Missouri S&T Library and Learning Resources

Scholars' Mine

Masters Theses | Student Theses and Dissertations

1971

A hardware and software interface between a graphics terminal and the SCC 650 computer

George Irvin Rhine Jr.

Follow this and additional works at: https://scholarsmine.mst.edu/masters_theses

 Part of the Electrical and Computer Engineering Commons

Department:

Recommended Citation

Rhine, George Irvin Jr., "A hardware and software interface between a graphics terminal and the SCC 650 computer" (1971). *Masters Theses*. 5508.
https://scholarsmine.mst.edu/masters_theses/5508

This thesis is brought to you by Scholars' Mine, a service of the Missouri S&T Library and Learning Resources. This work is protected by U. S. Copyright Law. Unauthorized use including reproduction for redistribution requires the permission of the copyright holder. For more information, please contact scholarsmine@mst.edu.

A HARDWARE AND SOFTWARE INTERFACE BETWEEN A GRAPHICS TERMINAL AND THE SCC 650 COMPUTER

BY

GEORGE IRVIN RHINE, JR., 1942-

A THESIS

Presented to the Faculty of the Graduate School of the

UNIVERSITY OF MISSOURI-ROLLA

In Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

1971

T2580
78 pages
c.1

Approved by

James R. Tracy (Advisor) William K. Trank

Herbert R. Alcorn

194213

ABSTRACT

This paper describes the design of a digital interface between a graphic terminal and the SCC-650 computer. The graphic terminal-computer combination can be used as a stand alone system for small applications or can be used as a satellite processor for a larger system such as the IBM System 360.

The interface is designed utilizing DTL NAND type integrated circuits. Its primary functions may be divided into three main categories: (1) to give level conversion, (2) control, and (3) data manipulation.

All requests for data transfers are initiated by the computer. The interface then assumes a control mode which handles the data transfer to or from the graphics terminal. Once the transfer is complete, the computer is notified that it may initiate another request. Data is converted from bit serial to parallel word form by the interface during the data transfer. A function keyboard has been implemented which may transfer any one of 2048 different command words to the computer.

A software package was written in SCC-650 Assembler which will utilize the graphics terminal as an input/output processor for an electronic circuit design program such as ECAP or CIRCUS. This program will allow a user to draw the exact circuit to be analyzed on the graphics terminal and then ask for specific results to be displayed in either numerical or graphical form. The circuit may then be changed by adding or deleting elements and re-analyzed.

ACKNOWLEDGEMENTS

The author wishes to thank Dr. James H. Tracey for his guidance and support during this project. Thanks are also extended to my wife, Jamie, for her patience during this period and to Linda Giem for typing the manuscript.

TABLE OF CONTENTS

Page

ABSTRACT. ii

ACKNOWLEDGEMENT iii

LIST OF ILLUSTRATIONS v

LIST OF TABLES. vi

I. INTRODUCTION. 1

II. A REVIEW OF COMPUTER GRAPHICS 4

A. Refresh Display Terminals. 4

B. Direct View Storage Tube Terminals 8

C. Graphic Input Devices. 10

D. Implementation of Two Systems. 15

III. HARDWARE INTERFACE. 17

A. Description of the Hardware System 17

B. Instruction Set for Graphics System. 32

IV. PROGRAM CIRCUITS. 38

A. Objectives of the Program. 38

B. Description of CIRCUITS. 40

V. AN EXAMPLE PROBLEM. 52

BIBLIOGRAPHY. 58

VITA. 60

APPENDIX A - FORMAL DESCRIPTION OF THE ARDS-SCC 650 INTERFACE . . 61

APPENDIX B - EXAMPLE INPUT/OUTPUT ROUTINES. 69

LIST OF ILLUSTRATIONS

Figures Page

1. Typical Random Scan Graphics Terminal 7
2. Direct View Storage Tube Terminal 9
3. Typical Characters Generated by Dot Matrix Techniques . . . 14
4. ARDS Graphical Data Structure 20
5. The Sequence 011010 Coded in NRZ and RZ Codes 23
6. Two Characters in Asynchronous Communication. 25
7. Block Diagram of Graphics System. 26
8. Block Diagram of Interface Control. 27
9. Block Diagram of Program Circuits 43
10. CIRCUITS Data Structure 47
11. Dictionary for CIRCUITS 50
12. An Example Circuit. 53

LIST OF TABLES

Table		Page
I	Properties of Different Graphics Terminals	5
II	Status Word	31
III	Graphic Interface Card Assignment	33
IV	Standard Elements for CIRCUITS	41
V	List of Function for CIRCUITS	46
VI	Note Table for Circuit Shown in Figure 12	56
VII	Element Table for Circuit Shown in Figure 12	57

I. INTRODUCTION

The main objective of this research project was to design an integrated hardware/software system for the support of interactive graphics[1]. The hardware system incorporated a small computer and direct view storage CRT graphics terminal and associated interface electronics. The computer used was a Scientific Control Corporation SCC-650, with 4,096 12-bit words of memory[2].

The graphics terminal is a Computer Displays, Incorporated, ARDS 100A[3]. This terminal uses a bistable storage tube for the display of alphanumeric and graphical data. The screen size is 8-1/4" x 6-3/8" with the long axis oriented vertically.

The interface electronics acts as a series/parallel converter for reformating the data transferred between the computer and graphics terminal. Parity checkers are incorporated to indicate any bit errors and several computer controllable functions are implemented in the interface. A function keyboard was designed and interfaced to the computer for use with the graphics terminal.

The total hardware configuration was designed to allow the small computer to act as a stand alone system or as a satellite processor to a more powerful computer such as an IBM System 360/50. The stand alone system could then be used for small scale problems, or the terminal could emulate a console typewriter. The minicomputer can also communicate to a central processor over voice grade telephone lines. This allows the larger program and data base of the large system to be concentrated on the solution of a problem while the satellite processor handles data transmission and certain local operations.

The software for this project was centered in the area of computer aided circuit analysis. Several programs exist for analyzing electronic circuits, however, most of these are designed to run in a batch mode environment. It was decided to write a preprocessor for an analysis program, such as ECAP, that would allow the user to draw a circuit using conventional circuit elements and then call for an analysis. This preprocessor program would be implemented on the minicomputer. This has the advantage of allowing the user to draw the circuit and make all necessary changes before the large processor enters into the problem.

The preprocessor program, CIRCUITS, has two parts 1) circuit construction, and 2) circuit analysis. The second of these, circuit analysis, was not attempted at this time, but is nearing completion under a separate project. A synopsis of the objectives of this part will be presented with the description of CIRCUITS.

CIRCUITS was written with three primary objectives. The first of these is that the user input to the program be in a free form and lend itself to the action of a person drawing a circuit. This was accomplished by using the graphic input device to show the position of an element and the function keyboard to define the element type. Problems of pattern recognition of user drawn elements are thus avoided.

The second objective is that the circuit can be changed at will. The action of the user to specify a change should be no more than that required to draw an element. Since the output media is a storage tube, this necessitates the requirement that the program be able to redraw the circuit at any time with no additional input.

The last objective is that the completed circuit should be stored, at the users request, on some permanent storage medium. A circuit that has been stored should be able to be reconstructed at a later time. This was accomplished by placing a description of the circuit on punched paper tape. The program can then read this paper tape and construct the original circuit.

II. A REVIEW OF COMPUTER GRAPHICS

A. Refresh Display Terminals

Graphics terminals can be divided into two classes depending on the method of presenting the display; these are the refresh and storage tube terminals[4]. The refresh terminal constantly presents a new picture on the screen of the CRT. As its name implies, the storage terminal presents the data once and relies on the ability of the storage tube to maintain the display. Refresh terminals can be further subdivided into two types 1) sequential scan and 2) random scan. Table I lists some of the attributes of all three of these types.

The sequential or raster scan terminal is the least expensive output device since the full benefit of television technology can be utilized. Its use for computer graphics is limited almost entirely to alphanumeric and simple graphical displays. The display must first be converted to a television raster scan. This space to time translation is both time consuming and expensive. One method of accomplishing this is to place the data on a small storage tube and then use a flying spot scanner to convert the display. Sequential scan terminals are only practical when the terminal is located near the computer since this system requires very high video bandwidths. The quality of the sequential scan is probably the lowest of the three types of terminals. Linearity is no better than fifteen percent and resolution and information density are equally poor. However, the display itself can be a reasonable size and has excellent brightness. These displays are flicker free since the frame rate is fixed.

TABLE I

PROPERTIES OF DIFFERENT GRAPHICS TERMINALS

PARAMETER	RANDOM SCAN	SEQUENTIAL SCAN	DIRECT VIEW STORAGE TUBE
Refresh memory	High speed random	High speed bulk	None
Driver circuitry	Very high speed high accuracy	High speed Low accuracy	Low speed High accuracy
Signal Bandwith	High	Very high	Low
Source memory Requirement	Large	Very large	Low
Information density	Moderate	Moderate	Very high
Resolution	Very good	Poor	Good
Brightness	Good	Good	Low
Size	Up to 18 in. sq.	Up to 18 in. sq.	6-1/4 x 8-1/2 in.
Interactive devices	Any	Cursor or light pen	Cursor
Response	Immediate	Intermediate*	Several seconds

*Depends on processor speed.

