as there is text on both sides of page ______
7. Indistinct, broken or small print on several pages √
8. Print exceeds margin requirements ______
9. Tightly bound copy with print lost in spine ______
10. Computer printout pages with indistinct print ______
11. Page(s) ______ lacking when material received, and not available from school or author.
12. Page(s) ______ seem to be missing in numbering only as text follows.
13. Two pages numbered _______. Text follows.
14. Curling and wrinkled pages ______
15. Other ______________________________________________________________________

University
Microfilms
International

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
TABLE OF CONTENTS

ACKNOWLEDGEMENTS........................................................................................................ii
LIST OF TABLES...............................................................................................................viii
Chapter

I. INTRODUCTION............................................................................................................1
II. GENERAL VIEW..........................................................................................................3
   Background.................................................................................................................3
   Overview of Literature...............................................................................................5
   Review of Computer Electronics................................................................................5
   Review of Graphics.....................................................................................................6
   Computer in Education...............................................................................................7
III. THE BASIC ELECTRONIC CONSTRUCTION OF GATES.................................10
   Part 1..........................................................................................................................10
   Resistor......................................................................................................................10
   Diode.........................................................................................................................11
   Transistor...................................................................................................................12
   Not Gate.....................................................................................................................13
   Or Gate......................................................................................................................13
   And Gate...................................................................................................................14
   Nor Gate....................................................................................................................14
   Nand Gate................................................................................................................15
   Part 2..........................................................................................................................15
   "1: Diode"...............................................................................................................16
"2: Transistor" ................................................................. 16
"3: Not Gate" ........................................................................ 17
"4: Or Gate" ........................................................................ 17
"5: And Gate" ....................................................................... 17
"6: Nor Gate" ....................................................................... 18
"7: Nand Gate" ..................................................................... 18
"8: Help" ................................................................................ 18
"9: Exit" ................................................................................ 18

IV. KARNAUGH MAP AND COMBINATIONAL CIRCUITS .......... 19
    Part 1 ............................................................................... 19
    K-map ............................................................................... 19
    Combinational Circuits ..................................................... 24
    Part 2 ............................................................................... 27

    "1: Display Map" ............................................................ 30
    "2: Selfcover" ..................................................................... 30
    "3: Suggested Solution" .................................................... 33
    "4: The Combinational Circuit" ........................................ 33
    "5: Help" ............................................................................. 33
    "6: Exit" ............................................................................. 33

V. APPLICATION OF COMBINATIONAL CIRCUITS ................. 34
    Part 1 ............................................................................... 34
    Binary Adder ...................................................................... 34
    Look-Ahead Carry Generator ......................................... 38
    4-Bit Full-Adders with Look-Ahead Carry Generator ........ 40
    Decoder ............................................................................. 40
VI. SEQUENTIAL CIRCUITS AND FLIP-FLOP TECHNIQUES

Part 1
Basic Flip-Flop Circuit
Clocked RS Flip-Flop
RST Flip-Flop
Clocked JK Flip-Flop
Clocked T Flip-Flop
D Flip-Flop
Master-Slave JK Flip-Flop Technique

Part 2
"1- RS Flip-Flop"
"2- RST Flip-Flop"
"3- JK Flip-Flop"
"4- T Flip-Flop"
LIST OF TABLES

TABLE

1. INPUT-OUTPUT FOR FIGURE 14 ............................................................. 26
2. ADDING TWO BINARY NUMBERS ........................................................... 35
3. TRUTH TABLE FOR HALF-ADDER ................................................................. 35
4. TRUTH TABLE FOR FULL-ADDER ................................................................. 37
5. FUNCTION TABLE FOR 4-TO-1 LINE MULTIPLEXER .................................. 45
6. TRUTH TABLE FOR 4X2 ROM ................................................................. 46
7. FLIP-FLOP TABLE .................................................................................. 95
CHAPTER I

INTRODUCTION

The main purpose of this project is to assist students in learning digital logic and computer structures on an Apple II system.

The goals of the project are accomplished through a series of teaching aid packages. These packages simulate the function and structure of some combinational and sequential circuits using graphics displays.

This project is one of new lines in computer science that use graphics techniques to study internal computer architecture. Computer-assisted instruction (CAI) is used as a method to educate in the area of internal computer design. In the literature, various studies of either graphics or the logical design of digital computers are found, but both have not been combined as an individual work, structured for teaching. Chapter two includes an overview of literature and a literary comparison between other studies and this study.

In chapter three the basic design of the electronic elements of the NOT, OR, AND, NOR, and NAND gates is illustrated.

In chapter four the K-map minimization technique and combinational logic are explained.

In chapter five the application of some combinational circuits in a computer is shown.
In chapter six sequential logic is presented and various designs of flip-flops are shown.

In chapter seven it is demonstrated how a certain flip-flop is constructed by using other types of flip-flops.

In chapter eight the functions and uses of shift registers and counters are discussed.

In chapter nine the basic construction of integrated and magnetic memories in computers is described.

The conclusion of this research and recommendations are made in the final chapter.
CHAPTER II

GENERAL VIEW

Background

In general, this thesis deals with two main subjects in logical design which can be graphically presented on an Apple II system. The first subject is combinational circuits, which consist of circuits whose outputs are determined at any time directly from the present combination of inputs without regard to previous inputs.

A Boolean expression is recursively defined as follows:
1. A constant or a variable is a Boolean expression.
2. If $X$, $Y$ are Boolean expressions then $(X \text{ AND } Y)$, $(X \text{ OR } Y)$, $(\text{NOT } X)$ are also Boolean expressions.

A Boolean function is a function which can be represented as a Boolean expression.

The terms "high" and "low" are used to describe the voltage level in relation to the operators OR, AND, and NOT. When the value of the voltage in an operator is high enough to carry information, the current in the operator is represented by the binary number one, and the voltage is said to be "high". When the value of the voltage is not high enough to carry information, the current in the operator is represented by the binary number zero, and the voltage is said to be "low." The logic "True" is associated with high voltage, while the logic "False" is associated with low voltage.

The gate NOT is called an inverter because the output of this gate
is the complement of its input. The OR gate's output is true if at least one of its inputs is true, otherwise the output is false. The AND gate's output is true if all of its inputs are true, otherwise the output is false.

Other operations which can be expressed in terms of NOT, OR, and AND operations are also used. Three such operations which are implemented as gates are NOR, NAND and XOR. The NOR gate is the complement of the OR gate, i.e., the output of the NOR gate is true if and only if both of its inputs are false. The NAND gate is the complement of the AND gate, i.e., the output of the NAND gate is true if and only if at least one of its inputs is false. The Exclusive-OR, XOR, gate's output is false if and only if both of its inputs are either true or false.

The second subject of this thesis is the investigation of sequential circuits. The sequential circuits employ memory elements (binary cells) in addition to logic gates. Their outputs are functions of the inputs and of the state of the memory elements. The state of the memory elements, in turn, is a function of previous inputs. As a consequence, the output of a sequential circuit depends not only on present inputs, but also on the past input-output sequence.

A flip-flop is a type of sequential circuit used as a storage unit in some memory devices. It has the capability to retain both the last output and the present input, then use these to generate the new output. The concept of feedback is used in its design because the last output becomes one of the new inputs by being fed back into the input line.
One application of sequential circuits is a shift register. A shifter is used to shift bits in a register to the right or to the left. A bit is the binary representation of logic-1 or logic-0. A register is a collection of Master-Slave flip-flops and a storage location for binary information. A Master-Slave is constructed from two separate flip-flops such that one circuit serves as a master which receives information from the inputs when a control switch is on. The other circuit serves as a slave which takes the information stored in the master circuit and stores it in its circuit when the control switch is off.

Another application of sequential circuits is a counter, which is used to count the number of pulses entering a circuit.

A memory unit is a collection of storage registers and the associated circuits needed to transfer information in and out of the registers. Some types of memories are: ROM (Read-Only Memory) and RAM (Random-Access Memory). ROM allows only reading information from the memory. RAM allows reading from and writing to storage registers.

Overview of Literature

Review of Computer Electronics

The first electronic digital computer, which was completed in 1946, was a collection of thousands of vacuum tubes, registers, capacitors, and switches. In 1947 three scientists invented a tiny, simple device called the transistor, which was much smaller than vacuum tubes, worked faster and had fewer failures. In the late 1950's, Jack
Kilby of Texas Instruments and Robert Noyce of Fairchild Semiconductor almost simultaneously realized that transistors could be etched directly onto a single piece of silicon along with the connections between them. Such integrated circuits (ICs) can contain entire sections of a computer which led to the development of the microchip. In 1971, the engineer Ted Hoff designed a microprocessor which contained the entire CPU of a simple computer on one chip. Soon thereafter the personal computer was developed.

Review of Graphics

In 1950 the first computer, attached to MIT's Whirlwind I Computer, was used to generate simple pictures. These displays made use of a cathode-ray tube (CRT) like those used in television sets. Several years earlier, a CRT had been used by the late F. Williams as an information storage device.

During the 1950's, interactive computer graphics made little progress, because the computers of that time were not suitable for interactive use. An important event that promoted the development of computer graphics was the publication of a brilliant thesis by Evin E. Sutherland in 1962. His project, entitled Sketchpad: A Man-Machine Graphical Communication System, proved to many people that interactive computer graphics was a practical, useful, and exciting field of research.

By the mid 1960's, large computer graphics research projects were under way at MIT, General Motors, and Bell Telephone Laboratories.
The 1970's were the decade in which computer graphics research began to gain successful progress. Interactive graphics displays are now in use in many countries and are widely used for educational purposes. 1

Computers in Education

The recent development of the combination of the microcomputer and video media has provided a communication tool between the interactive video and user to make vivid information responsive to the needs of the receiver. Many studies have been made on classroom students in efforts to provide better opportunities for learning. A brief account of some samples and conclusions of these studies are quoted from literature to show the impact of CAI in education.

(1) The following questionnaire was presented to teachers:
When was the last time that you found it necessary to:
- discourage students from coming to school too early?
- limit the amount of time that students spend on a learning activity?
- remind students that lunch period was necessary?
- tell students that it is time to go home? 2

(2) Evidence is mounting which supports the conclusions that CAI can be a cost-effective alternative, as well as an effective supplement, to more traditional modes of instruction. In spite of this evidence, faculty in higher education institutions have been reluctant to make significant moves toward its use. Some of the
socio-economic reasons for this resistance to change have been explored, and based on a case study concerning the unsuccessful implementation of a television-based technology in a university, recommendations have been drawn for successful change models in higher education. 3

(3) During the last two decades computer-based learning systems have been developed and used in educational settings ranging from preschool classrooms (Atkinson & Fletcher, 1972) to graduate schools (Kearsley, Romaniuk, & Rowand, 1978).

While much of the available literature is anecdotal or descriptive in nature, there are a number of empirical studies dealing with the absolute or relative effectiveness of computer-assisted learning (Roe and Aiken, 1976). For example, CAI simulation was found to be as good or better than traditional teaching methods in a course designed to teach students to administer an informal reading inventory. A similar study also reported that computer simulation was superior to traditional methods in some aspects of teaching students to administer an informal reading test (Henry & Boysen, 1979). Other papers have described successful applications of computer-based teaching systems in courses for educational administrators. 4

Researchers are still working with microcomputers to design CAI packages for elementary and high schools to provide easier education in mathematics, chemistry, and physics using the BASIC or LOGO graphics packages on different kinds of microcomputers. The CAI packages of this project are designed for college students to provide knowledge
about the electronic design of gates and the design of combinational and sequential logic circuits using the Pascal graphics package on an Apple II system.
CHAPTER III

THE BASIC ELECTRONIC CONSTRUCTION OF GATES

This chapter discusses the basic electronic elements, their functions, and electronic circuits of gates. Part 1 explains the electronic elements resistor, diode, and transistor, as well as the electronic circuits of the gates NOT, OR, AND, NOR, and NAND. The main family used in the construction of the gates is the Resistor Transistor Logic (RTL). Part 2 deals with the graphical construction of the electronic circuits and logic gates on an Apple II system.

Part 1

The electronic elements, or elementary circuits, resistor, diode, and transistor can be considered as basic functional blocks of hardware that produce a logic-1 or logic-0 output signal, if input logic requirements are satisfied. The elementary circuits can be combined to form electronic circuits, often called gates, which function as a complete set of logic operations.

Resistor

A resistor resists the current flow in electronic circuits and is used to control the current value. Figure 1 shows the symbol of a resistor.

10
A diode, which is widely used in electronic circuits, allows current to flow through itself in one direction only. If the diode is reversed, keeping the current direction fixed, no current can flow through it. Or, if the current direction is reversed, keeping the diode fixed, no current can flow through it.

Figure 2. Current passing through the diode

Figure 3. No current passing through the diode

Note that in Figure 2 the current is allowed to pass through the diode and creates high voltage on the output side, and that in Figure 3 the
current is blocked.

**Transistor**

A transistor is made of a semiconducting material and acts as a switch. It consists of a collector, a base, and an emitter. (See Figure 4a.)

![Diagram of Transistor](image)

Figure 4. The transistor: (a) schematic, (b) and (c) the switch equivalents.

Whenever current \( I_b \) flows into the base of the transistor, a short results between the collector and the emitter, which permits a current \( I_c \) to flow from the collector, limited only by the circuit external to the transistor. This is illustrated by the switch equivalent circuit shown in Figure 4b.

Figure 4c shows the resulting pattern of current flow in the transistor when current from the base is zero or negative. Under either of these conditions the circuit path between the collector and emitter is broken and the current \( I_c \) from the collector is zero.
Not Gate

The NOT gate is sometimes called an inverter circuit because it inverts a binary signal. It consists of one resistor and one transistor. There is one input to the circuit and one output which responds to the input signals. The graphic representation of the NOT gate is shown in Figure 5.

\[
\text{High} \quad \rightarrow \quad \text{Low}
\]

Figure 5. Not gate symbol

Or Gate

The function of the OR gate is to allow information to pass through it if at least one of its inputs carries information (logic-1) and to show that the output is of logic-0 whenever none of its inputs is of logic-1. The circuit consists of one resistor and three diodes. There are three inputs to the circuit and one output. The symbol of the OR gate is shown in Figure 6.

\[
\begin{align*}
\text{A} & \quad \rightarrow \quad \text{A} + \text{B} + \text{C} \\
\text{B} & \\
\text{C}
\end{align*}
\]

Figure 6. Or gate symbol
And Gate

The function of the AND gate is to allow information to transfer from input lines to output lines if and only if all of its inputs carry information (logic-1). The circuit has the same electronic elements as the OR gate. The graphic symbol of the AND gate is shown in Figure 7.

![Figure 7. And gate symbol](image)

Nor Gate

The NOR gate's function is to act as the negation of the OR gate. The output of this gate is true if and only if all of its inputs are false. The circuit consists of three transistors and one resistor. There are three inputs to the circuit and one output. Figure 8 shows the symbol of the NOR gate.

![Figure 8. Nor gate symbol](image)
Nand Gate

The NAND gate's function is to act as the negation of the AND gate. The output of this gate is true if at least one of its inputs is false. The circuit contains the same elements as the NOR gate. The NAND gate symbol is shown in Figure 9.

\[
\begin{array}{c}
A \\
B \\
C \\
\end{array} \quad (A \cdot B \cdot C)'
\]

Figure 9. Nand gate symbol

Part 2

This part explains the procedures used on an Apple system to display the design of the circuits which are described in part 1.

The first package of this project is called PACKAGE1 and is built on face number 1 of the disk. (Refer to appendix A to learn how to access the machine.) When the user executes this package, the following appears on the screen.
THE FOLLOWING LIST SHOWS THE PROCEDURE NUMBER CORRESPONDING TO EACH OF THE NAMED ITEMS. INSERT THE PROCEDURE NUMBER WHICH YOU WANT TO ACCESS.

1: DIODE
2: TRANSISTOR
3: NOT GATE
4: OR GATE
5: AND GATE
6: NOR GATE
7: NAND GATE
8: HELP
9: EXIT

It is useful if the user executes procedure number 1 first and then number 2 to display the diode and then the transistor. These procedures should familiarize the user with the functions of the diode and the transistor, making it easier to understand the function of the basic elements of the gates.

"1: Diode"

When the user selects procedure number 1, the screen will display an orange symbol of a diode with two green wires connected to it. The screen will also show that the diode in a certain position allows the current to pass through it, and that the diode in the reversed position does not allow the current to flow.

"2: Transistor"

If the user selects procedure number 2, a transistor circuit will appear on the screen. It consists of a green transistor symbol and three orange wires. The display will show the direction of the current.
and that the output of the circuit is low when the input is high.

