

1990

# HyperCard-based learning environment for DIADES

Ali A. Shamsapour  
*Portland State University*

Follow this and additional works at: [https://pdxscholar.library.pdx.edu/open\\_access\\_etds](https://pdxscholar.library.pdx.edu/open_access_etds)



Part of the [Electrical and Computer Engineering Commons](#)

Let us know how access to this document benefits you.

---

## Recommended Citation

Shamsapour, Ali A., "HyperCard-based learning environment for DIADES" (1990). *Dissertations and Theses*. Paper 4128.

<https://doi.org/10.15760/etd.6011>

This Thesis is brought to you for free and open access. It has been accepted for inclusion in Dissertations and Theses by an authorized administrator of PDXScholar. Please contact us if we can make this document more accessible: [pdxscholar@pdx.edu](mailto:pdxscholar@pdx.edu).

AN ABSTRACT OF THE THESIS OF Ali A. Shamsapour for the Master of Science in Electrical and Computer Engineering presented February 16, 1990.

Title: HyperCard-based Learning Environment for DIADES

APPROVED BY THE MEMBERS OF THE THESIS COMMITTEE:



Marek Perkowski, Chairman



Rajinder Aggarwal



Maria Balogh

This thesis is an attempt to create a HyperCard-based learning environment to teach DIADES and other related material. It is a departure from the classical Computer Aided Instruction methods towards a more flexible and user-controlled design. The goal was to set the foundation of a new CAI design which would closely resemble a Hyper-Text system. These systems are characterized as having interconnections between related concepts in the CAI environment.

Creating HyperCard stacks to carry out the DIADES-related material is the first of its kind. The environment provides text editing capability as well as drawing and painting tools. Using HyperCard to create CAI lessons has made it much easier to design

and use such lessons. The most important characteristic of this Composite lesson design probably lies in the fact that it provides an open-end structure for future DIADES developers to incorporate their pieces of DIADES documentation into the existing database.

The unique features of the tutorial can be described as:

- a) The user has control over the presentation of the material. Usually there are several options available to the user to choose from.
- b) It can be changed and modified at any time if DIADES research group has produced new results.
- c) The information is organized into several major topics. Subtopics are included wherever they exist. The organization of the lesson can be compared to a tree-like structure. More specific and more specialized subjects reside on remote branches.
- d) Incorporation of the related subjects into the DIADES tutorial allows expansion of the lesson to a degree which would be a comprehensive tutorial for most digital courses.

**HYPERCARD-BASED LEARNING ENVIRONMENT FOR DIADES**

by

**ALI A. SHAMSAPOUR**

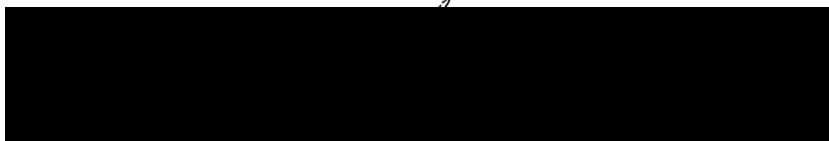
A thesis submitted in partial fulfillment of the  
requirement for the degree of

**MASTER OF SCIENCE**  
in  
**ELECTRICAL AND COMPUTER ENGINEERING**

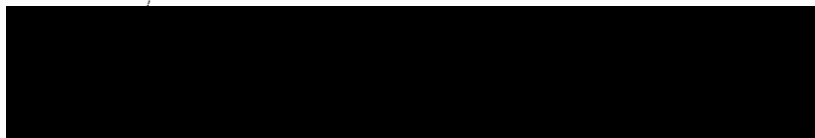
**Portland State University**  
1990

TO THE OFFICE OF GRADUATE STUDIES:

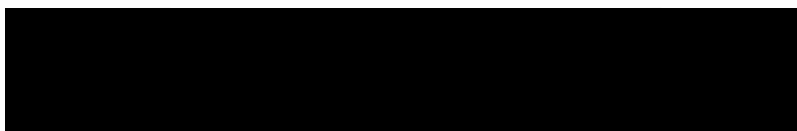
The members of the committee approve the thesis of Ali A. Shamsapour presented February 16, 1990.



Marek Perkowski, Chairman

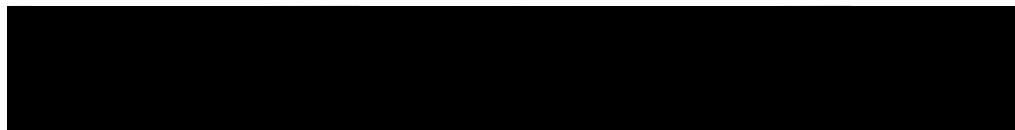


Rajinder Aggarwal

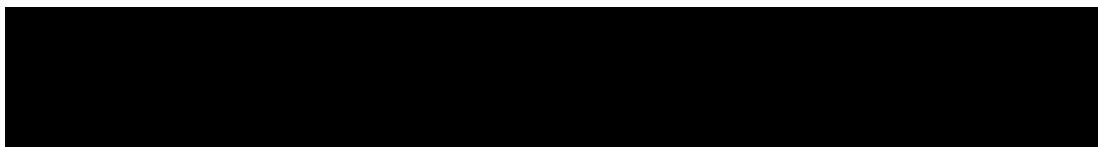


Maria Balogh

APPROVED:



Rolf Schaumann, Chairman, Department of Electrical Engineering



C. William Savery, Interim Vice Provost for Graduate Studies and Research

## ACKNOWLEDGEMENT

I would like to thank Dr. Marek A. Perkowski for providing valuable instructions, new ideas, hints, related material and overall guidance all the way through the preparation of this paper.

I also appreciate comments and guidance of Dr. Maria Balogh and Dr. Rajinder Aggarwal which helped me to improve the quality of the thesis.

Research of the PSU DIADES group provided valuable documentation about DIADES design automation system which were used during the preparation of this paper.

## TABLE OF CONTENTS

	PAGE
ACKNOWLEDGEMENTS .....	iii
LIST OF TABLES .....	vi
LIST OF FIGURES .....	vii
CHAPTER	
I INTRODUCTION .....	1
II CHARACTERIZATION OF THE CLASSICAL CAI	
METHODS .....	5
CAI Lesson Designs .....	6
CAI Systems .....	13
III USE OF HYPERCARD FOR COMPUTER AIDED INSTRU-	
TION .....	16
Buttons, Fields, Cards, Backgrounds and Stacks .....	21
Message Box .....	25
Multifinder .....	27
Mouse Functions .....	27
Power Keys and Blind Typing .....	30
Click and Double Click .....	31
Option, Command and Shift Keys .....	31
HyperTalk .....	32

	Home Stack .....	38
	Copying and Pasting from Other Stacks .....	39
	HyperCard in CAI .....	39
	The Idea of the Learning Environment for DIADES .....	40
	How HyperCard Interacts with the User .....	42
IV	COMPOSITE CAI METHOD AND HIERARCHICAL, GRAPHICAL HELP SYSTEMS .....	45
	Introduction to Our New Concept .....	45
V	THE DIADES SYSTEM .....	52
	What is DIADES? .....	54
VI	LESSON DESIGN .....	60
	The Actual Design .....	61
VII	CONCLUSION .....	76
	Preparing the Tutorial .....	77
	What Was Achieved and What Not? .....	78
	How Can the Current Design be Improved? .....	79
	REFERENCES .....	81
	APPENDICES	
A	SOME SAMPLE SCRIPTS .....	83
B	DESIGN DATA FOR SIGNAL DELAY PROCESSOR .....	114



## LIST OF TABLES

TABLE		PAGE
I	Comparison Between Composite and Classical CAI Methods .....	47

## LIST OF FIGURES

FIGURE		PAGE
1.	Inheritance Path for HyperCard .....	18
2.	Macintosh Menus and Tools Menu .....	20
3.	Some Icons and a Button Dialogue Box .....	22
4.	Field Properties and Available Fonts .....	24
5.	The Layers and the Message Box .....	26
6.	HyperCard Object Hierarchy for Resources .....	35
7.	"Home" Stack's "User Preferences" Card .....	37
8.	Dialogue Boxes for "ASK" and "ANSWER" Commands .....	43
9.	Design Process Steps in DIADES .....	53
10.	First and Second Card of the "Start" Stack .....	62
11.	Sample Cards of the "HELP" Stack .....	65
12.	Two Sample Cards of the "DIADES" Stack .....	66
13.	Sample Pop Quiz and Evaluation Cards .....	71
14.	Sample Index Cards of the "ADL" Stack .....	72
15.	First and Second Card of the "Home" Stack .....	74
16.	The Parallel Control Flow Diagram .....	126
17.	Layout of the Control Part of the System .....	127

## CHAPTER I

### INTRODUCTION

Using computers in education has concerned most educators from the time that computers became more available to the public. CAI (Computer Aided Instruction), is one of the areas that has been developed in the past to help the teacher and students to improve their educational performance. Along with the development of the computer hardware and consequently computer software, CAI developers started to create more sophisticated programs and tools to help satisfy increasing demand for supplementary means for education. Considering the amount of dependency that the modern life has on computers, there is no doubt that their use in education is no exception.

Computer Engineering and computer science students are two groups of people who are most involved in creating and using educational software. Electrical and Computer Engineering department in Portland State University is now preparing grounds to create and develop a unified CAI lesson design which not only will be used by the students, but also will be modified and improved as time goes on and the computer technology grows rapidly. A CAI lesson environment in Electrical and Computer Engineering department is being created and developed under the supervision of Dr. Marek Perkowski to aid the computer engineering students to acquire knowledge about DIADES, a hardware design automation system, and its programming language ADL.

In creating such an environment there were few decisions to make as to what kind of computer or what sort of software is to be used to make the task of creating, modifying and learning easier for both users and designers of the CAI lesson. Choice was made to use the Macintosh computer for its user friendly interface, and HyperCard software

which resides on Macintosh. HyperCard gives the designer of a CAI lesson the ability to create HyperText systems which are characterized by having linkages between similar ideas and concepts in different parts of the system.

The idea for creating such environment is to be able to describe DIADES system as it is known at the time of this writing, and to expand the environment later as it gets used by the students and engineers. It is expected that the use of the system will introduce some practical problems, and consequently will require the system to be improved and explained in more details. In HyperCard, both textual and graphical materials are stored in form of a stack. Any new material can be incorporated to an existing one by making a link between them. The existing material also can be modified or left unchanged.

With these goals in mind we started to create a HyperCard-based learning environment for DIADES design automation system. A tutorial about microprogramming was already written by Mark Von Presentin which is also included as part of the whole tutorial. In the following chapters some characteristics of the classical CAI lesson are described. This is intended to serve two purposes. first it will give us some ideas as to where we stand with regard to capabilities of the present CAI lessons. Second, it will introduce the existing shortcomings and aspects to be improved. Then we will see how the Macintosh and HyperCard will help us to overcome some of the obstacles that the early CAI lesson designers had to face.

According to Robert Burke [5], in the book "CAI source book", a classical Computer Aided Instruction (CAI) program approximates the performance of a human tutor and frequently asks questions which require the learner to formulate an answer after some thought. Ordinarily short answers composed by a learner must match perfectly, character for character, with the correct answer in order to be recognized by a basic CAI program. A simple computer subroutine analyzes the essentials of the correct answer and

of the learner's input. The learner's answer is then categorized as:

- a) A perfect match to the correct answer;
- b) Not a perfect match but correct in the essentials;
- c) Incorrect.

Reasonable misspellings and correct answers in excess verbiage fall into category (b). The computer can be programmed either to accept category (b) answers or to require a perfect match. This subroutine can be used to create lessons in conjunction with text files containing embedded commands from a word processor. In other words the questions in the lesson will be presented in a way that the answers to them can be categorized by the subroutine as correct or incorrect.

In classical CAI the computer largely controls the process of learning, presenting the reading material to the student to read and then periodically giving the student questions to answer or problems to solve. Language for creating CAI lessons is called CAI authoring language. Common PILOT is name of one of the most popular languages used for CAI. The big advantage of CAI authoring language is that it saves time for the CAI author. It was estimated to be about six times faster to create CAI lesson by using language like PILOT than using a more general purpose language like Basic. The first generation of CAI authoring systems were limited in capability and such systems used also lots of memory. They were also prone to bugs which rendered them inoperable until regenerated by the vendor, and they sometimes lost data or lessons generated by users.

In classical CAI, since a classroom teacher may not be present when the courseware is to be used, it is important to apply a systematic model of instruction design to CAI courseware development to make the end product usable and effective. Classical CAI programs did not use graphics, contained more dogma and less experimentation. It exploited only small part of the computer's power and the student had to memorize right answers and enter them exactly word by word.

However, classical CAI approach can be developed and made more flexible and useful when implementing HyperText systems by using the HyperCard. HyperCard is an authoring tool and an information organizer for Macintosh computers. According to the creator of the HyperCard, Bill Atkinson, one can use it to create stacks of information to share with other people or to read stacks of information made by other people. Stacks are largest units of data and graphics which are created by HyperCard. These stacks can contain everything that can be found and used by a HyperCard author. It is both an authoring tool and sort of a "video cassette player" for information. HyperCard is like many different programs combined into one, graphics program, programming environment, and ability to do a lot of things with these tools are the characteristics of HyperCard. It has an "opening up" potential which allows a user to go inside a stack that somebody else has written, see how it was done, and modify it and tweak it a little to tune it for his uses and learn from what someone else has done.

## CHAPTER II

### CHARACTERIZATION OF THE CLASSICAL CAI METHODS

Classical CAI used several design methods which will be discussed shortly. These methods have evolved from CAI practice with large computers. Albert E. Hickey [6], in the book "Computer Assisted Instruction", and Robert L. Burke [5], in the book "CAI source book", have classified classical CAI methods with respect to their use and design. They have mentioned about three aspects of CAI lesson design that must be considered by the CAI author in choosing a design.

First is a functional design, what instructional purpose or educational function is expected. Is that primary or supplementary means of education, and so on. These are the questions to be considered when determining functional aspects of a lesson design.

Second aspect of a CAI lesson design is that of the computer usage (using the HyperCard for CAI lesson design will improve this aspect of the consideration). This applies to the kind of hardware and instructional software available for a CAI lesson designer.

Third is the logical design of the lesson that depends largely on instructional considerations in the learning task analysis. This aspect applies to the way in which a lesson can or should be learned. Which material should be presented first, and which one second. Or, what successive steps should be included into the entire learning process. According to Robert Burke [5], CAI lesson designs basically depend on instructional function. These methods are described below.

## CAI LESSON DESIGNS

### 1. Drill and Practice Design

This method is used to facilitate the learning of a drill and practice material, which requires also additional reinforcement which is not available within most classroom environments. Drill and practice programs do not teach, they simply allow students to practice skills they have already learned somewhere else. These programs present the learner with a problem to be solved or a question to be answered. Questions could be assigned to have a yes/no format, or multiple choice format. Fill in blanks are also common.

The criticism about drill and practice design is that computers are too expensive to be used as automatic flashcard machines. Another concern is that such an approach may influence a student's perception of instruction by reducing learning to a game that implies that all questions have only one correct answer.

Another criticism is that drills do not teach but merely practice with the assumption that the student is already familiar with the information to some degree. In computer aided instruction this means that the drill should be preceded with an appropriate tutorial. It may also mean preceding the computer based drill with reading a textbook or with a classroom lesson.

It has been also criticized of having a low quality, of not incorporating good instructional principles, and of not collecting enough useful information to show the instructor how well a student is progressing. The response-judging procedures are frequently poorly programmed so that reasonable responses are sometimes judged as being incorrect.

Advantage of using this method of instruction is that computers are well suited for presenting repetitious tasks and patiently responding to students.



## 2. Tutorial Design

In this design method the computer simulates the actions of a very good tutor. It works best for the initial presentation of a new material. It also works best with highly verbal material which lends itself to the narrative description techniques as well as to the question and answer techniques. The characteristics of this design include:

- a) An ordered sequence of instruction.
- b) The presentation of information in small increments.
- c) Active student responses.
- d) A narrow range of possible answers.
- e) Provisions to reinforce correct responses while informing the learner about his or her progress.

Some students find this design to be tedious and boring. The linear instructional design in a tutorial lesson requires a learner to successfully complete each task in sequence.

## 3. Gamelike Design

The motivation of a student can sometimes benefit greatly when the objectives of a CAI lesson can be accomplished by using a gamelike approach. In this type of design the subjects to be learned are usually hidden inside a game, and the student will encounter them as he makes progress with the game. The gamelike type of design is especially useful for subjects that are either not easy to learn, or for an abstract type of material which requires special interest and attention to learn. The student will be motivated by the game which will ease the task of learning.

An example of a computer game is for instance when the computer provides the player with four letters and asks the player to create as many words as possible. Each letter can be used once or more or not used at all. These letters can be specified either by

the player or by the computer and may be constrained by having a specific letter in a specific position in the word. After all the words are created by the player, the computer fills in the remaining words from its dictionary and assigns a score to the player.

#### 4. Problem Solving Design

This method allows the student to use computer to solve problems. In some problem-solving models the computer is used as an intelligent calculator which monitors the student's actions step by step. The process which the student uses is analyzed at each step and the student is given a feedback immediately. In other models the student only proposes a solution and the computations are carried out by the computer.

An example of using this method is when the student must solve the problem of "making the turtle move in a square". Through the experimentation and help from other people, the student discovers how to make the turtle move along a straight line, turn in different directions, and eventually move in a square. The student deals then with progressively more difficult problems, such as making the turtle move in a circle and getting the turtle to solve math problems.

#### 5. Combined Functional Design

Combinations of some functional designs can make an interesting single synergistic system. Example of a combined functional design might be combining the drill and practice design with a gamelike design. Students get some time to enjoy the fruits of their expertise if they provide the number of expected answers per minute.

#### 6. Linear Design

Linear design is the most common of all CAI designs. It is directly patterned after linear, paper and pencil programmed instruction and after a great deal of CAI courses which have been written for large computers. Linear CAI does not exploit the powers of the computer fully, but it works. It is the easiest design to use and to revise or

validate by making necessary changes to the lesson, therefore it is a good design for the beginning CAI author to implement.

In a linear lesson design, the lesson progresses from one topic or concept to the next one, first presenting information and then asking questions. All students go through the presentations and questions in the same order, and the order does not change regardless of whether students answer questions correctly or incorrectly. The sequence in a linear lesson is determined by the author. One way to determine sequence is through the hierarchy of information. For instance in math, being able to perform addition is necessary before one can learn multiplication. Addition, subtraction and multiplication are next needed to learn long division. Thus, most arithmetic curricula begin with addition, followed by subtraction, multiplication and division.

The determination of the sequence may be also based on the familiarity or difficulty of the information. Vocabulary instruction usually begins with words of higher usage frequency. Next come words that are less common.

In a CAI lesson which has a linear design, each student is presented with the same material as every other student. Even if a linear design is able to provide different branches of information depending on different response from the students, it is still a linear design from the instructional point of view, and not a branching design.

## 7. Spiral Design

In this method the lesson logic flows in a spiral through the material to be learned, each time dwelling on a different property of the material. An example of the spiral design would be for instance first to provide information about what is a logic design and what steps are involved in going through the logic design process. Then to start the lesson over again dwelling on how to design each part of the process.

The logic behind spiral design is that it constantly keeps the learner aware of the whole lesson, and this way reminds him what is the most important to learn and where

the emphasis should be given to. In a lesson design which does not use the spiral method, the student may spend too much time on learning something that is not really essential in the whole lesson structure.

### 8. Branching Design

The term "branching design" denotes the kind of lesson design in which there are several different branches. A branching design is based on an instructional design which includes alternative tracks through the lesson. Depending on the performance of the student he may see different material than other students going through the same lesson. Typically the branching occurs following a criterion question frame. Students making a correct response would all take the same successive track. Students making incorrect responses would take one or another alternative track, depending upon the exact nature of their incorrect answer. Branching designs take advantage of the great power of the computer to individualize the lesson and to provide a personalized learning experience for each student. As a result the student who is capable of moving quickly through a lesson, is enabled to do so. The student who needs more time and more practice, may progress slower, at his pace, but also accomplish the assumed learning task.

### 9. Multitrack Design

Material is written on several distinctly different levels to permit the individualization of the lesson. The top track or the highest level is likely to be the shortest. Material is likely to be more abstract, with less explanation and at a higher reading level. With the lowering of tracks more explanations will be added and more examples will be given. Depending on the performance of the students multitrack branching will allow students to branch to another level or to stay in the same level.

A multitrack design allows more advanced students to be able to choose the top track which is the shortest and will take much less time to complete. If a learner feels

that he is not able to follow the lesson as it is being presented, then he can switch to a lower track which has more explanations and more examples.

### 10. Regenerative Design

The lesson can generate a different set of problems for each student or for each iteration of the lesson for the same student. Advantage of this method is that the student can use the same program over and over again. Each time, the program will present a new set of problems and examples which will look like a new program, so that the student can strengthen his skills. Some of the newer regenerative lesson designs permit the teacher to specify the type of arithmetic problems that the student will encounter in each iteration.

### 11. Adaptive Design

This method uses the data that is accumulated as they are used with a particular population as a basis for the self improvement of the lesson. This type of design improves as it is used and more experience is achieved. In other words, the adaptive design is more like a self-test on other types of CAI lessons. It does not pertain much as to how a lesson is presented, but rather to how a design can be improved. In fact it is an additional part of other design types, and can be implemented by installing appropriate handlers in some parts of the tutorial.

### 12. Logical Design

This is another way of considering different types of lesson designs in which the logic of the design is the main consideration. The terminology used to describe lesson logic comes largely from the behavioral psychology of learning. Indeed CAI lesson design owes much to the development of the behavioral learning theory. This type of design may get considerable attention in the future. In the same way as with the adaptive design, logical design does not represent a specific design type. The term is used to

denote a category of design in which the focus is on the issue of human learning habits with regard to some specific subjects.

### 13. Didactic Design

In this type of lesson design the student is presented with information and then asked to respond to questions, basically giving the same information back. Typically the student is presented the information in small "steps" and is asked to do minimal synthesis or manipulation of the material. The purpose is to convey information to the student, to provide him or her with a minimum opportunity for practice, and then to check retention, if not understanding.

### 14. Discovery Design

Discovery design can be very effective when it is used with material for which it is well suited. Design involves creating conditions within which students can reach insights on their own. Normally students are supplied with only as much material as is needed to reach an insight, for example about the relationship among a set of facts. This type of design maybe more useful in subjects areas such as humanities, philosophy and social science, since most people are more or less familiar with these subjects. Presenting a set of facts can stir student's curiosity to find out about the missing part.

### 15. Egrul Design

Egrul is an acronym for a lesson design in which the instructional logic proceeds from an example (EG) to a rule (RUL). Typically the student would be provided with a training and an opportunity to practice using sets of examples in order to determine some property which all the members of the class had in common. EGRUL designs are in fact a form of discovery learning and are inductive in nature.

Difference between EGRUL and DISCOVERY design is that the former starts with examples only, while in the latter any kind of information is used to help the

students think about the subject to get an insight. This type of design is also more applicable to humanities and similar subjects.

### 16. Rule Design

In this type of lesson design the instructional logic proceeds from a rule (RUL) to an example (EG), and basically is a deductive process. The student is typically taught to apply a law, a principle or some other form of a rule, to a set of examples.

### 17. Fading Design

Fading design is useful for content which has to be memorized, such as poetry or anatomy terms. The term "fading" refers to the fading of prompts. The prompts can be thought of as the aids that the CAI author builds to help the student develop the answer to the questions of the lesson. The lesson starts with frames (amount of information that is presented on the computer display screen at any time), containing very strong prompts, and then changes to frames with gradually weaker prompts as the lesson progresses.