The random scan CRT terminal is the one used most commonly in computer graphics systems. The scanning method differs from the sequential scan in that the beam is moved only over the area where the pattern is to appear. With random scan systems, the advantages of brightness and linearity are offset by flicker. Flicker results from the varying time required to complete a display. This variation is a function of the amount of information being presented at the face of the terminal. High information densities are possible with a random scan terminal by using hardware vector and character generators. This will also reduce the size of the memory needed to support the terminal. Figure 1 is a block diagram of a typical random scan terminal.

The second advantage of the random scan terminal is its ability to convey real time motion to the observer. This ability is lost in the raster scan and storage tube terminals because of the time spent in reformating the display. The best that can be accomplished is a series of "still pictures" at fixed time intervals.

Both types of refresh terminals require extensive supporting hardware. Some form of memory must accompany the terminal to store the display data. Currently, magnetic cores and discs, delay lines, and flying spot scanners are used as memory elements. This memory and associated control circuitry adds to the complexity and cost of the terminal. Most terminals in this class sell for $50,000 and up[5]. Highly interactive graphic communications which require either rapid response or flexible graphic input are the most likely applications for the random scan refresh system.

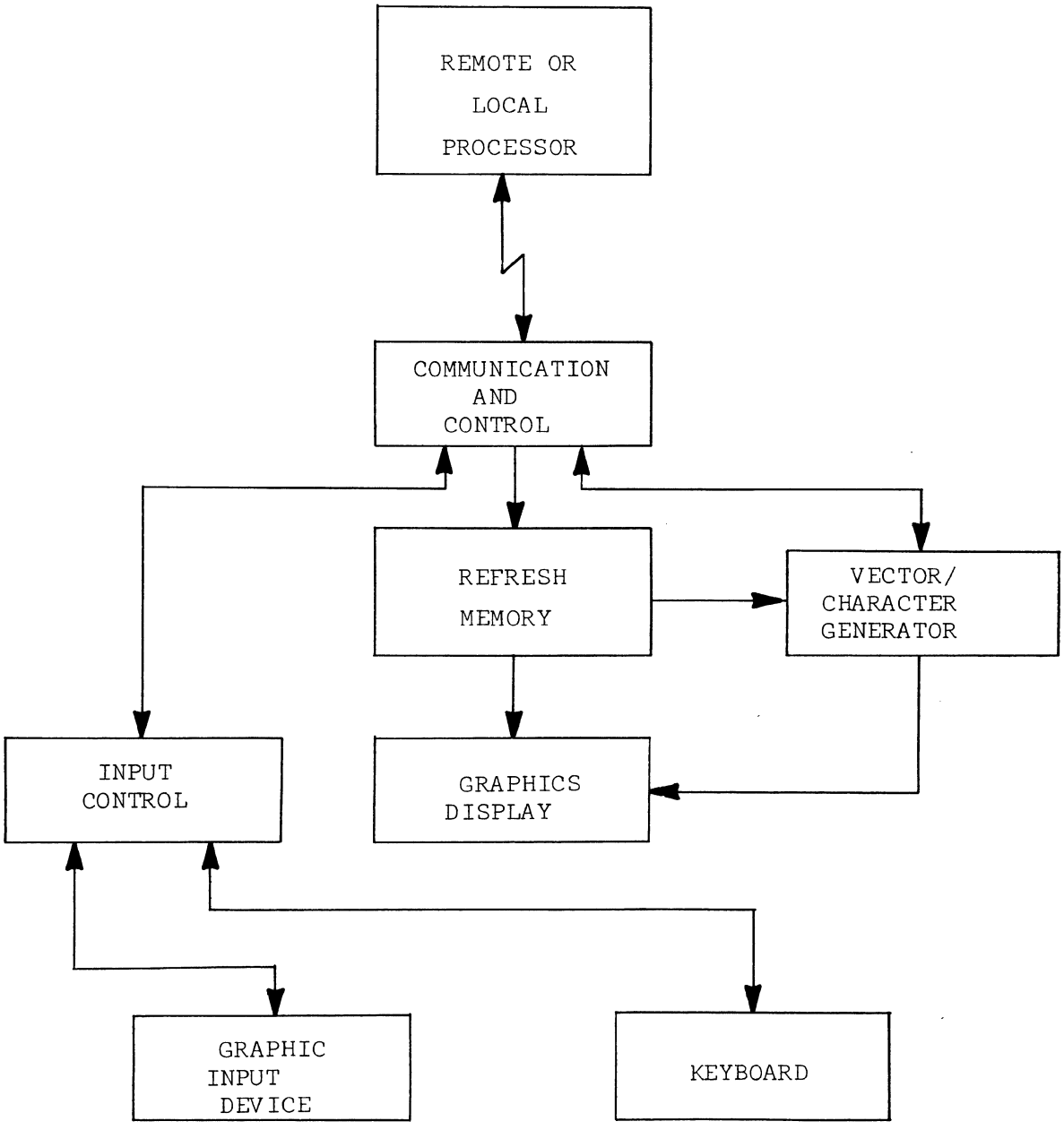


Figure 1. Typical Random Scan Graphics Terminal.

B. Direct View Storage Tube Terminals

The direct view storage tube has the ability to store a visual image as it is written. This eliminates the need for an associated refresh memory and also greatly reduces the necessary information rate to the terminal. These two facts are responsible for the inherent low cost of the Direct View Storage Tube (D.V.S.T.) display system. A graphics terminal utilizing a bistable storage tube can be purchased for under $10,000[4]. This drastic reduction in price has made computer graphics attractive in many new and diversified areas.

Figure 2 presents one form of a computer graphics system built around a D.V.S.T terminal. In this system, a small general purpose computer operates on line with the terminal. All control functions originate in the computer under command of the input devices.

The D.V.S.T. is driven from an analog function generator controlled by the computer. The use of these function generators is the key to low system cost. The second key to low cost is the heavy use of software, as opposed to hardware, to do nearly all data formating and data control functions. It is the relatively slow speed requirements of the D.V.S.T. which allows the use of software in this manner.

All of the D.V.S.T. systems have similar performance characteristics except for response speed. The display itself does not flicker, but contrast and brightness are less than that achievable with refresh systems, although adequate. No existing refresh system can match the information density of the D.V.S.T., and it is in this area the terminal is superior.

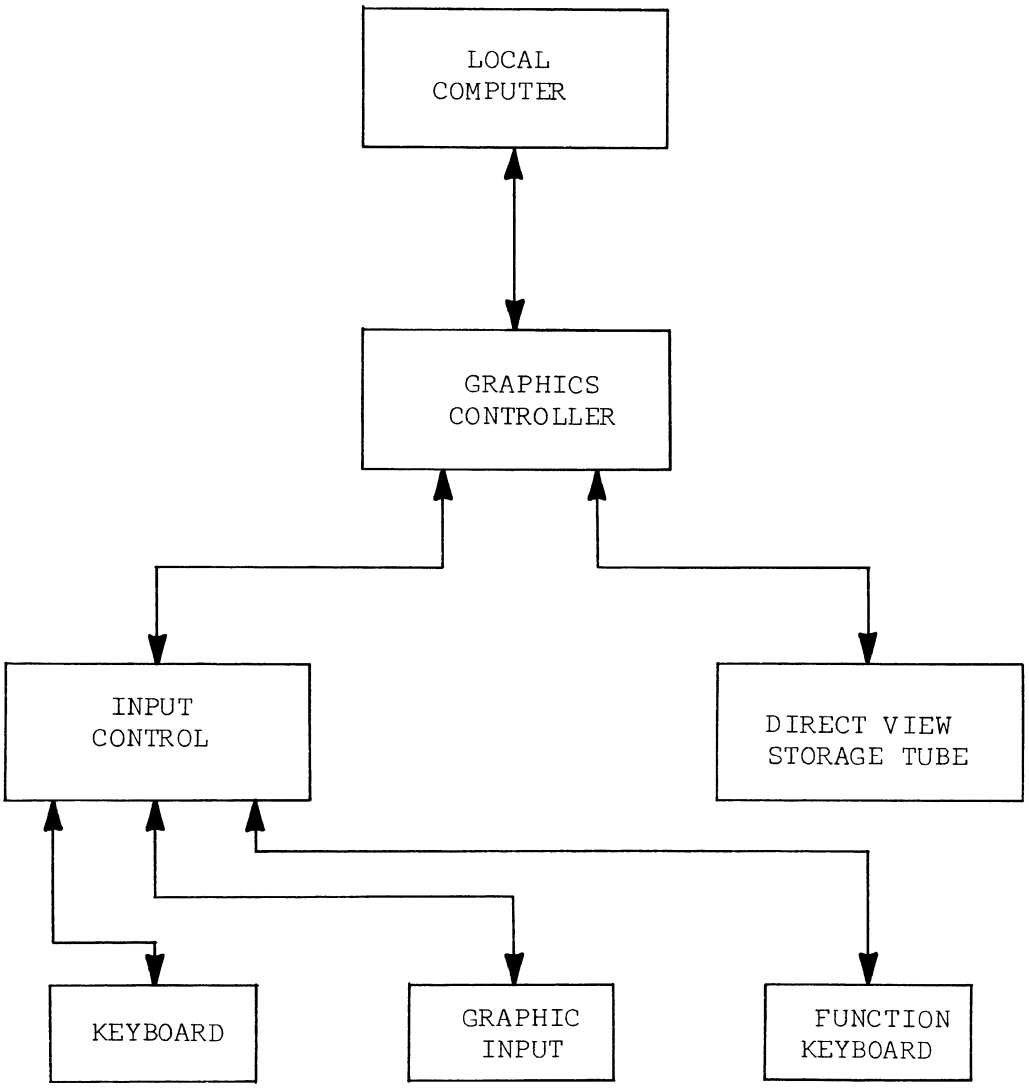


Figure 2. Direct View Storage Tube Terminal.