"3: Not Gate"

If the user selects procedure number 3, the screen will display a NOT gate circuit, which uses a green transistor and a violet resistor in its design. Blue wires are used to connect the elements of the circuit. It will also show that if the input carries high value, the transistor is open, and the current of the base flows through the transistor, causing more voltage than if it is cutoff. The voltage of the output drops between the resistor and the earth, causing the output of the circuit to be low. An orange symbol of the NOT gate will appear in the lower, right corner of the screen.

"4: Or Gate"

Procedure number 4 will display the electronic design of an OR gate. The circuit consists of three orange diodes, a violet resistor, and green wires to connect the elements of the circuit. A green OR gate symbol will show in the lower, right corner of the screen.

"5: And Gate"

When procedure number 5 is chosen, the screen will display the electronic circuit of an AND gate which has the same elements as the OR gate. A green symbol of the AND gate will also show in the lower, right corner of the screen.
"6: Nor Gate"

Procedure number 6 will display the electronic circuit of a NOR gate and its green symbol. It consists of 3 violet transistors, a violet resistor, and green wires to connect the elements of the circuit.

"7: Nand Gate"

Procedure number 7 will display a NAND gate circuit and its green symbol. The circuit consists of one blue, one green, and one violet transistor, a violet resistor, and green wires to connect the elements of the circuit.

"8: Help"

This procedure will help the user to understand the concepts of package 1 and how to select the other procedures.

"9: Exit"

Procedure number 9 will allow the user to exit from package 1.
CHAPTER IV

KARNAUGH MAP (K-MAP) AND COMBINATIONAL CIRCUITS

The purpose of this chapter is to define the Karnaugh maps and to explain how they are used to minimize Boolean expressions (part 1). The minimum expressions are then used to design minimum complexity combinational circuits. Apple II graphics are used to illustrate the use of Karnaugh maps and the design of combinational circuits (part 2).

Part 1

K-map

E. Veitch first introduced the representation of Boolean functions in a diagram known as the Veitch diagram. Karnaugh improved the representation introduced by Veitch and the new representation is called the Karnaugh map.

An illustration that may be used to visualize the relationships among the variables of a Boolean expression is the Venn diagram. This diagram consists of subsets of space inscribed in a rectangle such as the one shown in Figure 10.
A binary variable may appear either in its normal form (X) or in its complement form (X'). Consider two binary variables X and Y combined with an AND operation. Since each variable may appear in either its normal or complement form, there are four possible combinations: X'Y', X'Y, XY', and XY. Each of these four AND terms represents one of the distinct areas in the Venn diagram of Figure 10 and is called a minterm.

A two-value Boolean function can be expressed in a unique form called the canonical form. The canonical form consists of the logical sum of all minterms of the function. If a function contains n variables then the number of its minterms is 2**n. Each minterm has a 0 or 1 coefficient associated with it.

The K-map is a graphical representation of a Boolean function obtained directly from the function's canonical form. The K-map for a n-value Boolean function is a square with dimension 2**(n/2). Thus the K-map can be divided into 2**(n/2) * 2**(n/2) = 2**(n/2+n/2) = 2**(2*n/2) = 2**n squares of n-dimension. Each square corresponds to a minterm of the function. The squares are labeled accordingly so they represent one and only one minterm. The labels of neighboring squares
must be adjacent i.e., they must differ by only one variable which is complemented in the one and uncomplemented in the other. The minterms $XYZW$ and $XYZW'$ of a four-variable function are adjacent. The coefficients of the minterms are inserted into the squares. Thus, each square contains either 0 or 1 depending on the coefficient of the corresponding minterm. Figure 11 illustrates a K-map representation of four-variable Boolean functions. Each square contains the corresponding minterm.

![Figure 11. Labeling system of four-variable k-map](a) (b)

The map in Figure 11a shows a label for each of the corresponding minterm in Figure 11b. The map in Figure 11b shows the relationship between the four variables. The columns and rows are numbered in a reflected-code sequence, with only one digit changing in value between two adjacent rows or columns. The minterm corresponding to each square can be obtained from the concatenation of the row number with the column number. For example, the numbers of the fourth column (10) and the second row (01), when concatenated, give the binary number 1001,
the binary equivalent of decimal 9. Thus, the square in the fourth column and second row represents the minterm M9. Outer rows are adjacent to each other and outer columns are adjacent to each other. The combination of adjacent squares that is useful during the simplification process is easily determined by searching the four-variable map.

The rules for finding covers for squares in a four-variable K-map are:

(1) One square represents one minterm, giving a term of four literals (variable or its complement). An example is the M1 minterm which represents $A'B'C'D$. (See Figures 11a and 11b.) There is no minimization for covering a square which has no adjacent squares to be covered. The number of squares in the row or column to be covered simultaneously must be one or an even number.

(2) Two adjacent squares (combined) represent a term of three literals. For example, minimizing M1 and M3 gives $A'B'D$. (See Figure 12.) Note that two horizontally adjacent squares or two vertically adjacent squares may be covered simultaneously, but two diagonally adjacent squares may not be covered simultaneously.

Figure 12. Minimizing two adjacent squares
(3) Four adjacent squares represent a term of two literals. For example, minimizing M0, M1, M12, and M8 gives C'D'. Note that four adjacent squares can be covered as horizontally linear, vertically linear, or as a square form, but diagonally linear squares may not be covered simultaneously.

(4) Eight adjacent squares represent a term of one literal. For example, minimizing M0, M1, M2, M3, M4, M5, M6, and M7 gives A'. Note that the minimized cover is over two adjacent rows of four variables or two adjacent columns of four variables.

(5) Sixteen adjacent squares represent the function equal to 1. All 16 squares may be covered simultaneously.

A K-map can be used to easily minimize Boolean functions. An example of the simplification of a Boolean function is:

\[
F(A, B, C, D) = \{0,1,2,3,4,5,6,7,8,9\} = A'B'C'D' + A'B'C'D + A'B'CD' + \\
A'B'CD + A'BC'D' + A'BC'D + A'BCD' + \\
A'BCD + AB'C'D' + AB'C'D
\]

Since the function involves four variables, a four-variable map must be used. The minterms listed in the sum are designated by 1's in the map. See Figure 13.
Eight adjacent squares containing 1's can be combined to form the one-literal term $A'$. The remaining two 1's on the right can be combined with the two minterms on the left to form the two-literal term $B'C'$. The simplified function is: $F = A' + B'C'$. Note that sometimes there are more than one possible covering for the same function.

Combinational Circuits

A combinational circuit consists of input variables, logic gates, and output variables. The logic gates accept signals as inputs and generate signals as outputs. This process transforms binary information from the given input data to the output data. The circuit does not provide feedback; the output of an element can never serve as input to the same element or to a previous element in the sequence of the circuit.

An example of a combinational circuit is illustrated by the diagram in Figure 14. The wires in the circuit are channels for the flow of electronic signals between gates. They may be regarded as
graphic indications of the flow of information in the process. The logic elements operate the computing processes which change the input data to the output information.

\[ X_1 \quad \text{AND} \quad X_2 \quad \text{OR} \quad Y \]

Figure 14. Example of a combinational circuit

The circuit receives inputs (X1 and X2) simultaneously. Several inputs are distinguishable from each other by their different labels (for example, X1, X2, X3, etc.) and enter the circuit on different input lines. The values of the inputs are either 0 or 1, which may be interpreted respectively as "no current flowing through" or "current flowing through", "low voltage" or "high voltage", and as the absence or presence of a pulse. (See chapter III, part 1.) The signals flow through wires and are manipulated by the logic elements in a variety of ways. Each element receives one or more signals as inputs and produces one or more signals as outputs, each of which is coded as 0 or 1. These outputs may serve as inputs to other elements in the circuit. They may be used to drive mechanical devices, to cause the printing of results, or to turn lights on or off, etc.

In the circuit illustrated in Figure 14, if the input signals are X1=1 and X2=0, then the output signals of the AND element and the NOT

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
element should be 0 and 1 respectively, and will serve as inputs to the OR element. Thus the output of the entire circuit is \( Y = 1 \). Table 1 shows the transformations of all combinations of input signals to output signals of the circuit in Figure 14. Even without a detailed explanation of each component, the reader may understand the operation of the circuit from the Table.

**TABLE 1**

**INPUT-OUTPUT FOR FIGURE 14**

<table>
<thead>
<tr>
<th>INPUT</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>X1</td>
<td>X1 AND X2</td>
</tr>
<tr>
<td>-------</td>
<td>-----------</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The logic of combinational circuits focuses on the following two facts:

(1) In a combinational circuit, the flow of electronic signals is one-way; it starts from the input lines and proceeds through the circuit in one direction toward the output lines.

(2) Because of this one-way characteristic of a combinational circuit, it is able to make use of only the present inputs to the circuit, putting a severe limitation on the scope of tasks it can perform.
This part corresponds to the graphic representation of minimizing eight Boolean functions on an Apple system. The eight functions are stored in a data file called DATF.TEXT. (See Appendix C on data file.) A "Don't Care" function is provided for each of the eight Boolean functions. In the design of logical circuits one often encounters cases in which the switching function is not completely specified. In other words, a function may be required to contain certain minterms or omit certain minterms, with the remaining minterms being optional. That is, certain minterms may be included in the minimizing of the logic design if they help simplify the logic circuit. A minterm which is optional (1 or 0) is called a Don't Care minterm.

The package relating to the contents of this chapter is stored on face number 1 of the disk and is called PACKAGE2. When the user accesses this package (see Appendix A), the first function of the data file and the list of main package procedures will appear on the screen as follows.
THE PRESENT FUNCTION WITHOUT DON'T CARE
F = 4 5 8 12 13 14 15

INSERT THE PROCEDURE NUMBER.
1 : DISPLAY MAP
2 : SELFCOVER
3 : SUGGESTED SOLUTION
4 : THE COMBINATIONAL CIRCUIT
5 : HELP
6 : EXIT

To execute one of the above main procedures, the number of the procedure desired should be inserted. When the user finishes executing a procedure, the list of main package procedures and the first function without don't care will reappear on the screen. If the user inserts number 6, for example, the following message will appear:
"DO YOU WANT DON'T CARE WITH THE CURRENT FUNCTION ? (Y/N)"
Upon typing the letter Y, the screen will display the following.
THE PRESENT FUNCTION WITH DON'T CARE

D = 9 11

INSERT THE PROCEDURE NUMBER.

1: DISPLAY MAP
2: SELFCOVER
3: SUGGESTED SOLUTION
4: THE COMBINATIONAL CIRCUIT
5: HELP
6: EXIT

Now the user can apply the present function \( f(x, y, z, w) = (4, 5, 8, 12, 13, 14, 15) \) with DON'T CARE function \( f(x, y, z, w) = (9, 11) \) to the above main package procedures. But if the user inserts the letter N in response to the message "DO YOU WANT DON'T CARE WITH THE CURRENT FUNCTION ? (Y/N)," the screen will show another message, "DO YOU WANT NEW FUNCTION ? (Y/N)." If the user types the letter Y, the next function \( f(x, y, z, w) = (5, 6, 7, 13) \) in the data file and the main procedures will appear on the screen. This action will make the next function \( f(x, y, z, w) = (5, 6, 7, 13) \) available for minimizing. The system will recognize the function just read from the data file as the present function. But if the letter N is the response to the message (DO YOU WANT NEW FUNCTION ? (Y/N),) the user will exit the package.
"1 : Display Map"

If the user selects number 1 from the main procedures of the package, the screen will display a picture of a K-map of four-variables and show the mapping of the present function without covering the minterms.

"2 : Selfcover"

This procedure will show the user how to cover the present function which is already mapped in the K-map. To execute the SELFCOVER procedure, number 2 must be inserted. The screen will display the mapping of the present function in a K-map. A message will also appear on the screen:

HOW MANY SQUARES DO YOU WANT TO COVER?
INSERT(1, 2, 4, 8, H(HELP), OR E(XIT)).

Each digit refers to the number of squares to be covered in the K-map. The character H refers to the Help option for procedure number 2. The character E allows the user to exit from the SELFCOVER procedure. If the user inserts one of the integers (1, 2, 4, or 8), the K-map for the present function will remain on the screen and another message will appear:

WHICH SQUARES DO YOU WANT TO COVER?
INSERT TERMS IN ASCENDING ORDER
(SUCH AS 1, 2, ...).

The K-map is divided into 16 squares, each understood to be labelled with a number from 0 to 15. See Figure 15 which shows this labelling that is also available to the user in the HELP option.
To respond to the last message (WHICH SQUARES DO YOU WANT TO COVER?), the user should insert the squares' numbers based on the numbering system of Figure 15. Note that the user must insert commas between the numbers when answering the second message and the numbers should be inserted in ascending order. A carriage return is needed when the user finishes the insertion.

Example 1: Use procedure number 2 to cover the squares 4, 5, 12, 13 of the present function \( F = (4, 5, 8, 12, 13, 14, 15) \).

Solution: Number 2 from the main package procedures should be inserted, and a message and the K-map will appear as follows.
HOW MANY SQUARES DO YOU WANT TO COVER?
INSERT (1, 2, 4, 8, H(ELP), OR E(XIT)).

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Insert number 4 followed by a carriage return. The next message will appear:

WHICH SQUARES DO YOU WANT TO COVER?
INSERT TERMS IN ASCENDING ORDER
(SUCH AS 1,2,...).

The input to the last message should be 4,5,12,13 followed by a carriage return. The picture will show the drawing of a square around the minterms 4, 5, 12, 13, and the first message (HOW MANY SQUARES DO YOU WANT TO COVER? INSERT (1, 2, 4, 8, H(ELP), OR E(XIT)) will reappear.

Figure 16 shows the state of the K-map after the covering.

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 16. Covering solution for example 1
"3 : Suggested Solution"

Procedure number 3 will display the K-map with the suggested solution for covering the present function.

"4 : The Combinational Circuit"

Procedure number 4 will display the design of the combinational circuit for the present function using the gates NOT, OR, and AND and the colors green, violet, blue, and/or orange in the construction.

"5 : Help"

The fifth procedure will help the user to know the purpose of this package and how to respond to the messages in the package.

"6 : Exit"

This command will let the user exit from the main package procedures.
CHAPTER V

APPLICATION OF COMBINATIONAL CIRCUITS

In this chapter, six combinational circuits are discussed: BINARY-FULL ADDERS, LOOK-AHEAD CARRY GENERATOR, FOUR-BIT FULL-adders WITH LOOK-AHEAD CARRY GENERATOR, DECODER, MULTIPLEXER, and READ-ONLY MEMORY. Part 1 discusses the detailed function of each device. Part 2 corresponds to the constructions of the systems which are designed on an Apple system.

Part 1

Binary Adder

One of the simplest digital arithmetic systems is the one bit binary adder, which adds two single digit binary numbers and is sometimes referred to as a half-adder. The system has two outputs, since the result of adding two single binary numbers is a two-digit binary number (the sum digit and the carry digit).

Table 2 shows the possible results of adding two binary numbers X and Y and Table 3 shows the truth table for a system to implement this operation. The two outputs are the Carry (C) and the Sum (S) digits.
TABLE 2

ADDITION OF TWO BINARY NUMBERS.

<table>
<thead>
<tr>
<th>INPUT</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Y</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

TABLE 3

TRUTH TABLE FOR HALF-ADDER.

<table>
<thead>
<tr>
<th>INPUT</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Y</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

From Table 3 it follows that C and S outputs are described in terms of the inputs X and Y as follows.

\[ C = X \cdot Y \]
\[ S = X' \cdot Y + X \cdot Y' \]

Various implementations of a half-adder can be designed from the above two functions. Figure 17 shows a half adder.
If two numbers, each containing more than one digit, are to be added, the system for adding the two last significant digits can be exactly as described above. For the other significant digits, however, the possibility of a carry digit appearing from the previous stage of the calculation must be taken into account. This means that a three-input, two-output, system is required. The inputs consist of the two digits to be added together with the carry digit from the previous stage of addition. The truth table for such a system is given in Table 4. The three-input, two-output system is known as a full-adder.
TABLE 4

TRUTH TABLE FOR FULL-ADDER.

<table>
<thead>
<tr>
<th>INPUT</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ai</td>
<td>Bi</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The operation of the full-adder is described in terms of Boolean functions as follows.

\[ S_i = A_i \cdot B_i \cdot C_i + A_i \cdot B_i \cdot C_i' + A_i \cdot B_i' \cdot C_i + A_i \cdot B_i' \cdot C_i' \]
\[ C_{i+1} = A_i \cdot B_i' \cdot C_i + A_i' \cdot B_i \cdot C_i + A_i \cdot B_i \cdot C_i' + A_i \cdot B_i \cdot C_i' \]

A full-adder can be comprised of two half-adders and one OR gate. Figure 18 shows a full-adder circuit.
Look-Ahead Carry Generator

The addition of binary numbers in parallel implies that all the bits of the addend and the augend are available for computation at the same time. As in any combinational circuit, the signal must propagate through the gates before the correct output sum is available in the output terminals. The total propagation time is equal to the propagation delay time of a typical gate multiplied by the number of gate levels in the circuit. The longest propagation delay time in a parallel adder is the time it takes the carry to propagate through the full-adders.