The classical CAI approach uses several systems for implementing the lessons. Examples of these systems include PLATO, LOGO and TICCIT.

## CAI SYSTEMS

In the same book (CAI source book [5]), the author talks about several computer instruction systems which were developed to aid users and authors of the CAI lessons. More common of these systems were called PLATO, LOGO and TICCIT.

### PLATO

PLATO was the name of the first project aimed at using computers in education. It began in 1960 with the goal of designing a large computer-based system for instruction. The PLATO project in early 1970's introduced PLATO IV, a large time shared instructional system. Students study on individual terminals, hundreds of which are

connected to a large computer on which all lessons and student data are stored. All program execution occurs on the main computer. PLATO IV allows up to 600 students to use the computer simultaneously. It also allows authors to develop instructional materials at the same time that students are studying lessons.

## LOGO

LOGO was a major development of Papert's project who claimed that the students can learn many problem-solving skills on their own, given the correct educational environment which was an easily programmed computer. Papert believes that the students can transfer the powerful ideas of LOGO through a discovery learning environment. LOGO used text as well as graphics, which is called Turtle graphics. In the Turtle mode, a small triangle is in the center of the screen. This Turtle moves about the screen making line drawings in response to simple program instructions. Since the students tend to identify with the movement of the Turtle, the activities in this packet are first introduced as spatial movements in which the student acts out the role of a Turtle, following commands of FORWARD, BACK, RIGHT and LEFT. To produce graphics with this software, commands need to be entered to give instruction to each part of the plot. This means that the graphics is created by command line arguments. Of course there are ways to combine several commands in one to do the job of creating the graphics easier.

## TICCIT

In 1972 the Mitre corporation and Brigham Young University started development of the TICCIT (Time Shared Interactive Computer Controlled Instructional Television) system. There are two primary considerations in the original TICCIT project. The first is that the instruction be expository, with direct explication to the student, of various facts, rules, concepts and principles. This presentation is followed by the examples of, and then practice with, these facts, rules, concepts and principles. The second primary



consideration in TICCIT was that the sequence of instruction be learner controlled. The student has control not only over the sequence through a curriculum, but also within any part of that curriculum, over the presentation order of initial explanations, examples, practice, exercises, tests and even the difficulty level of the material. The drawback of the TICCIT system is that it restricts the instruction primarily to a strictly expository approach, which precludes simulations, games, and other more creative uses of the computer.

With the TICCIT system the students can study lessons presented on standard color televisions and interact through modified typewriter keyboards, all of which are controlled by a minicomputer. Intended for adult instruction, TICCIT embodied an instructional philosophy called learner controlled instruction (LCI). Lessons on TICCIT always included a variety of information presentations, examples, practice problems, tests and a map of the structure of the curriculum. Keyboard had additional keys labeled RULE, EXAMPLE, PRACTICE, EASY, HARD and ADVICE. By pressing these keys the student can change instructional activities, ask for easier or harder activities or ask for advice as to what to do. LCI had two advantages. First, the students can adopt the sequence of instruction to their own pace and learning styles. Second, the instructional developers did not have to make complex decisions about the content sequence, because the students can make their own sequencing decisions via the keyboard.

## CHAPTER III

### USE OF HYPERCARD FOR COMPUTER AIDED INSTRUCTION

In summer of 1987 Apple Computer Company introduced a new software for Macintosh computers. This software was called HyperCard. In the short life of the HyperCard a lot of material and programs, including graphical data bases, instructional material and computer games have been created by using this software. There are certain characteristics of HyperCard which make it a unique programming environment. They can be described as below.

a) Flexibility of the software.

This is accomplished by introducing several new ideas such as for instance fields and buttons which are treated as objects and can be placed in any desired locations on the display screen. Buttons are especially useful in creating links between different parts of a database. This is intended to enable the author to create HyperText systems which have been discussed in many conferences. The user of a HyperText system will not only learn about the subject matter, but also will be given access to any other subject which has something in common with the present material. Philosophically the idea of HyperText is extracted from the way that the human mind works. HyperCard takes a big step forward in helping HyperText systems come to existence.

b) Ease of use.

After a HyperCard-based material is created, using of the material will not take more than clicking a mouse button. A button which can activate hundreds of lines of code, is activated by a mouse click. This applies to fields, cards and

backgrounds as well. A mouse click on top of a card can activate all or part of the code written in the script of that card. Scripts of the cards, backgrounds, fields, buttons and stacks are part of these objects. They can be left empty, or a number of codes can be placed in them. If these scripts contain codes, they will respond to a handler. Handlers are discussed later in this chapter. The code is written in HyperTalk, the programming language for HyperCard, and placed in the script of a card which is a part of the card. A background (explained later in this chapter) has the same kind of script, with the only difference that it is one layer down in the inheritance path (HyperCard's hierarchy in receiving messages from a mouse, see Figure 1) and will receive messages only after they have gone through the card's script. A field has a name and a number and can be referenced by either one from anywhere in the stack. A stack is a collection of cards, backgrounds, fields and buttons which can be moved or copied altogether. Unlike the conventional computers in which the user has to know certain commands and procedures to be able to use a software, in HyperCard what the user does is only to follow instructions on the screen.

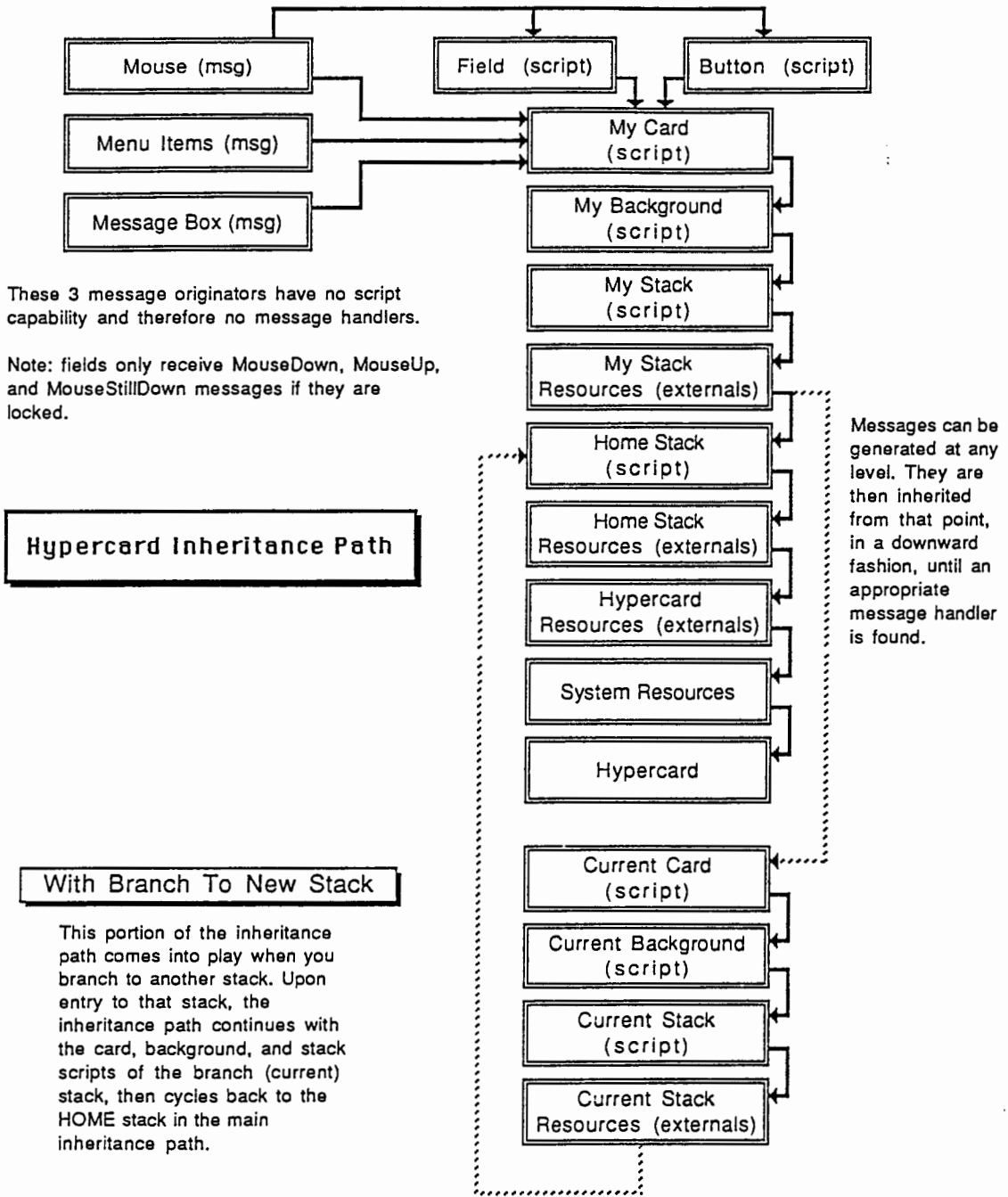
c) English-like language.

HyperTalk which is the programming language for HyperCard, is more like English. HyperTalk commands can be easily understood and used. Consider following lines of code:

*get the number of cards in this stack*

*put it into first line of card field id 20*

Using natural language makes it much easier to understand HyperCard commands as well as to know what they are supposed to do. It also eliminates the need to comment lines inside programs which is common in other programming languages. External commands and functions are very common in HyperCard. One can write a program in C or Pascal or even in assembly language, and make a



These 3 message originators have no script capability and therefore no message handlers.

Note: fields only receive MouseDown, MouseUp, and MouseStillDown messages if they are locked.

**Hypercard Inheritance Path**

**With Branch To New Stack**

This portion of the inheritance path comes into play when you branch to another stack. Upon entry to that stack, the inheritance path continues with the card, background, and stack scripts of the branch (current) stack, then cycles back to the HOME stack in the main inheritance path.

Messages can be generated at any level. They are then inherited from that point, in a downward fashion, until an appropriate message handler is found.

Figure 1. Inheritance path for HyperCard

new command by placing the executable code in an appropriate place. This ability is known as the external interface. It provides the means to rewrite or create new HyperTalk language commands (XCMD's) and functions (XFCN's). According to Danny Goodman in the book "HYPERCARD developer's guide", writing an XCMD requires working knowledge of a programming language Pascal, C, or Assembler, and a working knowledge of the Macintosh programming environment.

The names and parameters of XCMD's and XFCN's are sent along the inheritance path (explained later in this chapter) like any other message. If no message handler is found in the current card, background or stack script, the resource fork of the current stack is checked for an XCMD or XFCN that corresponds to the name of the message sent. See Figure 1 for the complete inheritance path for HyperCard.

d) Graphics included.

HyperCard uses several graphics tools which are under "tools" menubar. Menubar is a list of available tools and functions which is provided at the top of the Macintosh screen. Each word of this list represents a column of items with similar functions. Creating pictures is as easy as drawing with pencil on a piece of paper, or even easier, since movement of the tools can be controlled by keyboard keys.

e) Sound and music.

Sound and music can be created and played in HyperCard. A HyperTalk command "play" allows playing notes of music, or a digitized sound.

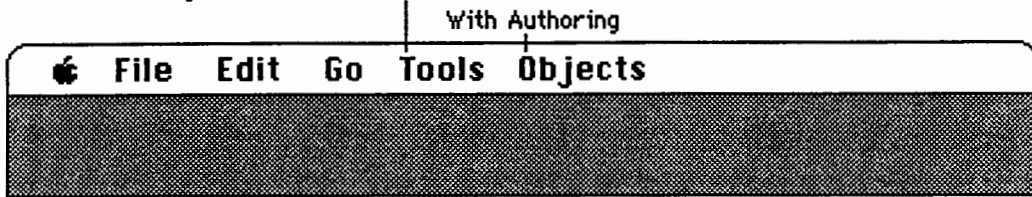
These were some of the features that HyperCard carries, and these are the reasons why this author has selected HyperCard to create computer aided instruction lesson designs. These characteristics make computers usable to those who do not have com-

---

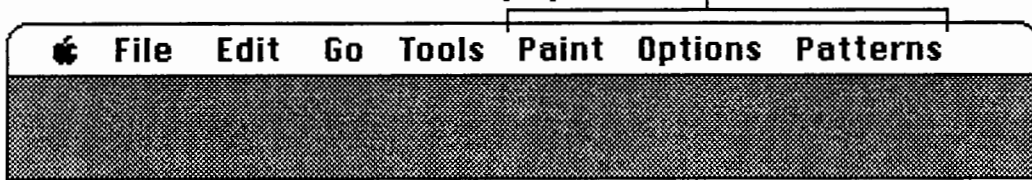
## Menus

---

With Painting checked on the Home stack User Preferences card

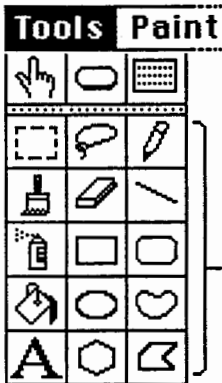


While using any Paint tool



### The Tools menu and window

HyperCard is preset to use just the Browse tool—the one that looks like a hand. When you've had some experience browsing and typing, go to the last card in the Home stack and set your User Level Preference to Painting, Authoring (to use the Button and Field tools), or Scripting (to edit scripts).



Choose tools from the Tool menu or "tear off" a tool window by dragging down and beyond the menu. Move the window on the screen by dragging its top bar. Click the close box to put the window away.

Paint tools

Choose patterns from the Patterns menu, and other options from the Options menu.

Figure 2. Macintosh menus and tools menu

puter knowledge. HyperCard takes the load off the user and carries out most of the work which makes Macintosh a better computer aided instruction tool to use.

## BUTTONS, FIELDS, CARDS, BACKGROUNDS AND STACKS

### Buttons

In the HyperCard, one is working with objects. These objects are buttons, fields, cards, backgrounds and stacks. Each of these objects can carry some amount of information and/or data in itself. When a button is selected, the entire script carried by it is also selected. Size of a button is determined by the author. It can be as small as a 0.5\*0.5 sq.cm square, or it can take the whole screen. Copying and pasting is done from the menubar, which saves time when creating buttons with similar scripts. They also take different shapes and icons or names. Some sample button icons along with a dialogue box for buttons are shown in Figure 3. Transparent buttons serve many useful purposes in designing tutorials. These buttons are exactly the same as other types of buttons, except that they are not visible to the user. The objects behind these buttons can be seen through the button.

Buttons can serve as a connection between two cards which have related information, or can be used to perform a sequence of actions when clicked on. Playing music, drawing a picture, creating a script, expanding a stack, quitting from HyperCard and displaying information, are few of the functions which can be performed by using the button scripts.

### Fields

In the same way as the buttons, the fields are objects which carry text. They can be installed either in the card or in the background. Each field has a unique ID number and a sequence number which can be referred to by these numbers. Very large volumes of text can be placed in a single field with a scrolling bar. Each field can be set to have a

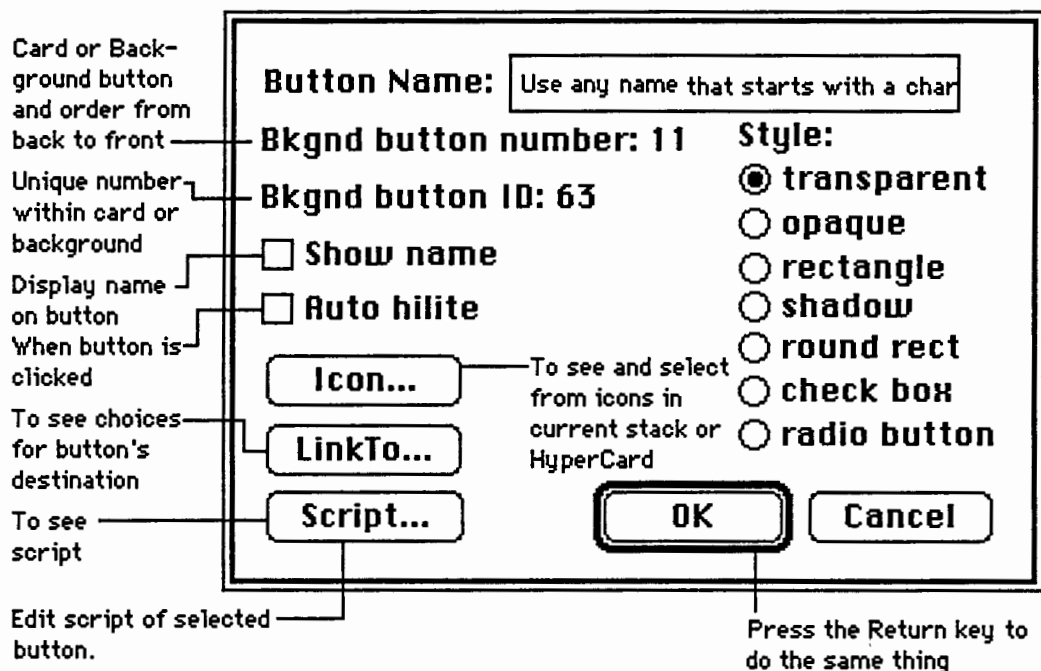


Figure 3. Some icons and a button dialogue box



certain text font, though one field can not have more than one font.

Card and background fields occupy an area on the screen, and are responsive to the mouse click on top of the occupied surface. This is true only if the field has a handler for mouse click. A handler in HyperTalk is the word "on" which is used along with different mouse functions. For example the HyperTalk code "on mouseup" will trap releasing of the mouse button on top of the object carrying the code. Creating and adjusting size and location of a field is very easy. A very specific area of a field can be accessed from within other cards and stacks. These areas are addressed by line numbers, items (comma separated statements), and words (space separated statements). This makes the fields suitable for use in database systems.

In the same way as the buttons, the fields are also treated as objects. They can be created from the menubar. Field dialogue box provides all the possible ways to set field properties. While editing inside a field, text will be entered after the cursor and will be automatically wrapped around at the end of each line. The author needs not to worry about pressing the return key at the end of each line, unless if he does not want to fill a line. Figure 4 shows a field dialogue box along with the available fonts for the text. Selected field properties are highlighted in the font dialogue box.

### Cards

A card is a collection of all the objects that are displayed on the screen. It has a background which can be shared with other cards in the stack. Hiding some of the objects in the card is a very common practice. These hidden fields or buttons are displayed in response to some parameter. When a card is being selected, all of the objects in the card are also selected.

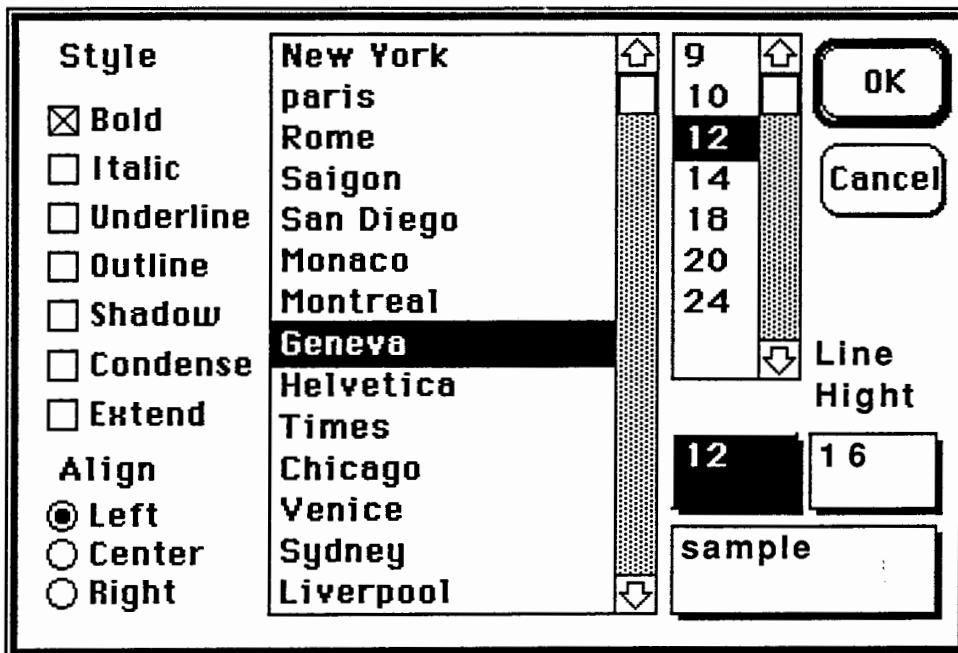
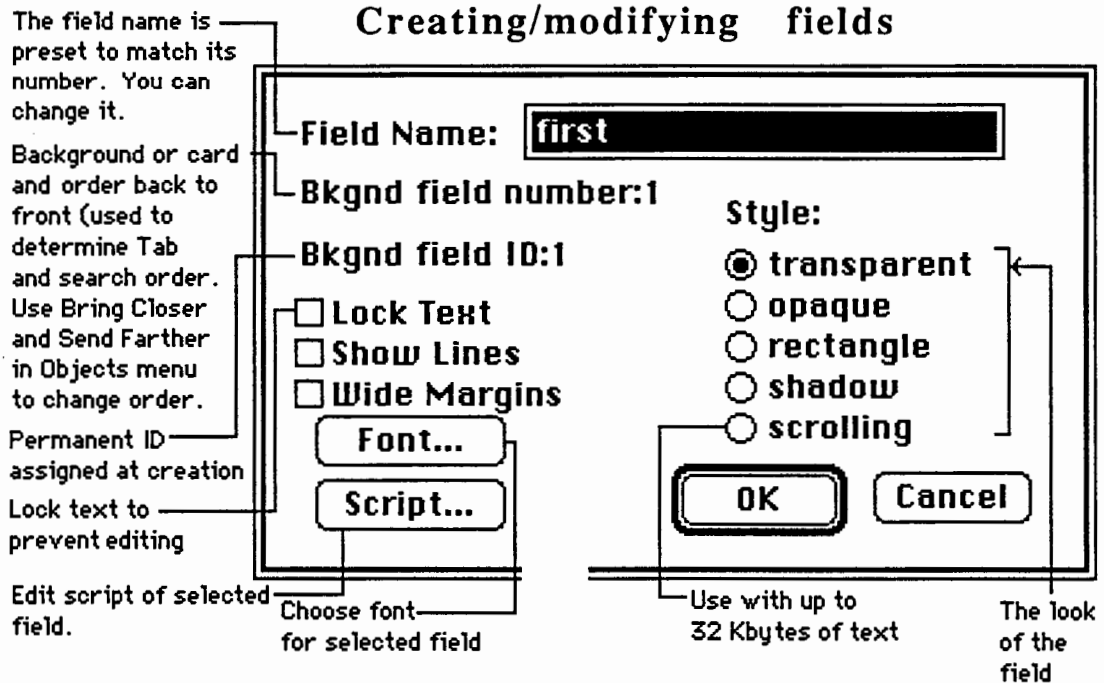


Figure 4. Field Properties and available fonts

## Backgrounds

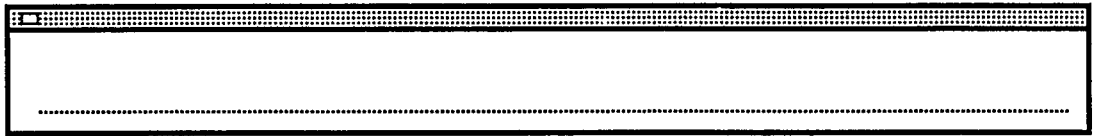
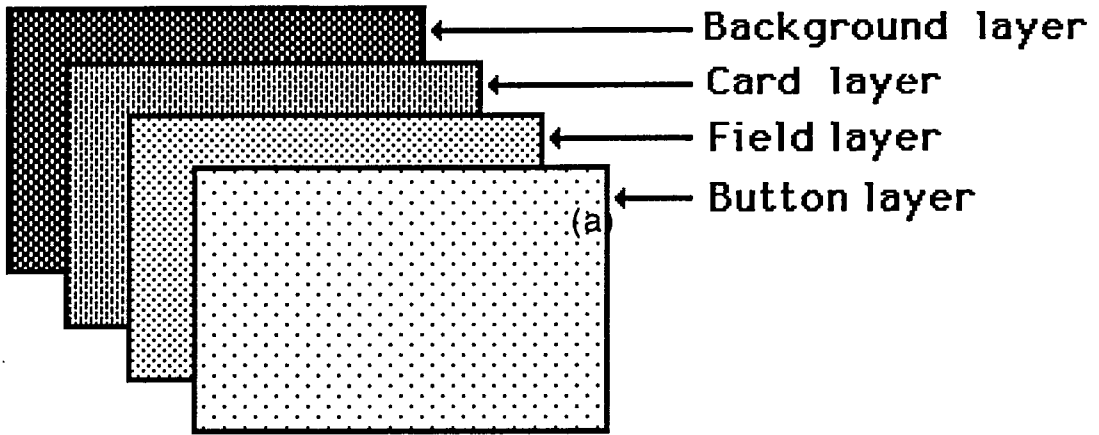
Each screenfull in HyperCard can be thought of as layers (Figure 5) of objects on top of each other. Buttons and fields are two layers on the top which can be eliminated. This means that a HyperCard screen does not necessarily need to have fields and/or buttons, but it always has a card and a background. Two other layers are cards and backgrounds. A card resides on top of a background. Each background can be shared by a number of cards, and this is a characteristic that is very useful when we want to see an object from several cards. Instead of placing that object in every card, we just put it in the background to be shared by a number of cards.