Certain applications do not lend themselves to the use of a D.V.S.T. terminal. These include real time systems or where fast response is required. Also the D.V.S.T. terminal lacks the ability of selective erase; any changes in the display means that the entire frame must be rewritten.

C. Graphic Input Devices

In order to make full use of a graphics terminal there must exist some means of inputting graphical information. There are currently several techniques for accomplishing this purpose. Some of these techniques are applicable only to a certain type of terminal while others are general enough to be used with any terminal. The various input schemes that will be considered are: 1) the light pen, 2) cursor, and 3) tablet.

The light pen is a photosensitive detector housed in a case resembling a large fountain pen. It is used almost exclusively with refresh CRT terminals. When the computer refreshes the area the light pen is pointing to, the pen senses the flash as the phosphor glow is renewed. If the vertical and horizontal deflection voltages are digitized at this time, the address of the light pen can then be sent to the computer. The user may draw on the CRT face by using the light pen. This is usually accomplished by indicating by some means that a drawing will be started and then placing the light pen against the screen. Once the computer has sensed that the light pen is positioned, it will track it across the face of the CRT intensifying points along the path.

Another type of graphic input device employs a cursor to show the position of the electron beam. This is used primarily with storage tubes as a substitute for a light pen. The cursor is usually a small spot of light with the intensity set low enough so that it will not store on the CRT yet bright enough to be visible. One manufacturer uses a set of full screen crosshairs to define a point. In either case, the cursor is designed to follow a mechanical device as it is moved through a 2-space. The usual devices employed are a joystick or mouse.

The joystick is a small box with a protruding handle. As it is moved by the user in a lateral and/or longitudinal direction, it transmits to the graphics terminal a pair of analog voltages equal to the x and y displacements from the center line. These voltages then form the deflection voltage for the cursor. The mouse is a device that fits in the palm of the hand with two orthogonal wheels on the bottom. As the mouse is moved across a hard surface, it causes the cursor to follow.

In addition to the two analog inputs, these two devices must provide control signals to the terminal. There are generally at least two such signals required, point and line. Once the user has the cursor at a required position on the screen he must press a button which will digitize the output of the graphic input device and send the address of the point to the computer. Once two such points have been defined, another button can be depressed which signals the terminal and computer to draw a line between the points. Spotting,

or pointing to a position on the screen with this type of graphic input device is relatively simple, but drawing any complicated figure is extremely difficult.

Tablets such as the RAND or sonic pen tablet[6] are another form of graphic input device. These can be used with any type graphics terminal. There are different principles involved in the construction of these tablets, but they all achieve the same result. A pen-like device is tracked as it moves across a plane by either electrostatic or electromagnetic sensors, or timing the propagation of a sonic wavefront. The position of the pen is then sent to the computer which plots its position on the graphics terminal. Accuracies on the order of 1 mm on a twelve inch square surface can be obtained. Tablets are useful for tracing a curve or the outline of an object that is to be displayed on a graphics terminal. Since the action of the pen on the tablet closely resembles the natural act of writing, these devices are very effective.

Most graphics terminals operate in two modes 1) text mode and 2) graphics mode. In the text mode, alphanumeric characters are displayed on the screen of the terminal. The action will resemble that of a typewriter. In graphics mode, the beam can be either positioned to a point on the screen, or displaced from a point causing a line to appear.

In the text mode, the alphanumeric characters are usually generated within the terminal. This decreases the amount of data the computer must transmit, but increases the complexity of the terminal. One popular method of generating characters involves using a seven by

nine or five by seven dot matrix. The characters are created by intensifying different combinations of dots. Figure 3 shows two characters generated by this method. Other methods of character generation include shaping the beam into the character shape by shadow mask techniques, or generating the character by using analog methods to produce the different strokes. The dot matrix is becoming more popular with the decreased cost of read only memories. All of the data necessary to draw the characters can be stored in a relatively small ROM. This has an additional advantage since different fonts can be used by simply selecting one of several ROM's.

Graphic modes involve at least two submodes 1) positioning, and 2) vector generation. Positioning the beam requires placing the terminal in a POSITION mode and then sending the new address to the terminal. The address may be either absolute or relative. Absolute addresses are with respect to some fixed origin on the graphics screen. Relative address are usually with respect to the current beam position.

Vectors are usually generated by placing the terminal in a VECTOR mode and then transmitting the data necessary for the vector. There are two methods of specifying vectors as there are with positions. Some terminals draw all vectors relative to the current position, in which case the data is actually an x and y displacement. In an absolute vector mode, the beam is left in an intensified state and moved to an absolute address specified by the data.

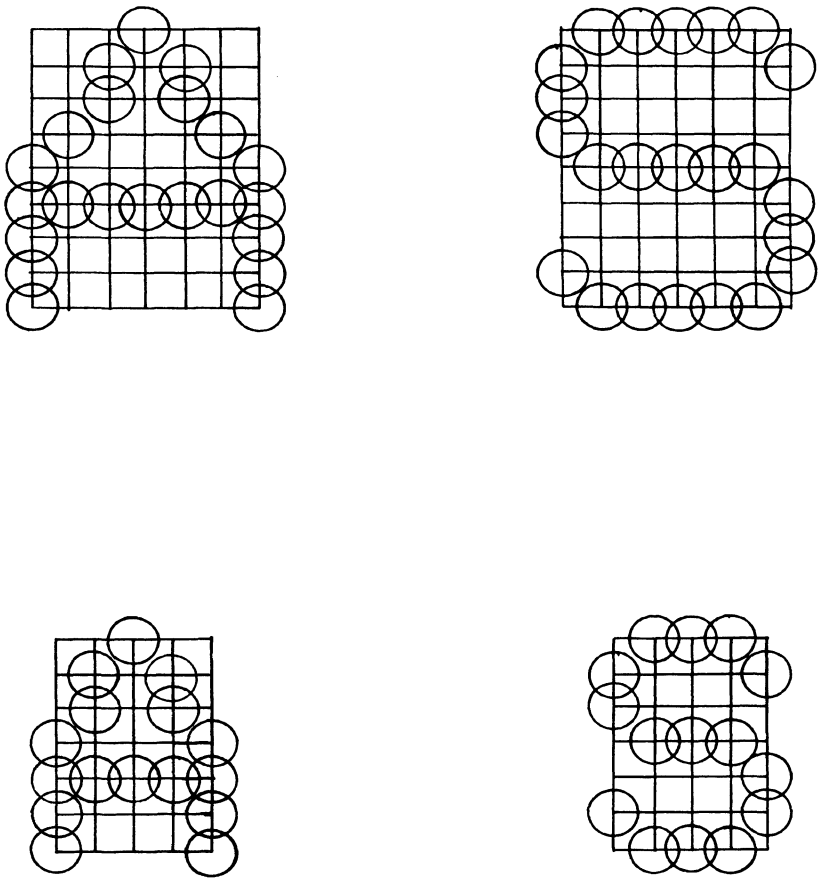


Figure 3. Typical Characters Generated by Dot Matrix Techniques.

D. Implementation of Two Systems

One approach to using a D.V.S.T. display as a remote graphics terminal is described by Pardee[7]. This approach utilizes a PDP 8/L minicomputer and a D.V.S.T. as a remote terminal. The D.V.S.T. display is simply the CRT and associated deflection circuits. The computer performs all the functions of a graphics controller. Software techniques are used to generate all characters and vectors. Two 10-bit digital to analog converters with a Z-axis control form the interface to the CRT display.

Since the control unit for the terminal is a programmable computer, several functions are available with this terminal that are not usually found on a low cost system. These functions include, among others, rotation of bodies and characters, and graphical subroutining[8].

Rotation of bodies is possible since the data for the display is stored in the memory of the minicomputer. All characters are generated as a dot matrix by a software routine so that it is a simple matter to rotate the characters.

Graphical subroutining allows the central processor to transmit data for one or more subpictures and then position the subpicture as many times as needed and whenever needed to complete the display. This technique increases the information rate between the central processor and remote terminal. Updating a display could then be as simple as redefining a basic subpicture.

The major drawback of this system is that it is not an interactive terminal. The user can not input alphanumeric or graphical information to the display unit.

An interactive graphic program is presented by Thornhill[9]. This program, named GRAPHSYS, allows a user to construct an electrical circuit composed of resistors, capacitors, wires, and nodes on a direct view storage tube terminal. The user can modify the circuit by adding or deleting elements and nodes. Once the circuit is properly constructed it can be named and used as a "black box" in constructing other circuits.

This program is designed primarily as a teaching vehicle and is not very useful as a circuit design tool. Many of the techniques and ideas it incorporates are applicable to a large design program and some were incorporated in CIRCUITS.

Graphics terminals, utilizing one or more graphic input devices, have proved themselves to be a very effective man machine interface. The graphical display provides the most flexiable and adaptable means of giving information to, and receiving information from, a computer. It has not yet fulfilled its potential because it probably is the most complicated single peripheral to be interfaced to a computer. The continuing research on hardware and software systems and their applications inspire an stmosphere of confident anticipation for the future of computer graphics. It appears to be generally accepted that the present trend toward decreasing terminal cost and increased flexability will be maintained and probably accelerated.