Since each bit of the output sum depends on the value of the input carry, the value of the sum in any given stage $S_i$ in the adder will be in its steady-state final value only after the input carry to that stage has been propagated.

There are many techniques for reducing the carry propagation time
in a parallel adder. The most widely used technique employs the principle of look-ahead carry.

Consider the full-adder circuit shown in Figure 18 of the previous section. If we define two new binary variables $G_i$ and $P_i$, where $G_i = X_i \cdot Y_i$ and $P_i = X_i \oplus Y_i$, the output carry and sum can be expressed as $C_{i+1} = G_i + P_i \cdot C_i$, $S_i = P_i \oplus C_i$. $G_i$ is called a carry generate and produces an input carry when both $X_i$ and $Y_i$ are equal to one, regardless of the input carry from the previous stage. $P_i$ is called a carry propagate because it is the term associated with the propagation of the carry from $C_i$ to $C_{i+1}$. The expression $P_i \oplus C_i$ describes the well known Exclusive-OR (XOR) operation. The Boolean function for each stage and substitute for each $C_i$ value from the previous equation ($C_{i+1} = G_i + P_i \cdot C_i$) are

\[
\begin{align*}
C_1 &= G_0 + P_0 \cdot C_0 \\
C_2 &= G_1 + P_1 \cdot C_1 \\
C_3 &= G_2 + P_2 \cdot C_2 = G_2 + P_2 \cdot (G_1 + P_1 \cdot C_1) \\
&= G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot C_1 \\
C_4 &= G_3 + P_3 \cdot C_3 \\
&= G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot C_1
\end{align*}
\]

Since the Boolean function for each output carry is the sum of products, each function can be implemented with one level of AND gates followed by an OR gate. The four Boolean functions for $C_1$, $C_2$, $C_3$, and $C_4$ are implemented in the look-ahead carry generator. $C_4$ does not need to wait for $C_3$ and $C_2$ to propagate, meaning that $C_4$ is propagated at the same time as $C_2$ and $C_3$. 

/
4-Bit Full-Adders with Look-Ahead Carry Generator

Up to this point, we can construct from the former description of the adder and the look-ahead carry generator a circuit consisting of 4-bit full-adders and a look-ahead carry generator. This device is used to add four binary bits. The look-ahead carry generator is used to avoid delay during the addition of two bits.

Decoder

An "n to 2**n" decoder is a multiple output combinational logic circuit that converts binary information from n input lines to a maximum of 2**n unique output lines. For each possible input condition, one and only one output signal will have logic 1. Therefore, the "n to 2**n" decoder may be considered simply as a minterm generator.

Multiplexer

A multiplexer is a device used to transmit a large number of information units over a smaller number of lines. It is a combinational circuit that selects binary information from many input lines and directs it to a single output line. Specific input is selected by a set of selectors. Usually there are 2**n input lines and n selection lines whose bit combinations determine which input is chosen.
ROM is a small piece of memory that cannot be erased either by reading it or by switching off the current. As we know, a decoder generates $2^n$ minterms from $n$ input variables. By inserting OR gates to sum these minterms of Boolean functions, it is possible to design any desired combinational circuit. ROM includes both a decoder and OR gates within a single integrated circuit (IC) package. The connections between the inputs of the decoder and the inputs of the OR gates can be specified for each particular configuration by programming the ROM.

ROM is used very often to eliminate all interconnecting wires by having a complex combinational circuit in one IC package. A familiar example of ROM is its use in video games, where the user does not write into the memory, but only reads from the permanently stored program in the memory.

Part 2

In part 1 the basic concepts of the six systems are presented. This part corresponds to the graphical implementation of the systems on an Apple system which is built on face number 1 of the disk and called PACKAGE3. When package 3 is accessed (see appendix A), a message and a list of eight procedures will appear on the screen:
THE FOLLOWING IS THE NUMBERING SYSTEM FOR
THE PROCEDURES IN PACKAGE 3. TYPE IN THE
NUMBER OF THE PROCEDURE WHICH YOU WOULD
LIKE TO EXERCISE.

1: FULL-ADDER
2: LOOK-AHEAD CARRY GENERATOR
3: 4-BIT FULL-ADDERS WITH LOOK-AHEAD
   CARRY GENERATOR
4: 3-TO-8 LINE DECODER
5: 4-TO-1 LINE MULTIPLEXER
6: 4X2 READ-ONLY MEMORY
7: HELP
8: EXIT

Selection of one of the six systems or help or exit procedures is
made by inserting its number. When the number of a system is inserted,
the screen will display the construction of the system. A carriage
return is needed after each display to return to the list of
procedures.

"1: Full-Adder"

If procedure number 1 is selected, the design of a full-adder
device will be displayed on the screen. (See Figure 18.) The circuit
is comprised of two half-adders and one orange OR gate. Each
half-adder consists of one orange XOR gate and one violet AND gate.
Green wires are used to connect the elements of the system. The input
to the first half is represented by Xi and Yi, the two binary bits to
be added. The outputs from the first half are Pi and Gi, Pi being the
output of the XOR gate and Gi being the output of the AND gate. These
outputs Pi and Gi become the inputs of the second half of the adder.
Pi is the input of gates XOR and AND of the second half, as is Gi which
represents carry input. The output of XOR of the second half is $S_i$ which represents the sum. The output of the AND gate in the second half is the input to OR gate. Both inputs to OR gate will generate $C_{i+1}$ output which represents the carry bit of adding the two bits $X_i$ and $Y_i$.

"2: Look-Ahead Carry Generator"

The second procedure of package 3 is a look-ahead carry circuit. It consists of six violet AND gates, three orange OR gates, and many green wires. The circuit has seven inputs ($P_3, G_3, P_2, G_2, P_1, G_1, \text{ and } C_1$) and three outputs ($C_4, C_3, \text{ and } C_2$). This device is one of several techniques for reducing the carry propagation time in a parallel adder. (See part 1 for more detail.)

"3: 4-Bit Full-Adders with Look-Ahead Carry Generator"

The system shown in this procedure is constructed from four full-adders and a look-ahead carry generator. A blue drawing of the look-ahead device separates the four half-adders and four orange XOR gates. The four half-adders sit above the carry generator. Each half-adder consists of one orange XOR gate and one violet AND gate. The input pairs of the four half-adders are: $B_4, A_4; B_3, A_3; B_2, A_2; \text{ and } B_1, A_1$. The four output pairs are $P_4, G_4; P_3, G_3; P_2, G_2; \text{ and } P_1, G_1$ which along with the carry $C_1$ go through the look-ahead carry generator as input. The outputs of the look-ahead are $C_2, C_3, C_4, \text{ and } C_5$. Four XOR
gates are located below the look-ahead to generate the final sums S1, S2, S3, and S4. The input pairs to the last gates are: C1,P1; C2,P2; C3,P3; and C4,P4. The final carry from the whole device is C5. Note that the addition is in parallel. C4 does not have to wait for C2 and C3 to propagate; C4 is propagated at the same time as C2 and C3.

"4: 3-to-8 Line Decoder"

A decoder of three to eight lines will be displayed if procedure number 4 is selected. Three inputs (X, Y, and Z) are decoded into eight outputs (D0, D1,... D7), each output representing one of the minterms of the three input variables. Three blue inverters provide the complement of the inputs, and eight violet AND gates generate the minterms. Green wires are used to connect the components of the system.

"5: 4-to-1 Line Multiplexer"

Selecting procedure number 5 will result in the display of a four to one line multiplexer. The term "four to one" means that one of four inputs is chosen under the control of selection lines and serves as the input to the multiplexer which is yielded as the output. This circuit consists of two blue inverters which are used to select a particular input, four violet AND gates, one orange OR gate, and several green wires to connect the logic elements. The input lines to the circuit are I1, I2, I3, and I4 which are controlled by the selection lines S1.
and S2. All of the input lines and S1 and S2 go directly as input to the AND gates. The control lines S1 and S2 act according to Table 5.

<table>
<thead>
<tr>
<th>INPUT</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>S2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The unique output of the circuit is Y. The outputs of the four AND gates go directly as input to the OR gate. This last gate generates the output Y. I1 is selected as Y if S1=0 and S2=0. The selection of other values of Y are shown in Table 5.

"4X2 Read-Only Memory"

The last device in this package is the ROM which has a very simple construction. It consists of two inputs (A and B), a green 2X4 decoder, two orange OR gates, and many green wires to link the logic elements of the circuit. The inputs A and B are decoded to four addresses (minterms). The four outputs of the decoder are connected through links to each OR gate. Table 6 is the truth table which specifies the combinational circuit with two inputs and two outputs.
TABLE 6

TRUTH TABLE FOR 4X2 ROM.

<table>
<thead>
<tr>
<th>INPUT</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

"7: Help"

This procedure assists the user in learning how to use package 3.

"8: Exit"

This procedure will allow the user to exit from the package.
CHAPTER VI

SEQUENTIAL CIRCUITS AND FLIP-FLOP TECHNIQUES

A sequential circuit consists of a combinational circuit to which memory elements are connected to form a feedback path. The memory elements (e.g., flip-flops) are devices capable of storing binary information within them. The binary information stored in the memory elements at any given time defines the state of the sequential circuit. The sequential circuit receives binary information from external inputs. These inputs, together with the present state of the memory elements, determine the binary value at the output lines.

A flip-flop is a circuit with two stable states, between which it can be flipped (or flopped) by means of an electronic pulse. The two states are designated by a 1 or 0 and, since they are both stable, they can be used to store a single bit indefinitely. The major differences among various types of flip-flops are in the number of inputs they process and the manner in which the inputs affect the binary state.

This chapter describes the flip-flops RS, RST, JK, T, D, and Master-Slave (part 1) and explains the differently designed flip-flops that have been graphically represented on an Apple II system (part 2).
Part 1

Basic Flip-Flop Circuit

A flip-flop (FF) circuit may be constructed with two NOR gates or two NAND gates. The logic diagrams and truth tables for the two constructions are shown in Figures 19 and 20, respectively.

<table>
<thead>
<tr>
<th>S</th>
<th>R</th>
<th>Q</th>
<th>Q'</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

(after $S=1$, $R=0$)

<table>
<thead>
<tr>
<th>S</th>
<th>R</th>
<th>Q</th>
<th>Q'</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

(after $S=0$, $R=1$)

Figure 19. Basic RS FF using Nor gates

<table>
<thead>
<tr>
<th>S</th>
<th>R</th>
<th>Q</th>
<th>Q'</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

(after $S=1$, $R=0$)

<table>
<thead>
<tr>
<th>S</th>
<th>R</th>
<th>Q</th>
<th>Q'</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

(after $S=0$, $R=1$)

Figure 20. Basic RS FF using Nand gates

Various designs of flip-flops depend on the basic combinations of flip-flops shown in 19 and 20. The input variables are $R$ (reset) and $S$ (set). The outputs are the present state $Q$ and the complement of $Q$. 

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
(Q'). The flip-flop in Figure 19 sets the storage to 1 if \( S=1 \) and \( R=0 \). It clears the flip-flop if \( S=0 \) and \( R=1 \). It stays in the previous state if \( S=0 \) and \( R=0 \). But if both \( S=1 \) and \( R=1 \), \( Q \) and \( Q' \) are not complemented as in the case of the other three conditions mentioned. This last condition must be avoided by not applying 1's to both inputs simultaneously since this will result in an unstable state. The truth table in Figure 19 shows all possible combinations of inputs and the resulting outputs of the flip-flop.

**Clocked RS Flip-Flop**

A clocked RS flip-flop can be constructed from one of the two basic flip-flop diagrams shown in Figures 19 and 20 with more gates added. The circuit in Figure 19 needs two AND gates and a clock pulse to form a clocked RS flip-flop. The circuit in Figure 20 needs two NAND gates and a clock pulse to form the clocked RS. If the circuit is constructed from the diagram in Figure 19, the storage unit will stay in the previous state as long as the clock pulse is at 0. If the clock turns to 1, the circuit will work according to the truth table in Figure 19. The circuit with NOR gates and the truth table are shown in Figure 21. \( Q(t+1) \) is the state of the flip-flop after the occurrence of a clock pulse (referred to as the next state).
RST Flip-Flop

Figure 22 shows an RST flip-flop and its truth table. The storage unit stays in the previous state if \( R=0 \), \( S=0 \), and \( T=0 \). A complement state occurs when the inputs are \( R=0 \), \( S=0 \), and \( T=1 \). The storage unit is set if \( R=0 \), \( S=1 \), and \( T=0 \). It is reset when the inputs are \( R=1 \), \( S=0 \), and \( T=0 \).

<table>
<thead>
<tr>
<th>R</th>
<th>S</th>
<th>T</th>
<th>Q(t+1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Q</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>1</td>
<td>D</td>
<td></td>
</tr>
</tbody>
</table>

Clocking JK Flip-Flop

The JK device acts like the RS flip-flop except under the
condition of $J=1$ and $K=1$, which switches the next state to its complement state. That is, if $Q=1$, it switches to $Q=0$ and vice versa. See Figure 23 below.

<table>
<thead>
<tr>
<th>J</th>
<th>K</th>
<th>$Q(t+1)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Q</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>$Q'$</td>
</tr>
</tbody>
</table>

Figure 23. JK flip-flop

Clocked T Flip-Flop

This is a single-input form of the JK flip-flop. If the input is 0, the next state stays in the previous state. It assumes the complement of the next state if the input is 1. The circuit is shown in Figure 24.

<table>
<thead>
<tr>
<th>$T$</th>
<th>$Q(t+1)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Q</td>
</tr>
<tr>
<td>1</td>
<td>$Q'$</td>
</tr>
</tbody>
</table>

Figure 24. Clocked T flip-flop

D Flip-Flop

This circuit is a modification of the clocked RS flip-flop shown
in Figure 20. Figure 25 shows the modified circuit.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>Q(t+1)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 25. Clocked D flip-flop

Gate 1 is used as an inverter to the input. Since the clock pulse input is at 0, the outputs of gates 2 and 3 are at 1, without regard to the input values. This means the basic flip-flop remains at level 1. When the clock pulse occurs, the D flip-flop input is selected. Suppose the input is 1, the output of gate 2 is at 0. This value switches the flip-flop to the set state. If the input is 0, the output of gate 3 goes to 0 and switches the flip-flop to the clear state.

Master-Slave JK Flip-Flop Technique

Figure 26 shows this system constructed with two RS flip-flops as master and slave, and an inverter to control the clock pulse. When the clock pulse C is 1, the information at external J and K inputs is transmitted to the master flip-flop (i.e., the data provided by the input lines is stored), keeping the slave flip-flop disabled as long as the pulse is at level 1, because the output of the inverter is 0. When the pulse becomes 0, the output of the inverter is 1. Since the clock pulse input of the slave is 1, it is enabled and the output Q
master) is equal to S2, while Q' (of the master) is equal to R2 (i.e., the inputs of the slave depend on the stored information of the master). At the same time, the master flip-flop is disabled because C=0.

![Master-slave JK flip-flop diagram]

Figure 26. Master-slave JK flip-flop

Part 2

This part corresponds to the graphic representation on an Apple system of the flip-flops discussed in part 1. Animation is used to show the dynamic movements of signals when they transfer between gates through wires.

Package 4 contains all programs which are designed for chapter VI. The name of the package on face number 2 of the disk is PACKAGE4. When package 4 is accessed (see appendix A), the screen will display eight
main procedures to demonstrate the functional input-output flip-flops.

The names and corresponding numbers of the procedures will appear on
the screen as follows.

INSERT THE NUMBER OF THE SELECTED FUNCTION.

1- RS FLIP-FLOP
2- RST FLIP-FLOP
3- JK FLIP-FLOP
4- T FLIP-FLOP
5- D FLIP-FLOP
6- MASTER-SLAVE FLIP-FLOP
7- HELP
8- EXIT

Binary inputs must be inserted when the user accesses one of the
first six procedures. For example, if RS flip-flop is chosen, the
message INSERT VALUES FOR R AND S IN BINARY will appear on the screen.
The inputs 00, 10, 01, or 11 can be inserted to respond to this last
message. Also, the system will display the result of each circuit of
procedures 1 through 5 on the screen according to their truth tables.

"1- RS Flip-Flop"

If procedure 1 is selected from the list of main procedures, a
clocked RS flip-flop will be displayed on the screen. It consists of
two blue AND gates, two violet NOR gates, an orange clock pulse, and
some green wires to connect the elements of the circuit.
"2- RST Flip-Flop"

Number 2 of the main procedures represents the RST flip-flop. The graphic circuit consists of four blue AND gates, two violet OR gates, two violet NOR gates and many green, white, and orange wires to connect the logic elements. The truth table of the circuit is shown in Figure 22.