## Stacks

Stacks are largest units of collecting data and information in the HyperCard. The smallest (in size) stack contains at least one card and one background. Size of a stack grows by adding fields and buttons, more cards and backgrounds, graphics, text and data. Upon opening a HyperCard application the existing stacks will show up on the Macintosh screen (or buttons which would open these stacks). They can be opened by double-clicking on them. A folder (provided from the Macintosh menu), can hold several stacks for the ease of stack management.

## MESSAGE BOX

Message box can be compared to the "command line" in more conventional computers (see Figure 5). One can enter HyperCard commands from within the message box by simply typing them and pressing the "return" key. The message box has some additional functions like calculating algebraic expressions and showing contents of the global variables. To calculate a numeric expression, one has to type it in the message box without the equal sign, and press the "return" key. The result of the calculation will show up in the box. To find out the value of a global variable, one types the name of that



(b)

Figure 5. (a) The Layers , and (b) The Message box

variable in the box and presses the "return" key. The value will appear in the message box.

## MULTIFINDER

The Multifinder, if installed on the Macintosh, allows using more than one application program at the same time. This means that one can switch from one application to another without quitting that application. To use the HyperCard with the Multifinder, it should be installed in the Multifinder. The HyperCard usually needs 800k of memory which does not leave much memory for another application if the Macintosh has only one megabyte of memory. This problem can be solved by upgrading the Macintosh memory.

## MOUSE FUNCTIONS

Almost the entire HyperCard environment depends on mouse functions for different purposes. Scripts of cards, buttons, fields, backgrounds and stacks are activated by a mouse click, and a user of the HyperCard needs not to know much about the HyperCard to be able to use the tutorial. All of the scripts of different objects are contained inside the handlers. These handlers are activated either by a mouse click, or by calling the name of the handler from inside of another script. All of them start with "on" keyword and end with "end" keyword. Mouse functions which can be used as handler names, are listed below.

### The mouse

This will return "up" or "down" values depending on the mouse button.

### The mouseclick

This function is sensitive to clicking on the mouse button. Whenever the Hyper-

Card user clicks on the mouse button, this function gets a "true" value. Clicking the mouse button must occur on top of the object that carries this function, and has a handler for that. This function is useful if the CAI lesson designer tries to monitor responses by the user on the computer screen.

### The mouseH

The function has the horizontal dimension of the mouse cursor. In a variety of mathematical calculations where the distance between the mouse cursor and the left edge of the screen is to be determined, this function becomes very useful.

### The mouseLoc

Both horizontal and vertical dimensions of the mouse location are stored here. This function gets dimensions of the mouse cursor. These are two numbers stored in the function reference and are separated by a single comma. These numbers are the number of pixels on the screen to the left and top of the mouse cursor. The original point is the top left corner of the HyperCard screen. The maximum horizontal distance is 512 pixels, and the maximum vertical distance is 342 pixels. A pixel is the smallest amount of distance visible on the screen.

### The mouseV

The vertical dimension of the mouse cursor is stored here. This is the same as mouseH except that it holds the vertical value of the mouse location.

### Mouseup

This pertains to the releasing of the mouse button. Each click of the mouse button has two steps involved, one is when the button is pushed down, and the other is when it is released. MouseUp refers to the releasing of the mouse button which will set the value of this function to "true". This function can act as a preventive step if the user has

pressed the mouse button in a wrong location. By moving the mouse cursor to another location and releasing the button he can avoid doing some mistake.

### Mousedown

Pressing on the mouse button is indicated by the function. The function works opposite the mouseUp function and detects the pressing of the mouse button. The value of the function will remain "true" for as long as the button is pressed.

### Mouseenter

The function gets "true" value when the mouse cursor enters the object. If an object has this function as a handler, the handler will be activated upon entering of the mouse cursor into the area of the object. The function is useful in detecting the movement of the mouse cursor.

### Mouseleave

The function holds "true" value when the mouse cursor leaves the object. This function works opposite to the MouseEnter function. It is responsive to the mouse cursor leaving the area of the object which carries such a handler, and is useful in the monitoring of the movement of the mouse cursor.

### MouseStillDown

It continues sending the "true" signal while mouse button is pressed. The function keeps sending the "true" message for as long as the mouse button is pressed and kept down.

### Mousewithin

It will send signal while mouse cursor is within the object. If the mouse cursor is inside the area of an object which carries a handler containing this function, the handler will be activated. The function will continue to send the "true" value for as long the

mouse cursor is within that object.

### idle (for mouseidle)

This will send the signal while the mouse is idle. If the mouse cursor is not being moved on the screen, it will set the value of this function to "true". A HyperCard-based CAI lesson designer can use the message sent by this function to account for the amount of time that the mouse is idle.

All of the above functions can be used to react to some state of the mouse tool. This allows a HyperTalk programmer to measure every possible move of the HyperCard user, and install a suitable handler to trap for that move.

## POWER KEYS AND BLIND TYPING

In the User Preferences Card in the home stack there is an option called "power Keys". This option appears by setting the user level to "painting" or higher. There are five user levels in HyperCard which are also numbered from one to five. The higher the user level is, the more ability that user has in working with the HyperCard. For example if the user level is set to one, that user will not be able to access scripts of the cards and stacks, neither will he have access to the painting tools. If the "power keys" option is selected, the HyperCard allows to use the keyboard to perform certain painting functions. This means that by using abbreviations from the keyboard one can enhance the painting ability. These abbreviations basically perform same functions as those provided in the painting menu. This is designed to save time while using the paint tools. The power keys can also be enabled by choosing from the "Options" menu. This menu item appears when one selects a paint tool. Once a menu item is selected, a check-mark appears in front of it to indicate the selection. To turn the option off, the item has to be chosen again, and the check-mark will disappear. However, the power keys can not be used with the text tool. A power key usually corresponds to the first letter of the word in the



options menu. For example, to "Darken" a painting, type "D", or to "Invert" the painting type "I". For a complete list of power keys refer to "HyperCard Users Guide" [7], or "The complete HyperCard handbook" given in the references section.

Another option comes with the user level set to "scripting". This option is called "Blind Typing" and allows typing of commands into message box without actually seeing the typed line. This property can also be set by a function called "BlindTyping" in a script. The function takes "true" or "false" values. An example is: set the BlindTyping to false.

### CLICK AND DOUBLE CLICK

For the most part, clicking once on an item chooses that item for further use. However, double clicking an item has a different purpose. It opens an application, or a dialogue box, a folder, etc. In the case of the "erase tool", double clicking erases the whole card or the background from the painted items. In the case of tools, double clicking opens some dialogue box to allow the author to make some changes in the selection. For the complete list of the effects of the double clicking on different tools the reader is referred to the "HyperCard User's Guide" [7].

### OPTION, COMMAND AND SHIFT KEYS

These keys are used to enhance the performance of the HyperCard author. For example pushing down both command and option keys at the same time will outline all of the buttons on the screen. Pressing down the shift key while drawing a line will keep that line on a straight direction. Option key allows copying objects on the screen just by selecting that object, clicking on that and dragging to another location.

## HYPERTALK

The programming language for the HyperCard is called HyperTalk. It is similar to an applications programming language which uses commands in each line of code. HyperTalk provides some basic commands to be used, but one can add extra commands written in Pascal or C languages. HyperTalk is written in English although HyperCard provides a facility to write programs in other languages as well. The language has its syntax but it is also forgiving if a line of code is typed differently. The name of the command must be first in the line.

### Variables in HyperTalk

There are two kinds of variables in HyperTalk: local and global. A local variable is used inside a script and is not known to any other script in other objects either in the same or other stack. It does not need to be declared at the top as it is common in other programming languages. Whenever name of a local variable is used for the first time, it is also initialized to a certain value. HyperCard will remember this value as long as the script has not reached the "end" keyword. Once that specific script ends, the value of the variable will be lost. On the other hand, a global variable should be declared at the top of the script as global. It can then be used in that or any other script as long as one does not quit the HyperCard. To use a global variable which has already been used in another script, it should be declared again as global to remind the HyperCard as to where it should look for its value. Both the local and global variables can hold any type of value without any need to declare its type first (integer, character or string).

### Indentation

While editing a script, there is no need to indent the lines of code since HyperCard will automatically do this. Every keyword which has a corresponding "end" keyword, may include several lines of code which are indented a certain number of

characters. This not only eliminates the need for taking care of the indentation, but also helps to find out if the corresponding "end" keyword has been typed or not. If it has not been typed, then indentation will not be complete, and that will notify the author of the missing keyword.

Comment lines can be included in the script by typing two minus signs at the beginning of the line (--).

### Executing HyperTalk Commands

There are two ways to execute a HyperTalk command. One is in a script, another is from within a message box. Each button, field, card, background and stack has a script. These scripts can be empty or they may include a large number of lines of code. They are responsive to some specific handlers which are inserted on top of a number of lines of code. A handler ends with the "end" keyword followed by the name of that handler. Except for a small number of HyperTalk commands which can not be entered and executed from the message box, all others can be executed from the message box as well. It can only carry one line of code at a time. After a line of code has been entered from a message box, to enter the next line there is no need to click on the box again. Just start typing and the previous line will be automatically erased and replaced with a new one.

Source of the HyperTalk commands can be placed everywhere from buttons and field's scripts to the HyperCard resources itself. The HyperCard application contains essential commands and functions to use the HyperCard. However, one can add extra commands and functions and call them from inside a script. These external commands and functions can be written in Pascal, C or assembly languages. External commands and functions are an important source for extension of HyperCard.

The way that HyperCard looks for external commands is through a path called inheritance path for resources. It starts from buttons and fields scripts and then goes

through card, background, stack, home stack and finally the HyperCard application itself. This means that if an external command or function is placed somewhere along the inheritance path, upon calling that will get executed and HyperCard will look no further. This path is shown in the Figure 6. A command or function is called upon entering its name in the script and executing that script.

Debugging is rather simple in HyperTalk programs. Indeed, no special debugging is needed. If an error occurs in a program, a dialogue box will appear on the screen and will notify the user of the kind of error. Not only this dialogue box will tell what it does not understand, but it will also provide access to the script (this is true only if the user's level is set to *scripting* in the *home stack's user preferences card*). Upon entering the script, cursor will be placed right in front of the word that HyperCard could not understand. The author can then make necessary changes to the script and make it work.

### Creating Sound and Music

Sound and music are provided in HyperCard with two commands, *play* and *beep*. A beep command simply beeps for the number of times specified as its argument. Macintosh has four different sounds for this command. Another command "play" is more powerful and it plays number of lines of code specified as its arguments. Lengthy lines of music notes can be written and played by this command. Codes of music sounds are recognized by the HyperCard. In the same way as composing a musical piece, one can write musical codes in HyperCard and play them by using the "play" command.

By adding some additional hardware and software to the Macintosh, one can also record and play external music and sounds. This allows the CAI lesson designer to be able to include some real sounds in the tutorial if necessary. If lengthy lines of code are to be played, then the whole code should be written in a card or a background field. Then the name or number of that field is given as an argument to the command "play".

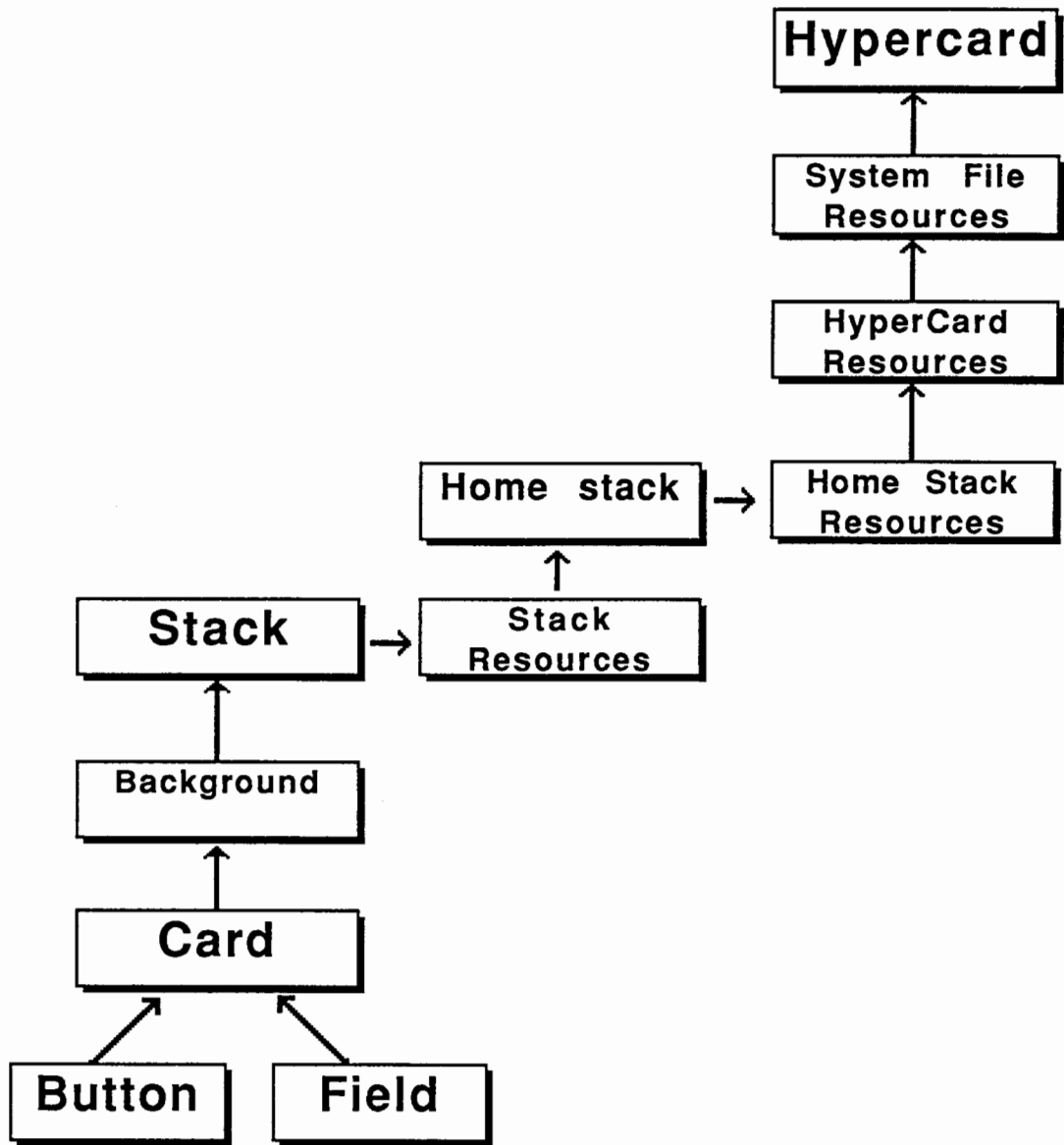


Figure 6. Hypercard object hierarchy for resources

Given that the picture of an object or a person can also be entered into Macintosh, combined with the real sound of the person or object can create a more real environment for the CAI lesson user. This is another reason to confirm that the use of the HyperCard to create CAI lessons will considerably improve the quality of the lesson, and will retain more characteristics of the real learning environment for the student.

### Different User Levels in HyperCard

There are five user levels in HyperCard. These levels are designed to restrict user abilities to avoid accidental deletions and unwanted changes. From low to high priority there are Browsing, Typing, Painting, Authoring, and Scripting levels. Browsing mode allows searching for information only. Typing level adds ability to enter text on cards. Painting level allows access to painting tools by adding a menubar item which provides these tools. Authoring level adds access to button and field tools, and finally Scripting level gives access to the HyperTalk scripts.

All of the user levels can be set from a message box or from inside a script. For example if the designer of a HyperCard stack does not want any modification to be done to that stack, then he can set the user level to browsing upon opening that stack, and set it back to previous level when closing that stack. All of the user levels are provided in the "Home stack" in the "user preferences" card (see Figure 7).

If user level is set to *painting* or higher, HyperCard provides tools to be used for painting and creating graphics. These tools can be accessed from the menubar. Creating complicated graphics by using these tools is no harder than creating scripts. Keyboard keys are very helpful while painting. They can be used to monitor movement of the tool so to avoid distortions in the movement of the hand.

## User Preferences

<b>User Name:</b>	
<b>User Level:</b>	
<input type="radio"/> Browsing	
<input type="radio"/> Typing	<input checked="" type="checkbox"/> Text Arrows
<input type="radio"/> Painting	<input checked="" type="checkbox"/> Power Keys
<input type="radio"/> Authoring	
<input checked="" type="radio"/> Scripting	<input checked="" type="checkbox"/> Blind Typing

Figure 7. "Home" stack's "User Preferences" card

## External Commands in HyperCard

HyperTalk provides some basic commands to be used. However, there is a way to add external commands to it and that way to expand the abilities of HyperTalk to do more specific tasks. These external commands (XCMD's) and functions (XFCN's) are written in high level languages, and after being processed through a compiler or assembler, attached to a stack as a resource. According to Danny Goodman when HyperCard sees that the message you send matches an XCMD or XFCN resource, HyperCard grabs a small chunk of memory to store what is called a parameter block. This parameter block acts as a staging area for information that goes back and forth between HyperCard and the resource's code.

In addition to source code for a XCMD some additional units are needed to make the command usable for HyperCard. These units are called interfaces or glue. The glue routines are short functions and procedures that do the communicating with HyperCard. They often condition the data in the XCMD code so that the information is in the right spots of the parameter block before jumping back into HyperCard. HyperCard object hierarchy is more complex when considering resources for XCMD's. The System File Resources are checked for a XCMD before a message reaches HyperCard. This hierarchy is shown in Figure 6.

### HOME STACK

The home stack is the most important stack of the HyperCard. It not only provides buttons to all other stacks, but also sets user preferences in a special card. It contains commands and functions which are not part of the HyperCard application, but are supposed to be used by all other stacks, as well as the list of documents to be looked for while searching for files. A card named "user preferences" provides buttons for five levels of using the HyperCard.



A customized home stack can replace HyperCard's home stack. Then HyperCard will retrieve information about pathnames for documents and applications and stacks as well as the user level from this stack.

### COPYING AND PASTING FROM OTHER STACKS

Modifying a HyperCard stack is no more difficult than creating a new stack. Modification comes handy when one wants to incorporate ideas expressed by other HyperCard lesson designers into his own. A stack is always expandable and its size is limited only by the size of the disk. A facility called "clip board" allows transferring material from one stack to another stack, or from one disk to another disk. Even if one quits HyperCard, the copied material will remain in the "clip board" to be pasted into another stack. Lengthy scripts written by others which may seem to serve our lesson design purposes, may be copied and pasted. This virtual connection between separate lesson designs can help improve the quality of the tutorial.

### HYPERCARD IN CAI

With the characteristics as mentioned in the above sections, combination of HyperCard software and Macintosh computer is a perfect tool to be used for creation of computer aided lessons. In CAI lesson design the goal is to make the material as easy to use and as much efficient as possible. The word efficient is used to describe the kind of lesson design in which the student will get most out of the tutorial in the least amount of time. Card, background, field and button properties seem to offer a lot in this respect. This is due to the special properties and characteristics of these objects as described earlier.

All of the classical CAI lesson designs discussed in chapter II can be implemented by HyperCard, plus some more features offered by HyperCard. We can easily combine the more useful characteristics of the classical methods to create a unique

design which eliminates a number of obstacles.

Lessons designed by using HyperCard allow the user to have control over the lesson and be able to examine different parts and sections which are related together, without losing the original point where he started from. The user gets the advantage of using the written material (since the access can be provided to all parts of the lesson at any time), plus the speed of access, sound effects, and.... The most important feature of the HyperCard is, however, the power of the software to interrelate different parts of the material which might have related information about a subject. If someone is learning material about digital systems, on the same screen, the user can have access to the related information about digital systems by just clicking on the corresponding screen areas.

In the classical lesson designs the user did not have much control on the flow of the lesson. Even though he could affect the computer by his performance, the decision making was left to the computer. The prepared lesson design would decide which direction the user should follow. This is not true for the HyperCard-based lesson designs. If in the middle of the tutorial a user decides to go back and forth and search for something, he will be able to do that, and then will be able to return to the same point as he left from.

For the reasons mentioned above, HyperCard program can dramatically improve quality of a lesson design. A well designed lesson in HyperCard can be assumed as being a collection of information about a subject on a disk which can be accessed by choice. Each screenfull of information is also connected to all related information which one may also want to know about.

## THE IDEA OF THE LEARNING ENVIRONMENT FOR DIADES

Computer aided instruction in the past was mostly based on following the steps provided by the computer. The student did not have much to do with the lesson design, neither he had much choice with the prepared design. In real life situations, however, a

person's learning ability increases if he is provided a variety of choices in each step through the lesson. With the introduction of HyperCard it seemed plausible to create such lesson designs with not much difficulty.

HyperCard provides such flexibility and user interaction that makes it possible to have a "floating" design. The word "floating" is used to denote a kind of design in which the user will be able to tell the computer how to present the material. In other words the learner will have ability to design the lesson. Of course this does not mean that the user should have HyperCard knowledge or any computer knowledge at all. The assumption is that a user of the CAI doesn't know much about computers, but still will be able to read the instructions on the computer screen and affect the way a lesson is presented.

The above mentioned ideas come to existence by using several methods of interaction provided by the HyperCard. For example a CAI lesson designer can ask few questions from the learner before presenting the material. Responses given by the user can then be used to present the material as requested. The persistence of a global variable to hold its value to the end of a HyperCard session, combined with the conditional statements in the HyperTalk code, provide such ability to present material differently according to the response from the learner.