III. HARDWARE INTERFACE

A. Description of the Hardware System

The computer graphics system located at the University of Missouri-Rolla, Department of Electrical Engineering, incorporates a Scientific Control Corporation SCC-650 digital computer and a Computer Displays, Inc. ARDS 100A graphics terminal. The SCC-650, considered to be in the minicomputer class, is equipped with 4,096 12-bit words of memory and has a basic cycle time of 2 microseconds. All arithmetic and logical operations, together with high speed data transfers between memory and the various registers, are fully parallel. The central processor is equipped with a single accumulator and one hardware index register. All input/output data must be directed through the accumulator. One interrupt channel is provided at the I/O buss. Using this facility, the computer can be interrupted and control transferred to the appropriate subroutine. Four addressing modes provide increased flexibility in obtaining data from the memory.

In addition to the graphics terminal,an ASR-35 teletype equipped with a paper tape reader and punch is available for input and output. A Kennedy model 1400R digital tape system is on-line for higher speed data transfers.

The ARDS graphics terminal is logically and physically composed of three units, the keyboard, the controller, and the display unit. In normal operation the keyboard provides alphanumeric information to the computer and the ARDS. It contains keys for generating all

128 possible codes of ASCII. Patterned after a normal typewriter, it contains a pair of shift keys and a shift lock. Depressing a key causes the character to be echoed on the screen of the display unit.

The controller contains all of the digital logic of the system. This includes the logic for code conversion and the analog signal generators, and vector generators which permit maximum utilization of a low speed data link. The controller also contains the logic necessary for formating and transmitting the data from the keyboard and graphic input device.

The display unit contains a direct view storage CRT. The surface of the display acts as the memory for the graphics terminal and the viewing screen. Once data is written on the surface of the CRT it remains visible for up to fifteen minutes without serious degradation. The viewing area measures 8-1/4" x 6-3/8" with the long axis oriented vertical in the manner of a page of text. With the excellent resolution of a D.V.S.T., over 4,000 easily legible symbols can be plotted. For graphics applications the screen is defined to contain 1081 x 1415 addressable points. The point 0, 0 is the origin and is located in the center of the screen. The display unit can draw vectors at the rate of one half inch per millisecond and characters can be plotted in one or two milliseconds.

The data format of the ARDS is compatible with ASCII; the symbol set contains the ninty six printable ASCII symbols. To accommodate graphic input and output, the ARDS operates under mode control. The ARDS is placed into the symbol mode by any of the ASCII control characters (bit 6= bit 7 = zero) with the exception of the three that are reserved for graphic mode.

Since ASCII does not have any provision for transmitting graphic information, an extension of this code has been adopted. This extension provides for a number of graphic commands each of which interpret characters as binary information. Binary characters are those with bit 7 = One, thus the problems of the ARDS interpreting data as graphic commands are eliminated. This leaves six bits per character available for information.

The graphic modes the ARDS responds to are set point, long vector, and short vector. Set point mode positions the beam to any absolute address on the screen. This mode is entered by control character GS (octal 35). While in this submode, the next four characters are interpreted as an absolute binary address.

The long vector mode will draw a relative vector any length up to 1023 increments long in both x and y. The vector may be visible or invisible, solid or dotted. Long vector is entered by control character RS (octal 36). While in this submode, the next four characters are interpreted as an x and y displacement from the current point.

The short vector mode will draw a relative vector any length up to 31 increments in x and y. A short vector is always solid and visible. The short vector submode is entered by the control character US (octal 37). While in this submode, the ARDS will interpret the next two characters as an x and y displacement from the current point. This mode reduces the amount of data transmitted since detailed drawing will have a large number of short solid lines. Figure 4 summarizes the different graphic modes.

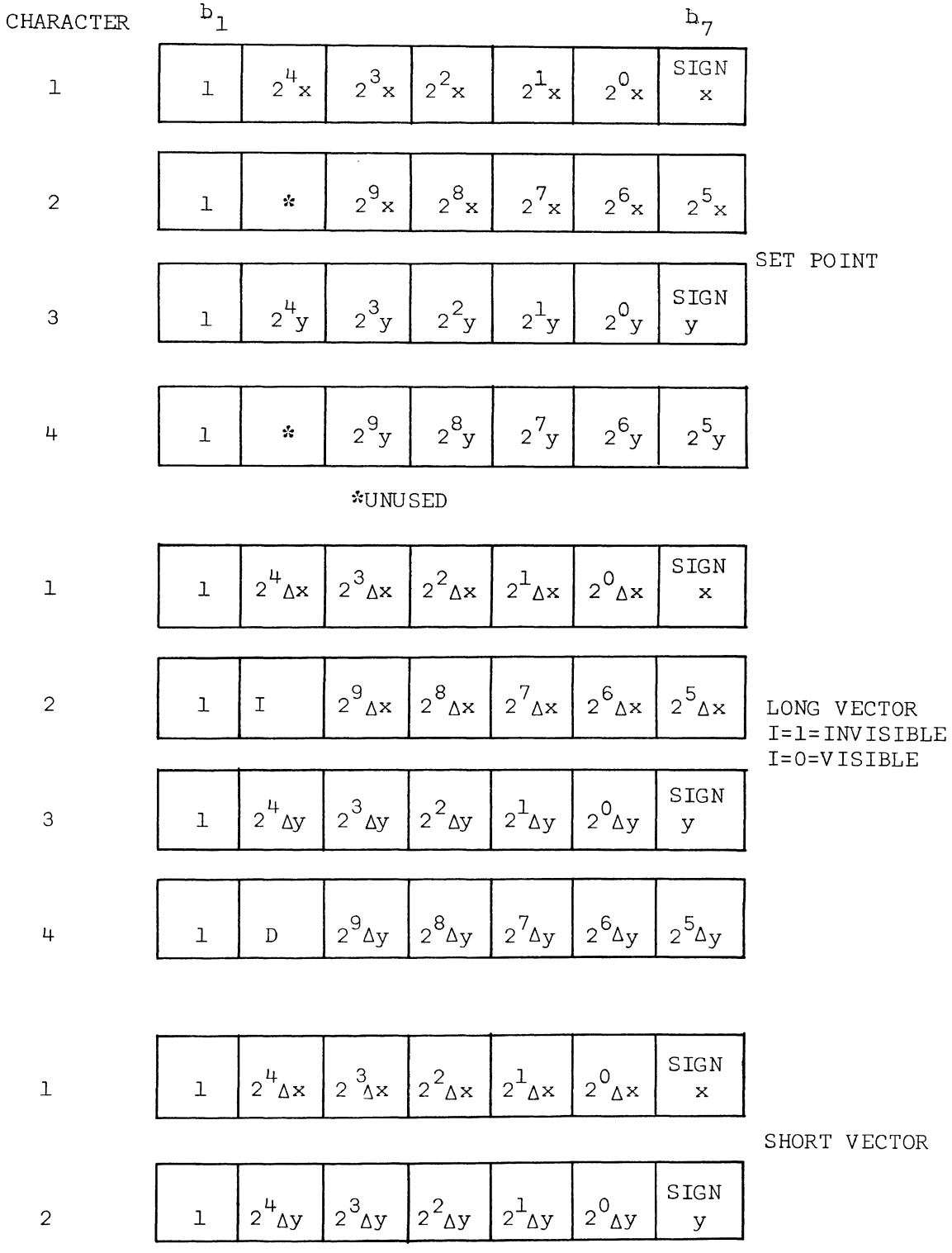


Figure 4. ARDS Graphical Data Structure.

The ARDS is equipped with a "mouse" as a graphic input device. Three switches located on the mouse enable the user to send graphic data to the ARDS and computer. As the mouse is moved across a hard surface, it will cause the cursor to follow in the same direction. Depressing the POINT (center) switch causes the ARDS to define the cursor position as the current point. This information is sent to the computer by transmitting the SET POINT character (GS) and four characters of address data. Depressing the LINE (left) switch, causes a solid visible line to be drawn on the graphics screen between the current cursor position and the last defined SET POINT. This information is transmitted to the computer by the LONG VECTOR (RS) character and four characters of displacement. The end of the line is then defined as the current point in the graphics terminal. The third switch is used for special applications.

Other options on the graphics terminal control echo mode and the keyboard bell. The echo mode allows data to be transmitted from the ARDS graphic input device and keyboard without having this data displayed on the screen. If ECHO is on, the ARDS will print on the screen all data that is input from the keyboard and will respond to all graphics data from the mouse. If ECHO is off, the display unit will not respond to any data generated by the ARDS. However, all data is transmitted to the computer and the ARDS will respond to all data received from the computer. The bell can be activated by the character BEL (octal 7) and is used to signal the operator of some event under control of the program.

Electrically the ARDS is designed to communicate with a device that meets EIA RS-232 standards. Several control signals are provided or recognized by the ARDS controller so that it may operate in a full or half duplex mode of operation. Since the control necessary for full duplex operation is somewhat simpler, this mode was chosen as the operating mode for the graphics terminal.

The ARDS - 650 interface[10] directs the flow of information and control between the graphics terminal and the CPU. The interface accepts signals from both the ARDS and the computer to control both the type and direction of data flow. All data that is transferred between the interface and CPU is in bit-parallel form, that is, all 12 bits are transferred simultaneously. Data transferred between the ARDS and the interface is in bit-serial, NRZ, asynchronous form. Each character, 7-bits plus parity, is preceded by a start bit and followed by a stop bit.