"3- JK Flip-Flop"

Upon selecting procedure number 3, the screen will display the graphic circuit of the JK flip-flop, which is constructed from two blue AND gates, two violet NOR gates, an orange clock pulse, and green wires to connect the gates and the source of clock pulses. The circuit works according to the truth table shown in Figure 23.

"4- T Flip-Flop"

This device will be displayed if the user selects procedure 4. The circuit consists of two blue AND gates, two violet NOR gates, an orange clock pulse, and many green wires. The result of the circuit depends on the truth table shown in Figure 6.6.

"5- D Flip-Flop"

The fifth procedure is the D flip-flop, which consists of five blue NAND gates, an orange clock pulse, and many green wires to connect
The elements. The truth table for the system is shown in Figure 25.

"6- Master-Slave Flip-Flop"

The last logic design of the package is the master-slave JK flip-flop system. It consists of four blue AND gates, an orange master flip-flop, a violet slave flip-flop, a white inverter, and many green wires to connect the logic elements. This system will recognize the result of the last access. That is, if the user stores 1 or 0 in the flip-flop, the system will remember the stored data for the next access.

"7- Help"

This procedure will help the user understand the concepts of package 4 and know how to access it.

"8- Exit"

This procedure will allow the user to exit from the package.
CHAPTER VII

SIMULATION OF FLIP-FLOPS

In this chapter, various types of flip-flops are simulated using other types of flip-flops. The simulation is achieved by referring to an FF (flip-flop) table (appendix B), which shows the relationship among different types of FFs. The table also shows the equations of the next state for each type of FF in the form of $Q^{(N+1)} = G_1 Q + G_2 Q'$, where $G_1$ and $G_2$ are variables.

Part 1 of this chapter explains the simulation of each flip-flop (RS, RST, T, or D) using other types of flip-flops in terms of equations which are taken from the FF table. Also, an example of a circuit is given for each type which shows the technique of constructing a flip-flop from other flip-flops. Part 2 describes the simulation of these flip-flops on an Apple II system.

Part 1

Simulating JK FF using RS, RST, T, or D FF

The third row of the FF table shown in appendix B contains all the equations needed to simulate a JK FF using other kinds of FFs. The following equations are quoted from the FF table.
1- Simulate JK using RS FF: \( R = K \cdot Q \), \( S = J \cdot Q' \)
2- " " " RST FF: \( R = K \cdot J' \), \( S = K' \cdot J \), \( T = K \cdot J \)
3- " " " T FF: \( T = K \cdot Q + J \cdot Q' \)
4- " " " D FF: \( D = K' \cdot Q + J \cdot Q' \)

Example 2: Construct a JK FF circuit using an RS FF and the following equations: \( R = K \cdot Q \), \( S = J \cdot Q' \).

Solution: The simulation of a JK FF is constructed with an RS FF and two AND gates. The proper connections of the wires are shown in Figure 27. The function of the circuit behaves as a JK FF.

![Figure 27. Simulation of JK FF using RS FF](image)

Simulating RS FF using JK, RST, T, or D FF

The fourth row in the FF table contains the equations needed for constructing an RS FF using other types of FFs. The equations are:

1- Simulate RS using JK FF: \( J = R \), \( K = S \)
2- " " " RST FF: \( R = S \), \( S = R \), \( T = - \)
3- " " " T FF: \( T = R \cdot Q + S \cdot Q' \)
4- " " " D FF: \( D = S + R' \cdot Q \)

Example 3: Construct an RS FF circuit using a T FF and the following
equation: \( T = R \cdot Q + S \cdot Q' \).

Solution: an RS FF can be simulated from one T FF, one OR gate, and two AND gates. Figure 28 illustrates the proper connection of the wires in the circuit. The behavior of the circuit is similar to that of an RS FF.

![Figure 28. Simulation of RS FF using T FF](image)

Simulating RST FF using JK, RS, T, or D FF

The fifth row in the FF table contains the equations needed to simulate an RST FF using other types of FFs. The following are the equations from the fifth row of the table.

1- Simulate RST using JK FF: \( J = S + T, \ K = R + T \)
2- " " " RS " : \( R = (R+T) \cdot Q, \ S = (S+T) \cdot Q' \)
3- " " " T " : \( T = (R+T) \cdot Q + (S+T) \cdot Q' \)
4- " " " D " : \( D = S + T \cdot Q' + R' \cdot T \cdot Q \)

Example 4: Construct an RST FF circuit using a JK FF and the given equations: \( J = S + T, \ K = R + T \).

Solution: The circuit of a simulated RST FF using a JK FF consists of two OR gates and a JK FF. The circuit is shown in Figure 29 and
behaves as an RST FF.

![Figure 29. Simulation of RST FF using JK FF](image)

Simulating T FF using JK, RS, RST, or D FF

The sixth row in the FF table contains the equations needed to simulate a T FF using other types of FFs. The following equations are taken from the sixth row of the table.

1- Simulate T using JK FF: \( J = T, K = T \)
2- " " " RS FF: \( R = T \lor Q, S = T \lor Q' \)
3- " " " RST FF: \( R = \neg, S = \neg, T = T \)
4- " " " D FF: \( D = T' \lor Q + T \lor Q' \)

Example 5: Design a circuit for a T FF using a D FF and the given equation: \( D = T' \lor Q + T \lor Q' \).

Solution: The circuit consists of one XOR gate and a D FF. The circuit behaves as a T FF. Figure 30 shows the proper connection of the wires in the circuit.
Simulating D FF using JK, RS, RST, or T FF

The last row of the FF table contains the equations needed to simulate a D FF using other types of FFs. The following equations are used to simulate a D FF using other kinds of FFs:

1- Simulate D using JK FF: $J = D$, $K = D'$
2- " " " RS FF: $R = D'$, $S = D$
3- " " " RST FF: $R = D'$, $S = D$, $T = -$ 
4- " " " T FF: $T = D'Q + D\overline{Q}'$

Example 6: Build a D FF circuit using an RST FF and the following equations: $R = D'$, $S = D$, $T = -$.

Solution: A D FF circuit can be simulated by using one NOT gate and an RST FF. The circuit behaves as a D FF. Figure 31 shows the construction of the circuit.
Figure 31. Simulation of D FF using RST FF

Part 2

Graphic Representation

This part describes the graphical representations of package 5 which is accomplished on an Apple II system. The name of the package on face number 2 of the disk is PACKAGE5. Several colors are used in the drawings of the circuits: blue for AND gates, violet for OR gates, green or orange for NOT gates, and green or white for wires.

When package 5 is accessed (see appendix A), a message and a set of main procedures appears in the following form.

```
INSERT THE NUMBER OF THE FUNCTION TO BE SIMULATED.

1: SIMULATE JK USING RS, RST, T, OR D FF
2:   " RS " JK, " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " 
3:   " RST " " , RS ; " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " 
4:   " T " " " " " , RST, OR D FF
5:   " D " " " " " , " " " " " T FF
6: FOR HELP
7: FOR EXIT
```
A choice of one of the first five procedures leads to the following displays.

(1) The equation of FFs in the form of $Q^{*\times(N+1)} = G_1^*Q + G_2^*Q'$ will appear on the screen. The picture on the screen will be as follows.

EQUATION OF FFs IN THE FORM

$$Q^{N+1} = G_1^*Q + G_2^*Q$$

$$JK - Q^{N+1} = K^*Q + J^*Q$$

$$RS - Q^{N+1} = R^*Q + S^*Q, \quad G_1^*G_2^* = 0$$

$$RST - Q^{N+1} = R^*T^*Q + (S + T)^*Q$$

$$T - Q^{N+1} = T^*Q + T^*Q, \quad G_1^*G_2^* = G_1^*G_2^* = 0$$

$$D - Q^{N+1} = D^*Q + D^*Q, \quad G_1^*G_2^* = G_1^*G_2^* = 0$$

"PRESS RETURN"

(2) The second display will appear after the user presses the carriage return. It is a list of submain procedures which simulate the FF being chosen from the main procedures. Suppose the user has chosen the procedure 1 from the main procedures. The picture of the submain procedures will appear as follows.

CHOOSE A FF TO BE USED IN THE SIMULATION OF JK FF.

1: SIMULATE JK USING RS FF
2: " " " RST "
3: " " " T "
4: " " " D "
5: HELP
6: EXIT

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
(3) The third display will depend on the decision which has been made by the user to simulate JK using one of the other FFs. Suppose the user selects number 1 from the submain procedures. The screen will show the truth table for RS FF and the equations for RS FF, both of which are taken from the FF table in Appendix B. The picture on the screen will show the following information.

**SIMULATE FFS USING RS**

```
+-----------+-----------+-----------+-----------+
| R=G1*Q    | S=J*Q     |
| R=K*Q     | S=J*Q     |
| R=(R+T)*Q | S=(S+T)*Q |
| R=T*Q     | S=T*Q     |
+-----------+-----------+-----------+-----------+
```

R=K*Q, S=J*Q

(4) The fourth display is a circuit design of the selected FF. In the present example, the screen will display a simulated JK FF circuit using RS FF in the construction. The circuit consists of two blue AND gates, a green RS FF, and six green wires. See Figure 27 in part 1 of this chapter.
The above four steps are used for every simulation of FFs using other types of FFs on an Apple II system throughout package 5. If the user wants to simulate an RS FF using a T FF, the fourth step will display the circuit diagram in Figure 28 which consists of a green T FF, two blue AND gates, a violet OR gate, and some green wires to connect the logic elements. The display of the circuit diagram in Figure 29 on the screen will show the simulation of an RST FF using a JK FF which consists of a green JK FF, two violet OR gates, and some green wires. The circuit diagram in Figure 30 on the screen will show the simulation of a T FF using a green D FF, a violet XOR gate, and two green wires. The display of Figure 31 will show a D FF circuit consisting of a green RST FF, a blue AND gate, and two green wires.
CHAPTER VIII

SHIFTERS AND COUNTERS

A wide variety of memory devices are used in the designs of computers. Some such devices are shifters and counters. They are sequential circuits embodied in the arithmetic operations and control systems of computers.

Part 1 of this chapter explains the theoretical view of both shifters and counters. Part 2 relates to the graphic design of the systems and their functions on an Apple II system.

Part 1

Shift Register

A shift register is a collection of flip-flops which are arranged for the storage and manipulation by left or right shifts of relatively small amounts of binary data. A single clock pulse, shifting the information by one place, can be thought of as performing the operation of dividing or multiplying (depending on the direction of the shift) by two the number in the register. A simple storage register becomes a shift register if the individual flip-flops are interconnected in such a way that the output of one flip-flop is allowed to become the input to another one. Figure 32 shows a shift-right register using clocked D flip-flops. Successive clock pulses will cause the state of outputs
(QA, QB, QC, and QD) to change in accordance with the table in Figure 32.

<table>
<thead>
<tr>
<th>Clock</th>
<th>QA</th>
<th>QB</th>
<th>QC</th>
<th>QD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 32. 4-Bit binary shift register

COUNTER

Counters are a class of sequential logic circuits which record a series of input pulses. They are used for counting the number of occurrences of an event and are useful for generating timing sequences to control operations in a digital system. The input pulses, called count pulses, may be clock pulses, or they may originate from an external source and may occur at prescribed intervals of time or at random.
Of various sequences a counter may follow, the straight binary sequence is the simplest and most straightforward. A counter that follows the binary sequence is called a binary counter. An n-bit binary counter consists of n flip-flops and can count in binary from 0 to \((2^n) - 1\). An example of a count sequence of a 3-bit binary counter and the design of the circuit are shown in Figure 33.

<table>
<thead>
<tr>
<th>Counter Sequence</th>
<th>Flip-Flop inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock A3 A2 A1</td>
<td>TA3 TA2 TA1</td>
</tr>
<tr>
<td>------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>- 0 0 0</td>
<td>0 0 1</td>
</tr>
<tr>
<td>0 0 0</td>
<td>1 0 0</td>
</tr>
<tr>
<td>1 0 1</td>
<td>0 0 1</td>
</tr>
<tr>
<td>2 0 1 1</td>
<td>1 1 1</td>
</tr>
<tr>
<td>3 1 0 0</td>
<td>0 0 1</td>
</tr>
<tr>
<td>4 1 0 1</td>
<td>0 1 1</td>
</tr>
<tr>
<td>5 1 1 0</td>
<td>0 0 1</td>
</tr>
<tr>
<td>6 1 1 1</td>
<td>1 1 1</td>
</tr>
</tbody>
</table>

Figure 33. A 3-Bit binary counter

In Figure 33 the next number in the sequence represents the next state reached by the circuit upon the application of a count pulse. The count sequence repeats after it reaches the last values, so that
state 000 is the next state after 111. The three flip-flops are given a variable designation (A3, A2, A1).

Binary counters are most efficiently constructed with T flip-flops (or JK flip-flops). The T flip-flop inputs are derived from inspection of the truth table of T flip-flop and the state transition (in count sequence table) from a given count (present state) to the next below it (next state). For example, consider the flip-flop inputs 001. The present state (in count sequence table) is 001 and the next state is 010, which is the next count in the sequence. Comparing these two counts, we note that A3 goes from 0 to 0; so TA3 is marked with a 0 because the flip-flop associated with A3 must remain unchanged when a clock pulse occurs. A2 goes from 0 to 1; so TA2 is marked with a 1 because this flip-flop must be complemented in the next clock pulse. Similarly, A1 goes from 1 to 0, indicating that it must be complemented; so TA1 is marked with a 1. The last row of the table with present state 111 is compared with first count 000 which is its next state. The last condition requires that all three flip-flops are complemented.

Part 2

Package 6 is designed to illustrate the shifter in Figure 32 and the counter in Figure 33. The name of the package, which is built on face number 2 of the disk, is PACKAGE6. When the user accesses the package (see appendix A), the screen will show the following:
INSERT THE PROCEDURE NUMBER.
1: SHIFTER
2: COUNTER
3: HELP
4: EXIT

"1: Shifter"

If procedure number 1 is selected, a design of a 4-bit shift register will be displayed on the screen. The device consists of four blue D flip-flops and green wires for connection. The initial input and output values of the flip-flops are DA=1, QA=0; DB=0, QB=0; DC=0, QC=0; DD=0, QD=0. The whole circuit is shown in Figure 32.

The screen will also display two messages. The first one

P= FOR INSERTING PULSES.

means that if the key P is pressed, a pulse will be generated by the clock. Then the system will show the dynamic movements of logic-1 and -0 when the signals are shifted from one flip-flop to another. Each time the user presses P, the state of the shifter will change. In the same time the table in Figure 32 will be built on the screen to show the changes of states. The other message which will show on the screen is

T= FOR TERMINATION
THEN TYPE RETURN.

which allows the user to exit from the shifter procedure and return to the main package.
"2: Counter"

Selecting procedure number 2 will display an illustration of a 3-bit binary counter circuit consisting of three violet T flip-flops, one orange AND gate, a clock, and some green wires. Initially, the values of A1, A2, and A3 are zero. The input of TA1 will always be equal to 1. See Figure 33. The screen will show the messages

P = FOR INSERTING PULSES.
T = FOR TERMINATION
   THEN RETURN.

These messages have the same meaning as they do in the shifter procedure, except that the circuit will respond as a counter to pulses. The table on the screen will show the count sequence as in Figure 33.

"3: Help"

This procedure will inform the user how to use package 6.

"4: Exit"

The last procedure will allow the user to exit from the package.
CHAPTER IX

INTEGRATED AND MAGNETIC MEMORIES

The memory might be considered as the most important component of a computer. Usually more than fifty percent of the cost of a processor is contributed to the memory. The speed organization of a memory can alter the functional arrangement of the processor and the way in which it is fitted into the rest of the processor organization. A memory consists of storage registers and stores programs and/or data.

In this chapter, two kinds of random-access memories (RAM) are considered:

- integrated-circuit memory (IC) and magnetic core memory (MC). IC is said to be a volatile memory because its components lose stored information with time or when the power is turned off. MC is considered as nonvolatile memory because it retains its stored information even after removal of power. This is because the information in magnetic components is manifested by the direction of magnetization, which is retained when the power is turned off.

Part 1 of this chapter discusses the theoretical view of seven items of both IC and MC memories: integrated-memory cell, integrated-circuit memory, storing a bit into a magnetic core, two-dimensional core array, four wiring magnetic core, three-dimensional magnetic array, and a stack memory of four planes. Part 2 relates to the graphic representation of the seven systems on an
Part 1

**Integrated-Memory Cell**

In the integrated-memory, the binary storage cell is the basic building block of a memory unit. The circuit diagram for the binary cell that stores one bit of information is shown in Figure 34.