As described earlier, a series of HyperCard commands can be activated by a simple click of the mouse button on top of the object carrying the script. These commands can access another objects even in different stacks, and activate their scripts. This unique capability of the HyperCard scripts allows a CAI lesson designer to execute a series of commands in different objects and stacks from a single object by a mouseclick. This way several lesson designs can be incorporated into the same lesson, and the user will have an opportunity to choose the one that may seem to serve him the best.

Each object in HyperCard has a unique name and a unique number. Even if two objects have the same name, once their full name is specified (including name of the

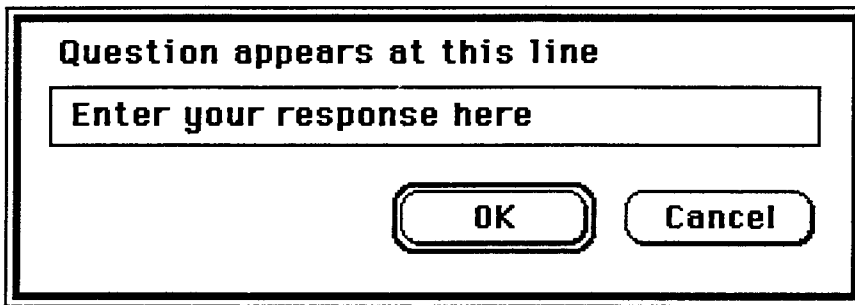
stack), then the names will differ. Of course no two stacks can have the same name. By referencing an object by the full pathname, it is possible to access that object and make necessary changes to the script or other contents it may have.

## HOW HYPERCARD INTERACTS WITH THE USER

The interaction between the user and the computer can be accomplished in several ways. Commands "ask" and "answer" provide different types of user interaction. With the command "answer" the user is presented with a dialogue box. Along with the dialogue box there can be one, two or three possible responses. The response goes into a local variable "it". Designer of the CAI lesson can use this response to decide about the way the lesson should be presented. Another command "ask" interacts differently. It also presents a question in a dialogue box, but does not present prepared replies. Instead it asks the user to enter a response to a question. The response is again stored in the local variable "it". The value of the "it" variable can be examined or used later in the tutorial. Figure 8 shows dialogue boxes for these two commands.

HyperCard provides also another way of evaluating the behavior of the user. The location of the mouse is monitored with a function called "The mouseLoc". This function gets the location of the mouse on the computer screen. CAI designer can then decide to evaluate this location and give an appropriate response according to that location. There are other mouse functions which serve almost the same purpose. The click of the mouse can be measured by "The mouseClick" function. If the user clicks on some location on the computer screen, it sets the value of the mouseClick function to "true".

The HyperCard lesson designer can use the answers from the user either at the beginning of the tutorial to decide about the presentation method, or he can place dialogue boxes in appropriate places to make sure that the user is making enough progress. This way if not enough progress have been made by the learner, then the computer will

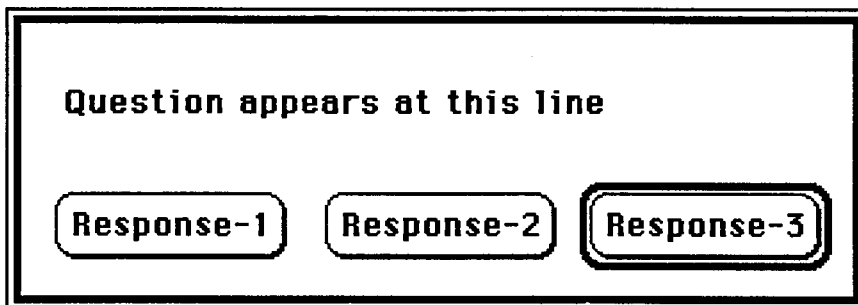


Question appears at this line

Enter your response here

OK Cancel

This diagram shows a rectangular dialog box with a double-line border. At the top, the text "Question appears at this line" is displayed. Below this is a rectangular input field containing the text "Enter your response here". At the bottom of the dialog box, there are two buttons: "OK" on the left and "Cancel" on the right, both with rounded corners and a double-line border.



Question appears at this line

Response-1 Response-2 Response-3

This diagram shows a rectangular dialog box with a double-line border. At the top, the text "Question appears at this line" is displayed. Below this, there are three buttons arranged horizontally: "Response-1", "Response-2", and "Response-3". Each button has rounded corners and a double-line border.

Figure 8. Dialogue boxes for "ASK" and "ANSWER" commands

suggest to provide alternative ways in which the learner can do a better job.

The above mentioned methods of interaction make it possible to have a very flexible lesson design in which the student will have a variety of choices to make. A HyperCard-based lesson designer is not able to tell in advance what the specific needs of each student will be while learning a subject with HyperCard. However, the designer has the capability to present a lesson in a manner that can be used differently by different users with diverse backgrounds.

Comparison between the classical CAI methods and the HyperCard-based CAI methods (both described in earlier chapters), shows how much these methods can be improved by using the HyperCard. It allows the user to take the control of the lesson, search for the subject, go back and forth in the lesson, create pictures and get much closer to a real learning environment.

This project attempted to use HyperCard to improve classical CAI lesson design methods, and design a new method for teaching the DIADES design automation system. The CAI lesson designs which use HyperCard, are not quite common yet, so there is not much previous work in this area to draw upon.

## CHAPTER IV

### COMPOSITE CAI METHOD AND HIERARCHICAL, GRAPHICAL HELP SYSTEMS

#### INTRODUCTION TO OUR NEW CONCEPT

With a look back into the history of the Computer Aided Instruction, and after taking into account all the instructional functions that could have been accomplished by different CAI methods, and properties that have not been accomplished, the author tried to create a lesson design style which would include most if not all of the previous design strategies. This new design system, which is called COMPOSITE Computer Aided Instruction system, has a unique structure. It allows to incorporate many design systems into one system. The Composite system is hierarchical and graphical. It is one step towards the future "ideal" HyperText system. The internal conceptual connections can be found in every stage of the lesson.

The Composite CAI System allows modification, addition and expansion of the graphical, hierarchical help system at any time. The authors of this system will have access to the expansion process which is mostly carried out by the system itself. The author clicks on the "expand help system" button, and the button goes to work. It will ask the index and the contents of the proposed help. After the help text and help index are entered in the pre-determined areas, the author will click on another button to install the new help text. The content and index of the new help will be installed in proper locations.

Each time someone uses the composite CAI system, he will be notified to have his comments about the system ready to enter in a suggestion box. At the end of the les-

son when he wants to quit, he will be asked if he has any suggestions to make. The suggestions from various users will be taken into account by the main CAI developer and will affect the modification process or creating of the new material.

### 1. Comparison of the Environment and Classical CAI Methods

A comparison between our Composite system and more common classical methods of CAI will reveal the ability, flexibility, efficiency and power of the Composite system. While each of the previous methods accommodates a certain way of learning and is limited in the presentation method, the new Composite system has all those methods combined into one in a comprehensive and expandable way. A comparison of the well known classical CAI methods and the Composite method is shown in Table I. Each method is given a score for the capabilities possessed by it. The accumulated score is an indication of the usefulness of the method.

The Composite design seems to have the combined capability of all other CAI methods. This is what makes it a new and improved lesson design. Some capabilities of the Composite lesson design are unique in itself, and are not implemented any time before. For example the conceptual connection of the similar material, as it is defined in a HyperText system, can be found only in our Composite system.

Now, let us see how each of the advantages offered by the other design methods can be accomplished by the Composite design. Games and discovery methods which would normally get attention of the users, can be created in the new system. HyperCard allows creation of all kinds of games. There have been numerous games written in HyperCard. The design is user-controlled and there is no one way or a prepared way through the lesson. This ability is provided only by two of the previous methods, branching and multitrack designs. They still lack the ability to be fully user-controlled since after choosing a branch or a track that will be the only way through the lesson.





The only classical method which can be still used to improve the Composite method, is the adaptive design. The Composite design can incorporate necessary changes into itself by using automatic methods to modify the contents of the stacks (not implemented yet, but can be implemented in HyperCard). Stacks of the Composite design are expandable and modifiable. The user-controlled capability of the Composite design allows each user to individualize the lesson. The previous design methods do not go too far in accommodating this aspect of the design. Those methods which allowed the user to practice with examples related to the lesson, are included in our Composite system. The random number generator in HyperCard also allows presenting different problems each time through the lesson.

Presenting of the new and graphical material is possible in the Composite design. Tests, exercises and comprehensive final examinations are given and scored in HyperCard-based systems. Probably a very important feature of the Composite design is that it provides means of presenting material in a hierarchical form. The more a student goes down in the hierarchy, the more detailed and specific information will be available to him. Ability of the Composite system to accommodate all of the above mentioned methods will obviously increase the efficiency of the system. This is actually the goal of a better CAI lesson design.

The prepared lesson design to teach about DIADES has incorporated many of the Composite characteristics as mentioned above. Use of the material by students and the feedback that we can get from them can help to enhance those characteristics that will be found most useful.

## 2. How General is the Environment?

The current design system is implemented to teach the students about DIADES and microprogramming. There are some specific elements about this design, such as a glossary and a final exam. However, once the Composite lesson design is established as

a useful and productive design method, the lesson structure can be used to produce lessons which would teach different subjects. The design environment is not limited to teach only few subject matters. Depending on the subject of the lesson, the environment should be modified to accommodate the specific ways in which the lesson can be learned.

For example, if one wants to create a HyperCard-based environment using the Composite design methodology to teach about a power engineering course such as distribution systems, the following steps will be taken to accomplish the task.

Step 1: The textual material about distribution systems will be written into computer. If it is already stored in the VAX system, it will be transferred to Macintosh.

Step 2: The major sections of the lesson will be recognized and put into different fields. The text inside these fields will be later divided into card fields. These sections will be further divided into sub-sections until the text is no more dividable.

Step 3: All necessary graphical diagrams and pictures will be created by using drawing and painting tools. They will be stored in the same stack as the pertaining text is stored.

Step 4: A card will be created which would include names of the major divisions of the lesson. For each major division which has sub-divisions, a new card will be created to carry titles of those sub-divisions. This way the hierarchical presentation system of the Composite design will be created.

Step 5: A card design will be chosen. At this step the author can copy the card design of the DIADES system. It consists of two text fields to enter the textual material, and a set of buttons in the lower side of the card to carry out connection between different parts of the lesson.

Step 6: Text will be copied to card fields. The graphics will be pasted into cards right after the explanation of the figures. These figures if part of an example, can be linked to the text of the example so that the user can go back and forth and view the

example and the figure at the same time.

Step 7: A glossary will be made of difficult technical words. These words will be highlighted in the text of the lesson to indicate the presence of more explanation.

Step 8: Help system will be created to carry out the task of informing the users about the presentation method. The help system for the most part depends on the subject matter and is created in the same way as other stacks. The connections must be maintained between the help system and the appropriate part of the lesson where the help will be needed.

Step 9: Tests will be added which would pop up in the middle of the lesson, if desired. The author is responsible to extract the test questions from the text and score the students. He can also give appropriate comments each time a student gives a right or a wrong answer.

The Composite system is created to teach about distribution systems. It has a hierarchical construct, includes help system, tests, glossary, graphics and connections between different parts of the lesson.

Another example of creating a Composite lesson design would be creating a data-base system which would carry information about a certain subject. The information will be divided into small components. Then a card will be created with a number of fields on it. This number will match the number of the components in a chunk of information. One can add graphics to the card to explain the subject of the information in a graphical form as well. This card will be copied and pasted as many times as the amount of information requires. Then the pieces of information will be placed in the appropriate fields.

After creating the graphical data-base system, one can easily search for information with respect to the contents of any individual field. These fields are recognized by HyperCard with their names. After copying and pasting a card, all of the fields inside the

new card will hold the same name as the previous card. This allows searching for contents of a specific field in all cards.

### 3. How a Composite Design Can be Expanded by New Users?

To create a Composite lesson design for each class, the material for that class should be first entered into Macintosh. Then according to the subject of the lesson, the high points of the presentation method will be determined. The steps involved in creating the lesson are similar to those explained in section 3. The question might arise how a user who does not know much about HyperCard, can go through those steps and create his projected design. The answer is that if the author of the Composite lesson design wants to make the system expandable by the users, then he will write the necessary scripts to carry out each step's function. Then he will place these scripts in several buttons and would give a name or a number to each button. The designer-user would then click on these buttons in sequence and will answer some questions wherever necessary. The rest of the design process will be carried out by the scripts.

An example of this in a data-base system would be creating a script that would ask questions from the user, and will place those responses in appropriate fields of the new cards. These new cards are already created by the script and placed at the end of the data-base system. The script can even be written such that it would check for the correctness of the responses. If a numerical response is needed, then it will not accept non-numeric responses. Or, if a three part response is required, a suitable script code will make sure that the response has three parts in it.

## CHAPTER V

### THE DIADES SYSTEM

DIADES which is created by Dr. Marek Perkowski, and is being developed by several graduate students from Portland State University, as a hardware design automation system, has not entered industrial applications yet. It is still in the stage of research and development. The prepared tutorial attempts to present DIADES in its present stage of development, including TAG user's guide, ADL bugs, and a glossary. The material is presented in a textual form along with graphics which are inserted wherever necessary.

The environment developed in the lesson is based on the text and diagrams from the class textbook that has been written in the winter of 1989 by Marek Perkowski and his graduate students David Smith, Jiuling Liu, Pan Wu and William Zhao. The goal of creating a learning environment for DIADES is to be able to collect all the information about DIADES in one place. This will help the students to have easy and quick access to that information, and will help the developers of the DIADES to add their new findings and improvements to the existing data base.

According to David Smith in the DIADES manual [14], the DIADES design automation system is a set of programs for the synthesis of digital circuits from high level behavioral descriptions. A digital system is described on the behavioral level in terms of variables and operations using the language ADL. The behavioral description is compiled to a structural description which is composed of specific hardware units such as adders, buffers and multiplexors.

DIADES has several components which work together to get the final result. As it can be seen from the Figure 9, the design process starts with writing an ADL program

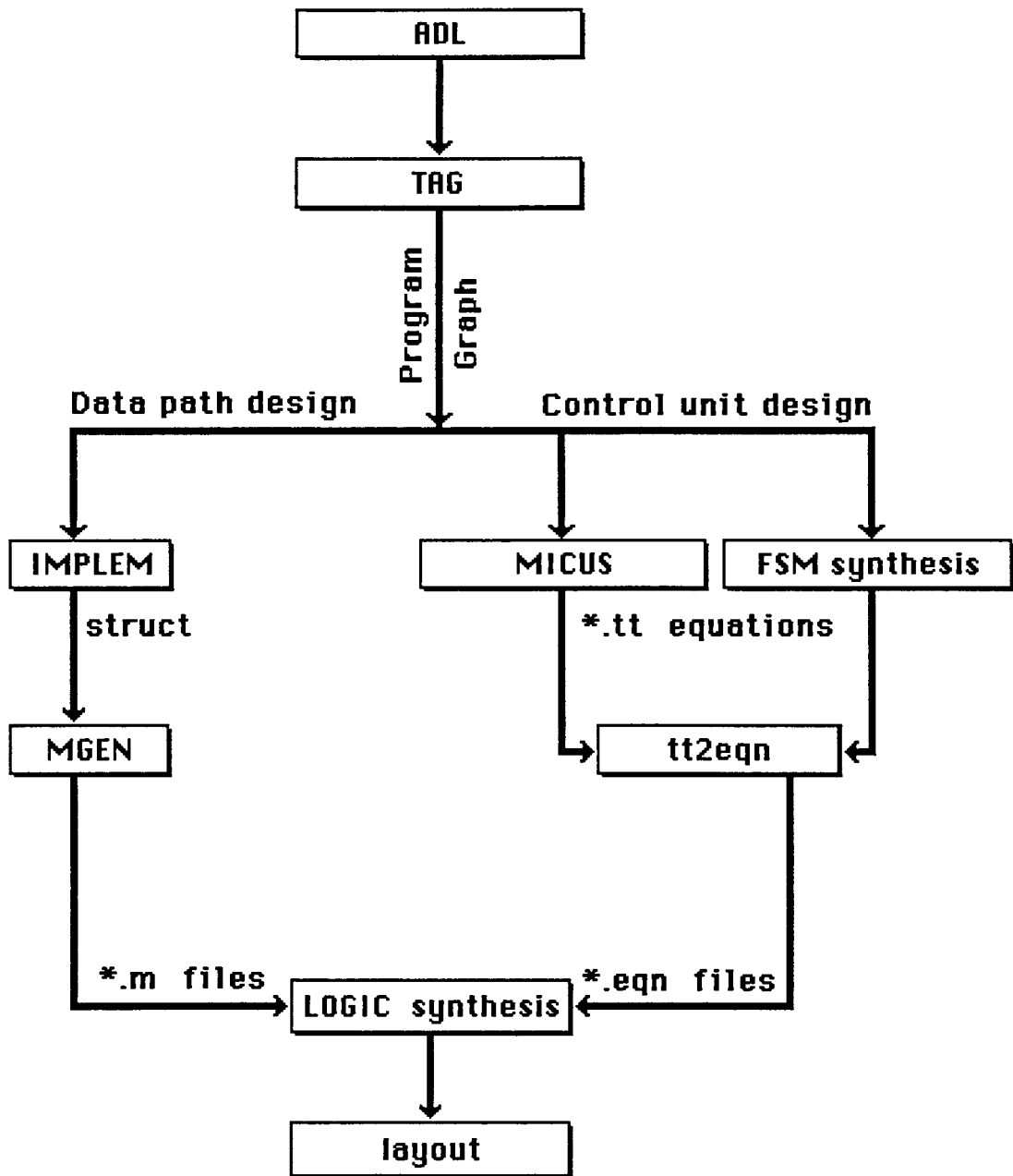


Figure 9. Design process steps in DIADES

to describe the desired system.

### WHAT IS DIADES?

DIADES is a set of programs which work together to automate the design process of the VLSI realization of a digital system. The purpose of a system such as DIADES is basically providing a tool which would allow the engineer to describe his chip in a behavioral level and leave the task of designing the hardware (logic and VLSI layout) to the DIADES system. The system will get few input data from the user during the design process and will output the final layout of the chip to the monitor.

To actually find out how one can use DIADES, and what steps are involved in the design process, an example is included in Appendix B. This example demonstrates a parallel program description. The goal of the design is to delay an input signal A by the time  $2T$ . The device will have information signal A as input, information signal B as output and a control input T. A and B are logic signals and T is an integer number of the specified clock cycles. The parallel control flow diagram is shown in Figure 16.

The first step in designing the system is writing the ADL high level description. This is shown in section 1 of the appendix B. Then the user of the DIADES will run the TAG compiler to transform the ADL description into a program graph format. As it can be seen from the example, section 2, list of arrows and nodes are created at this stage. A node represents an operation or program statement. An operation can be a variable transfer, arithmetic operation, logical operation or comparison. An arrow represents the flow of control from one node to another. Syntax checking, macro expansion and identity replacements are carried out at this stage.

The other sections of the example are inputs and outputs of the different programs in DIADES. They do not make sense for a reader but they in fact represent different transformation stages from the high level description to the final layout. The layout of



the control part of the system is shown in Figure 17.

### ADL Programming Language

ADL is a block-structured algorithmic language developed for DIADES. It is very similar to a Pascal or C program. Variables and arithmetic and logical operations are not mapped to specific hardware elements. An ADL program is specified by a set of input and output ports, internal or intermediate variables, and the algorithm. The DIADES system translates an ADL program into a hardware structure executing the algorithm. DIADES analyzes the program and generates computer hardware and a control program. The translation process is called synthesis. Steps of the algorithm are analyzed and the appropriate hardware elements are generated and wired together. There are usually several different hardware designs possible for the same algorithm. Some hardware designs will be small in area but take longer to execute. Others will have large area with maximum parallelism and run quickly.

ADL uses a LISP-like syntax. It consists of a series of statements and programs enclosed in multiple levels of parenthesis. Each part of an ADL program is a list. The simple ADL program is first broken down into two lists, a declaration list and an algorithm list. At the beginning of an ADL program instructions are placed for the TAG compiler which are not part of any lists. They can be summarized as follows:

- Listing : Echoes the program to the screen.
- Adl : Tells the compiler this is an ADL program.
- Graph : Tells the compiler to evaluate the main program first.
- Subgraph: Tells the compiler to evaluate the sub programs last.

The adl command is necessary, others are optional. Structure of an ADL program can be described in the following manner:

```
<clock list> ::= ((clock (<natural number>)));
```

```

<input variable list> ::= (input <variable declaration>+ | empty);
<internal variable list> ::= (intern <variable declaration>+ | empty);
<output variable list> ::= (output <variable declaration>+ | empty);
<variable declaration> ::= (<description> (<type>));
<description> ::= <name> | <indexed variable>;
<indexed variable> ::= (<name> [<size>]);
<size> ::= <natural number> | <index variable>;
<type> ::= <p variable> | <d variable>;
<p variable> ::= (p k1 <number of bits>);
<number of bits> ::= <positive integer>;
<d variable> ::= (d);
<subroutine list> ::= (subr <subroutine element>+ | empty);
<subroutine element> ::= (<subroutine type> <symbol> <name> (<fix flag> <parameter>+));
<subroutine type> ::= macro | block | logmacro | logblock;
<fix flag> ::= fix | empty;
<parameter> ::= <number> | <vector> | <parameter name>;
<parameter name> ::= <name>;
<constant list> ::= (const <list of parameters> | empty);
<identity list> ::= (iden <identity> | empty);
<identity> ::= (<name> <expression>);
<name> ::= Any combination of letters and numbers;
<expression> ::= <constant> | <variable> | <arithmetic expression> | <logical expression> | <predicate> | <expression> @ <expression>;

```

Subroutines in ADL Program. Subroutines can be used in ADL program. The subroutine itself is listed at the end of the ADL program. Subroutine names and parameters are listed in the declaration section. There are two types of subroutines, *macros* and

*blocks*. A subroutine can be structural or behavioral. Macros and blocks can be behavioral. A logmacro subroutine is a special form of the macro and describes structure only. A macro is a sequence of statements which is represented by a single macro call statement in the main program. Macros share the same resources with the main program. A block is a separate system. DIADES generates a separate data path and control unit for the block. The block operates asynchronously with the main program and no resources are shared.

Logical Statements. Logical statements in ADL are composed of an assignment statement and a logical expression. A logical expression takes a number of operands and maps them into a logical operator. Any expression can be an operand. The number of operands is unlimited except for "exor" which takes two operands and "not" which takes one operand. If all operands of an expression have one bit, the result of the logical operator is one bit. If the operands have different sizes, the result of the logical operator is equal to the size of the largest operator. Smaller operands have zeros added to the more significant sides.

Control Flow Statements. In the same way as other programming languages, ADL language contains a number of control flow statements. These statements are used to control execution of the branches in the program. They are GO, IF-THEN-ELSE, WHILE, COND and WAIT statements. GO statements branch unconditionally. IF-THEN-ELSE and WHILE statements allow the program to have several branches. The branch executed depends on the value of the predicate in the control statement. COND statements select one of several branches depending on the value of the predicate at the beginning of each branch. WAIT statements hold the system in a loop for a specific time or until some input value changes. Each statement except GO uses a predicate to control branching. Predicates are expressions which have two values, true and false. True is represented by one bit with value one, and false with one bit with value zero.

System Control Statements. System control statements are divided into two groups, statements controlling the starting and stopping of the digital system, and statements controlling parallel operations. There are three statements controlling the execution of the system. They are *start*, *stopadl* and *return*. The "start" statement begins the algorithm description. This statement always appears as the first line of the algorithm. The control unit always jumps to this location at the start of the program execution. The "stopadl" statement marks the end of the algorithm description. It can appear anywhere in the program but is not necessary. The "return" statement is used in subroutines. When the return statement is reached, the subroutine is done and a signal is sent to the main system, allowing it to continue.