Under normal conditions, the data line is held in the MARK or "1" state when there is no activity on the line. The start bit is always a SPACE or "0" and last for one bit time. The data is formated in a NRZ, Non-Return to Zero, code. The difference between a non-return to zero and a return to zero code is best illustrated by an example. Figure 5 shows the sequence "011010" coded in both a NRZ and RZ code. As the illustration shows, the signal stays in the appropriate mark or space state for the entire bit time in NRZ coding. In an RZ coding scheme, the bit is always in the SPACE or "0" state and changes to the "1" state only briefly at each appropriate bit time.

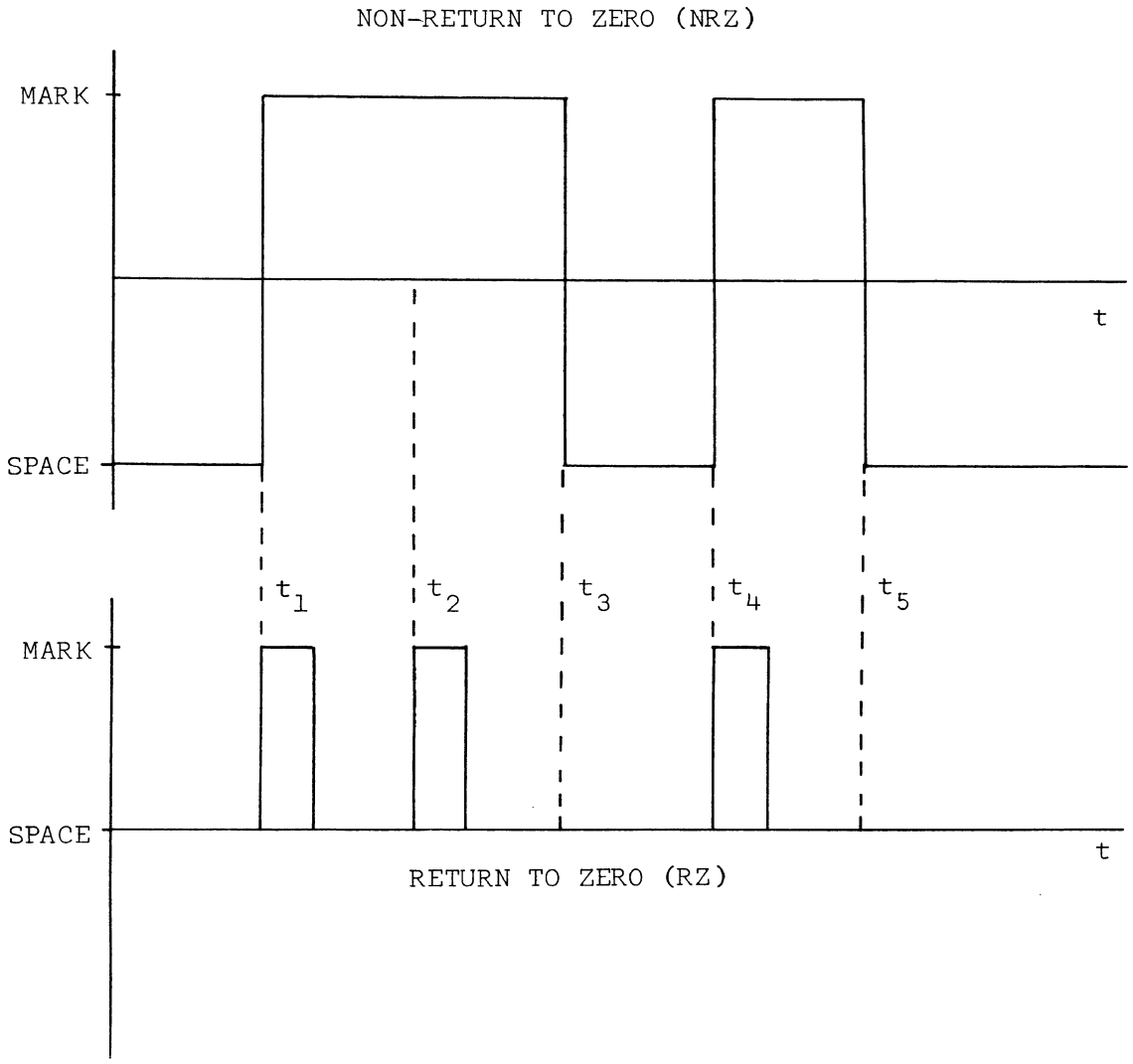


Figure 5. The Sequence 011010 Coded in NRZ and RZ Codes.

The stop bit which follows each character is always a MARK and must last at least one bit time. In the ARDS, and its interface, the stop bit is 2 bit-times in duration.

The asynchronous communication method is the simpliest method of bit serial data transmission. Since the data line is always left in a known state (MARK) and the first bit (the start bit) is the other state, then the change of state can be used to signify that data is present on the line. Since the bit time and number of bits are always known, the detection of data is relatively straight forward. Figure 6 shows the characters "a" and "U" as they would appear in start-stop format.

The graphics interface is composed of four main sections:

1) the parallel/serial data converter,
2) control section,
3) command decoders, and
4) data buffers.

The interface accepts data from the graphics terminal, function keyboard, and computer, and directs it along the proper data path, and in the proper format. Figure 7 is a block diagram of the graphics system and Figure 8 is a block diagram of the interface controller.

The parallel to serial converter used in the interface is a 10-bit shift register. Outgoing data is placed into bits two through eight in a parallel transfer and the data is sent to the ARDS by shifting to the right one bit each bit time. Shift register bits one and ten are the start and stop bits respectively, and bit nine is the parity bit. Data coming from the ARDS is shifted into the

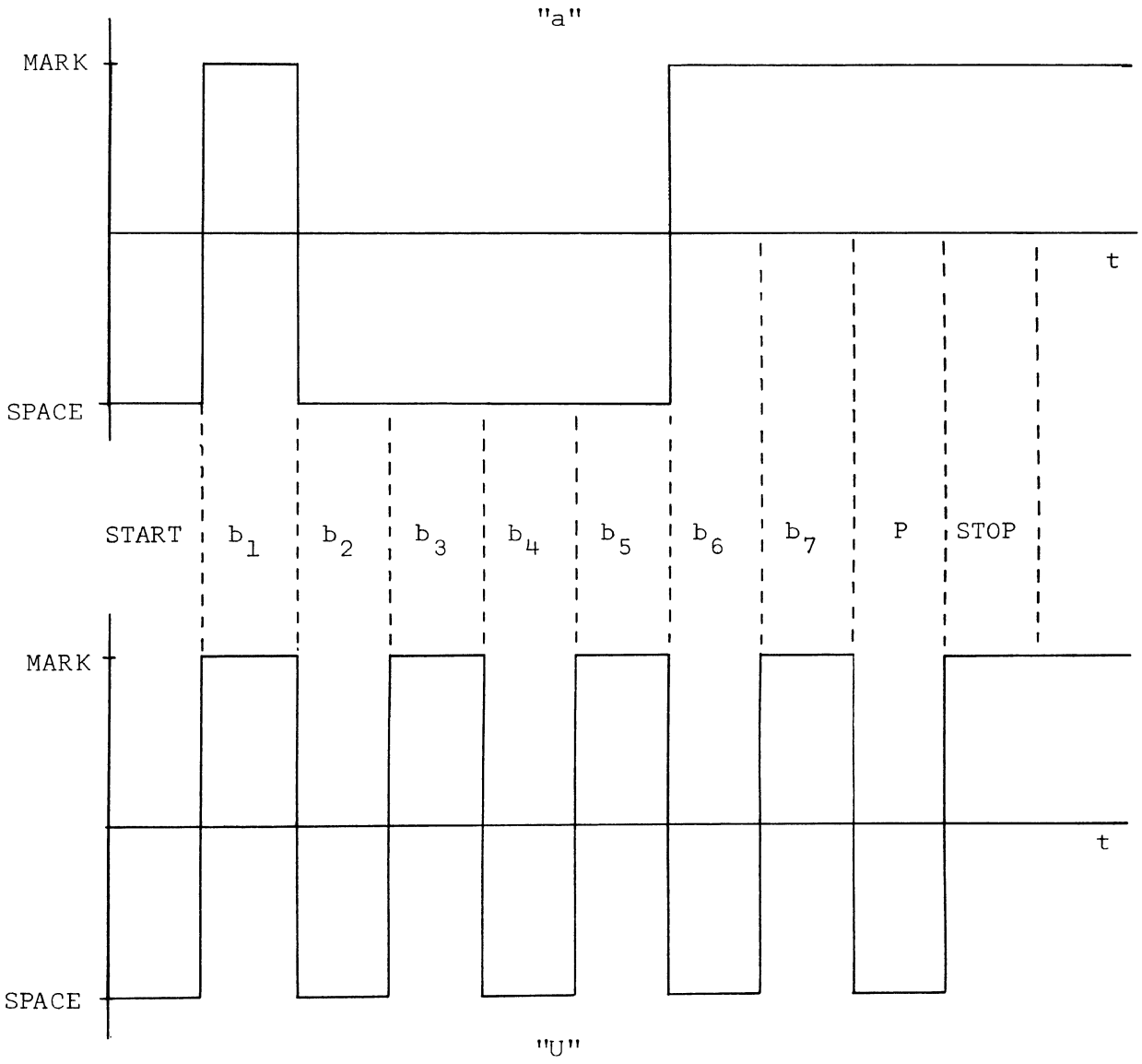


Figure 6. Two Characters in Asynchronous Communication.