![Figure 34. Integrated-memory cell](image)

The system is provided with a select control and a read/write control. The select control enables or disables the circuit and depends on the value it is given (1 enables and 0 disables). The read/write control determines whether the circuit is in read or write mode. Setting the read/write equal to 1 puts the circuit in the read operation. the read/write control set equal to 0 puts the circuit in the write operation. The read operation reads the content of the flip-flop and passes it to the output line. The write operation stores
the input information in the flip-flop.

**Integrated-Circuit Memory**

The integrated-circuit random-access memory (IC RAM) in Figure 35 consists of four words (groups of binary information) of three bits each, making a total of twelve storage cells which are represented by the small boxes in the Figure. The two address input lines are decoded by a 2-to-4 decoder. The decoder is enabled by the memory enable control. When the memory enable is at 0, all the outputs of the decoder are zero and no memory words are selected. If the memory enable is 1, a word is selected, depending on the value of the two input address lines. If the read/write control is 1, the bits of the selected word go through the three OR gates to the output lines. The nonselected binary cells produce zeros in the inputs of OR gates, so they do not have an effect on the outputs. If the read/write control is at 0, the information available on the input lines is transferred into the binary cells of the selected word. The nonselected binary cells in the other words are disabled by their selection inputs. Their previous values remain unchanged.
Storing a Bit into a Magnetic Core

A magnetic-core memory uses magnetic cores to store binary information. A magnetic core is a tiny ring of magnetic material having three physical quantities: current, magnetic flux, and voltage. The signal which excites the core is a current pulse in a wire passing orthogonally through the center of the core. The direction of magnetic flux through the core causes the binary information to be stored and determines the binary value. The output binary information is extracted from a wire linking the core in the form of a voltage pulse. With zero current, a flux which is either positive (counterclockwise direction) or negative (clockwise direction) stays in the magnetized
A counterclockwise magnetization represents a 1 and a clockwise magnetization represents a 0.

A pulse of current applied to the wires, which go through cores, can shift the direction of magnetization. Figure 36a shows current flowing through a core from right to left and producing flux in the core in the clockwise direction, causing the core to store binary information of value 0. Figure 36b shows the current and flux directions for storing a 1.

Two-Dimensional Core Array

Consider a two-dimensional array of cores as shown in Figure 37. If a current \( +1/2 I \) is applied to any one X wire and any one Y wire, only
one core in the array is switched (i.e., one selected core changes its direction of magnetization). Only one core in the matrix can be switched at a time, since only one core can have the full switching current passing through it. The material of which the cores are made has the property such that if insufficient current passes through a core to switch it, the magnetization reverses and sends the pulse in the opposite direction (where it came from).

In this memory, as in all core memories having destructive read-out, the data in the word which is read is obliterated. Since it is usually desired to retain this word in a memory location until new data is written there, the word which has just been read must be written back into the same location immediately, if it is to be preserved. For this reason memories with destructive read-out have a
read-write cycle of operation. In a memory read operation, the data is first read and stored temporarily in a register, and is then rewritten in the same location before the memory selection address is changed. During a write operation a similar cycle is followed, in which the data in a location is first read-out, but not allowed to set the memory register, to clear the location. The new data to be entered is then written in during the write part of the cycle.

Four Wiring Magnetic Core

The most popular type of ferrite core memory at the present time is the 3D-4 wire system. Each core has the same design of threaded wires as in Figure 37 with two extra wires going through each core. One of these extra wires is called a sense wire and is used for sensing a voltage for reading. The second additional wire is called an inhibit wire and is used to inhibit writing on any core. The diagram in Figure 38 shows the wiring system for a three-dimensional core.

![Figure 38. Four Wiring magnetic core system](Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.)
Three-Dimensional Magnetic Array

In this system one plane of cores is used for each bit in a word and a stack of n planes forms a memory of n bit words. The three dimensions of a four wiring system and a single 3X4 array are shown in Figure 39.

A particular core in the plane is selected by passing 1/2 I current through an X wire and a Y wire. Only one core receives the field due to both X and Y currents and is therefore selected. These pulses are positive for writing and negative for reading. A single sense wire threads every core in the plane in a zig-zag manner, with cores arranged as shown. The zig-zag arrangement has the effect of cancelling the disturb voltages from alternate cores during the reading, so that the overall effect of these is much reduced. The inhibit wire threads each core in the plane, and can inhibit writing on any core in that plane. If a 0 is to be written for that bit or plane, a current of -1/2 I is passed through the inhibit wire. The selected core receives

\[ (+1/2 I + 1/2 I - 1/2 I = ) 1/2 I \text{ current} \]

and the core, which had been previously cleared to 0, does not change state (remains at 0). Other cores in the plane receive either

\[ (+1/2 I - 1/2 I = ) 0 \text{ or } -1/2 I \text{ current,} \]

and in both cases do not change state. But if no current is passed through the inhibit wire, the selected core is set to 1.
Stack Memory of Four Planes

A diagram of the wiring arrangement for a 4-bit memory having four core planes and twelve address locations is shown in Figure 40. Large numbers of memories have been manufactured using this type of formation, in which each memory bit plane is separately constructed on a frame. A number of these planes, one for each bit, are then laid above each other to form a stack of core planes, which are then wired as indicated in Figure 40. It is now practical to make planes each having 64X64 cores, representing 4096 memory locations when stacked. Larger memories are made by adding more planes to a stack, which then
makes more significant control bits in the address register.

Figure 40. Stack memory of Four planes

Part 2

In this part, a graphically-aided package demonstrating two kinds of memories is described. The package program is designed on an Apple II system. Various colors are used in the graphics of the package. The two kinds of memories are explained in part 1. Package 7 is stored on face number 2 of a disk and is called PACKAGE7. When the package is
accessed (see appendix A), the screen will show a list of nine main procedures in the following form.

**INSERT THE PROCEDURE NUMBER DESIRED.**

1: MEMORY CELL  
2: INTEGRATED-CIRCUIT MEMORY  
3: STORING A BIT INTO A MAGNETIC CORE  
4: TWO-DIMENSIONAL ARRAY  
5: FOUR WIRING CORE  
6: THREE-DIMENSIONAL ARRAY  
7: STACK MEMORY OF FOUR PLANES  
8: HELP  
9: EXIT

To choose one of the above main procedures, the user needs to insert its number, then the screen will display the selected system.

"1: Memory Cell"

This memory cell consists of one violet RS flip-flop, three blue AND gates, two white NOT gates, and many green wires to connect the elements of the circuit. The design of the circuit is shown in Figure 34 of part 1.

When the user selects procedure number 1, a message will appear on the screen asking for insertion of the input values. After inserting the binary information, the binary logics (1 and 0) are shown to move dynamically through gates along the connecting wires. A second message will ask for the value of the select control. The last message will ask the user to select either read or write operation. The result of storing a bit in the memory cell (RS flip-flop) will be displayed on the screen.
Suppose we want to select the memory cell procedure and store a logic-1 in the memory cell. The process is to first select number 1 from the list of main procedures; the screen will display the system along with the message:

INSERT INPUT:(1/0) ?

The user should insert the binary value 1. The binary value will move from the input line to the two AND gates. Then the first message will disappear and the second message will appear:

INSERT SELECT:(1/0) ?

The response should be to insert 1 to enable the circuit. Then the second message will disappear and the third message will appear:

INSERT R/W:(1/0) (READ=1/WRITE=0).

Since the user wants to write the input 1 into the flip-flop, a 0 should be the response to the third message. The system will type the result of storing the bit on the screen as follows:

STORE BIT IN MEMORY =1.

The user's input to the three messages will also appear on the screen at the time of response to remind the user of the input values to the system. For example, the last version of the inserted values for the above example will appear as follows.

<table>
<thead>
<tr>
<th>I</th>
<th>SEL</th>
<th>R/W</th>
<th>OUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

The value of OUT is for the read operation only. A carriage return should be used to return to the list of main procedures. The unit cell will retain the previously stored information for future access of the
same procedure during one accessing of package 7.

"2: Integrated-Circuit Memory"

The second system consists of twelve orange memory cells, one violet 2x4 decoder, three blue OR gates, and many green wires to connect the circuit. There are two address input lines and three data output lines. The system is provided with three data input control lines, a memory enable control, and a read/write control. The decoder decodes the two address inputs to four word lines: WORD0, WORD1, WORD2, WORD3. (See part 1 of this chapter for more detail.)

"3: Storing a Bit into a Magnetic Core"

The display in this procedure will show the directions of the magnetic flux to store the binary values 1 and 0 in magnetic cores. The first diagram will show how to store 0. It consists of a green magnetic ring which is threaded by a white wire. The second diagram shows how to store 1, using the same components that are used in the first diagram. (See Figure 36a and 36b.)

"4: Two-Dimensional Array"

The display of the fourth procedure will show the construction of a two-dimensional core array. It is formed from twelve green cores, three blue X rows, and four white Y columns. X wires and Y wires are threaded through the cores. A 1/2 current is passed through an X row
and a Y column. The selected core is shown. (See Figure 37.)

"5:Four Wiring Core"

The display of the fifth procedure will show a typical three-dimensional magnetic core. It consists of a green magnetic ring, a blue X wire, a white Y wire, a violet inhibit wire, and an orange sense wire. All wires are threaded through the magnetic ring to form a four wiring core. (See Figure 38.)

"6:Three-Dimensional Array"

If the user selects procedure number 6 from the list of main procedures, the screen will display a three-dimensional core array consisting of twelve green cores, three blue X wires, four white Y wires, a violet sense wire, and an orange inhibit wire. The X wires thread the cores in rows and the Y wires thread the cores in columns. The sense wire threads each core in the plane in a zig-zag manner. The inhibit wire also threads each core in the plane. (See Figure 39.)

"7:Stack Memory of Four Planes"

The last system consists of four white planes, three orange X wires, four green Y wires, and inhibit and sense wires. The display shows how the wires pass through the stack of planes using three-dimensional graphics. Each plane represents positions of bits in words. For example, there are four planes: 2**0, 2**1, 2**2, and 2**3.
All of these planes form a 4-bit memory with twelve address locations.

"8:Help"

The eighth procedure will help the user to briefly learn how to use package 7.

"9:Exit"

This procedure will allow the user to exit from the package.
CHAPTER X

CONCLUSION AND RECOMMENDATIONS

Conclusion

The goal of this research is to build computer-aided programs to assist students in learning digital logic and computer structures on an Apple II system. The function of the project has been performed successfully through a series of seven packages which are stored on a double-sided floppy disk.

Packages 1 through 3 are on face number 1 of the disk. Package 1 demonstrates the basic design of the electronic elements of logics (NOT, OR, AND, NOR, and NAND gates). Package 2 shows the use of the K-Map technique and the construction of combinational circuits. The application of some important combinational circuits are presented in package 3.

Packages 4 through 7 are built on face number 2 of the disk. The fourth package shows the functional input-output techniques of various flip-flops. Package 5 simulates different types of flip-flops using other types of flip-flops. Package 6 demonstrates the functional implementation of a shift register and a counter. The last package demonstrates two kinds of random access memories (integrated-circuit and magnetic core). All programs are written in the Pascal language and the code files are kept in libraries to protect the programs, save space on the disk, and handle the large programs.
The demonstration of the packages shows that computers can teach about themselves (i.e., electronics inside the computer). From literature it is found that the uses of computer in education have changed markedly over past ten years. Micros will be inexpensive and common. There is now a concentrated effort on the part of many school boards and provincial, state, and federal governments to create consortiums of users in order to create, exchange, and adapt existing software in the area of the educational uses of computers. A fear that computers may replace teachers, administrators, and educational support staff has been provoked by a new awareness.

The main problems in the project are:

(1) The amount of storage on disk is small.
(2) The Apple II device sometimes destroys the directory of a disk.
(3) The size of the screen is not always large enough to display a particular picture.
(4) The time for the project is not enough to test the packages on a sample of students.

Recommendations

The following recommendations are suggested to people who like to choose the new line of technology and support CAI packages.

(1) It is well if CAI packages are tested before using them regularly for teaching classes.
(2) Instructors are urged to use CAI in their planned curriculum because it provides an enjoyable atmosphere, enough time for learning,
less cost of education, and high confidence to students for
self-learning and self-teaching.

(3) Others are encouraged to further research for more advancements of
electronic devices.

(4) The Pascal language is recommended as a programming language
because of its flexibility.
REFERENCES


APPENDIX A

EXECUTION OF A PACKAGE
ONE-DRIVE OR TWO-DRIVE STARTUP

If your system has two diskette drives, insert a disk in the slot labelled DRIVE 1. If it has one diskette drive, put a disk in the slot and close the door to the disk drive, and turn on the Apple and the screen (or TV). First, the message (APPLE II) appears at the top of the monitor screen (or TV), and the disk drive's IN USE light comes on. The disk drive emits sound indicating that it is working. The screen lights up for displaying of black signs (@'s) on a white background, then shows a black screen. The message

WELCOME APPLE1, TO
U.C.S.D. PASCAL SYSTEM II.1
CURRENT DATE MAY-27-1983

Appears on the first page of the screen (if for example you are using Apple II+), followed by a line at the top of the screen:

COMMAND: E(DIT, R(UN, F(ILE, C(OMP, L(IN

And the rest of the line shows on the second page of the screen (it can be displayed by pressing control A (C/A), then again pressing C/A to return to the first page of the screen) which is:

K, X(ECUTE, A(SSEM, D(EBUG,? [1.1]

This line at the top of the screen is running the Apple Pascal system and waiting for the user's selected command.

To run one of the packages on face no. 1 or face no. 2 of the disk, type the letter "X" (for Execute). The system will send a message on the first page of the screen

EXECUTE WHAT FILE
at this point the user should insert the name of the package to be run and follow it with a carriage return.

For example, if the user intends to execute package 1, his response to the last message (EXECUTE WHAT FILE?) should be as follows.

PACKAGE1 (followed by a carriage return).

then package 1 will start running.

The user is advised to follow the instructions which are provided by the system.
APPENDIX B

FLIP-FLOP TABLE
<table>
<thead>
<tr>
<th>Table 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>N+1</td>
</tr>
<tr>
<td>J</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
APPENDIX D

PROGRAMS
(* FALAH REDA AL-SAFFAR *)

(* A PROGRAM FOR ELECTRONIC CIRCUIT DESIGN OF GATES *)
(* (NOT, OR, AND, NAND, and NOR) *)

(*$S+*)
PROGRAM PACKAGE1;
(* CALL SUBROUTINES FROM THE LIBRARY OF THE PROJECT *)
USES TURTLEGRAPHICS,APPLESTUFF,STUFF;
VAR CH,N:CHAR;MM:INTEGER;
PROCEDURE HEAD;
BEGIN
FILLScreen(81);
PENCOLOR(NONE);
MOVETO(100,180);
WSTRING('PACKAGE1');
MOVETO(100,170);
WSTRING('--------------');
MOVETO(0,150);
WSTRING('THE BASIC ELECTRONIC CONSTRUCTION');
MOVETO(50,130);
WSTRING('OF GATES');
MOVETO(0,100);
WSTRING('"NOT", "OR", "AND", "NOR", and "NAND"');
MOVETO(80,70);
WSTRING('BY');
MOVETO(80,50);
WSTRING('FALAH AL-SAFFAR');
MOVETO(180,0);WSTRING('(PRESS RETURN)');
REPEAT UNTIL KEYPRESS;
END;
PROCEDURE HELP;
BEGIN
WRITELN;WRITELN;
WRITELN('HELP PROCEDURE');
WRITELN;WRITELN;
WRITELN('THIS PACKAGE DEMONSTRATES THE FUNCTION');
WRITELN('OF THE BASIC ELECTRONIC ELEMENTS DIODE,');
WRITELN('RESISETER, AND TRANSISTOR AND THE LOGICS');
WRITELN('"NOT", "OR", "AND", "NOR", AND "NAND"');
WRITELN('GATES,');WRITELN;
WRITELN('EACH PROCEDURE IS LABELLED WITH A');
WRITELN('NUMBER. IF YOU WANT TO SELECT A');
WRITELN('PROCEDURE, INSERT ITS NUMBER. REMEMBER');
WRITELN('TO PRESS RETURN AFTER EACH DISPLAY TO');
WRITELN('RETURN TO THE MAIN PACKAGE.');
FOR MM:=1 TO 8 DO WRITELN;
WRITELN('PRESS RETURN TO CONTINUE');
REPEAT UNTIL KEYPRESS;
READ(CH);
END;
BEGIN
INITTURTLE;
GRAFMODE;HEAD;FILLSCREEN(B1);
REPEAT
TEXTMODE;
WRITELN;WRITELN;WRITELN;WRITELN;WRITELN;
WRITELN('THE FOLLOWING LIST SHOWS THE PROCEDURE');
WRITELN('NUMBER CORRESPONDING TO EACH OF THE');
WRITELN('NAMED ITEMS. INSERT THE PROCEDURE');
WRITELN('NUMBER WHICH YOU WANT TO ACCESS.‘’);
WRITELN;WRITELN;WRITELN;
WRITELN('1: DIODE');
WRITELN('2: TRANSISTOR');
WRITELN('3: NOT GATE');
WRITELN('4: OR GATE');
WRITELN('5: AND GATE');
WRITELN('6: NOR GATE');
WRITELN('7: NAND GATE');
WRITELN('8: HELP');
WRITELN('9: EXIT');
WRITELN;WRITELN;
REPEAT READ(N) UNTIL ((N='1')OR(N='2')OR(N='3')OR(N='4')
OR(N='5')OR(N='6')OR(N='7')OR(N='8')OR(N='9'));
IF N='8' THEN HELP;
IF ((N='1')OR(N='2')OR(N='3')OR(N='4')OR(N='5')OR(N='6')
OR(N='7'))
THEN
BEGIN
GRAFMODE;
CASE N OF
‘1’:DDIODE;
‘2’:DTRANS;
‘3’:NOTGCIR;
‘4’:ORGCIR;
‘5’:ANDGCIR;
‘6’:NORGCI;
‘7’:NANDGCI;
END;
PENCOLOR(NO);
MOVETO(180,180);
WSTRING('(PRESS RETURN)’);
REPEAT UNTIL KEYPRESS;
READ(CH);
FILLScreen(B1);
END;
UNTIL N='9';
END.
(* K-MAP AND COBINATIONAL CIRCUITS ARE PROGRAMMED IN *)
(* THIS PACKAGE. *)