DIADES can design a system with parallel operations. Parallel operations use the same control unit but multiple hardware elements are enabled at the same time. The *sim* instruction is used to execute more than one assignment instruction in the same cycle. All instructions within the scope of the "sim" instruction are executed in the same cycle.

Besides describing a digital systems in behavioral level, ADL has the capability to describe them in a structural level. Instead of describing a system in terms of operations and variable transfers, the system is described in terms of hardware operators and connections between them. The system is still described with statements syntactically similar to behavioral statements, but their meaning is different.

### TAG Compiler

The DIADES system takes ADL descriptions of digital systems and generates structural descriptions. The program TAG is the top program in DIADES. It takes an ADL program and generates a program graph which is used by other programs to generate structural description. TAG is written in LISP and is run from the LISP environment. The name of a file containing the ADL program is entered and TAG goes to work. The program graph is written out to two files, one each for the data path generator

IMPLEM, and the control unit generators MICUS or CERTIF. The ADL program is listed to the terminal along with debugging and error information. To run TAG one should do the following:

1. Be in a directory where files can be read and written.
2. Prepare the ADL program and name it such that it will have the extension ".adl".
3. At the prompt type: /slush/dsmith/bin/tag
4. Follow the instructions on the screen.
5. Once the message "tag loaded" is printed on the screen, then you can compile your ADL program by typing "tag87 <filename>".
6. If the program is correct, the message "Graph is cohesive, no errors" will appear on the screen.

The output files will be generated and placed in the working directory. Follow the instructions on the screen and the final result will be shown on a separate monitor.

## CHAPTER VI

### LESSON DESIGN

This chapter is intended to explain the lesson design as it is implemented, and to be a guide for the developers of the tutorial, as well as user's manual for the users. There are two main branches in the lesson design. One of these branches teaches microprogramming which was written by another student (Mark Presentin), and is incorporated into DIADES lesson. Another branch teaches about DIADES and related subjects such as ADL language and TAG compiler. With regard to the large size of the whole tutorial it is installed on the hard disk. This also corresponds to our long term goal which is creating a HyperCard-based learning environment for a complete DIADES system which is expected to take a large disk space.

In the lesson design primary consideration is given to the following.

- The ease of use of the material
- Giving control of the tutorial mostly to the student
- Making tutorial interesting to work with
- Future expansion of the material by adding new cards and stacks
- Incorporating graphics along with written text if applicable

The whole material consists of several subjects which have been presented separately in different stacks. This will allow more experienced user to be able to branch out just to the subject that he needs to learn about. Of course, less experienced users are provided a predetermined path which will lead them through the tutorial from one card to another card and from one stack to another stack.

The idea of dividing tutorial into several stacks not only helps the students in searching for specific subjects, but it will also make it easy to expand the tutorial in the future by adding new stacks rather than by modifying the existing ones. This ability is also included by creating new cards. This will be explained in details later in this chapter.

## THE ACTUAL DESIGN

It was decided to divide the lesson into several stacks each containing a homogeneous subject matter. There are several major components in the DIADES system. ADL programming language, TAG compiler, ADL errors and bugs are each a separate stack. Besides these main core stacks, there are four other stacks with special purposes. These are "start", "help", "home" and "glossary" stacks. Each and every one of these stacks is explained in details in the following sections.

### 1. The "Start" Stack

The tutorial for DIADES starts from the "start" stack (Figure 10). This stack does not contain any material related to DIADES. It is solely used to control flow of the program. This is done by providing several buttons at the beginning of the tutorial to start with. The user has these options:

- a) To go to the "HELP" stack and find out more about the tutorial and lesson design. This path is primarily intended to help new users who are not quite familiar with the computer aided instruction using HyperCard.
- b) To start the lesson immediately. This path takes the student to a pre-determined stack "DIADES", and then continues with "ADL" stack and so on. If a student quits tutorial in some point, next time when he tries to use it again, he will be placed in the same stack and card from which he had quitted.
- c) To go to a specific stack. An experienced user will probably want to directly go to a certain stack and search for some subject. This button provides such ability.

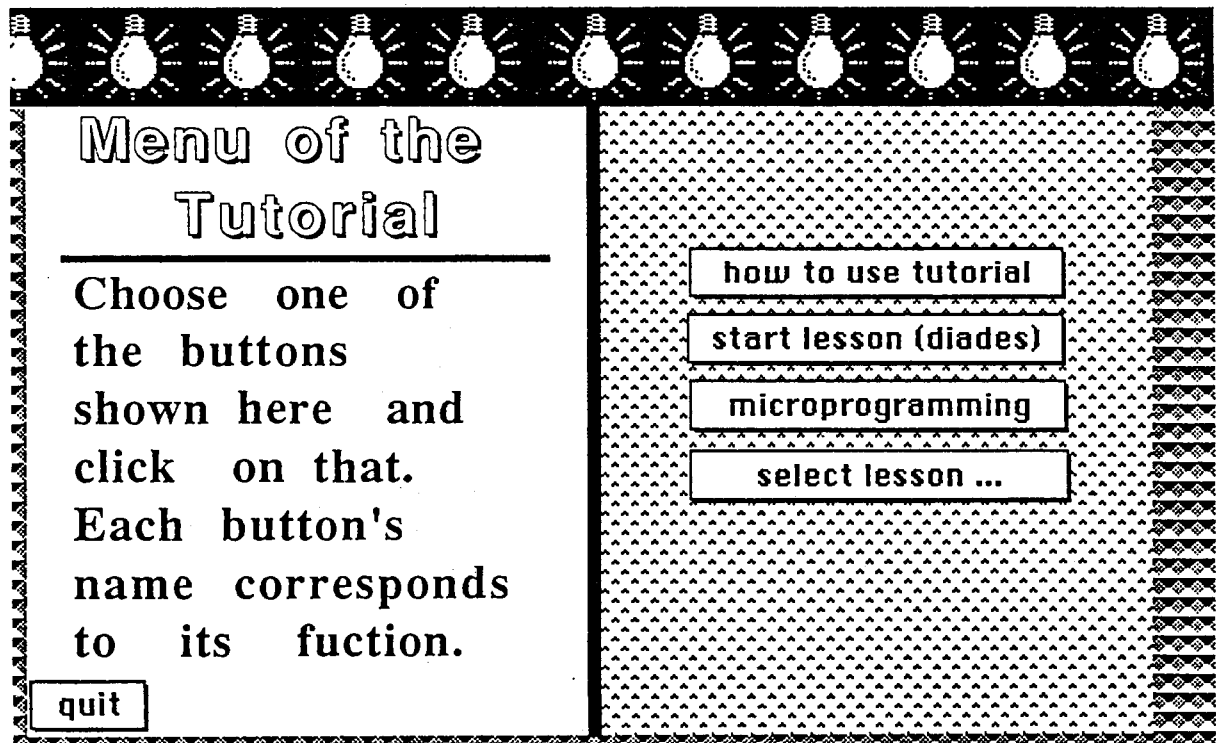
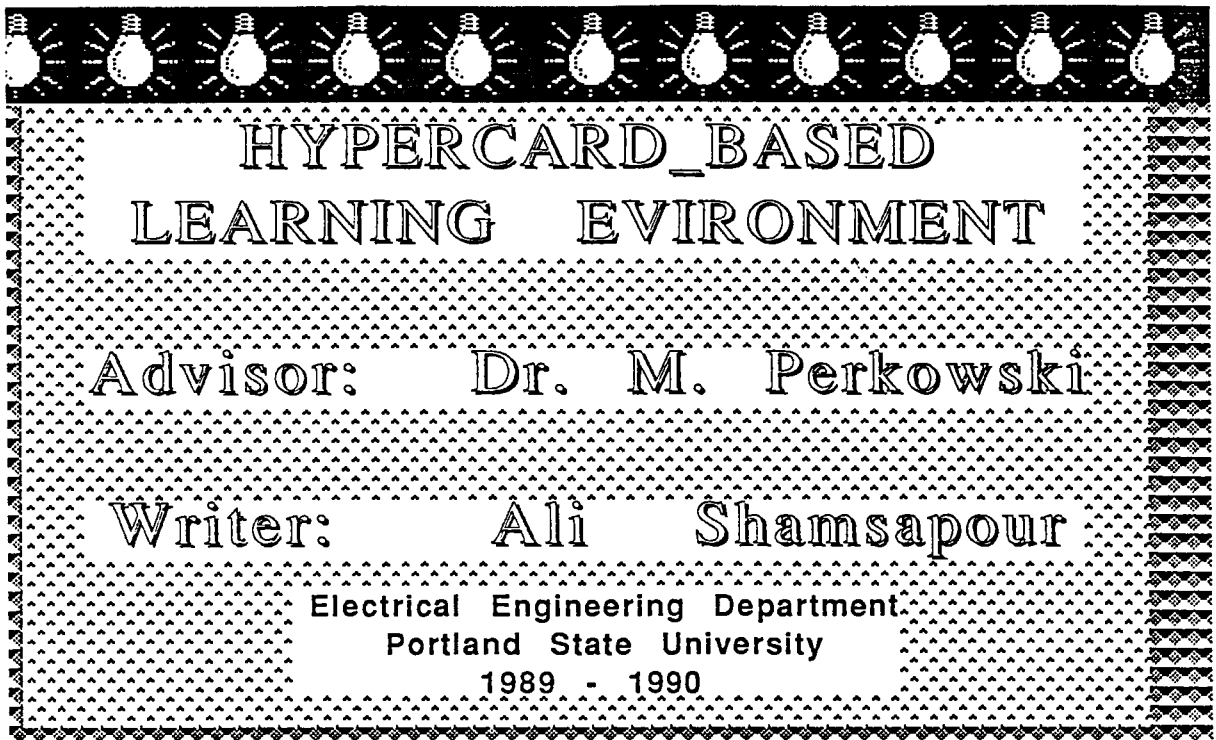


Figure 10. First and second card of the "start" stack



Eight buttons show up when a user clicks at this button, each going to a different stack. The user is asked to choose one and click on that.

- d) To use the second branch of the lesson and learn about microprogramming. This branch has its own environment with help, glossary and final exam stacks. There is no interconnection between these two branches except at the beginning of each branch.

These are options that are provided at this time, But one can add more buttons to the same card and give users another alternative paths to choose from. Quit button is provided in almost every card of every stack to allow users to stop reading whenever they want to.

Saving Card Numbers. An important feature of "start" stack is the ability to save card numbers of the stacks which have been reviewed by the user. When a user tries to quit the tutorial from any stack by clicking on the quit button, he will be asked if the card numbers should be saved. If the answer is "yes", then the card numbers will be kept in a hidden background field. This will allow the user next time to start from where he had left the tutorial. If the answer is "no", then no card number will be saved, and next time the user tries to use the material, he will start from beginning of each stack.

If the answer is "cancel", then the "quit" button will be ignored and the user will be provided a "resume" button. Clicking on this button will take the user to the same card in the last stack where the user pressed the "quit" button.

Third card of the "start" stack contains a schematic of the steps involved in a DIADES design process. At present time it only serves as a guide for the users of the DIADES as to what the organization of the whole system is and how it works. It is also intended to be used and expanded in the future by the developers of the material to organize their work.

## 2. The "HELP" Stack

"HELP" stack is designed to provide every possible assistance which users may need before or during the use of the material. This stack can be accessed from all other stacks. The first card of the stack is an index of all kinds of help available. In every stage of the tutorial if a user clicks on the "HELP" button, (this button is the one with question mark on it), he will be directed to the help index card and will be asked to choose the kind of help needed. The choice of help will take the user to the appropriate card in the "HELP" stack where he can find the necessary information.

"HELP" stack is given a special format (Figure 11). Instead of presenting the entire help text in a lump, it is being typed slowly on the screen. This is supposed to get more attention from the user to read the text. The text cursor stays at the end of the text if there is more help to come. But it will disappear at the end to indicate that the user needs to click on the "more" button if he needs more help. The "more" button stays inoperable during the time when the text is being typed. This is shown by the changing of the color of the button to black. It turns white when it is ready to be used again.

How To Expand The "HELP" Stack. The design of the help stack is such that it allows the expandability of the stack in the future. To add a different type of help to the "HELP" stack, simply a button is added to the index card (which is the first card of the stack), and this button is linked to the appropriate card in the stack where the help can be found. The help itself is added to the end of the stack. Two background buttons are provided in the "HELP" stack. One of these will take the user back to the previous stack, and the other will provide more help, if available.

## 3. The "DIADES" Stack

This stack (Figure 12) contains main and comprehensive explanation to the DIADES system, although it is not the largest stack. First card of the stack is an actual beginning of the lesson. In this card several buttons can be found in the bottom row of

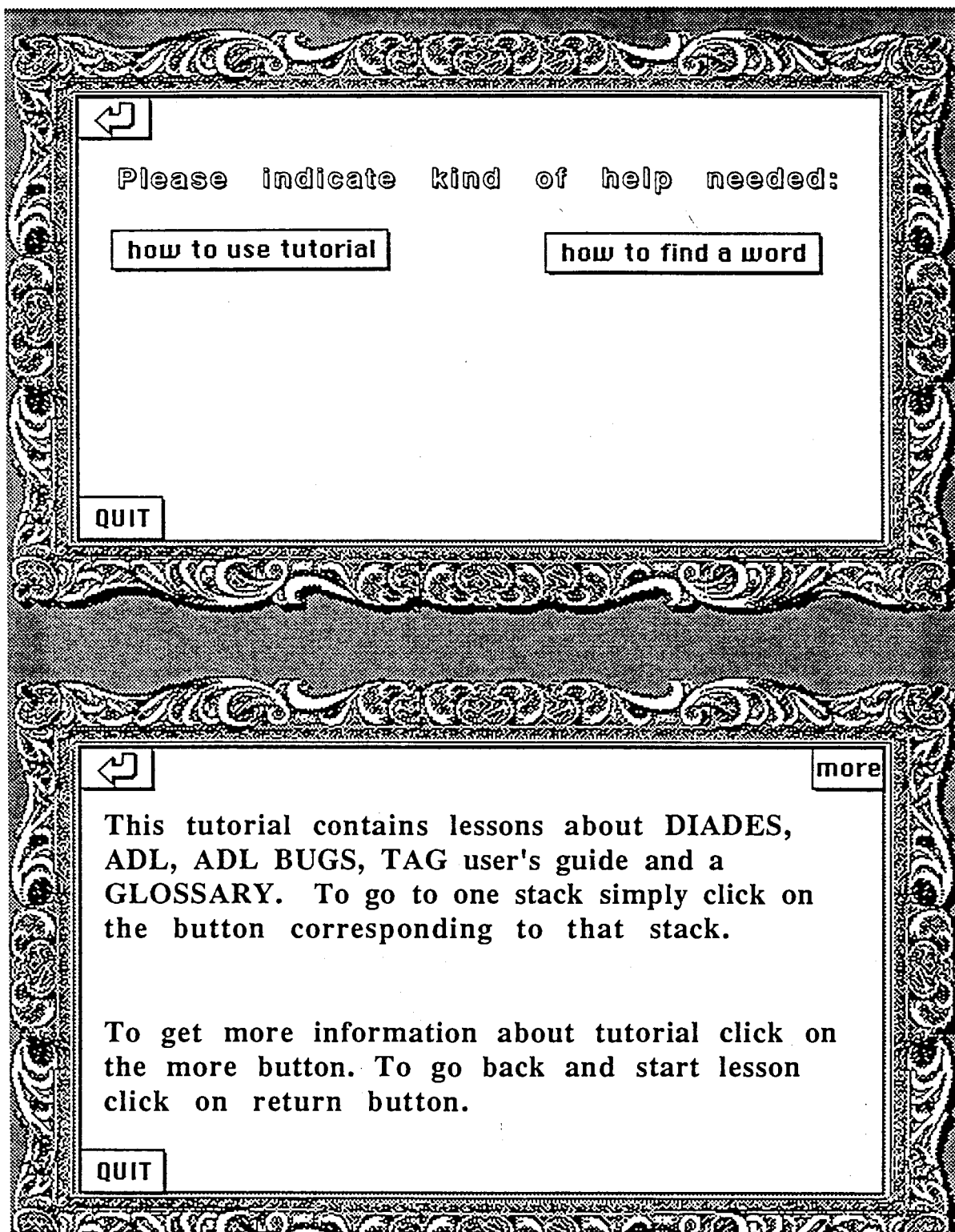


Figure 11. Sample cards of the "HELP" stack

card 1 of 2	2/22/90	2:45 PM
-------------	---------	---------

**DIADES**

Design Automation System

1. Introduction

The DIADES design automation system is a set of programs for the synthesis of digital circuits from high-level, behavioral descriptions. A **digital system** is described on the **behavioral** level in terms of variables and operations using a language called **ADL** [ref].

The behavioral description is compiled to a **structural** description, which is composed of specific hardware units such as adders, buffers, and multiplexors.

The need for complicated, special purpose digital systems is increasing. One of the problems is the time required to design such systems.

QUIT	?	BUGS	TAG	ADL	→			open	find	↩
card 2 of 2		2/22/90		2:44 PM						

Input variables:      operand1, operand2 - 8 bit numbers.  
Output variables:     answer - 8 bit number  
Internal variable:    temp - holds answer

```

line 1: (((adl a example_circuit ....
line 2:     (input (op1 (p k 1 8)) (op2 (p k 1 8)) )
line 3:     (intern (temp (p k 1 8)))
line 4:     (output (answer (p k 1 8)))
line 5:     ((start) a
line 6: 10 (temp := (op1 + op2))
line 7:     (if (temp = 10) then (temp := (op1 - op2)))
line 8:     (answer := temp)
line 9:     (go 10)
line 10: )))
line 11: end

```

Figure 12. Two sample cards of the "DIADES" stack

the card, and three fields at the top of the card. All these features are the background properties, and will appear in all cards of the stack which share the same background.

Each button in the stack is used for some purpose. There are five buttons which serve as connection between DIADES stack and five other stacks. These buttons are named: ADL, GLOSSARY, ADL-BUGS, TAG-USERS-GUIDE, and a button with question mark on it for HELP stack. The QUIT button is used to end a session.

The important property of the above mentioned buttons is that using them to move from one stack to another will save card id number of the present stack. This number is used to bring the user back to the same card where he had left the stack for another stack. This property will allow the user to be able to observe different cards from different stacks with one click. These card numbers are later saved to be used next time when the user opens the tutorial.

When the "QUIT" button is used to end a session, the user will be taken to the "start" stack which is also used to save the card numbers. The user will be asked if the card numbers of the stacks should be saved. If the answer is "yes" then all the card numbers of stacks will be saved in a background field. This background field is invisible and has smallest possible size. The card numbers are saved in global variables which makes them usable from and within all stacks. These global variables are: A\_ID (for ADL stack), B\_ID (for ADL\_bugs stack), D\_ID (for DIADES stack), G\_ID (for GLOSSARY stack), and T\_ID (for TAG-users-guide stack). If the answer is "no", then no card number will be saved, and next time the user will start from beginning of each stack. And finally, if the answer is "cancel", a button will appear right beneath the "cancel" button. This button is called "RESUME" and if used, will take the user to the last stack where he clicked on the "QUIT" button.

There are three buttons in the right hand side of the stack in each card. The one with the "RETURN" icon on it is used to take the user back to the stack which he was

reading before coming to the present stack. This button will not work if the present stack is the first one being opened.

The other two buttons have different properties. One of them is the "find" button. As the name implies, it is used to find words in a stack. Upon clicking on this button the user will be asked for the word to be found. After the user enters the word and clicks on the "ok" button (or presses "RETURN" key), present stack will be searched for the given word. Along with the search procedure, the message box is shown along with the text of the desired word in it. The user is also instructed to press the "RETURN" key to find the next occurrences of the word. If there is no such word in the present stack, the user will be notified, and the message box will disappear. The same button is provided in all stacks so that the user can open some other stack and look for the words.

With the revision 1.2 of the HyperCard the "find whole" function can be used to look for a whole statement. This will include several words with white spaces between them. After installing new version of HyperCard on our Macintosh the "find Whole" function replaced "find" function.

There is another button called "open". This is the button which will be used by authors to open the locked text and add, change or modify it, if necessary. This button might as well be hidden for the user level of less than 4, so that the users will not see this button at all. When a CAI author clicks on this button, a statement will appear right on top of the button to inform the author that the fields are now open for writing.

There are three fields in the top part of the background. These fields display helpful information. The field in the top left is used to display the total number of cards in the stack plus the current card number. This way the user will be notified of the amount of progress that he has made. A message such as *Card 12 of 77* appears in the field, with 77 being the total number of cards in the stack. The second field is used to display the date in a short form. The third field displays the current time. The time in the third field

is updated every minute.

A very significant feature of the stack is that all the technical and other difficult words are highlighted. A transparent button is installed on the top of each highlighted word. The purpose of installing these buttons is to lead the user to the explanation of the word in the *glossary* stack. The button will locate the explanation of the word in the glossary stack and will highlight it for the ease of search. After reading the explanation, the user can go back to the previous card and stack. This is accomplished in two ways:

- a) By pressing the "RETURN" button which will take the user back to the last stack where he clicked the highlighted button.
- b) By using any other button in the bottom row of the glossary stack.

Some of the words in the stack which have been highlighted, have a short explanation which is provided in the same card by showing a field. Field is located in an appropriate place and will be hidden again when clicked on it.

Graphics in the stack are created by using HyperCard tools. In examples which have also graphical explanations, different parts are connected to each other with buttons. The user can go back and forth in the stack to see the example text itself, the graphics and the explanations, without moving out of the examples environment. The buttons can be considered as windows to related information which can be opened upon clicking.

Tests In The "DIADES" Stack. Since design of the whole material is intended to be used by the students and engineers in the industry, tests are included in each stack. These tests are pop quizzes which unexpectedly pop up during the use of the stack. During the tests the menubar is hidden and the user is provided only with the "QUIT" button. Each test consists of three questions. Each question is given four possible responses, while only one of them is a correct answer. The tests are graded and the user is given the result. In the case he fails to respond correctly to two out of three questions, he will be given a choice of reviewing the material. There are various numbers of tests in each

stack depending on the size of the stack (Figure 13).

Modifying The "DIADES" Stack. The DIADES stack, like all other stacks, is designed in a way that makes it very easy to modify, to add more material or to change the existing material. All of the text is stored in two card fields (in some cards only one field is used). The background button with the name "open" is used to unlock the text in both fields for modification. There is no need to lock the text after opening it, since this has been already taken care of in the card script. Locking of the text at all times prevents unwanted changes to the text.

Adding new material to the stack is as easy as modifying the existing text. A "NEWCARD" handler is included in the script of the stack. Whenever an author wants to create a new card in the stack, this handler will set the script of the new card and will put two text fields into the card. All the background fields and buttons are visible from all the cards.

Adding graphics to the stack is also possible. The author can simply delete one or both text fields (depending on how much space is required to create the graphics) and use HyperCard tools to create pictures. Size of a stack is only limited by the disk space, otherwise there is no other limit.

#### 4. The "ADL" Stack

So far, the "ADL" stack is the largest stack of the tutorial. It takes up to 400k of the disk space. The design of the stack is basically the same as DIADES stack to make it easier to use. It has a different background pattern, has more tests and more graphics. It is modified in the same way as the DIADES stack is, and has the same number of background fields and buttons. The space in the DIADES stack's background which has been allocated to the "ADL" button, becomes now allocated to the "DIADES" button.