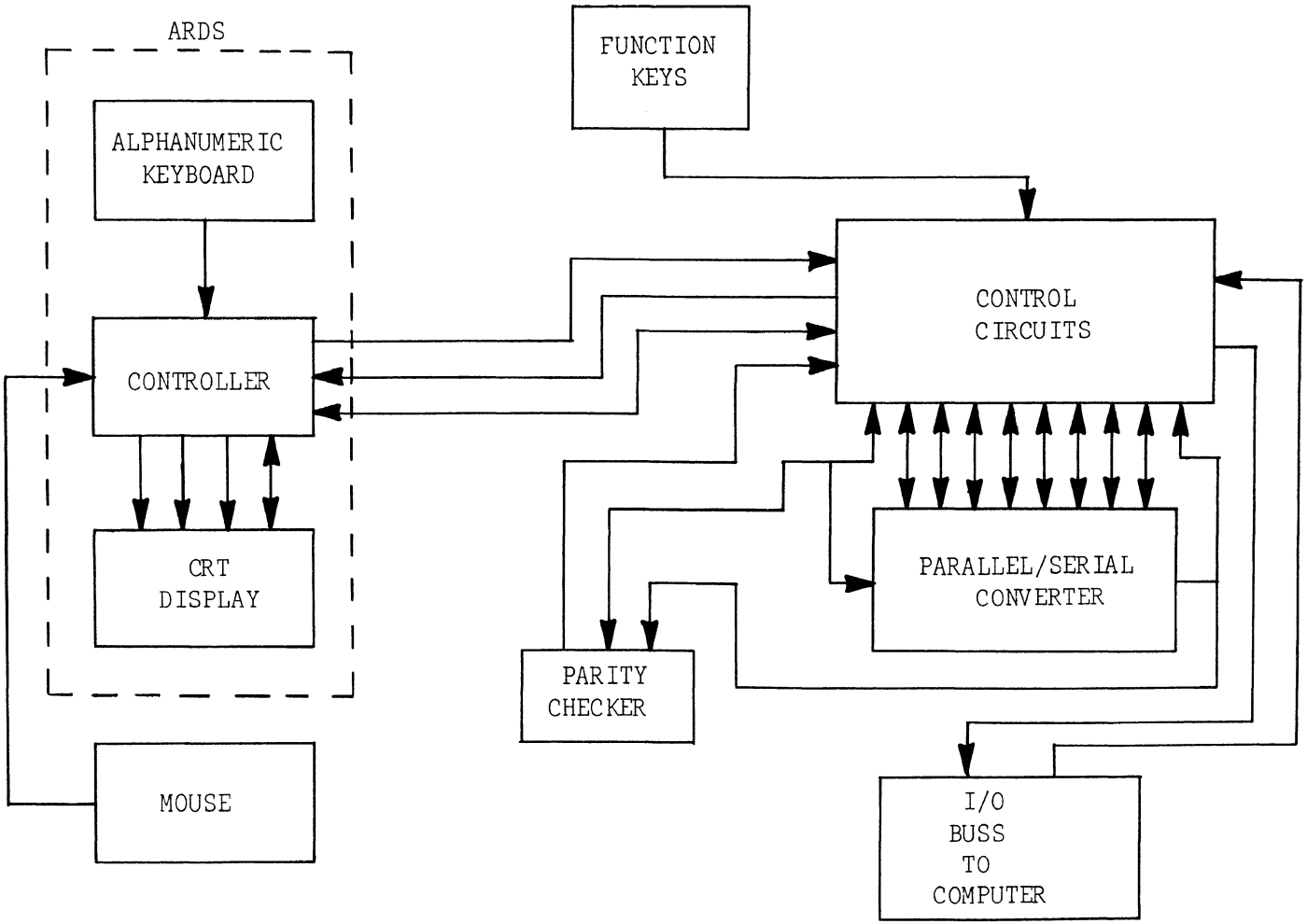


Figure 7. Block Diagram of Graphics System.

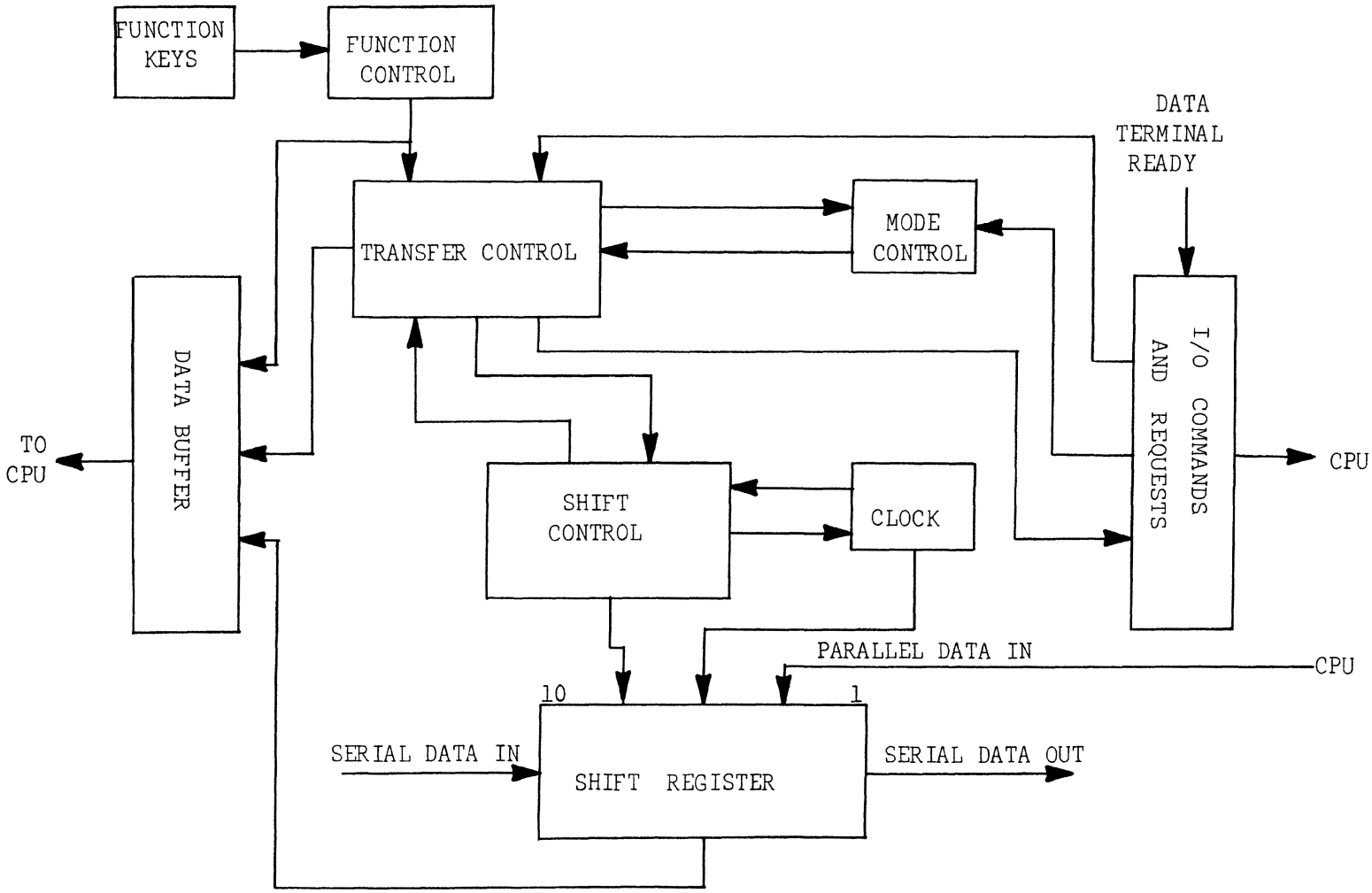


Figure 8. Block Diagram of Interface Control.

shift register one bit at a time and then bits two through eight form the parallel output. All data enters the shift register under the direction of the SHIFT CONTROL circuitry.

In the transmit mode, this section accepts the parallel data from the computer, clears the shift register, and then loads the data. The clock is then started and twelve clock pulses are counted. AFter the twelfth shift, the data is out of the shift register, the clock is stopped, and the control section is notified that the transfer is complete.

In the receive mode, the start bit from the ARDS is sensed (the line changes from MARK to SPACE) and the clock is started. At each clock pulse, a new bit is shifted into the shift register. After ten clock pulses the clock is stopped, the data is ready to be transferred to the computer, and the CONTROL SECTION is advised that data is ready. Once the computer requests the data, the CONTROL SECTION transters it and the SHIFT CONTROL circuitry resets itself for the next transfer.

A parity checker is used to ensure that the transmitted serial data is always even parity and that the received serial data is correct parity. As the data is transmitted to the graphics terminal, the number of ONE's in the data are counted and the parity bit is set to make the sum of the bits that are ONE even. As data is received from the ARDS, the parity of the data is computed and if it is not even, the CONTROL SECTION notifies the computer that an error exists in the character.

The CONTROL SECTION is divided into two parts 1) mode control, and 2) transfer control. MODE CONTROL sets the interface into either

a transmit or receive mode. (All data directions are with respect to the computer, e.g., transmitted data is from the computer to the ARDS.) The interface is normally in the receive mode. It is placed into the transmit mode only when the computer requests that data be transmitted to the ARDS, and stays in this mode until the transfer is complete.