(*$S++*)
PROGRAM PACKAGE2;
(* CALL PROCEDURES FROM THE LIBRARY *)
USES TURTLEGRAPHICS,APPLESTUFF,STUF3,STUF4,STUF5,STUF6;
PROCEDURE FIRST;
BEGIN
  GRAFMODE;
  FILLSCREEN(BLACK1);
  PENCOLOR(NONE);
  MOVETO(100,180);
  WSTRING("PACKAGE2");
  MOVETO(100,170);
  WSTRING("--------");
  MOVETO(0,150);
  WSTRING("KARNAUGH MAP AND COMBINATIONAL CIRCUITS");
  MOVETO(80,70);
  WSTRING("BY");
  MOVETO(80,50);
  WSTRING("FALAH AL-SAFFAR");
  MOVETO(180,0);WSTRING("(PRESS RETURN)");
  REPEAT UNTIL KEYPRESS;READ(CH);FILLSCREEN(BLACK1);
END;
(* MAIN PROGRAM *)
BEGIN
  INITTURTLE;
  FIRST;
  RESET(INPUT0,'#4:DATA.TEXT');
  CHR:="Y";
  COLOR:=GREEN;
  CONTM:=0;
  WHILE (NOT EOF(INPUT0)AND(CHR="Y")) DO
    BEGIN
      FILLSCREEN(BLACK1);
      FOR I:=0 TO 15 DO
        BEGIN
          MAP[I].S:=0;
          MAP[I].U:=FALSE;
        END;
      TEXTMODE;WRITELN("THE PRESENT FUNCTION WITHOUT DON'T CARE:");
      WRITE("F=");
      DONT:=FALSE;
      CONTM:=CONTM+1;
      READLINE;
      COMMANDS;
      TEXTMODE;
      FILLSCREEN(BLACK1);
    END;
  END;
END.
ABCDEFGHIJKLMNOPQRSTUVWXYZ
0123456789
ABCDEFGHIJKLMNOPQRSTUVWXYZ
0123456789

WRITELN;WRITELN;
WRITELN(’DO YOU WANT DON”T CARE WITH THE’);
WRITELN(’CURRENT FUNCTION ? (Y/N’);
REPEAT READ(CH); UNTIL ((CH=’Y’)OR(CH=’N’));
CONTM:=CONTM+1;
IF CH=’Y’ THEN
BEGIN
  DONT:=TRUE;
  WRITELN;WRITELN(’THE PRESENT FUNCTION WITH DON”T CARE’);
  WRITE(’D=’);
  READLINE;
  FILLSCREEN(BLACK1);
  COMMANDS;
END
ELSE READLN(INPUTO);
WRITELN;
IF CONTM=16 THEN WRITELN(’END OF DATA FILE ’)
ELSE BEGIN
  WRITELN(’DO YOU WANT NEW FUNCTION ? (Y/N’);
  REPEAT READ(CHR);UNTIL ((CHR=’Y’)OR(CHR=’N’));
END;
WRITELN;
END;
CLOSE(INPUTO);
END.
(* APPLICATION OF COMBINATIONAL CIRCUITS *)

(*$S++*
PROGRAM PACKAGE3;
USES TURTLEGRAPHICS,APPLESTUFF,STUF1,STUF2;
VAR N:CHAR;I:INTEGER;
PROCEDURE FIRST;
BEGIN
  FILLSCREEN(BLACK1);
  PENCOLOR(NONE);
  MOVETO(100,180);
  WSTRING("PACKAGE3");
  MOVETO(100,170);
  WSTRING("----------");
  MOVETO(0,150);
  WSTRING("APPLICATION OF COMBINATIONAL CIRCUITS");
  MOVETO(180,0);
  WSTRING("BY FALAH AL-SAFAAR");
  MOVETO(80,70);
  WSTRING("PRESS RETURN");
  REPEAT UNTIL KEYPRESS;
END;
PROCEDURE HELP;
BEGIN
  WRITELN;
  WRITELN("HELP PROCEDURE");
  WRITELN;
  WRITELN("THIS PACKAGE CONSISTS OF SEVERAL");
  WRITELN("EXAMPLES OF COMBINATIONAL CIRCUITS.");
  WRITELN;
  WRITELN("EACH COMBINATIONAL CIRCUIT IS");
  WRITELN("LABELLED WITH A NUMBER. TO VIEW");
  WRITELN("A PARTICULAR CIRCUIT, INSERT ITS");
  WRITELN("NUMBER WITHOUT PRESSING RETURN.");
  WRITELN("WHEN WANTING TO RETURN TO THE MAIN");
  WRITELN("PACKAGE PROCEDURES, PRESS CARRIAGE");
  WRITELN("RETURN.");
  FOR I:=1 TO 10 DO WRITELN;
  WRITELN("PRESS RETURN");
  REPEAT UNTIL KEYPRESS;READ(N);
END;
BEGIN (* MAIN *)
  INITTURTLE;GRAFMODE;FIRST;
  FILLSCREEN(BLACK1);
  REPEAT
    TEXTMODE;
    FOR I:=1 TO 10 DO WRITELN;
    WRITELN("THE FOLLOWING IS THE NUMBERING SYSTEM");
  })
WRITELN('FOR THE PROCEDURES IN PACKAGE 3. TYPE');
WRITELN('IN THE NUMBER OF THE PROCEDURE WHICH');
WRITELN('YOU WOULD LIKE TO EXERCISE.');
WRITELN;
WRITELN('1: FULL-ADDER');
WRITELN('2: LOOK-AHEAD CARRY GENERATOR');
WRITELN('3: 4-BIT FULL-ADDER WITH LOOK-AHEAD');
WRITELN('4: CARRY GENERATOR');
WRITELN('5: 3-TO-8 LINE DECODER');
WRITELN('6: 4X2 READ-ONLY MEMORY');
WRITELN('7: HELP');
WRITELN('8: EXIT');
REPEAT READ(N); UNTIL (N='1' OR N='2' OR N='3' OR N='4' OR
N='5' OR N='6' OR N='7' OR N='8');
IF N='7' THEN HELP;
IF N IN ['1', '2', '3', '4', '5', '6'] THEN
BEGIN
GRAFMODE;
CASE N OF
'1': FULADDER;
'2': LOOKAHEAD;
'3': FOURADDERSWLH;
'4': DECODER;
'5': MULTIPLEXER;
'6': ROM;
END;
REPEAT UNTIL KEYPRESS; READ(N);
FILLSCREEN(BLACK1);
END;
UNTIL N='8';
END.
(* FLIP-FLOP PROGRAM *)

(*$S++*)

PROGRAM PACKAGE4;
USES TURTLEGRAPHICS,APPLESTUFF,STUFF1,STUFF3;
PROCEDURE FIRST;
BEGIN
  GRAFMODE;
  FILLSCREEN(BLACK1);
  PENCOLOR(NONE);
  MOVETO(100,180);
  WSTRING('PACKAGE4');
  MOVETO(100,170);
  WSTRING('-------------');
  MOVETO(0,150);
  WSTRING('SEQUENTIAL CIRCUITS AND FLIP-FLOP');
  MOVETO(0,130);
  WSTRING('TECHNIQUES');
  MOVETO(80,70);
  WSTRING('BY');MOVETO(80,50);
  WSTRING('FALAH AL-SAFFAR');
  MOVETO(180,0);WSTRING('PRESS RETURN');
  REPEAT UNTIL KEYPRESS;FILLSCREEN(BLACK1);
END;
BEGIN (* MAIN *)
  INITTURTLE;
  Q1:="0";
  INIT:=TRUE;
  NOTQ1:="0";
  Q:="0";FIRST;
  REPEAT
    FILLSCREEN(BLACK1);
    COM4;
    WRITELN;
    IF G<>"8" THEN
      BEGIN
        GRAFMODE;
        CASE G OF
          '1':RS;
          '2':RST;
          '3':JK;
          '4':TF;
          '5':DF;
          '6':MASTERSLAVE;
          '7':HELP4;
        END;
        REPEAT UNTIL KEYPRESS;
      END;
  END;
END;

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
UNTIL G="8";
END.
(*) SIMULATION OF FLIP-FLOPS *)

(*$S++*)
PROGRAM PACKAGE5;
USES TURTLEGRAPHICS,APPLESTUFF,STUFF1,STUFF2;
VAR JJ:INTEGER;G:CHAR;
BEGIN
  INITTURTLE;
  FILLSCREEN(BLACK1);TEXTMODE;
  REPEAT
    COMMANDS;
    WRITELN;
    CASE NO OF
      '1':SIMJK;
      '2':SIMRS;
      '3':SIMRST;
      '4':SIMT;
      '5':SIMD;
      '6':BEGIN
        HELP5;
        REPEAT UNTIL KEYPRESS;READ(G);
        END;
      END;
    END;
  UNTIL NO='7';
END.

(* THE NEXT IS THE MAIN PROGRAM OF PACKAGE 5 *)

(*$S++*)
PROGRAM PACKAGE5;
USES TURTLEGRAPHICS,APPLESTUFF,CHAINSTUFF;
BEGIN
  INITTURTLE;
  FILLSCREEN(BLACK1);WHITE;
  PENCOLOR(NONE);
  MOVETO(100,180);
  WSTRING('PACKAGES ');
  MOVETO(100,170);
  WSTRING(' ---------------' ) ;
  MOVETO(80,150);
  W STRINGC'SIMULATION OF FLIP-FLOPS') ;
  MOVETO(80,150);
  WSTRING(' BY');MOVETO(80,50);
  WSTRING(' FALAH AL-SAFFAR');
  MOVETO(180,0);WSTRING(' (PRESS RETURN)');
  REPEAT UNTIL KEYPRESS;

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
(* CALL PACKAGE5 *)
SETCHAIN('LINKWITH5');
END.
(* SHIFTER AND COUNTER *)

(*$S++*

PROGRAM PACKAGE6;
USES TURTLEGRAPHICS,APPLESTUFF,STUFF1,STUFF4;
PROCEDURE FIRST;
BEGIN
  GRAFMODE;
  FILLSCREEN(BLACK1);
  PENCOLOR(NONE);
  MOVETO(100,180);
  WSTRING('PACKAGE6');
  MOVETO(10,170);
  WSTRING('-------------');
  MOVETO(60,150);
  WSTRING('"SHIFTERS AND COUNTERS"');
  MOVETO(80,150);
  WSTRING('BY

  FALAH ALSAFFAR');
  MOVETO(180,0);
  WSTRING('PRESS RETURN');
  REPEAT UNTIL KEYPRESS;
END;
BEGIN
  INITTURTLE;FIRST;
  REPEAT
    FILLSCREEN(BLACK1);
    COM6;
    WRITELN;
    IF G>'4' THEN
    BEGIN
      GRAFMODE;
      CASE G OF
        '1':SHIFT;
        '2':COUNTER;
        '3':HELP6;
      END;
      REPEAT UNTIL KEYPRESS;
    END;
  UNTIL G='4';
END.
(* THIS PROGRAM IMPLEMENTS MEMORIES *)

(*$S++*)
PROGRAM PACKAGE7;
USES TURTLEGRAPHICS,APPLESTUFF,TRANSCEND,STUFF1,STUFF5;
PROCEDURE FIRST;
BEGIN
  GRAFMODE;
  FILLSCREEN(BLACK1);
  PENCOLOR(NONE);
  MOVETO(100,180);
  WSTRING('PACKAGE7');
  MOVETO(100,170);
  WSTRING('--------------');
  MOVETO(20,150);
  WSTRING('"INTEGRATED AND MAGNETIC MEMORIES"');
  MOVETO(80,70);
  WSTRING('"BY"');MOVETO(80,50);
  WSTRING('FALAH AL-SAFFAR"0;
  MOVETO(180,0);WSTRING('"PRESS RETURN"');
  REPEAT UNTIL KEYPRESS;
END;
BEGIN (* MAIN *)
INITTURTLE;FIRST;
REPEAT
  FILLSCREEN(BLACK1);
  COM7;
  WRITELN;
  IF G<>'9' THEN
  BEGIN
    GRAFMODE;
    CASE G OF
      '1':MCELL;
      '2':ICM;
      '3':MAG;
      '4':DD2;
      '5':CORES;
      '6':D3;
      '7':ALLPLANES;
      '8':HELP7;
      END;
      REPEAT UNTIL KEYPRESS;
  END;
  UNTIL G="9";
END.
(* FALAH AL-SAFFAR *)

(* A SAMPLE OF ROUTINES FROM THE LIBRARY #1 WHICH CONSISTS *)
(* OF SIX UNITS. EACH UNIT CONTAINS SEVERAL ROUTINES *)
(* TO IMPLEMENT CERTAIN FUNCTIONS IN THE PROJECT. *)

(*$S+*)

UNIT STUF1; INTRINSIC CODE 16;
INTERFACE;
USES TURTLEGRAPHICS, APPLESTUFF;
CONST
  B1=BLACK1;
  G=GREEN;
  V=VIOLET;
  O=ORANGE;
  BL=BLUE;
  W=WHITE1;
  NO=NONE;
PROCEDURE PM(C:SCREENCOLOR;X,Y:INTEGER);
PROCEDURE MP(X,Y:INTEGER;C:SCREENCOLOR);
PROCEDURE M2(X1,Y1,X2,Y2:INTEGER);
PROCEDURE PMPM(C1,C2:SCREENCOLOR;X1,Y1,X2,Y2:INTEGER);
PROCEDURE P2M(C:SCREENCOLOR;X1,Y1,X2,Y2:INTEGER);
PROCEDURE P3MP(C1,C2:SCREENCOLOR;X1,Y1,X2,Y2,X3,Y3:INTEGER);
PROCEDURE P4MP(C1,C2:SCREENCOLOR;X1,Y1,X2,Y2,
               X3,Y3,X4,Y4:INTEGER);
PROCEDURE P5MP(C1,C2:SCREENCOLOR;X1,Y1,X2,Y2,X3,Y3,
               X4,Y4,X5,Y5:INTEGER);
PROCEDURE VR(C1,C2,C3:SCREENCOLOR;X,Y,I:INTEGER);
PROCEDURE LRESISTOR(X,Y,I:INTEGER);
PROCEDURE LINPUT(X,Y:INTEGER;CH:CHAR);
PROCEDURE DIODES(X,Y,H,L,W:INTEGER;CH:CHAR;LTOR:BOOLEAN;
                 C:SCREENCOLOR);
PROCEDURE TRANS(X,Y:INTEGER;C:SCREENCOLOR);
PROCEDURE TR(C:SCREENCOLOR;X,Y:INTEGER;CH:CHAR);
PROCEDURE DNOT(X,Y:INTEGER);
PROCEDURE NOTGCIR;
PROCEDURE DANDG(X,Y:INTEGER);
PROCEDURE ANDGCIR;
PROCEDURE DORCIR;
PROCEDURE ORGCIR;
PROCEDURE DNAND(X,Y:INTEGER);
PROCEDURE NANDCIR;
PROCEDURE DNOR(X,Y:INTEGER);
PROCEDURE NORGCIR;
PROCEDURE DDIODE;
PROCEDURE DTRANS;
IMPLEMENTATION