Because of the large size of the "ADL" stack, the index of the whole contents of the stack is provided at the beginning (Figure 14). A user can go directly to a section by



**pop quiz**

1- ADL program describes a digital system in:

- A** Behavioral level
- B** Functional level
- C** Structural level
- D** All of the above

**QUIT****pop quiz**

**Your performance in pop quizzes shows that you probably need to review this section of the tutorial.**

**To review click here --> **review****

**To continue click here--> **continue****

Figure 13. Sample pop quiz and evaluation cards

<p><b>3.4</b> Statements and Expressions</p> <p><b>3.4.A</b> Introduction</p> <p><b>3.4.B</b> Labels</p> <p><b>3.4.C</b> Assignment Statements</p> <p><b>3.4.D</b> Arithmetic Statements</p> <p><b>3.4.E</b> Logical Operations</p> <p><b>3.4.F</b> Complicated Expressions</p> <p><b>3.5</b> Control Flow Statements</p> <p><b>3.5.A</b> Go Statements</p> <p><b>3.5.B</b> If-Then-Else Statements</p> <p><b>3.5.C</b> While Loops</p>	<p><b>3.5.D</b> Cond Statement</p> <p><b>3.6</b> Special Statements</p> <p><b>3.6.A</b> Wait Statements</p> <p><b>3.6.B</b> Set and Reset Statements</p> <p><b>3.7</b> System Control Statements</p> <p><b>3.7.A</b> Starting and Stopping Statements</p>
<p><b>3.8</b> Parallel Execution Statements</p> <p><b>3.8.A</b> Multiple Statement Execution</p> <p><b>3.8.B</b> Parallel Program Execution</p> <p><b>3.9</b> Subroutines</p> <p><b>3.9.A</b> Macro Subroutines</p> <p><b>3.9.B</b> Block Subroutines</p>	<p><b>3.10</b> Structural Descriptions</p> <p><b>3.10.A</b> Declarations</p> <p><b>3.10.B</b> Structural Assignment Statements</p> <p><b>3.10.C</b> Structural Conditional Statements</p> <p><b>3.10.D</b> Subroutine Calls</p> <p><b>3.10.F</b> Structural subroutines</p>

QUIT ? BUGS TAG DIADES GLOSSARY back open find ↩

card 2 of 2 2/22/90 2:40 PM

Figure 14. Sample index cards of the "ADL" stack

clicking on the corresponding button in the index. The button with the name "back" is used in the "ADL" stack to take the user to the last card in the stack before coming to the present card. This is useful in taking the user back to the index cards.

### 5. The "GLOSSARY" Stack

This is the stack that contains all the technical words along with their definitions. "GLOSSARY" stack can be accessed directly by clicking on the "GLOSSARY" button from every stack, or it can be accessed by clicking on the highlighted words. If accessed through the highlighted words, the beginning of the definition of the word in the stack will be highlighted. New words along with their definitions can be added to the GLOSSARY stack, but the author is responsible to find the correct alphabetic order to place the new word. This stack has a different background pattern, but it has the same number of background fields and buttons as the ADL and DIADES stacks.

### 6. The "ADL-bugs" And "TAG-users-guide" Stacks

These two small size stacks contain information about TAG compiler and BUGS in the ADL language. Their structure is the same as the ADL and DIADES stacks.

### 7. The "HOME" Stack

A customized "Home" stack is created to set different user attributes if necessary (Figure 15). First card of the stack provides small icons of all of the stacks on the disk. It should be updated if a new stack is added to the tutorial or if one is deleted. Last card of the stack contains buttons to set the user level. Probably it is a good idea to set the user level to browsing during the use of the tutorial, so that the stacks can not be accidentally modified or deleted.

External command and function codes are kept in home stack to make them usable from all other stacks. No external command or function is used at the time of this writing. The inheritance path of the HyperCard goes from the card, the background, the

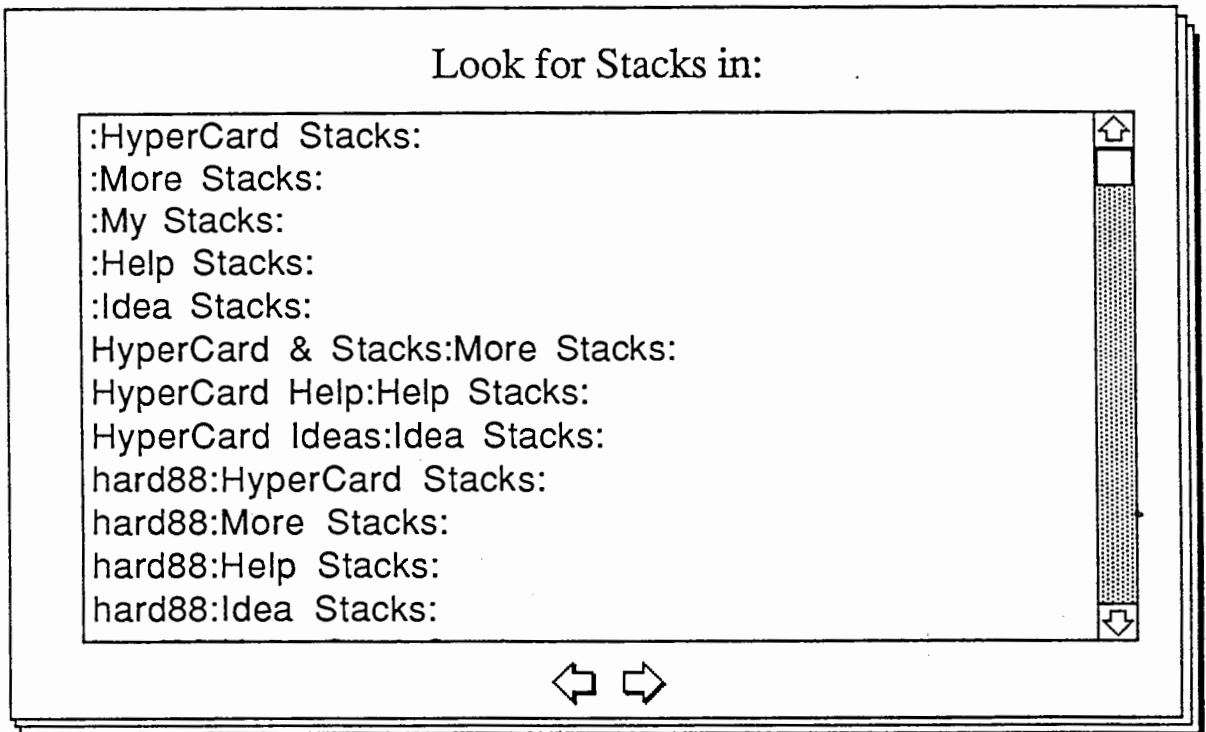
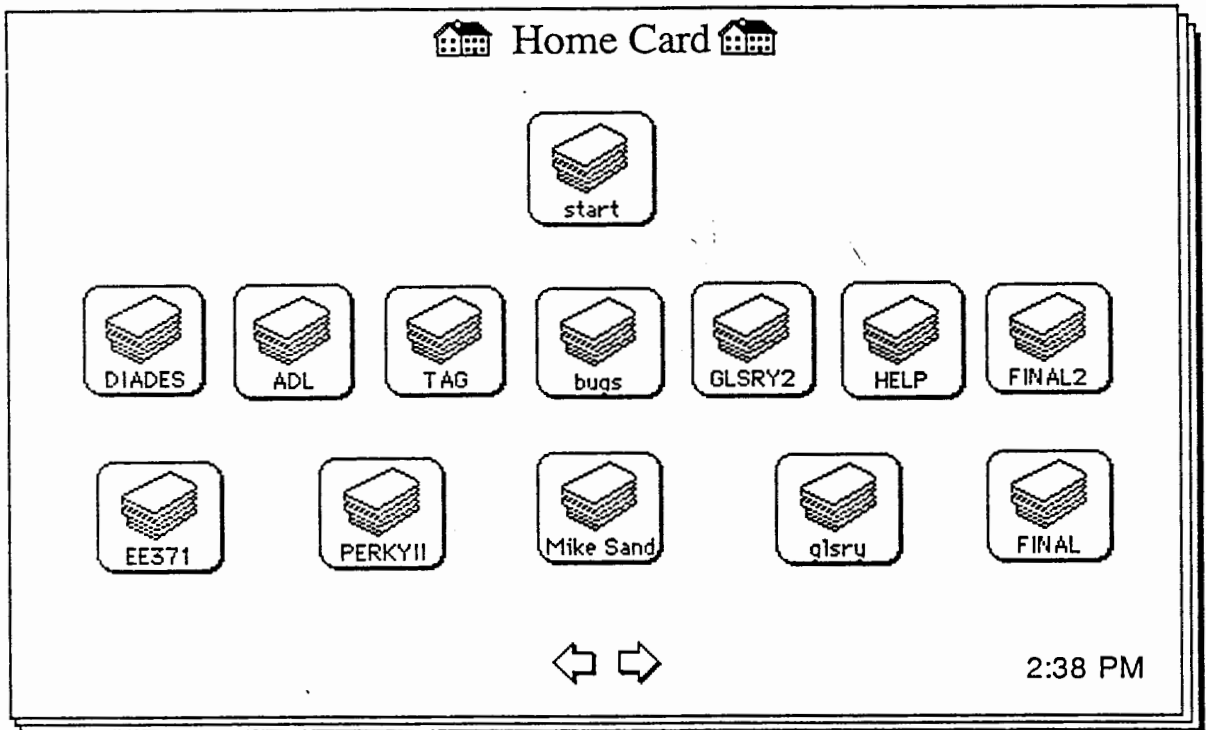


Figure 15. First and second card of the "Home" stack

stack scripts to the home stack. Any external command in the customized home stack will be available to all the stacks on the disk.

## 8. The "FINAL" Stack

There are two final stacks in the tutorial. One is written to test the microprogramming knowledge of the students after they have completed the related material. The test consists of twelve multiple answer questions. Students have one hour to complete the test. During a test one can find out about the remaining time by clicking on a button. At the end of the test the student's performance is evaluated and he is given a score. Then the student will be given a certificate if he passes the test, and if he does not pass the test, he will be suggested to review the material and take the test again.

Another final stack is created to test the knowledge of the students on the subjects related to DIADES system and related material. The structure of the stack is basically the same as the previous one. It is called "FINAL2" to distinguish that from other final stack.

## CHAPTER VII

### CONCLUSION

Computer aided instruction has been used in the past by different groups. However, these classical CAI methods (as described in many papers) were very rigid in the way they presented the material. Students had to follow the computer and submit to the way that the lesson was designed. There was no way to ask the computer questions, and if the student did not understand a subject, he could not go back in the lesson and review the material one more time. This would require starting the whole lesson from the beginning. In other words, the learning from the computer was very different from the learning in the class. It was not able to provide the graphics and sound along with the textual material. One aspect of this lesson design is that it attempts to take advantage of the known classical CAI methods by combining them and by improving them. Of course it can be improved in many ways which are discussed later on in this chapter.

With the present design, knowledge of the student is often checked by the system which is impossible in normal classes for obvious reasons. The material together with all figures, is presented with "cards" (windows). There are automatic quizzes after each subsection, section, and at the end of the tutorial. The quizzes are scored. There is also a final exam. Grading system for final exam is the same as for the quizzes.

HyperCard is really a revolutionary kind of software of a long lasting importance. Its importance is compared to the invention of such mediums as data bases, spreadsheets or laser printers. This system permits creating authoring systems that include CAI, CAD video presentations, animation of physical processes, graphical data bases and many other. This technology is expected to have applications particularly in the improvement

of education, since everybody who has to teach something and is not an expert programmer, will be able to write programs quickly to teach their personnel in the area of their expertise. These programs will include text, graphics, voice, music, and in the future even the video CD ROM, that have either been created by the easy to use tools provided by Apple Inc. (like MacPaint or MacDraw), or acquired from practical media such as books or records (for instance with use of programs like Image Grabber or ThunderScan).

### PREPARING THE TUTORIAL

There were several topics to be studied and used while developing the tutorial. This includes the HyperCard as our lesson-design tool, the well-known CAI lesson design methods to get some ideas as to what is expected from a computer aided instruction, and DIADES as the subject of the tutorial. All of the above-mentioned topics constitute a major part of the work in preparing the whole tutorial.

The author had to study classical CAI lesson designs and the tools that they were created with. This would clarify as to what goals have been achieved and what goals were not. Attempt was made to preserve such aspects of a lesson design which still are valuable to the user. At the meantime, the priority was given to finding ways which would satisfy the specifications of a more advanced CAI method. The HyperCard had to be learned well before one is able to write CAI programs in the HyperCard environment. It was to provide means of creating the projected lesson design. Finally the DIADES system as the subject of the lesson was studied. There still are some aspects about DIADES that are not clear enough and need to be developed, such as for example writing structural and functional descriptions.

DIADES is taught in Portland State University every year. The students for this course will finally be provided with a comprehensive computer-based instruction to supplement their in-class efforts. Computerizing the whole DIADES system will allow the

future developers to incorporate their contributions to the system. An example of this is combining DIADES tutorial with microprogramming stacks as part of the whole lesson. Especially since DIADES is in the development stage, a unified environment will gather all the related information in one place to be reviewed.

## WHAT WAS ACHIEVED AND WHAT NOT?

Creating a graphical data-base for DIADES which would serve as a tutorial for users of the DIADES, was the goal of the thesis. At the meantime it had to be prepared in such a manner that would make it easy for the future developers of the tutorial to incorporate their findings into the existing data-base. This also was an attempt to use HyperCard to prepare a CAI lesson, to examine the advantages that one would get in working in such an environment. The prepared design can be characterized with visual effects, graphical presentations, connections between similar subjects in different stacks, hierarchical organization of the lesson design, automatizing the tests and scoring them, ease of use and expansion, comprehensiveness and a resemblance of a HyperText system.

In recent years, there have been many conferences about HyperText systems. These systems are defined as having connections between related pieces of material. The connection is not necessarily between major categorical divisions. It can include anything that may help the learner to understand and absorb the subject matter. At the present development stage of the software, HyperText systems seem to be more of an ideal system which leads software developers to create better programs. Designing such a system was considered while creating our system. Our goal has been creating a HyperText environment which would closely resemble a real-life learning environment.

An educational software developer needs to be concerned about taking care of the internal connections in HyperText systems. This is a conscious effort and requires writ-



ing hundreds of lines of code to carry out the design goals. With the advancement of artificial intelligence and expert systems we can look forward to creating HyperText systems in much less time and effort with more efficiency.

There were few problems with regard to preparing and using of the tutorial which can be summarized as below:

- a) The compiler for DIADES resides on the VAX system. This means that the user needs to go out of HyperCard environment to do some real work.
- b) There are some uncertain areas about DIADES. Not everything is clear with DIADES and ADL.
- c) Transferring files and figures from VAX to Macintosh and vice versa is not easily possible.
- d) To use Multifinder (which would allow using several application programs at the same time), a large Macintosh memory space is required. The plan was to allow the user to stay in the HyperCard environment and be able to access VAX to use the TAG compiler. The "network login" application would let the Macintosh users to login into the VAX, but doing this from inside the HyperCard would require using "Multifinder" which for the reasons mentioned above was not possible.

#### HOW CAN THE CURRENT DESIGN BE IMPROVED?

The present design provides the tutorial and the reference for those who are interested to learn about DIADES. However, there is a part that has not been completed yet. This is the actual implementation of the DIADES system as is shown in the third card of the "start" stack. The purpose of this section is to provide the user with the ability to write, compile, debug and execute DIADES programs from Macintosh. The DIADES compiler and debugger are located in dsmith directory in VAX system. The ideal design

will allow the user to write hardware description program by using Macintosh, and then access the VAX system to compile and debug the program. If this is done in every stage of the learning process, it can give the user ability to learn DIADES quickly and efficiently.

The written text and graphics from David Smith's reports about DIADES were primary source of information for DIADES. These documents are transferred from VAX to Macintosh and placed in HyperCard fields. A user of the tutorial can access this information in a linear method. A better presentation would be extracting main topics of each subject, placing them in one card and allowing the user to choose one of them to learn about. Later he will make some more choices to narrow down just to what he wants to learn about. To accomplish this objective, a more organized DIADES manual is needed. At the time being only the "ADL" stack has some index cards. The task of searching for a specific subject is still left to the user. A better system would lead the user just to the information that he wants to learn about.

Considering the amount of available manuals for DIADES, it will require a good deal of programming to create such a design. To reduce the number of program codes, one solution is to have a fixed format for presenting material. This way only the contents of some fields will change depending on the choice subject of the learner. This kind of design requires organized DIADES manuals to draw upon.

Another plan can be providing an automatic hardware design procedure. The DIADES user will write his ADL program in a field, then will click on a button to initialize design process. At present time a user of the TAG responds to several questions before he is able to see the actual design on a separate monitor. A program which would take care of the answers to these questions, can simplify and automatize the design process. Installing TAG compiler on a Macintosh or accessing VAX from within HyperCard are steps in this direction.

## REFERENCES

- [1] Gary Bond, XCMD's for HyperCard, 1988, Management information source, Inc. 1107 N.W. 14th Avenue, Portland, Oregon 97209 tel: (503)222-2399
- [2] William Lee, "'?': A context sensitive Help System based on Hypertext, Design Automation Proceedings, 1987, Pages 429-435, VHSIC Test Systems, SENTRY Schlumberger 1601 Technology Drive, San Jose, CA 95110-1397
- [3] Danny Goodman, The complete HYPERCARD handbook, 1987 Bantam Books, 666 Fifth Avenue, New York, New York, 10103
- [4] Danny Goodman, HYPERCARD developer's guide, 1988 Bantam Books, 666 Fifth Avenue, New York, New York, 10103
- [5] Robert L. Burke, CAI source book, Englewood Cliffs, N.J. : Prentice-Hall, 1982
- [6] Albert E. Hickey, Computer assisted instruction
- [7] Hypercard User's Guide, Apple Computer Inc. 20525 Mariani Avenue, Cupertino, California 95014 (408)996-1010 TLX 171-576
- [8] Journal of Computer Based Instruction, 1986-1987, Association for the development of Computer-Based Instruction systems, Computer Center, Western Washington University
- [9] Journal of Educational Computing Research, Farmingdale, N.Y. : Baywood Pub. Co., 1986
- [10] Shirley Torgerson, LOGO in the classroom, 1984, ICCE Publications, Eugene, OR, University of Oregon, 1787 Agate St., Eugene 97403-1943, International Council for Computers in Education
- [11] Stephen Alessi, Computer Based Instruction, 1985 Prentice Hall Inc., Englewood Cliffs, N.J.
- [12] Etienne Wenger, Artificial Intelligence and Tutoring Systems, 1987 Morgan Kaufmann Publishers, Inc. 95 First Street, Los Altos, CA 94022
- [13] IEEE Software, IEEE Computer Society, Los Angeles, CA, Jan. 1989
- [14] Smith, D., "DIADES Design Automation System", Technical Report 88-9, Electrical Engineering Dept., Portland State University, Portland, OR, 1988.

- [15] Smith, D., "ADL Reference Manual", Technical Report 88-10, Electrical Engineering Dept., Portland State University, Portland, OR, 1988.
- [16] Smith, D., "MGEN: STRUCT to M-Language Translation", Technical Report 88-11, Electrical Engineering Dept., Portland State University, Portland, OR, 1988.
- [17] Smith D., "Graph Language", Technical Report 89-1, Electrical Engineering Dept., Portland State University, Portland, OR, 1989.
- [18] Marek Pekowski, Ideas about new approach to computer aided instruction for elementary logic design
- [19] Academic Computing, P.O.Box 804, McKinney, TX 75069  
May 1989, Page 20  
February 1988, Page 22  
Dec 87/Jan 88, Page 22, 26
- [20] Wheels for the mind, 1988, Volume 4, number 2  
Pages 17, 95-98
- [21] Coed, Computers in education division of ASEE  
Vol. VIII, No. 3, July/Sep 1988, Page 20, 28  
Vol. VIII, No. 4, Oct/Dec 1988, Page 38
- [22] Perkowski, M, Integration of logic synthesis and high-level synthesis into the DIADES design automation system, Proceedings of the 1989 IEEE International symposium on circuits and systems, Volume 2, pages 748-751
- [23] Perkowski, M, DIADES - A high level synthesis system, Proceedings of the 1989 IEEE International symposium on circuits and systems, Volume 3, pages 1895-1898

APPENDIX A

SOME SAMPLE SCRIPTS

## APPENDIX A

This appendix contains scripts of all stacks along with script of chosen cards, backgrounds, buttons and fields which perform a special function.

\*\*\* Scripts of the stack "start" \*\*\*

\* Stack script

```
on closestack
  global RET, LAST_STK
  put "second card" && "of" && "stack start" into RET
  put RET into LAST_STK
end closestack
```

\* Script of the first card

```
on opencard
  hide menubar
  hide card button 1
  choose browse tool
  set TextFont to New York
  set TextSize to 18
  set TextStyle to outline
  set TextAlign to Center
  click at 240,70
  type "HYPERCARD_BASED"
  click at 240,100
  type "LEARNING ENVIRONMENT"
  click at 240,160
  type "Advisor: Dr. M. Perkowski"
  click at 240,220
  type "Writer: Ali Shamsapour"
  set TextFont to geneva
  set TextSize to 12
```

```
set TextStyle to Bold
set TextAlign to Center
click at 240,250
type " Electrical Engineering Department"
click at 240,270
type " Portland State University"
click at 240,286
type "1989 - 1990"
click at 5,5
wait 2 seconds
go next
end opencard
```

```
on closecard
  global ST_STK
  put "ok" into ST_STK

  put empty into card field 1
  put empty into card field 2
end closecard
```

\* Script of the second card

```
on opencard
  global A_ID,B_ID,D_ID,G_ID,T_ID,ST_STK
  show menubar
  set userLevel to 5
  hide card field id 59
  repeat with x = 164 to 178
    hide card field id x
  end repeat
  hide card button id 2
  hide card button id 53
  hide card button id 54
  hide card button id 55
  hide card button id 56
  hide card button id 57
  hide card button id 62
```

```
hide card button id 102
hide card button id 103
show card button id 1
show card button id 4
show card button id 52
show card button id 179
if ST_STK is "ok" then
  if first line of background field 1 is not empty then
    get first line of background field 1
    put it into A_ID
    put empty into first line of background field 1
  end if
  if second line of background field 1 is not empty then
    put second line of background field 1 into B_ID
    put empty into second line of background field 1
  end if
  if third line of background field 1 is not empty then
    put third line of background field 1 into D_ID
    put empty into third line of background field 1
  end if
  if fourth line of background field 1 is not empty then
    put fourth line of background field 1 into G_ID
    put empty into fourth line of background field 1
  end if
  if fifth line of background field 1 is not empty then
    put fifth line of background field 1 into T_ID
    put empty into fifth line of background field 1
  end if
  --put 164 into begin
  --repeat until the mouseClick
  --repeat with x = 14 to 24
  --set hilite of card button x to true
  --end repeat
  --show card field id begin
  --add 1 to begin
  --if begin is "179" then
  --put 164 into begin
  --repeat with x = 164 to 178
```



```
--hide card field id x
--end repeat
--end if
--repeat with x = 14 to 24
  --set hilite of card button x to false
--end repeat
--show card field id begin
--add 1 to begin
--if begin is "179" then
  --put 164 into begin
  --repeat with x = 164 to 178
    --hide card field id x
  --end repeat
--end if
--end repeat
--repeat with x = 164 to 178
  --hide card field id x
--end repeat
end if
end opencard

on closecard
  global ST_STK
  put empty into ST_STK
  hide card field id 59
  hide card field id 60
  hide card button id 53
  hide card button id 54
  hide card button id 55
  hide card button id 56
  hide card button id 57
  hide card button id 62
  hide card button id 102
  hide card button id 103
end closecard
```

\* Script of the "quit" button

```

on mouseUp
  global A_ID,B_ID,D_ID,G_ID,T_ID
  hide card button id 1
  hide card button id 2
  hide card button id 4
  hide card button id 52
  hide card button id 53
  hide card button id 54
  hide card button id 55
  hide card button id 56
  hide card button id 57
  hide card button id 62
  hide card button id 102
  hide card button id 103
  hide card button id 179
  hide card field id 59
  hide card field id 60
  Answer "Save card numbers before quitting?" with "yes" →
  or "no" or "cancel"
  if it is "cancel" then
    show card button id 1
    show card button id 2
    show card button id 4
    show card button id 52
    show card button id 179
  end if
  if it is "yes" then
    put A_ID into first line of background field 1
    put B_ID into second line of background field 1
    put D_ID into third line of background field 1
    put G_ID into fourth line of background field 1
    put T_ID into fifth line of background field 1
  end if
  if it is "yes" then domenu quit hypercard
  if it is "no" then domenu quit hypercard
end mouseUp

```

\* Script of the "go to lesson ..." button

on mouseUp

```
hide card button id 52
show card field id 59
--select Text in card field id 59
show card button id 53
show card button id 54
show card button id 55
show card button id 56
show card button id 57
show card button id 62
show card button id 102
show card button id 103
repeat until the mouseClick
    set hilite of card button id 53 to true
    set hilite of card button id 54 to true
    set hilite of card button id 55 to true
    set hilite of card button id 56 to true
    set hilite of card button id 57 to true
    set hilite of card button id 62 to true
    set hilite of card button id 102 to true
    set hilite of card button id 103 to true
    set hilite of card button id 53 to false
    set hilite of card button id 54 to false
    set hilite of card button id 55 to false
    set hilite of card button id 56 to false
    set hilite of card button id 57 to false
    set hilite of card button id 62 to false
    set hilite of card button id 102 to false
    set hilite of card button id 103 to false
end repeat
hide card field id 59
show card field id 60
--select Text in card field id 60
end mouseUp
```

\* Script of the "microprogramming" button

```

on mouseUp
  global M_ID,TEMP
  put M_ID && "of" && "EE371 stack" into TEMP
  repeat 5 times
    set hilite of card button ID 179 to true
    wait 1
    set hilite of card button ID 179 to false
    wait 1
  end repeat
  if M_ID is empty then go to card one in stack "EE371 stack"
  else go to TEMP
end mouseUp

```

Note: Other buttons in the "start" stack have scripts similar to "microprogramming" button, except that they are linked to different cards and/or stacks.