The transfer control circuitry determines when data is transferred to and from the computer. In addition, it determines which data word is sent to the computer. It accepts inputs from all other sections of the interface and produces essentially three outputs to the computer. The first output (IDRDY) informs the CPU that the requested action is complete. The second output is an interrupt signal (EXTINT) which informs the CPU that some action is required. The last output signal (IOE) informs the computer that a character from the graphics terminal is in error. Several other signals are also generated for internal control of the interface. These provide timing information for functions such as clearing the shift register, and starting the clock.

The COMMAND DECODER section interprets all of the commands that apply to the graphics interface. The input for this circuitry originates at the input/output buss of the computer and the graphics terminal. The graphics terminal provides a signal (DTR) whenever the terminal is connected to the interface and is ready to operate. The computer provides several I/O instructions which have been decoded in the computer, and the instruction register so that the interface may decode the additional I/O instructions. In addition, all of the

CPU timing signals are provided to the interface so that it may synchronize itself with the computer. All of the inputs to this circuitry are buffered so that any anomaly in the graphics interface will not affect another periphral device. In a similar manner, all of the control outputs to both the graphics terminal and computer are brought to this section for buffering.

The DATA BUFFER accepts three parallel words as input and gates one of these, determined by the TRANSFER CONTROL circuitry, to the computer. The input words are 1) data from the shift register, 2) data from the function keyboard, and 3) the Status word. The data from the shift register represents the character received from the ARDS. The function word is the input from the function keyboard, and the status word is a collection of the state of various control signals within the interface. The status word can be input to the computer so that the precise state of the interface is known. Table II describes the various signals in the status word.

The function keyboard that was added to the graphics facility consists of keyboard and control circuitry. The keyboard is a set of switches located on a console near the graphics user. It consists of eleven function switches, a CLEAR switch and a TRANSFER switch. The function switches are numbered 1-11 and correspond to accumulator bits 11-1 respectively. There are 2,048 different combinations of the function switches and these are entered by concatenating different switches. Function 23, for example, would be entered by depressing switches two and three. The control circuitry consists of

TABLE II

STATUS WORD

INPUT BIT	MNEMONIC	STATUS
0	Not used	Not used
1	INT	Interrupt is Pending
2	AFCN	Function Keyboard Armed
3	ECHO	Echo Mode is Set
4	SRF	Shift Register is Full
5	SIP	Shift in Progress
6	AXINT	Xmit Interrupt Armed
7	ARINT	Rec Interrupt Armed
8	MODE	1 for Rec, 0 for Xmit
9	DTR	Data Terminal Ready
10	IDL	Interface Idle
11	ACT	Interface Active

the function flip-flops and associated control. Depressing the CLEAR switch on the function keyboard clears the function flip-flops, and initializes the control. Depressing any combination of the function switches causes the corresponding flip-flops to be set. When the transfer switch is depressed, and if the function keyboard is armed, the TRANSFER CONTROL section is notified that data is ready. The function word will then be transferred to the CPU at the next read request.

A detailed formal description of the interface can be found in APPENDIX A.

Physically the graphic interface occupies one card file in the System Interface Cabinet. The electronics was implemented by utilizing digital integrated logic packages. DTL (diode transistor logic) and TTL (transistor transistor logic) NAND logic packages were chosen since both are readily obtainable. The logic for the interface is mounted on 17 printed circuit boards mounted in slots one thru 17 in the card file. Slot number 0 contains level converters for interfacing the IC logic with the ARDS. Slot number 18 contains the clock for determining the serial data shift rate. Table III summarizes the card assignment.

B. Instruction Set for Graphics System

The instruction set necessary for efficient communications with the ARDS terminal consists of six instructions. All of these instructions have the form of a mneumonic plus a device code. The device code informs the interface that the instruction concerns

TABLE III

GRAPHIC INTERFACE CARD ASSIGNMENT

SLOT	FUNCTION
0	Level converters
1	Data Buffer (bits 8-11, 0)
2	Data buffer (bits 4-7)
3	Data buffer (bits 1-3)
4	Command decode #1
5	Command decode #2
6	Transfer control #1
7	Transfer control #2
8	Transfer control #3
9	Shift register (bits 5-10)
10	Shift register (bits 1-4)
11	Shift control
12	Clock control
13	Clock decoder
14	Function keys (bits 1-8)
15	Function keys (bits 9-11)
16	Function control
17	Parity generator
18	Clock

the graphics terminal. The graphics interface is assigned device number 11. Since instructions are represented in octal in Assembler notation, the device code will be shown as '13 where the apostrophe designates an octal number.

SELECT. The instruction SEL '13 performs the function of electrically connecting the interface and CPU. This instruction is conditioned on the ARDS being connected to the interface and operative (DTR=1). If this condition is met, the ACTIVE flip-flop in the interface is set. At this point, the interface will respond to any command from the computer.

READ. This instruction is represented as TTA '13. It will transfer data to the accumulator if there is data ready. If the transfer is accomplished, the CPU will execute the next sequential instruction. The CPU will skip the next instruction if the data is not ready. The data that is input by this instruction is either the character in the shift register or the function word. If it is the data from the shift register, the character will be in bits 5-11 of the accumulator with bits 0-4 set equal to zero. If the function word was the data that was input, it will be in accumulator bits 1-11, and bit zero will be a one. Since bit zero is also the sign bit in two-complement notation, a negative number will result when the function word is input and a positive number will result for a character. Testing the input data for a positive or negative sign will be able to differentiate between the two types of data. This provides a simple method for inputting data from two devices using the same instruction.

The WRITE instruction (TFA '13) transfers data from the accumulator to the shift register. Accumulator bits 5-11 will be considered to have the character to be transmitted. This instruction is conditioned on the interface being able to accept the data. If the interface is not ready, the next sequential instruction will be skipped. If the interface is ready for the data (IDL = 1), the data will be transferred and the CPU will execute the next sequential instruction.

The TEST (IOT) instruction is used to check on the validity of a received character. An IOT command following a READ will cause the CPU to skip an instruction if the character just received had correct parity. If the parity was incorrect, the next instruction will be executed. An IOT instruction following a WRITE command is meaningless.

EXECUTE. There are several EXU '13 instructions. These are used to control certain functions of the interface. Upon receipt of an EXECUTE command, the interface looks to the accumulator to determine which function is to be performed. These auxillary functions are:

A10 = A11 = 0	Disarm interrupt system
A11 = 1, A10 = 0	Arm receive interrupt system
A11 = 1, A10 = 1	Arm transmit interrupt system
A7 = 0, A6 = 1	Disarm Function Keyboard
A7 = 1, A6 = 0	Arm Function Keyboard
A8 = 1, A9 = 0	Disarm Echo Mode
A8 = 0, A9 = 1	Arm Echo Mode

The interrupt system allows the computer to read or write data without having to wait for the graphics terminal. With the Receive Interrupt System armed, the interface will interrupt the CPU only when data is ready to be transferred to the computer. If the Transmit Interrupt System is armed, the interface will interrupt the CPU only when the last character has been sent to the ARDS and the interface is idle. With both of these systems armed, the interface will interrupt whenever it requires action by the CPU.

The Function Keyboard can be enabled by the program using the appropriate Execute command. This allows the program to disregard this as an input device.

Echo mode, for the present time, is always enabled when the interface is active. The Echo Off mode is used only for special application such as the later addition of a hard copy unit connected to the graphics system.

TERMINATE. The TMR '13 command performs the reverse action of the SELECT command. This instruction places the interface in a state (ACT = 0) where it will not respond to further commands.

STATUS. The DST '13 instruction causes the interface status word to be read into the accumulator. This instruction is unconditional and is not dependent on the interface being previously selected. As long as the interface is cabled to the computer and the interface power is on, the status word can be read. This is useful for determining the exact state of the interface.

These instructions compose the entire set necessary to communicate with the ARDS. Because of the small number of instructions, the instruction set is simple. The instructions are powerful enough, however, to effect an efficient means of communication. Appendix B presents some sample routines for communicating with the graphics terminal.

IV. PROGRAM CIRCUITS

A. Objectives of the Program

There are currently many large scale circuit analysis programs such as ELECTRONIC CIRCUIT ANALYSIS PROGRAM (ECAP), CIRCUIS and SPECTER[11]. These are designed to run in a batch mode environment. The input is typically a tabular description of the interconnections of the elements and their values. The generated output is typically a tabular description of some selected voltage or current at specified frequency increments. Several of these programs incorporate plotting facilities which are either an on-line plotter, or a print-plot technique. Most of these programs are designed to run on a medium to large computer with a memory of at least 150K bytes.

Several versions of these programs have been designed to run in a semi-interactive environment. Usually this means that the user can access the program from a remote teleprocessing terminal. The input is again a tabular description of the circuit and the output will be the same. If an on-line plotter is utilized, the resulting plot will have to be mailed to the user.

The next step in this chain is to have an analysis program that has graphical input and output. With this goal in mind, CIRCUITS was developed.

CIRCUITS[12] is a program that resides on an SCC-650 computer located with the ARDS graphics terminal. It interactively converts user input specifications into a schematic diagram of an electronic

circuit which is then drawn on the graphics terminal. This computer then acts as a satellite processor to an IBM 360/50 which hosts the analysis program[13]. Communications between the two computers is via a dial up telephone line.

There were three main reasons which prompted the decision to use a satellite processor. The first of these is that all analysis programs are insensitive to the physical layout of a circuit. They require only a description of the interconnection and element values. The simplest method to achieve this is utilizing a preprocessor for the analysis program. The preprocessor can then handle all of the details of drawing a circuit, and pass the analysis routines and the circuit description in the required form.