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
PROCEDURE PM;
BEGIN
  PENCOLOR(C);
  MOVETO(X,Y);
END;

PROCEDURE MP;
BEGIN
  MOVETO(X,Y);
  PENCOLOR(C);
END;

PROCEDURE M2;
BEGIN
  MOVETO(X1,Y1);
  MOVETO(X2,Y2);
END;

PROCEDURE PMPM;
BEGIN
  PM(C1,X1,Y1);
  PM(C2,X2,Y2);
END;

PROCEDURE P2M;
BEGIN
  PENCOLOR(C);
  M2(X1,Y1,X2,Y2);
END;

PROCEDURE P3MP;
BEGIN
  P2MP(C1,X1,Y1,X2,Y2);
  MP(X3,Y3,C2);
END;

PROCEDURE P4MP;
BEGIN
  P2M(C1,X1,Y1,X2,Y2);
  M2(X3,Y3,X4,Y4);
  PENCOLOR(C2);
END;

PROCEDURE P5MP;
BEGIN
  P2M(C1,X1,Y1,X2,Y2);
  M2(X3,Y3,X4,Y4);
  MP(X5,Y5,C2);
END;

PROCEDURE LRESISTOR;
BEGIN
  PM(NONE,X,Y);
  CASE I OF
    0:WSTRING("R");
    1:WSTRING("R1");
    2:WSTRING("R2");
    3:WSTRING("R3");
4:WSTRING("R4");
5:WSTRING("R5");
6:WSTRING("R6");
7:WSTRING("R7");
8:WSTRING("R8");
9:WSTRING("R9");
END;
END;
PROCEDURE LINPUT;
BEGIN
  PM(NONE,X,Y);
  CASE CH OF
    "A":WSTRING("A");
    "B":WSTRING("B");
    "C":WSTRING("C");
    "D":WSTRING("D");
    "1":WSTRING("D1");
    "2":WSTRING("D2");
    "3":WSTRING("D3");
    "4":WSTRING("D4");
    "5":WSTRING("D5");
  END;
END;
PROCEDURE LINPUT;
BEGIN
  PM(NONE,X,Y);
  CASE CH OF
    "A":WSTRING("A");
    "B":WSTRING("B");
    "C":WSTRING("C");
    "D":WSTRING("D");
    "1":WSTRING("D1");
    "2":WSTRING("D2");
    "3":WSTRING("D3");
    "4":WSTRING("D4");
    "5":WSTRING("D5");
  END;
END;
PROCEDURE LINPUT;
BEGIN
  PM(NONE,X,Y);
  CASE CH OF
    "A":WSTRING("A");
    "B":WSTRING("B");
    "C":WSTRING("C");
    "D":WSTRING("D");
    "1":WSTRING("D1");
    "2":WSTRING("D2");
    "3":WSTRING("D3");
    "4":WSTRING("D4");
    "5":WSTRING("D5");
  END;
END;
PROCEDURE DIODES;
BEGIN
  PENCOLOR(NONE);
  IF LTOR THEN BEGIN
    MP(X+W,Y,C);
    P5MP(C,C,X+H,X,Y-L,X+W,Y+H,X+W,Y-L);
    LINPUT(X+W+3,Y+H,CH);
    MOVETO(X,Y);
  END ELSE BEGIN
    MOVETO(X-W,Y);
    LINPUT(X+3,Y+H,CH);
    MOVETO(X,Y);
  END;
END;
END;

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
PROCEDURE TRANS;
BEGIN
  P4MP(C, C, X-5, Y-30, X, Y-30, X-2, Y-26, X, Y-30);
END;

PROCEDURE TR;
BEGIN
  PMPM(NONE, C, X, Y, X, Y-10);
  TRANS(X, Y-10, VIOLET);
  PM(C, X, Y-50);
  PMPM(NONE, C, X-6, Y-50, X+6, Y-50);
  PMPM(NONE, C, X-4, Y-52, X+4, Y-52);
  PMPM(NONE, C, X-2, Y-54, X+2, Y-54);
  PMPM(NONE, C, X-30, Y-25, X-50, Y-25);
  LINPUT(X-50, Y-25, CH);
END;

PROCEDURE DNOT;
BEGIN
  PM(NO, X, Y);
  P5MP(0, NO, X+30, Y, X+30, Y+15, X+50, Y, X+30, Y-15, X+30, Y);
  PMPM(NO, O, X+50, Y, X+60, Y);
  PENCOLOR(NO);
END;

PROCEDURE NOTGcir;
VAR X, Y: INTEGER;
BEGIN
  X:= 50; Y:= 50;
  PM(NO, X+88, Y+133);
  WSTRING('+VO');
  PMPM(NO, BL, X+88, Y+130, X+88, Y+120);
  VR(NO, BL, V, X+88, Y+120, 0);
  P2M(BL, X+88, Y+76, X+108, Y+76);
  MOVETO(X+110, Y+76);
  WSTRING("OUTPUT");
  M2(X+88, Y+76, X+88, Y+70);
  TRANS(X+88, Y+70, G);
  MOVETO(X+88, Y+20);
  PMPM(NO, BL, X+82, Y+20, X+94, Y+20);
  PMPM(NO, BL, X+84, Y+18, X+92, Y+18);
  PMPM(NO, BL, X+86, Y+16, X+90, Y+16);
  PMPM(NO, BL, X+58, Y+55, X-10, Y+55);
  WSTRING("INPUT");
  PENCOLOR(NO);
  DNOT(X+150, Y);
  PM(NO, X+160, Y-40);
  WSTRING("THE SYMBOL");
  MOVETO(0, 0);
  WSTRING("BASIC "NOT" GATE CIRCUIT");
  MOVETO(X+150, Y+3);
  WSTRING("1");
END.
PROCEDURE DANDG;
BEGIN
PM(NO,X,Y);
P5MP(G,NO,X,Y+15,X+10,Y+15,X+10,Y-15,X,Y-15,X,Y);
MOVETO(X+10,Y);
PMPM(G,NO,X+20,Y,X+23,Y);
WSTRING('ABC');
LINPUT(X-18,Y+18,'A');
MOVETO(X-15,Y+10);
P3MP(G,G,X,Y+10,X,Y,X-15,Y);
LINPUT(X-18,Y-2,'B');
LINPUT(X-18,Y-12,'C');
PMPM(NO,G,X-15,Y-10,X,Y-10);
X:=X-25;
Y:=Y-40;
PM(NO,X,Y);
WSTRING('THE SYMBOL');
END;
PROCEDURE ANDGCIR;
VAR X,Y:INTEGER;
BEGIN
X:=50;Y:=50;
PM(NO,X+40,Y+133);
WSTRING('+');
PMPM(NO,G,X+40,Y+133,X+40,Y+120);
VR(NO,G,Y,X+40,Y+120,0);
PM(G,X+60,Y+76);
WSTRING('OUTPUT');
P3MP(G,G,X+40,Y+76,X+40,Y+70,X+10,Y+70);
DIODES(X+10,Y+70,5,5,10,'1',FALSE,0);
PMPM(NO,G,X,Y+70,X-20,Y+70);
LINPUT(X-20,Y+70,'A');
MOVETO(X+40,Y+70);
P2M(G,X+40,Y+40,X+10,Y+40);
DIODES(X+10,Y+40,5,5,10,'2',FALSE,0);
PMPM(NO,G,X,Y+40,X-20,Y+40);
LINPUT(X-20,Y+40,'B');
MOVETO(X+40,Y+40);
P2M(G,X+40,Y+10,X+10,Y+10);
DIODES(X+10,Y+10,5,5,10,'3',FALSE,0);
PMPM(NO,G,X,Y+10,X-20,Y+10);
LINPUT(X-20,Y+10,'C');
MOVETO(10,10);
WSTRING("BASIC "AND" GATE CIRCUIT");
DANDG(X+130,Y+50);
END;
PROCEDURE DORG;
BEGIN
PM(NO,X,Y);
P3MP(G,N0,X-10,Y+20,X+25,Y,X+33,Y);
MOVETO(X+35,Y-2);
WSTRING('A+B+C');
MOVETO(X+30,Y);
P3MP(G,G,X+25,Y,X-10,Y-20,X,Y);
LINPUT(X-25,Y+8,'A');
PMPM(NO,G,X-20,Y+10,X-6,Y+10);
LINPUT(X-25,Y-3,'B');
PMPM(NO,G,X-20,Y,X,Y);
LINPUT(X-25,Y-13,'C');
PMPM(NO,G,X-20,Y-10,X-6,Y-10);
X:=X-25;
Y:=Y-40;
PM(NO,X,Y);
WSTRING('THE SYMBOL');
END;

PROCEDURE ORGCR;
VAR X,Y:INTEGER;
BEGIN
 X:=50;Y:=50;
PM(N0,X+40,Y+133);
WSTRING('-');
PMPM(N0,G,X+40,Y+133,X+40,Y+120);
VR(N0,G,V,X+40,Y+120,0);
PM(N0,X+60,Y+76);
WSTRING("OUTPUT");
P3MP(G,G,X+40,Y+76,X+40,Y+70,X,Y+70);
DIODES(X,Y+70,5,5,10,'1',TRUE,0);
PM(G,X-20,Y+70);
LINPUT(X-20,Y+70,'A');
MOVETO(X+40,Y+70);
P2M(G,X+40,Y+40,X,Y+40);
DIODES(X,Y+40,5,5,10,'2',TRUE,0);
PM(G,X-20,Y+40);
LINPUT(X-20,Y+40,'B');
MOVETO(X+40,Y+40);
P2M(G,X+40,Y+10,X,Y+10);
DIODES(X,Y+10,5,5,10,'3',TRUE,0);
PM(G,X-20,Y+10);
LINPUT(X-20,Y+10,'C');
MOVETO(10,10);
WSTRING("BASIC "OR" GATE CIRCUIT");
DORG(X+130,Y+50);
END;

PROCEDURE DNO R;
BEGIN
 Y:=Y-80;
PM(NO,X,Y);

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
P3MP(G,NO,X-10,Y+20,X+25,Y,X+33,Y);
MOBET0(X+35,Y-2);
WSTRING(‘A+B+C’);
M2(X+30,Y,X+35,Y+10);
WSTRING(‘———’);
MOBET0(X+25,Y);
P3MP(G,G,X+25,Y+3,X+30,Y+3,X+30,Y-3,X+25,Y-3,X+25,Y);
P3MP(G,G,X+25,Y,X-10,Y-20,X,Y);
LINP1(X-25,Y+8,’A’);
PM1(G,G,X-20,Y+10,X-6,Y+10);
LINP1(X-25,Y-3,’B’);
PM1(NO,G,X-20,Y,X,Y);
LINP1(X-25,Y-13,’C’);
PM1(NO,G,X-20,Y-10,X-6,Y-10);
PM(NO,X-20,Y-40);
WSTRING(‘THE SYMBOL’);
END;

PROCEDURE NORCIR;
VAR X:INTEGER;
BEGIN
X:=30;Y:=63;
PM(NO,X+90,Y+120);
WSTRING(‘+VO’);
MOBET0(X+90,Y+120);
VR(NO,G,V,X+90,Y+120,0);
P2M(G,X+90,Y+80,X+120,Y+80);
WSTRING(‘OUTPUT’);
M2(X+90,Y+80,X+90,Y+70);
MOBET0(X+20,Y+70);
TR(G,X+20,Y+70,’A’);
MOBET0(X+90,Y+70);
TR(G,X+90,Y+70,’B’);
PM1(NO,G,X+90,Y+70,X+160,Y+70);
TR(G,X+160,Y+70,’C’);
PM(NO,0,0);
WSTRING(‘BASIC TRANSISTOR "NOR" GATE CIRCUIT’);
X:=50;
DNO(RX+145,Y+70);
END;

PROCEDURE D skateboard;
BEGIN
Y:=Y-30;
PM(NO,X,Y);
P5MP(G,NO,X,Y+15,X+10,Y+15,X+10,Y-15,X,Y-15,X,Y);
MOBET0(X+10,Y);
PM1(NO,X+20,Y,X+23,Y);
WSTRING(‘ABC’);
MOBET0(X+10,Y+3);
P3MP(G,NO,X+15,Y+3,X+15,Y-3,X+10,Y-3);
MOBET0(X+25,Y+10);

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
WSTRING("---");
LINPUT(X-18,Y+8,"A");
MOVETO(X-15,Y+10);
P3MP(G,G,X,Y+10,X,Y,X-15,Y);
LINPUT(X-18,Y-2,"B");
LINPUT(X-18,Y-12,"C");
MOVETO(X-15,Y-10);
PMPM(G,NO,X-10,X-20,Y-40);
WSTRING("THE SYMBOL");
END;

PROCEDURE NANDGCIR;
VAR X,Y:INTEGER;
BEGIN
  X:=0;Y:=63;
  PMPM(NO,X+90,Y+120);
  WSTRING("+V0");
  MOVETO(X+90,Y+120);
  VR(NO,G,V,X+90,Y+120,0);
  P2M(G,X+90,Y+80,X+120,Y+80);
  WSTRING("OUTPUT");
  M2(X+90,Y+80,X+90,Y+76);
  TRANS(X+90,Y+76,BL);
  PM(G,X+90,Y+36);
  TRANS(X+90,Y+36,G);
  MOVETO(X+90,Y-4);
  TRANS(X+90,Y-4,V);
  PM(G,X+90,Y-50);
  PMPM(NO,X+84,Y-50,X+96,Y-50);
  PMPM(NO,G,X+86,Y-52,X+94,Y-52);
  PMPM(NO,G,X+88,Y-54,X+92,Y-54);
  PMPM(NO,G,X+60,Y+61,X+30,Y+61);
  LINPUT(X+30,Y+61,"A");
  PMPM(NO,G,X+60,Y+21,X+30,Y+21);
  LINPUT(X+30,Y+21,"B");
  PMPM(NO,G,X+60,Y-19,X+30,Y-19);
  LINPUT(X+30,Y-19,"C");
  PM(NO,0,0);
  WSTRING("BASIC TRANSISTOR "NAND" GATE CIRCUIT");
X:=50;
DNAND(X+145,Y+70);
END;

PROCEDURE DDIODE;
VAR X,Y:INTEGER;
BEGIN
  X:=50;Y:=50;
  PM(NO,X-40,Y+97);
  WSTRING("HIGH");
  PMPM(NO,G,X-10,Y+100,X+20,Y+100);
  DIODES(X+20,Y+100,5,5,10,"",TRUE,0);
  PMPM(NO,G,X+30,Y+100,X+60,Y+100);
PM(NO,X+63,Y+97);
WSTRING('HIGH');
MOVETO(X-40,Y+70);
WSTRING('CURRENT IS ALLOWED TO PASS');
MOVETO(X-40,Y+60);
WSTRING('THROUGH THE DIODE');
MOVETO(X+130,Y+130);
WSTRING('"PRESS RETURN"');
Y:=20;
MOVETO(X-40,Y+27);
WSTRING('HIGH');
MOVETO(X-10,Y+30);
PMPM(G,NO,X+30,Y+30,X+40,Y+30);
DIODES(X+40,Y+30,5,5,10,' ',FALSE,0);
PMPM(G,NO,X+70,Y+30,X-40,Y);
WSTRING('CURRENT IS NOT ALLOWED TO PASS');
MOVETO(X-40,Y-10);
WSTRING('THROUGH THE DIODE');
MOVETO(X+80,Y+25);
WSTRING('LOW');
REPEAT
  MOVETO(X+10,Y+140);
  WSTRING('-----> ');
  MOVETO(X+10,Y+140);
  WSTRING(' ');
  MOVETO(X-5,Y+47);
  WSTRING('----> ');%
  MOVETO(X-5,Y+47);
  WSTRING(' ');
  MOVETO(X-5,Y+40);
  WSTRING('<----');%
  MOVETO(X-5,Y+40);
  WSTRING(' ');
UNTIL KEYPRESS;
MOVETO(X+130,Y+160);
WSTRING('');
END;

PROCEDURE DTRANS;
VAR X,Y:INTEGER;
BEGIN
  X:=50;Y:=50;
  PM(NO,X-40,Y+97);
  WSTRING('HIGH');
  PMPM(NO,0,X-10,Y+100,X+20,Y+100);
  PMPM(NO,0,X+50,Y+115,X+50,Y+120);
  MOVETO(X+70,Y+120);
  WSTRING('LOW');
  P3MP(0,0,X+50,Y+120,X+50,Y+125,X+50,Y+115);
  TRANS(X+50,Y+115,G);%
  PMPM(NO,X+50,Y+70,X,Y+95);
END.
MOVETO(X-2,Y+75);
WSERVED(‘I’);
MOVETO(X-10,Y+60);
WSERVED(CURRENT’);
MOVETO(X-10,Y+50);
WSERVED(‘DIRECTION’);
MOVETO(X-40,Y-20);
WSERVED(THE TRANSISTOR IS CONDUCTED ”);
MOVETO(X-40,Y-30);
MOVETO(180,180);
WSERVED(”(PRESS RETURN)”);
REPEAT
MOVETO(X,Y+90);
P5MP(W,NO,X+15,Y+95,X+15,Y+70,X+20,Y+75,
X+15,Y+70,X+10,Y+75);
MOVETO(X,Y+95);
P5MP(B1,NO,X+15,Y+95,X+15,Y+70,X+20,Y+75,
X+15,Y+70,X+10,Y+75);
UNTIL KEYPRESS;
MOVETO(180,180);
WSERVED(””);
END;
BEGIN
END.
(*) FALAH AL-SAFAAR (*)