\*\*\* Script of the "DIADES" stack \*\*\*

\* Stack script

```

on openstack
  global D_STK
  put "stack DIADES" into D_STK
end openstack

on closestack
  global D_ID,RET,D_STK,LAST_STK
  put D_ID && "of" && D_STK into RET
  put RET into LAST_STK
end closestack

```

on NewCard

```

  put script of card id 5075 into scr
  get the number of this card
  set the script of card it to scr
  doMenu new field
  drag from 164,126 to 24,31
  drag from 345,195 to 241,287
  drag from 128,152 to 371,152 with OptionKey, ShiftKey
  choose browse tool
  put "First field of the new card." into card field 1
  set TextStyle of card field 1 to Bold
  put "Second field of the new card." into card field 2
  set TextStyle of card field 2 to Bold
end NewCard

```

\* Background script of the "DIADES" stack

on openbackground

```

  put the date into background field 2
  put the time into background field 3
end openbackground

```

on idle

```

  put the time into background field 3
end idle

```

\* Script of the first card of the "DIADES" stack

on opencard

```

global D_ID,STK,RET
get the ID of this card
put it into D_ID
put D_ID && "of" && STK into RET
set hilite of card button id 17 to true
put the date into background field 2
put "card " into background field one
get the number of this card
put it after background field one

```

```

put " of " after background field one
get the number of cards
put it after background field one
set lockText of card field 1 to true
set lockText of card field 2 to true
set hilite of card button 1 to true
set hilite of card button 2 to true
set hilite of card button 3 to true
set hilite of card button 4 to true
set hilite of card button 5 to true
end opencard

```

```

on closecard
  global CARD_ID,D_ID
  put D_ID into CARD_ID
end closecard

```

\* Script of the second card

```

on opencard
  global D_ID
  put the date into background field 2
  get the ID of this card
  put it into D_ID
  put "card " into background field one
  get number of this card
  put it after background field one
  put " of " after background field one
  get number of cards
  put it after background field one
  set hilite of card button 1 to true
  set hilite of card button 2 to true
  set hilite of card button 3 to true
  set hilite of card button 4 to true
  set lockText of card field 1 to true
  set lockText of card field 2 to true
end opencard

```

```

on closecard
  global CARD_ID,D_ID
  put D_ID into CARD_ID
end closecard

```

\* Script of the card 3

```

on opencard
  global D_ID
  put the date into background field 2
  get the ID of this card
  put it into D_ID
  put "card " into background field one
  get number of this card
  put it after background field one
  put " of " after background field one
  get number of cards
  put it after background field one
  set hilite of card button 1 to true
  set hilite of card button 2 to true
  set hilite of card button 3 to true
  set hilite of card button 4 to true
  set lockText of card field 1 to true
  set lockText of card field 2 to true
end opencard

```

```

on closecard
  global CARD_ID,D_ID
  put D_ID into CARD_ID
end closecard

```

\* Script of the card 4

```

on opencard
  global D_ID
  set hilite of card button 1 to true
  put the date into background field 2
  get the ID of this card

```

```

put it into D_ID
put "card " into background field one
get number of this card
put it after background field one
put " of " after background field one
get number of cards
put it after background field one
set lockText of card field 1 to true
set lockText of card field 2 to true
end opencard

```

```

on closecard
  global CARD_ID,D_ID
  put D_ID into CARD_ID
end closecard

```

\* Script of the card 5

```

-- Do not modify script of card id 5075.
-- It is used in creating new cards.

```

```

on opencard
  global D_ID
  set hilite of card button 1 to true
  set hilite of card button 2 to true
  put the date into background field 2
  get the ID of this card
  put it into D_ID
  put "card " into background field one
  get number of this card
  put it after background field one
  put " of " after background field one
  get number of cards
  put it after background field one
  set lockText of card field 1 to true
  set lockText of card field 2 to true
end opencard

```



```
on closecard
  global CARD_ID,D_ID
  put D_ID into CARD_ID
end closecard
```

\* Script of card 6

```
on opencard
  global D_ID
  put the date into background field 2
  get the ID of this card
  put it into D_ID
  put "card " into background field one
  get number of this card
  put it after background field one
  put " of " after background field one
  get number of cards
  put it after background field one
  set lockText of card field 1 to true
  set lockText of card field 2 to true
end opencard
```

```
on closecard
  global CARD_ID,D_ID
  put D_ID into CARD_ID
end closecard
```

\* Script of the card 7 (pop quiz)

```
on opencard
  hide menubar
  hide card field id 9
  hide card field id 10
  hide card field id 11
  hide card field id 12
  set hilite of card button 1 to true
  set hilite of card button 2 to true
  set hilite of card button 3 to true
```

```
    set hilite of card button 4 to true
end opencard
```

\* Script of the card 8 (pop quiz)

```
on opencard
    hide card field id 9
    hide card field id 10
    hide card field id 11
    hide card field id 12
    set hilite of card button 1 to true
    set hilite of card button 2 to true
    set hilite of card button 3 to true
    set hilite of card button 4 to true
end opencard
```

\* Script of the card 9 (pop quiz)

```
on opencard
    hide card field id 8
    hide card field id 9
    hide card field id 10
    hide card field id 11
    hide card field id 12
    hide card field id 13
    hide card field id 16
    hide card button id 14
    hide card button id 15
    set hilite of card button 1 to true
    set hilite of card button 2 to true
    set hilite of card button 3 to true
    set hilite of card button 4 to true
end opencard
```

```
on closecard
    hide card field id 12
    hide card field id 13
```

```
hide card field id 16
hide card button id 14
hide card button id 15
show menubar
end closecard
```

\* Script of a answer button (A)

```
on mouseUp
  global score
  put "1" into score
  show card field id 9
  wait 2 seconds
  show card field id 10
  wait 4 seconds
  hide card field id 9
  hide card field id 10
  go next
end mouseUp
```

\* Script of a answer button (B)

```
on mouseUp
  global score
  put "0" into score
  show card field id 11
  wait 2 seconds
  show card field id 12
  wait 5 seconds
  hide card field id 11
  hide card field id 12
  go next
end mouseUp
```

\* Script of answer button (C)

```
on mouseUp
  global score
```

```
put "0" into score
show card field id 11
wait 2 seconds
show card field id 12
wait 5 seconds
hide card field id 11
hide card field id 12
go next
end mouseUp
```

\* Script of answer button (D)

```
on mouseUp
global score
put "0" into score
show card field id 11
wait 2 seconds
show card field id 12
wait 5 seconds
hide card field id 11
hide card field id 12
go next
end mouseUp
```

\* Script of the card 10

```
on opencard
global D_ID,CARD_ID
set hilite of card button id 7 to true
put the date into background field 2
get the ID of this card
put it into D_ID
put "card " into background field one
get number of this card
put it after background field one
put " of " after background field one
get number of cards
put it after background field one
```

```
set hilite of card button 1 to true
set hilite of card button 2 to true
set lockText of card field 1 to true
set lockText of card field 2 to true
end opencard
```

```
on closecard
  global CARD_ID,D_ID
  put D_ID into CARD_ID
end closecard
```

\* Script of the card 11

```
on opencard
  global D_ID
  put the date into background field 2
  get the ID of this card
  put it into D_ID
  put "card " into background field one
  get number of this card
  put it after background field one
  put " of " after background field one
  get number of cards
  put it after background field one
  set hilite of card button 1 to true
  set lockText of card field 1 to true
  set lockText of card field 2 to true
end opencard
```

```
on closecard
  global CARD_ID,D_ID
  put D_ID into CARD_ID
end closecard
```

\* Script of the card 12

```
on openstack
  global D_STK
```

```
    put "stack DIADES" into D_STK
end openstack

on closestack
    global D_ID,RET,D_STK,LAST_STK
    put D_ID && "of" && D_STK into RET
    put RET into LAST_STK
end closestack
```

```
on NewCard
    put script of card id 5075 into scr
    get the number of this card
    set the script of card it to scr
    doMenu new field
    .....
    .....
```

Note: All of the cards in the "DIADES" stack have almost the same script.

\* Script of a sample "pop quiz" card in the "DIADES" stack

```
on opencard
    hide card field id 8
    hide card field id 9
    hide card field id 10
    hide card field id 11
    hide card field id 12
    hide card field id 13
    hide card field id 16
    hide card button id 14
    hide card button id 15
    set hilite of card button 1 to true
    set hilite of card button 2 to true
    set hilite of card button 3 to true
    set hilite of card button 4 to true
end opencard
```

```

on closecard
  hide card field id 12
  hide card field id 13
  hide card field id 16
  hide card button id 14
  hide card button id 15
  show menubar
end closecard

```

\* Script of the "quit" button

```

on mouseUp
  go to card id 4505 of stack "start"
  click at 49,291
end mouseUp

```

\* Script of the "?" button

```

on mouseUp
  repeat 5 times
    set hilite of bkgnd button id 33 to true
    set hilite of bkgnd button id 33 to false
  end repeat
  go to card id 2964 of stack "HELP"
end mouseUp

```

Note: Buttons which link one stack to another, have the same script as "?" button, except that they are linked to different stacks.

\* Script of the "open" button

```

on mouseUp
  get the number of card fields
  if it is 1 then
    set lockText of card field 1 to false
  else if it >= 2 then
    set lockText of card field 1 to false
  end if
end mouseUp

```

```

    set lockText of card field 2 to false
end if
show background button id 97
wait 3 seconds
hide background button id 97
end mouseUp

```

\* Script of the "find" button

```

on mouseUp
    global temp
    Ask "Enter the word to be found."
    if it is not empty then
        show message box at 18,361
        put "find whole " & quote & it & quote into message box
        put " -- To find next occurrence press return key." after
        message box
        do message box
        if the result is not empty then
            put quote & it & quote && "not found" && "in this stack." into
            temp
            Answer temp
            hide message box
        end if
    end if
end mouseUp

```

\* Script of the "return" button

```

on mouseUp
    global LAST_STK
    repeat 5 times
        set hilite of background button ID 78 to true
        set hilite of background button ID 78 to false
    end repeat
    if LAST_STK is not empty then go to LAST_STK
end mouseUp

```



\* Script of the "right arrow" button

```
on mouseUp
  get number of this card
  put it into num
  get number of cards
  if it=num then go to card ID 2892
  else go to next card
end mouseUp
```

\* Script of the left arrow button

```
on mouseUp
  visual effect scroll right
  go to prev card
end mouseUp
```

\* Script of a "answer" button in a pop quiz

```
on mouseUp
  global score
  put "1" into score
  show card field id 9
  wait 2 seconds
  show card field id 10
  wait 4 seconds
  hide card field id 9
  hide card field id 10
  go next
end mouseUp
```

\* Script of an "evaluation" button in a pop quiz

```
on mouseUp
  global score
  show card field id 8
  wait 2 seconds
  show card field id 9
```

```

wait 5 seconds
hide card field id 8
hide card field id 9
if score = 2 then
  show card field id 16
  wait 3 seconds
  go next
else
  show card field id 13
  show card button id 14
  show card button id 15
  set hilite of card button id 14 to true
  set hilite of card button id 15 to true
end if
end mouseUp

```

\*\*\* Script of the "ADL" stack "

\* Script of a sample card

```

on opencard
  global A_ID,NAM
  put the date into background field 2
  set hilite of card button 1 to true
  set hilite of card button 3 to true
  set hilite of card button 4 to true
  set hilite of card button 5 to true
  set hilite of card button 6 to true
  get the ID of this card
  put it into A_ID
  if NAM is "intro" then set hilite of card button 2 to true
  put "card " into background field one
  get the number of this card
  put it after background field one
  put " of " after background field one
  get the number of cards
  put it after background field one
  set lockText of card field 1 to true

```

```
    set lockText of card field 2 to true
end opencard
```

```
on closecard
  global A_ID,CARD_NO,NAM
  put A_ID into CARD_NO
  put empty into NAM
  set hilite of card button 2 to false
end closecard
```

Note: Most of the buttons and cards in the "ADL" stack have similar scripts as "DIADES" stack. So, most of the scripts in this stack are not printed out, except those that are different from other stacks.

\* Script of a "pop quiz" card in "ADL" stack

```
on opencard
  set hilite of card button 1 to true
  set hilite of card button 2 to true
  set hilite of card button 3 to true
  set hilite of card button 4 to true
  hide card field id 13
  hide card field id 14
  hide card field id 15
  hide card field id 16
  hide card field id 17
  hide card field id 18
  hide card field id 22
  hide card field id 23
  hide card button id 24
  hide card button id 25
end opencard
```

```
on closecard
  hide card field id 13
  hide card field id 14
  hide card field id 15
```

```

hide card field id 16
hide card field id 17
hide card field id 18
hide card field id 22
hide card field id 23
hide card button id 24
hide card button id 25
show menubar
end closecard

```

\*\*\* Scripts in the "HELP" stack \*\*\*

\* Stack script

```

on closestack
  global RET, LAST_STK
  put "first card" && "of" && "stack HELP" into RET
  put RET into LAST_STK
end closestack

```

\* Script of a sample card in "HELP" stack

```

on opencard
  show background button id 8
  set hilite of background button id 8 to true
  choose browse tool
  click at 66,79
  set TextFont to New York
  set TextSize to 14
  set TextStyle to Bold
  type "This tutorial contains lessons about DIADES, ADL, ADL BUGS,"
  type " TAG user's guide and a GLOSSARY."
  wait 3 seconds
  type " To go to one stack simply click on the button corresponding"
  type " to that stack."
  wait 3 seconds

```

```

click at 66,180
type "To get more information about tutorial click on the more
button."
wait 3 seconds
type " To go back and start lesson click on return button."
set hilite of background button id 8 to false
end opencard

```

```

on closecard
  put empty into card field 1
end closecard

```

\* Script of the "return" button

```

on mouseUp
  global LAST_STK
  repeat 8 times
    set hilite of background button ID 9 to true
    set hilite of background button ID 9 to false
  end repeat
  if LAST_STK is not empty then go to LAST_STK
end mouseUp

```

\*\*\* Scripts of the "GLOSSARY" stack \*\*\*

\* Script of a sample card

```

on opencard
  global G_ID,G_STK,RET,NAME
  get the ID of this card
  put it into G_ID
  put G_ID && "of" && G_STK into RET
  set hilite of card button id 15 to true
  put the date into background field 2
  put "card " into background field one
  get number of this card
  put it after background field one
  put " of " after background field one

```

```

get number of cards
put it after background field one
set lockText of card field 1 to true
set lockText of card field 2 to true
if NAME is "ALGORITHM" then set hilite of card button 1 to true
if NAME is "ADL" then set hilite of card button 2 to true
if NAME is "ASSIGNMENT" then set hilite of card button 3 to true
end opencard

```

```

on closecard
global NAME
set hilite of card button 1 to false
set hilite of card button 2 to false
set hilite of card button 3 to false
put empty into NAME
end closecard

```

Note: "GLOSSARY" stack and most of it's cards have almost the same script as "DIADES" stack with minor differences. So these scripts are not printed.

\*\*\* Scripts in the "Home" stack \*\*\*

\* Stack script (this is copied from HyperCard's "HOME" stack

```

on startUp
getHomeInfo
pass startUp -- to a startUp XCMD, if present
end startUp

```

```

on resume
getHomeInfo
pass resume -- to a resume XCMD, if present
end resume

```

```

on getHomeInfo
global stacks,applications,documents,userName
set lockScreen to true

```

```

set lockMessages to true
push this card
go to card "User Preferences" of stack "Home"
put card field "User Name" into userName
set userLevel to card field "User Level"
set powerKeys to the hilite of button "Power Keys"
--set textArrows to the hilite of button "Text Arrows"
set blindTyping to the hilite of button "Blind Typing"
put field "paths" of card "stacks" into stacks
put field "paths" of card "applications" into applications
put field "paths" of card "documents" into documents
pop card
set lockScreen to false
set lockMessages to false
end getHomeInfo

on searchScript pattern,stackName -- search all scripts of a stack
set lockMessages to true
if stackName is not empty then go to stack stackName

if the script of this stack contains pattern
then edit script of this stack

repeat with i = 1 to the number of bkgnds
go to card 1 of bkgnd i
if the script of this bkgnd contains pattern
then edit script of bkgnd

repeat with j = 1 to the number of bkgnd buttons
if the script of bkgnd button j contains pattern
then edit script of bkgnd button j
end repeat

repeat with j = 1 to the number of bkgnd fields
if the script of bkgnd field j contains pattern
then edit script of bkgnd field j
end repeat
end repeat
end repeat

```

```
repeat with i = 1 to the number of cards
  go card i
  if the script of this card contains pattern
  then edit script of this card

  repeat with j = 1 to the number of card buttons
    if the script of card button j contains pattern
    then edit script of card button j
  end repeat

  repeat with j = 1 to the number of card fields
    if the script of card field j contains pattern
    then edit script of card field j
  end repeat

end repeat
set lockMessages to false
end searchScript

* Script of the first card of the "Home" stack

on idle
  put the time into card field "Time"
  pass idle
end idle

on startup
  show card field "Copyright"
end startup

on closeCard
  hide card field "Copyright"
end closeCard

* Script of the field "paths" in the second card of the "Home" stack

on closeField
  global stacks,applications,documents
```



```

    put field "paths" of card "stacks" into stacks
    put field "paths" of card "applications" into applications
    put field "paths" of card "documents" into documents
end closeField

```

\* Script of the "user preferences" card of the "Home" stack

```

on openCard

```

```

    setUserLevel the userLevel

```

```

    if card field "User Name" is empty

```

```

        then click at the loc of card field "User Name"

```

```

    end openCard

```

```

on setUserLevel whatLevel

```

```

    set userLevel to whatLevel

```

```

    if the userLevel is whatLevel then

```

```

        put the userLevel into card field "User Level"

```

```

        set hilite of button "Browsing" to the userLevel = 1

```

```

        set hilite of button "Typing" to the userLevel = 2

```

```

        set hilite of button "Painting" to the userLevel = 3

```

```

        set hilite of button "Authoring" to the userLevel = 4

```

```

        set hilite of button "Scripting" to the userLevel = 5

```

```

        set visible of button "Text Arrows" to the userLevel >= 2

```

```

        set visible of button "Power Keys" to the userLevel >= 3

```

```

        set visible of button "Blind Typing" to the userLevel = 5

```

```

        set hilite of button "Text Arrows" to the textArrows

```

```

        set hilite of button "Power Keys" to the powerKeys

```

```

        set hilite of button "Blind Typing" to the blindTyping

```

```

    else

```

```

        set hilite of the target to false

```

```

    end if

```

```

end setUserLevel

```

\* Script of the field "User Nmae" in the "user preferences" card

```

on closeField

```

```

    global userName

```

```

    put card field "User Name" into userName

```

end closeField

\* Script of the button "Scripting" in the "User Preferences" card

```
on mouseUp
  setUserLevel 5
end mouseUp
```

\* Script of the button "Power Keys" in the "User Preferences" card

```
on mouseUp
  set powerKeys to the hilite of button "Power Keys"
end mouseUp
```

\*\*\* Scripts in the "FINAL" stack \*\*\*

\* Script of the button "start final" (see note at the end of scripts)

```
on mouseUp
  global Finalsum
  global startfinal
  global endtime
  put 0 into Finalsum
  put 1 into startfinal
  put the seconds into endtime
  add 3600 to endtime
  visual effect iris close
  go to next card
  visual effect iris open
end mouseUp
```

\* Script of the button "time remaining"

```
on mouseUp
  global endtime
  put endtime - the seconds into timeleft
  put timeleft div 60 into minleft
  put timeleft mod 60 into secleft
```

```

put minleft into background field id 18
if secleft < 10 then
  put "0" & secleft into background field id 15
else
  put secleft into background field id 15
end if
show background field id 18
show background field id 17
show background field id 15
wait 1 seconds
hide background field id 18
hide background field id 17
hide background field id 15
end mouseUp

```

\* Script of the background of the "FINAL" stack

```

on idle
  global startfinal
  global endtime
  if startfinal = 1 and endtime < the seconds then
    visual effect zoom close very slow to black
  end if
  pass idle
end idle

```

Note: Most of the "FINAL" stack scripts are copied from the microprogramming stacks written by Mark Presentin.