(* A UNIT OF SUBROUTINES FROM LIBRARY #2 *)

(*$S++*)
UNIT STUFF4;INTRINSIC CODE 24 DATA 25;
INTERFACE
USES TURTLEGRAPHICS,APPLESTUFF,STUFF1;
VAR I,P :INTEGER;G:CHAR;
PROCEDURE COUNTER;
PROCEDURE SHIFT;
PROCEDURE MOVES(N:CHAR;L1,L2,Y:INTEGER;FR:BOOLEAN);
PROCEDURE HELP6;
PROCEDURE COM6;
IMPLEMENTATION
PROCEDURE HELP;
BEGIN
WRITELN('"HELP PROCEDURE"');
WRITELN('TWO APPLICATIONS OF FLIP-FLOPS ARE SHOWN');
WRITELN('IN THIS PACKAGE: SHIFTER AND COUNTER.');
WRITELN;
WRITELN('WHEN THE USER INSERTS THE NUMBER OF A');
WRITELN('DEVICE FROM THE MAIN COMMANDS, ITS');
WRITELN('CIRCUIT WILL BE DISPLAYED ON THE SCREEN.');
WRITELN('THE DEVICE EXPECTS TO RECEIVE CLOCK');
WRITELN('PULSES TO ACTIVATE IT. THE RESPONSE OF');
WRITELN('THE SHIFTER IS SHIFTING A SIGNAL FROM');
WRITELN('THE INPUT OF THE LEFT FLIP-FLOP TO THE');
WRITELN('OUTPUT OF THE LAST RIGHT FLIP-FLOP.');
WRITELN('THE COUNTER COUNTS THE NUMBER OF PULSES');
WRITELN('BEING INSERTED BY THE USER. TO INSERT A');
END;
PROCEDURE HELP6;
BEGIN
TEXTMODE;
HELP;
WRITELN('PULSE FROM THE TERMINAL, THE KEY "P"');
WRITELN('SHOULD BE Pressed. THE MAXIMUM NUMBER');
WRITELN('OF PULSES THAT CAN BE INSERTED FOR THE');
WRITELN('SHIFTER IS 5 AND FOR THE COUNTER IS 7.');
WRITELN('IF THE USER WANTS TO EXIT BEFORE');
WRITELN('COMPUTING THE 5 OR THE 7 PULSES, THE');
WRITELN('KEY "T" SHOULD BE Pressed FOLLOWED BY');
WRITELN('CARRiAge RETURN.');
WRITELN('PRESS RETURN');
END;
PROCEDURE COM6;
BEGIN
TEXTMODE;
FOR I:=1 TO 10 DO WRITELN;
WRITELN('INSERT THE PROCEDURE NUMBER.');
WRITELN;
WRITELN('1:SHIFTER');
WRITELN('2:COUNTER');
WRITELN('3:HELP');
WRITELN('4:EXIT');
FOR I:=1 TO 7 DO WRITELN;
REPEAT READ(G);UNTIL((G='1')OR(G='2')OR(G='3')OR(G='4'));
END;
PROCEDURE MOVES;
BEGIN
PENCOLOR(NONE);
IF FR THEN
FOR I:=L1+2 TO L2-14 DO
BEGIN
MOVETO(I,Y+1);
WSTRING('');
IF N='1' THEN WSTRING('1')
ELSE WSTRING('0');
END
ELSE
FOR I:=L1-14 DOWNTO L2+2 DO
BEGIN
MOVETO(I,Y+1);
WSTRING('');
IF N='1' THEN WSTRING('1')
ELSE WSTRING('0');
END;
END;
PROCEDURE DRAW;
PROCEDURE T(X,Y:INTEGER);
BEGIN
PMP4(NONE,VIOLET,X,Y,X+40,Y,X+40,Y-30,X,Y-30,X,Y);
PMPM(NONE, GREEN,X+30,Y,X+30,Y+10);
PMPM(NONE, GREEN,X+30,Y-30,X+30,Y-60);
PMPM(NONE, GREEN,X+10,Y-30,X+10,Y-45);
PM(NONE,X+15,Y-15);
WSTRING('FF');
MOVETO(X+30,Y-29);WSTRING('T');
END;
BEGIN
T(30,170);T(130,170);
PMPM(NONE,GREEN,240,125,0,125);
PM(NONE,0,126);
WSTRING('CLOCK');
PMPM3(NONE,GREEN,260,175,220,175,220,105,100,105);
PMPM3(NONE,GREEN,160,175,120,175,120,115,100,115);
PENCOLOR(ORANGE);
MOVE4(100,120,80,120,80,100,100,100);
MOVETO(100,115);
PMPM(NONE,GREEN,80,110,60,110);
MOVETO(60,140);PM(NONE,260,181);
WSTRING("A1");
MOVETO(160,181);
WSTRING("A2");
MOVETO(60,181);
WSTRING("A3");
MOVETO(260,100);
WSTRING("1");
END;
PROCEDURE TABLE(I:INTEGER);
BEGIN
  PENCOLOR(NONE);
  IF I=0 THEN
    BEGIN
      MOVETO(10,80);
      WSTRING(" 0 0 1 0");
    END;
  IF I=1 THEN
    BEGIN
      MOVETO(10,72);
      WSTRING(" 1 0 1 0");
    END;
  IF I=2 THEN
    BEGIN
      MOVETO(10,64);
      WSTRING(" 2 0 1 1");
    END;
  IF I=3 THEN
    BEGIN
      MOVETO(10,56);
      WSTRING(" 3 1 0 0");
    END;
  IF I=4 THEN
    BEGIN
      MOVETO(10,48);
      WSTRING(" 4 1 0 1");
    END;
  IF I=5 THEN
    BEGIN
      MOVETO(10,40);
      WSTRING(" 5 1 1 0");
    END;
  IF I=6 THEN
    BEGIN
      MOVETO(10,32);
      WSTRING(" 6 1 1 1");
      MOVETO(10,30);
PROCEDURE MSS;
VAR P:INTEGER
CH:CHAR;
PROCEDURE M1;
BEGIN
MOVES('I',40,60,125,TRUE);
PMPM(NONE,GREEN,40,125,40,140);
MOVES('1',45,160,125,TRUE);
PMPM(NONE,GREEN,60,125,60,140);
PMPM(NONE,GREEN,120,125,120,140);
PMPM(NONE,GREEN,140,125,140,140);
MOVES('1',145,240,125,TRUE);
PMPM(NONE,GREEN,160,125,160,140);
PMPM(NONE,GREEN,220,125,220,140);
PM(NONE,232,126);
END;
PROCEDURE MAND(A1,A2,A3:CHAR);
BEGIN
MOVES(A1,260,220,175,FALSE);
MOVETO(222,176);WSTRING(' ');
MOVES(A1,220,140,105,FALSE);
MOVETO(163,171);
IF (((A1='1') AND (A2='1')) OR ((A1='0') AND (A2='0'))) THEN WSTRING('O')
ELSE WSTRING('1');
PMPM(NONE,GREEN,160,105,160,115);
MOVES(A1,155,100,115,FALSE);
MOVES(A2,122,176);WSTRING(' ');
MOVES(A2,120,100,115,FALSE);
IF ((A1='1') AND (A2='1')) THEN MOVES(A1,80,60,110,FALSE)
ELSE MOVES('0',80,60,110,FALSE);
MOVETO(102,106);
WSTRING(' ');
MOVETO(102,116);
WSTRING(' ');
MOVETO(62,111)WSTRING(' ');
MOVETO(63,171);
IF (A3='0') THEN IF ((A2='1') AND (A1='1')) THEN WSTRING('1')
ELSE WSTRING('0');
ELSE
BEGIN
PM(NONE,264,171);
WSTRING('0');
MOVETO(164,171);
END;
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
WSTRING('0');
MOVETO(64,171);
WSTRING('0');
P:=-1;
REPEAT
  READ(CH);UNTIL ((CH='P')OR(CH='T'));
IF CH='P' THEN
  BEGIN
    P:=P+1;
    IF P=0 THEN
      BEGIN
        NAND('0','0','0');
        ML;
        PM(0,263,171);
        WSTRING('1');
      END;
    IF P=1 THEN
      BEGIN
        MAND('1','0','0');
        ML;
        PM(0,263,171);
        WSTRING('0');
      END;
    IF P=2 THEN
      BEGIN
        MAND('0','0','0');
        ML;
        PM(0,263,171);
        WSTRING('1');
      END;
    IF P=3 THEN
      BEGIN
        MAND('1','1','0');
        ML;
        PM(0,263,171);
        WSTRING('0');
      END;
    IF P=4 THEN
      BEGIN
        MAND('0','0','1');
        ML;
        PM(0,263,171);
        WSTRING('1');
      END;
    IF P=5 THEN
      BEGIN
        MAND('1','0','1');
        ML;
        PM(0,263,171);
        WSTRING('0');
      END;
  END;
IF P=6 THEN
BEGIN
MAND("0", "1", "1");
M1;
PM(NONE, 263, 171);
WSTRING("1");
END;
END;
TABLE(P);
UNTIL ((P=6) OR (CH="T"))
END;
PROCEDURE COUNTER;
BEGIN
DRAW;
PM(NONE, 0, 0);
WSTRING("3-BIT BINARY COUNTER");
MOVETO(10, 100);
WSTRING("CLOCK");
MOVETO(10, 91);
WSTRING("PULSE A1 A2 A3");
MOVETO(10, 90);
PMPM(GREEN, NONE, 130, 90, 145, 70);
WSTRING("P= FOR INSERTING");
MOVETO(145, 60);
WSTRING("PULSES.");
MOVETO(145, 40);
WSTRING("T= FOR TERMINATION");
MOVETO(145, 30);
WSTRING("THEN TYPE RETURN.");
MSS;
END;
PROCEDURE TABL(I:INTEGER);
BEGIN
PENCOLOR(NONE);
IF I=0 THEN
BEGIN
MOVETO(10, 70);
WSTRING("0 1 0 0 0");
END;
IF I=1 THEN
BEGIN
MOVETO(10, 60);
WSTRING("1 0 1 0 0");
END;
IF I=2 THEN
BEGIN
MOVETO(10, 50);
WSTRING("2 0 0 1 0");
END;
IF I=3 THEN
BEGIN
  MOVETO(10,40);
  WSTRING(‘3 0 0 0 1’);
END;
IF I=4 THEN
BEGIN
  MOVETO(10,30);
  WSTRING(‘4 0 0 0 0’);
  MOVETO(10,29);
  PM(GREEN,125,29);
END;

PROCEDURE MOVESS;
VAR P:INTEGER;
  CH:CHAR;
PROCEDURE M1;
BEGIN
  PM(NONE,13,171);
  WSTRING(‘1’);
  MOVETO(62,17);
  WSTRING(‘0’);
  MOVETO(83,17);
  WSTRING(‘0’);
  MOVETO(132,171);
  WSTRING(‘0’);
  MOVETO(153,171);
  WSTRING(‘0’);
  MOVETO(202,171);
  WSTRING(‘0’);
  MOVETO(223,171);
  WSTRING(‘0’);
END;
PROCEDURE M2;
BEGIN
  MOVES(‘1’,240,200,100,FALSE);
  MMPM(NONE,GREEN,220,100,220,110);
  PM(NONE,223,161);
  WSTRING(‘1’);
  MOVES(‘1’,215,130,100,FALSE);
  MMPM(NONE,GREEN,150,100,150,110);
  PM(NONE,153,161);WSTRING(‘1’);
  MOVES(‘1’,145,60,100,FALSE);
  MMPM(NONE,GREEN,80,100,80,110);
  PM(NONE,83,161);WSTRING(‘1’);
  MOVES(‘1’,75,10,100,FALSE);
  MOVETO(12,101);WSTRING(‘1’);
  MOVETO(13,161);WSTRING(‘1’);
BEGIN
M1;
P:=0;
REPEAT
REPEAT READ(CH);UNTIL ((CH='P')OR(CH='T'));
IF CH='P' THEN
BEGIN
P:P+1;
IF P=0 THEN
BEGIN
M2;
MOVETO(13,171);
WSTRING('0');
MOVES('0',60,90,170,TRUE);
MOVES('0',130,160,170,TRUE);
MOVES('0',200,230,170,TRUE);
MOVETO(62,171);
WSTRING('1');
END;
IF P=1 THEN
BEGIN
M2;
MOVETO(62,171);
WSTRING('0');
MOVES('1',60,90,170,TRUE);
MOVETO(132,171);
WSTRING('1');
MOVES('0',130,160,170,TRUE);
MOVES('0',200,230,170,TRUE);
MOVETO(132,171);
WSTRING('1');
END;
IF P=2 THEN
BEGIN
M2;
MOVES('0',60,90,170,TRUE);
MOVETO(132,171);
WSTRING('0');
MOVES('1',130,160,170,TRUE);
MOVETO(202,171);
WSTRING('1');
MOVES('0',200,230,170,TRUE);
MOVETO(202,171);
WSTRING('1');
END;
IF P=3 THEN
BEGIN
M2;
MOVES('0',60,90,170,TRUE);
MOVES("0",130,160,170,TRUE);
MOVETO(202,171);
WSTRING("0");
MOVES("1",200,230,170,TRUE);
MOVETO(272,171);
WSTRING("1");
END;
IF P=4 THEN
BEGIN
M2;
MOVES("0",60,90,170,TRUE);
MOVES("0",130,160,170,TRUE);
MOVES("0",200,230,170,TRUE);
MOVETO(272,171);
WSTRING("0");
END;
END;
TAB(P);
UNTIL ((P=4)OR(CH="T"))
END;
PROCEDURE SHIFT;
PROCEDURE FF(X,Y:INTEGER;CH:CHAR);
BEGIN
PM(NONE,X+2,Y-10);
CASE CH OF
  "A":BEGIN
    WSTRING("DA");
    MOVETO(X+26,Y-10);
    WSTRING("QA");
    END;
  "B":BEGIN
    WSTRING("DB");
    MOVETO(X+26,Y-10);
    WSTRING("QB");
    END;
  "C":BEGIN
    WSTRING("DC");
    MOVETO(X+26,Y-10);
    WSTRING("QC");
    END;
  "D":BEGIN
    WSTRING("DD");
    MOVETO(X+26,Y-10);
    WSTRING("QD");
    END;
END;
PMMP4(NONE,BLUE,X,Y,X+40,Y,X+40,Y-40,X,Y-40,X,Y);
PMMP(NONE,GREEN,X,Y-20,X-10,Y-20);
MOVETO(X-10,Y-80);
PMMP(NONE,GREEN,X+40,Y-10,X+70,Y-10);
PM(NONE,X+10,Y-25);
WSTRING("FF");
END;
BEGIN
PMPM(NONE,GREEN,20,170,20,170);
FF(20,180,"A");
FF(90,180,"B");
FF(160,180,"C");
FF(230,180,"D");
PMPM(NONE,GREEN,10,100,275,100);
PM(NONE,240,101);
WSTRING("CLOCK");
PMPM(NONE,GREEN,0,170,20,170);
PM(NONE,50,0);
WSTRING("4-BIT SHIFT REGISTER");
MOVETO(145,70);
WSTRING("P = FOR INSERTING ");
MOVETO(145,60);
WSTRING("PULSES.");
MOVETO(145,50);
WSTRING("T = FOR TERMINATION");
MOVETO(145,40);
WSTRING(" THEN TYPE RETURN.");
PM(NONE,10,90);
WSTRING("CLOCK");
MOVETO(10,81);
WSTRING("PULSE QA QB QC QD");
MOVETO(10,80);
PM(GREEN,125,80);
MOVESS;
END;
BEGIN
END.
BIBLIOGRAPHY

Books


Magazines


Johnson D. Lamont; Willis, J.; and Danley, D. "A Comparison of Traditional Computer-Base Methods of Teaching Students to Administer Individual Intelligence Test," AEDS Journal, 16 (Fall, 1982): 56-63.


Unpublished Materials


Kountanis, D. "Computer Structure." Notes compiled at Western Michigan University, Kalamazoo, Michigan, Fall, 1981.