## APPENDIX B

### DESIGN DATA FOR SIGNAL DELAY PROCESSOR

## APPENDIX B

## DESIGN DATA FOR SIGNAL DELAY PROCESSOR

1) The description of signal delay device in ADL :

```

graph
subgraph
  (((adl c opimp
    ((clock(1000)))
    (input(a(d))(t(p k1 8)))
    (intern(l(p k1 8))(lt(p k1 8))(f(p k1 8)))
    (output(b(d))))
    ((start) c
10 (sim (l := 0) (lt := t)(f := 0))
    (if(! a)then(go 10))
    (fork
      (15 (f := (f + 1))
        (if !(f = 100)) then (go 15)))
      (12(lt := lt)
        (lt := (lt - 1))
        (if(!(lt = 0))then(go 12)))
      (11 (l := (l + 1))
        (if(and a (!(lt = 0))) then(go 11))
        (drop))
      dand)
13(if a then (b == 1)(go 13) )
14(sim
  (l := (l - 1))
  (b == 1))
    (if(l = 0)then(go 10)else(go 14))
    )))
end

```

2) The cf-graph description of signal delay device :

a) List of arrows :

```

(setq *coplisset* '((1
  ((not 22) 22 19)
  (22 22 2)
  (x 19 22)
  ((not 6) 17 19)
  (x 18 17)
  (6 17 18)
  (e 16 17)
  (12 12 16)

```

```

(9 9 16)
((not 14) 14 15)
(14 14 13)
(x 13 14)
(e 7 13)
((not 12) 12 10)
(x 11 12)
(x 10 11)
(e 7 10)
((not 9) 9 8)
(x 8 9)
(e 7 8)
(6 6 7)
((not 6) 6 2)
(x 2 6)
(x 1 2)))) )

```

b) List of node properties :

```

(setq *nolisser* ' ((1
((cond 22 nil)
(19 19 nil)
(18 18 nil)
(cond 17 nil)
(dand 16 nil)
(drop 15 nil)
(cond 14 nil)
(13 13 nil)
(cond 12 nil)
(11 11 nil)
(10 10 nil)
(cond 9 nil)
(8 8 nil)
(fork 7 nil)
(cond 6 nil)
(2 2 nil)
(start 1 nil)))) )

```

c) List of Assignments :

```

(setq *nalisset* ' ((1
((19 (18 20))
(20 (:= 1 (plus 1 (minus 1))))
(18 (== b 1))
(13 (:= 1 (plus 1 1)))
(11 (:= lt (plus lt (minus 1))))
(10 (:= lt lt))
(8 (:= f (plus f 1)))
(2 (3 4 5))
(5 (:= f 0))
(4 (:= lt t))
(3 (:= 1 0)))))) )

```

d) List of predicates :

```

(setq *plisset* ' ((1

```

```

((22 (equal l 0))
 (14 (and a (not (equal lt 0))))
 (12 (equal lt 0))
 (9 (equal f 100))
 (6 a)))) )

```

e) List of node groups :

```

(setq *anlisset* ' ((1
((cond (6 9 12 14 17 22))
 (19 (19))
 (18 (18))
 (dand (16))
 (drop (15))
 (13 (13))
 (11 (11))
 (10 (10))
 (8 (8))
 (fork (7))
 (2 (2))
 (start (1)))))) )

```

f) List of memory variables :

```

(setq *lzmset* ' ((1 (b f lt 1))) )

```

g) List of systems under design :

```

(setq *symlis* ' ((c 1)) )

```

3) Compact parallel cf-graph description :

a) Transition description list :

```

((1 (1 1))
 (9 (7 2))
 (9 (9 3))
 (7 (7 4))
 (7 (2 5))
 (8 (9 6))
 (8 (8 7))
 (6 (9 8))
 (6 (5 9))
 (3 (3 10))
 (8 (7 11))
 (2 (8 12))
 (6 (7 13))
 (5 (6 14))
 (2 (5 15))
 (3 (4 16))
 (2 (3 17))
 (2 (2 18))
 (1 (2 19)))

```

b) Relation description list :

```

((1 (1) ((not (start))))

```

(2 (18 20) ((not 6)))  
 (3 (18) (6))  
 (4 (18 20) ((not 22)))  
 (5 (3 4 5) (22))  
 (6 (18) (6 9))  
 (7 (8) ((not 9)))  
 (8 (18) (6 12))  
 (9 (10) ((not 12)))  
 (10 (13) (14))  
 (11 (18 20) ((not 6) 9))  
 (12 (8) (6))  
 (13 (18 20) ((not 6) 12))  
 (14 (11) nil)  
 (15 (10) (6))  
 (16 (drop) ((not 14)))  
 (17 (13) (6))  
 (18 (3 4 5) ((not 6)))  
 (19 (3 4 5) ((start)))

c) Passing *FORK* transitions list :  
 ((2 8) (2 5) (2 3))

d) Passing *DAND* transitions list :  
 ((8 9) (6 9) (8 7) (6 7))

4) *Kiss* format and state table descriptions of FSM control unit :

```

----0 st1 st1 0000000
----1 st1 st2 1110000
----1- st2 st3 0001010
----0- st2 st2 1110000
-1-11- st3 st4 0000110
-0-11- st3 st5 0000100
-1-0-- st3 st6 0001110
-0-0-- st3 st7 0001100
-0-10- st3 st5 0000100
--111- st7 st8 0000000
--011- st7 st9 0000000
--101- st7 st10 0001000
--00-- st7 st11 0001000
--100- st7 st10 0001000
--010- st7 st9 0000000
--110- st7 st12 0000001
---11- st11 st5 0000100
---0-- st11 st7 0001100
---10- st11 st5 0000100
---11- st10 st8 0000000
---0-- st10 st10 0001000
---10- st10 st12 0000001
----- st9 st5 0000100
-0111- st6 st8 0000000
-1011- st6 st13 0000010
-0011- st6 st9 0000000
  
```



-0101- st6 st10 0001000  
 -100-- st6 st14 0001010  
 -000-- st6 st11 0001000  
 -0100- st6 st10 0001000  
 -0010- st6 st9 0000000  
 -0110- st6 st12 0000001  
 -1-11- st14 st4 0000110  
 -0-11- st14 st5 0000100  
 -1-0-- st14 st6 0001110  
 -0-0-- st14 st7 0001100  
 -0-10- st14 st5 0000100  
 -1---- st13 st4 0000110  
 -0---- st13 st5 0000100  
 --1-1- st5 st8 0000000  
 --0--- st5 st9 0000000  
 --1-0- st5 st12 0000001  
 -01-1- st4 st8 0000000  
 -10--- st4 st13 0000010  
 -00--- st4 st9 0000000  
 -01-0- st4 st12 0000001  
 0----- st12 st12 0000001  
 1----- st12 st2 1110000  
 ----0- st8 st12 0000001  
 ----1- st8 st8 0000000

.ip  
 -----0  
 -----1  
 --111-  
 --011-  
 --101-  
 --00--  
 --100-  
 --010-  
 --110-  
 ---11-  
 ---0--  
 ---10-  
 -----  
 -0111-  
 -1011-  
 -0011-  
 -0101-  
 -100--  
 -000--  
 -0100-  
 -0010-  
 -0110-  
 -1-11-  
 -0-11-  
 -1-0--  
 -0-0--  
 -0-10-

-1----

-0----

--1-1-

--0---

--1-0-

-01-1-

-10---

-00---

-01-0-

0-----

1-----

----0-

----1-

0001010

0001000

0001110

0001100

0000110

0000100

0000010

1110000

0000001

0000000

.dr

0100

1000

00011111011001111111001110001101110010

001011110110100111111001110010110010010

001101111010111011111110010001101110010

0011101111010111100111110010010110010000

0011110111010111111011110010011011100001

0011111011100111111101111100010110010001

0011111101100111111110111100011011100001

0000111110110000111111001110000100010010

001100011101011100001111001000000000000000

0011111001100111111100111100010010000001

00

0001111110110011111111101111001101110010

0010111110110101111111011110110110110010

0010111110110110111111101111010111010010

001101111101011101111111011001101110010

0011101111010111101111110110110110110000

0011101111010111110111111011010111010000

0011110111010111111011111011011011100001

0011111011100111111101111101010111010001

000011111011010111111011110100110110010

000011111011001011111101111000101010010

0011000111010111101111110110100010110000

0011000111010111010011111011000001000000

0011111001100111111100111101010011000001

0000000000000101101111010110100010110000

00000000000000010010000101001000001000000

```

0001011110010011011111000010001101110010
0010101010000100100101000000010110010000
0011110101000111111010110000011011100001
0001011110010011011111101011001101110010
0010101010000101101111010110110110110000
0010101010000110110101101001010111010000
0011110101000111111010111001011011100001
00000000000000000000000000000000000000100
000000000000000000000000000000000000001000
0011100001000111100000110000010010000001
0000001110010000000111000010000100010010
end

```

### 5) Minimized *Kiss* format format description :

```

-----0 st 1 st 1 0000000
-----1 st 1 st 2 1110000
----0- st 2 st 2 1110000
----1- st 2 st13 0001010
-01-1- st 3 st 7 0000000
-10--- st 3 st12 0000010
-00--- st 3 st 8 0000000
-01-0- st 3 st11 0000001
--1-1- st 4 st 7 0000000
--0--- st 4 st 8 0000000
--1-0- st 4 st11 0000001
-0111- st 5 st 7 0000000
-1011- st 5 st12 0000010
-0011- st 5 st 8 0000000
-0101- st 5 st 9 0001000
-100-- st 5 st13 0001010
-000-- st 5 st10 0001000
-0100- st 5 st 9 0001000
-0010- st 5 st 8 0000000
-0110- st 5 st11 0000001
--111- st 6 st 7 0000000
--011- st 6 st 8 0000000
--101- st 6 st 9 0001000
--00-- st 6 st10 0001000
--100- st 6 st 9 0001000
--010- st 6 st 8 0000000
--110- st 6 st11 0000001
----0- st 7 st11 0000001
----1- st 7 st 7 0000000
----- st 8 st 4 0000100
---11- st 9 st 7 0000000
---0-- st 9 st 9 0001000
---10- st 9 st11 0000001
---11- st10 st 4 0000100
---0-- st10 st 6 0001100
---10- st10 st 4 0000100
0----- st11 st11 0000001
1----- st11 st 2 1110000

```

```

-1---- st12 st 3 0000110
-0---- st12 st 4 0000100
-1-11- st13 st 3 0000110
-0-11- st13 st 4 0000100
-1-0-- st13 st 5 0001110
-0-0-- st13 st 6 0001100
-0-10- st13 st 4 0000100

```

## 6) Input and output implications

input-implication:

```

((equal l 0) (equal f 10) (equal lt 0) (and a (not (equal lt 0))) a
(start))

```

output-implication:

```

( (= 1 0)
  (= lt t)
  (= e 0)
  (= f 0)
  (= 1 (plus l 1))
  (= lt (plus lt (minus 1)))
  (= e (plus e 1))
  (= f (plus f 1))
  (= 1 (plus l (minus 1))))

```

## 7) Encoded states :

```

st1      0 0 0 0
st2      0 0 0 1
st3      1 1 0 1
st4      0 1 0 0
st5      1 0 0 0
st6      1 1 1 0
st7      1 1 0 0
st8      0 1 1 0
st9      1 0 1 0
st10     0 0 1 0
st11     1 0 0 1
st12     1 1 1 1
st13     0 1 0 1

```

## 8) Truth-table description for the combinational part of FSM :

```

.i 10
.o 11

----1 0000 0001 1110000
----0 0000 0000 0000000
----1- 0001 0101 0001010
----0- 0001 0001 1110000
-01-0- 1101 1001 0000001
-00-- 1101 0110 0000000
-10-- 1101 1111 0000010

```

```

-01-1- 1101 1100 0000000
--1-0- 0100 1001 0000001
--0--- 0100 0110 0000000
--1-1- 0100 1100 0000000
-0110- 1000 1001 0000001
-0010- 1000 0110 0000000
-0100- 1000 1010 0001000
-000-- 1000 0010 0001000
-100-- 1000 0101 0001010
-0101- 1000 1010 0001000
-0011- 1000 0110 0000000
-1011- 1000 1111 0000010
-0111- 1000 1100 0000000
--110- 1110 1001 0000001
--010- 1110 0110 0000000
--100- 1110 1010 0001000
--00-- 1110 0010 0001000
--101- 1110 1010 0001000
--011- 1110 0110 0000000
--111- 1110 1100 0000000
---1- 1100 1100 0000000
---0- 1100 1001 0000001
----- 0110 0100 0000100
---10- 1010 1001 0000001
---0-- 1010 1010 0001000
---11- 1010 1100 0000000
---10- 0010 0100 0000100
---0-- 0010 1110 0001100
---11- 0010 0100 0000100
1---- 1001 0001 1110000
0---- 1001 1001 0000001
-0---- 1111 0100 0000100
-1---- 1111 1101 0000110
-0-10- 0101 0100 0000100
-0-0-- 0101 1110 0001100
-1-0-- 0101 1000 0001110
-0-11- 0101 0100 0000100
-1-11- 0101 1101 0000110
.e

```

9) Optimized truth-table description for the combinational part of FSM :

```

.i 10
.o 11
.p 36
-001--1000 01100000000
-0-11-1--0 01000000000
-1011-1000 11110000010
-0110-1--0 00010000001
-0-0--0101 10100001000
-100--1000 01010001010
-1-0--0101 10000001110
---11--1-0 01000000000

```

```

-1-11-0101 11010000110
--110-11-0 00010000001
--01--1110 01100000000
---11-1010 11000000000
----0-0001 000111110000
-01-1-110- 11000000000
-----10000 000111110000
-0-0--10-0 00100001000
---10-1010 10010000001
-0---0101 01000000100
-01-0-110- 10010000001
1-----1001 000111110000
--0---0100 01100000000
--0---1101 01100000000
-10---11-1 10010000010
----1-0001 01010001010
-1---1111 10010000010
0-----1001 10010000001
--0--1-10 00100001000
--1-1--100 11000000000
--1-0--100 10010000001
--1---1-10 10000000000
-01---1--0 10000000000
---0---010 10100001000
-----1111 01000000100
----1-1100 11000000000
----0-1100 10010000001
-----0-10 01000000100
.e

```

10) The result of the conversion from truth-table format to EQN format :

```

Q1 = IN2 & IN3 & IN4 & ! IN6 & IN7 & ! IN8 & ! IN9
Q2 = IN1 & IN2 & ! IN3 & ! IN6 & IN7 & ! IN8 & ! IN9
Q3 = ! IN2 & IN3 & ! IN4 & ! IN6 & IN7 & ! IN8 & ! IN9
Q4 = IN2 & IN3 & ! IN4 & ! IN6 & IN7 & ! IN8 & ! IN9
Q5 = IN2 & ! IN3 & ! IN4 & ! IN6 & IN7 & ! IN8 & ! IN9
Q6 = ! IN2 & ! IN3 & ! IN4 & ! IN6 & IN7 & ! IN8 & ! IN9
Q7 = ! IN2 & IN4 & ! IN6 & IN7 & ! IN8 & ! IN9
Q8 = IN5 & ! IN6 & ! IN7 & ! IN8 & ! IN9
Q9 = IN6 & ! IN7 & ! IN8 & IN9
Q10 = ! IN6 & ! IN7 & IN8 & IN9
Q11 = IN6 & ! IN7 & IN8 & ! IN9
Q12 = IN6 & IN7 & ! IN8 & ! IN9
Q13 = IN6 & ! IN7 & ! IN8 & ! IN9
Q14 = ! IN6 & IN7 & IN8 & IN9
Q15 = ! IN6 & ! IN7 & ! IN8 & IN9
Q16 = IN6 & IN7 & IN8 & ! IN9
Q17 = ! IN6 & IN7 & IN8 & ! IN9
Q18 = ! IN6 & IN7 & ! IN8 & IN9
OUT1 = Q7 | Q12 | Q15 | Q16
OUT2 = Q3 | Q6 | Q7 | Q9 | Q18
OUT3 = Q2 | Q4 | Q5 | Q6 | Q10 | Q11

```

```
| & IN3 & IN4 & ! IN6 & IN7  
| Q13 | Q14 | Q16 | Q17 | Q18  
OUT4 = Q3 | Q8 | Q15 | Q17 | Q18  
OUT5 = Q3 | Q6 | Q7  
OUT6 = Q8  
OUT7 = Q12  
OUT8 = Q15 | Q16 | Q17 | Q18  
OUT9 = Q9  
OUT10 = Q9  
OUT11 = Q4  
OUT12 = Q11 | Q14  
OUT13 = Q10 | Q11  
OUT14 = Q2 | Q5  
OUT15 = Q1  
OUT16 = Q1  
OUT17 = Q15 | Q16 | Q17 | Q18  
OUT18 = Q13
```

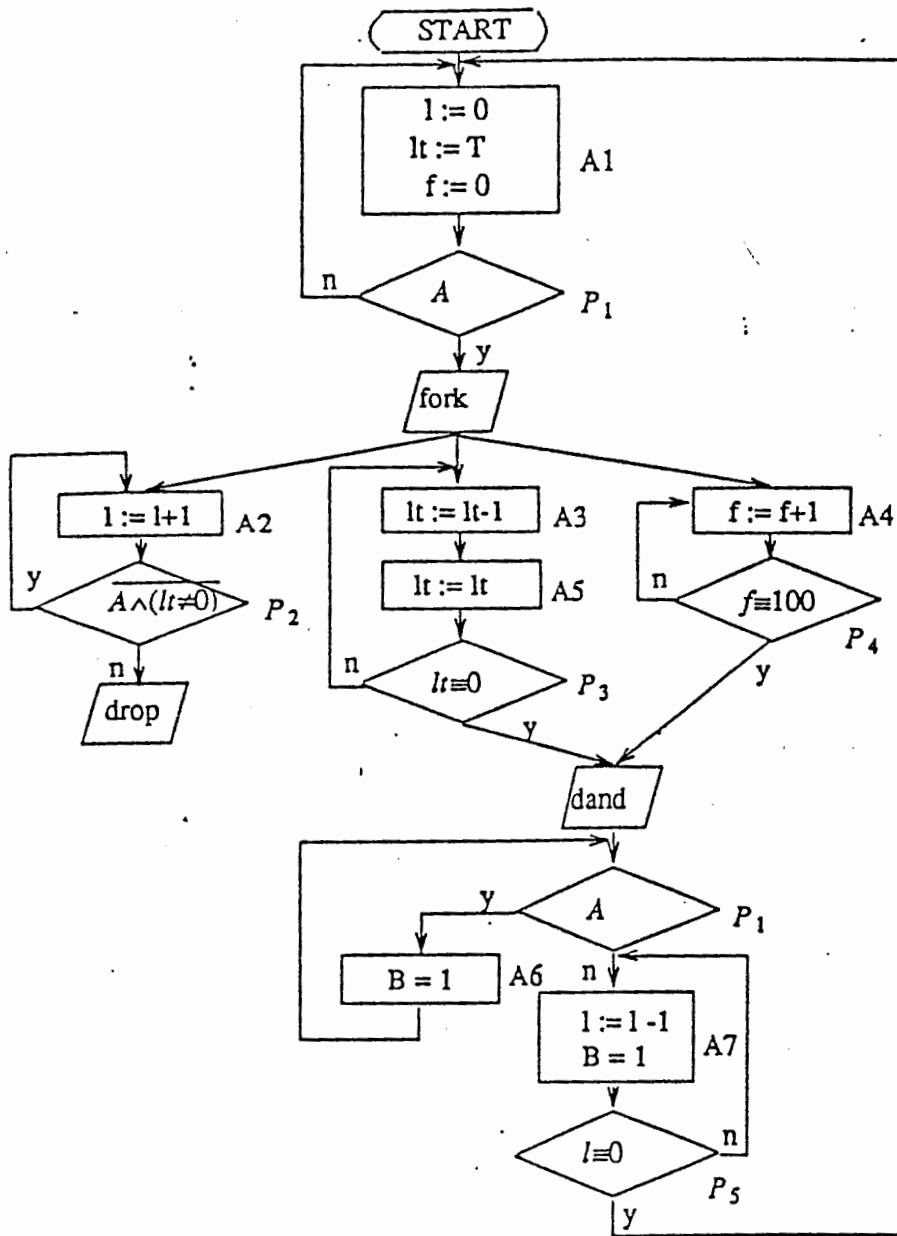


Figure 16. The parallel control flow diagram



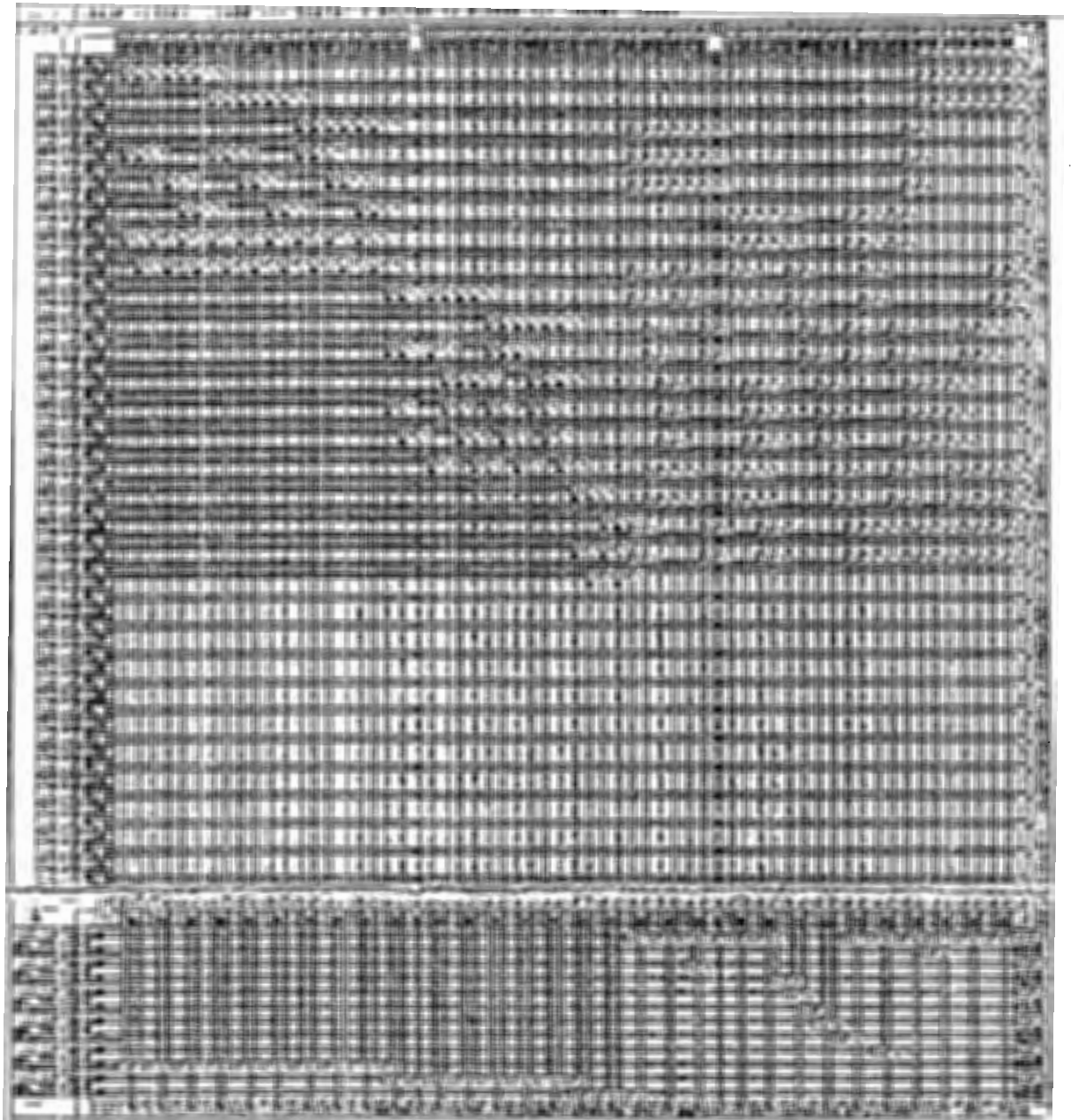


Figure 17. Layout of the control part of the system