The second reason that prompted this decision is that the cost of computation time on a large machine can be significant. One way to minimize this cost was to decrease the amount of data transferred between the graphics terminal and central processor. The satellite processor acts to reformat the graphical data into a compressed form. For example, in order to draw a resistor from node 1 to node 2 would require the graphics terminal to send the address of nodes 1 and 2 (10 characters) and some code for resistor (2 characters). The computer would respond with approximately 42 characters necessary to draw and label the resistor. The satellite processor could convey the same information by the six character sequence "N1N2R3". This results in a 9:1 reduction in data.

The last reason for the satellite processor involves the environment this program will be used in. The satellite processor

allows the formation of the circuit under study to take place independently of the large central processor. The description of the circuit can then be prepared "off-line" and all corrections made before the services of the central computer are required.

CIRCUITS has two primary functions. One, it acts interactively with the user to draw a schematic diagram on the face of the graphics terminal and, at the same time, constructs a dictionary which defines all of the interconnections. The second function is to present this dictionary, at the users request, to the analysis program and convey the output of the analysis program to the graphics terminal. The first of these functions has been completed under this research project while the second is currently nearing completion as a second project. The description of the program in this thesis will be of the completed section and a synopsis of the objectives of the analysis portion.

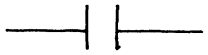
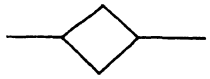
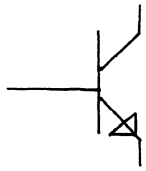
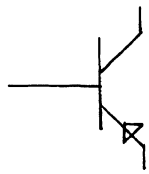
B. Description of CIRCUITS

CIRCUITS is capable of drawing a schematic diagram of any electronic circuit composed of the elements shown in Table IV. These elements are drawn by using the "mouse" to define the position of the element and the function keyboard to define the type of element desired.

In order to have the circuit drawn by a set of standard sub-routines, several restrictions were imposed on the user. The first of these restrictions is that all two terminal elements must be drawn between a pair of nodes which are located on a vertical or

TABLE IV

STANDARD ELEMENTS FOR CIRCUITS

ELEMENT	NAME	SYMBOL
NODE	N	
RESISTOR	R	
CAPACITOR	C	
INDUCTOR	L	
WIRE	W	
RIGHT/DOWN DIODE	D	
LEFT/UP DIODE	E	
SOURCE	S	
BATTERY	B	
MINUS BATTERY	A	
NPN-TRANSISTOR	Q	
PNP-TRANSISTOR	P	

horizontal grid, and that no more than four elements may be connected to any node. Since any practical circuit, and most analysis programs, do not impose this restriction, the concept of a wire was introduced.

A wire is a zero resistance connection between two nodes located at different physical points on the screen. Most analysis programs, however, do not allow a zero resistance connection between nodes, so that during the transmission of the dictionary to the central processor all nodes connected by wires will be assigned the same node number. This element does, however, allow the satellite processor to distinguish between two nodes located at different points.

Another restriction is that the node which will be at ground potential must be the first node specified by the user. This will insure, for the analysis program, that all node voltages can be referenced to the lowest node.

Basically CIRCUITS is composed of four main sections as shown in Figure 9. These sections are input/output and error handler, device and function handler, dictionary entry and modification handler, and analysis interface. The first section handles all input and output to the graphic terminal and handles error conditions. The input/output section will decide whether the input is valid graphical information and decode it into x and y screen coordinates. All inputs from the function keyboard are decoded and control is directed to the proper routine. Output data is converted to a form acceptable to the graphics terminal depending on its class, that is text or graphics. The error handler will create an appropriate error message and display it on the screen for any error condition that may arise.

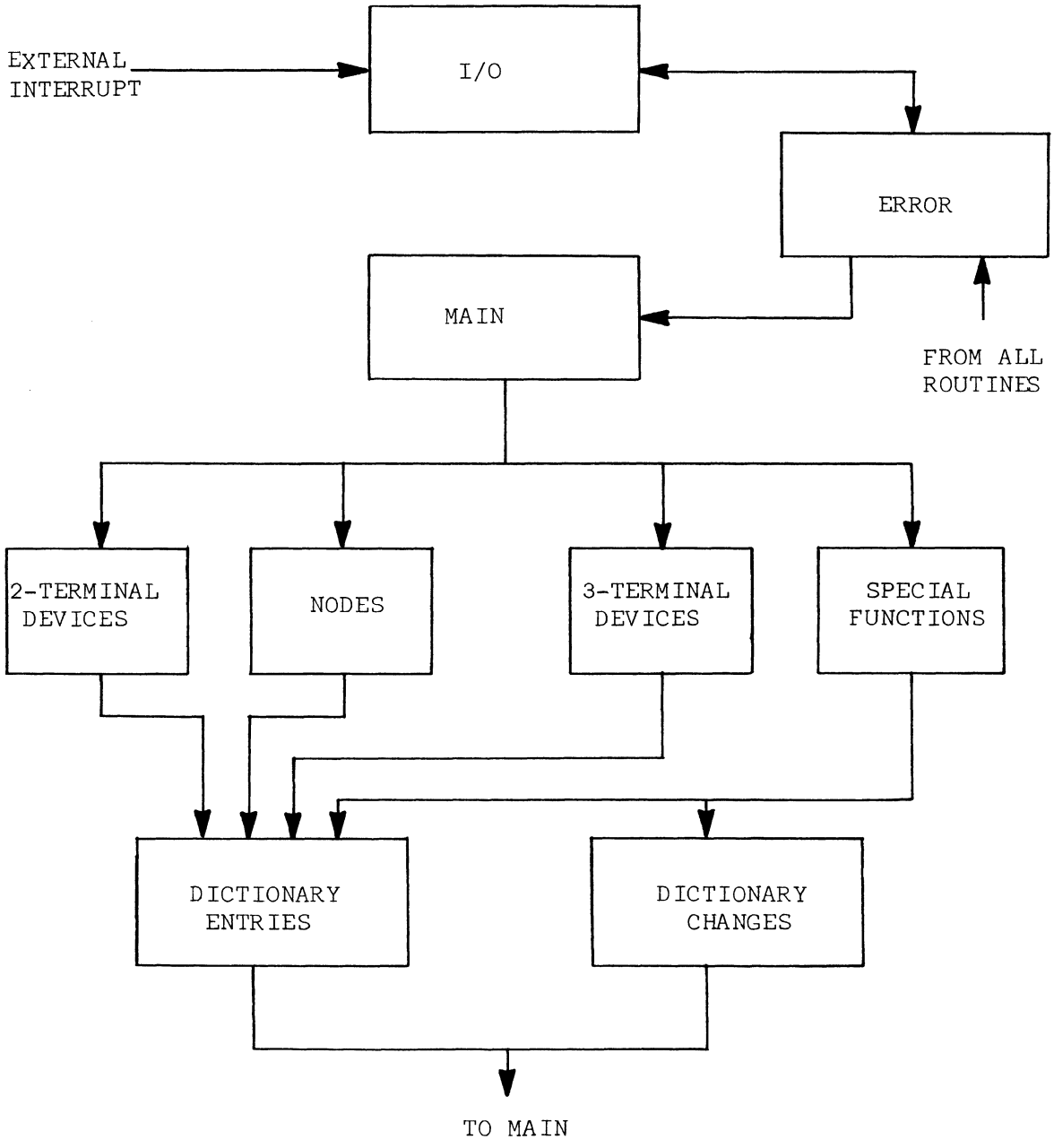


Figure 9. Block Diagram of Program CIRCUITS.

The second section handles the circuit elements and special functions and is composed of four subsections. The first of these handles node definitions. When the user requests a node definition, via the "mouse" and function keyboard, the symbol for a node is placed on the screen along with the number of that node. An entry is then made in the dictionary which defines the number of the node and its x and y coordinates. The x and y coordinates of this node will also be compared to the x and y coordinates of the last refenced node. If either of these is within a specified range of the last node, the current node will have its coordinates modified so that it lies on the same grid line as the last node. This will insure that all elements lie on a grid line. If the x and y coordinates do not lie within this range, then they are left alone. The program also keeps track of two nodes called the last node and the current node. Whenever a new node is requested, it is entered as the current node and the node that was the current node becomes the last node. It is between these two nodes that all elements are drawn.

The second subsection handles all two terminal elements. Upon entry into this routine the functions key input is matched against an element name. This name is a single letter which is unique for each element, for example, a left diode is named "E" while a right diode is named "D". These names will then be used to make entries in the dictionary so that the proper element and polarity can later be reconstructed. Next, the last node and current node are compared. An error condition will be generated if the two nodes are the same, or do not lie on a grid line, or are too close. If the nodes are

correctly positioned a flag will be set to indicate whether they are oriented vertically or horizontally. At the same time, the node to the left, or down in vertical orientation, will be designated as node one and the other as node two. Starting at node one, the element will be drawn by first drawing a lead from the node, then the element, and then another lead connecting to node two.

Each element has associated with it an element definition table. This table comprises all of the vectors necessary to draw that element. Figure 10 shows the form of a typical element definition table. The first word is the count of data words in the table. Each data word is composed of up to three vectors which are interpreted from right to left. Each vector entry comprises four bits with the right most bit (T) indicating whether it is a long or short vector. The short vector "0000" indicates the element is complete and the other vectors to the left are ignored. This is used if a data word holds less than three vectors. The long vector 2, -2 INV is an invisible vector or beam displacement. The table number (T) will determine which vector table will be used, and the vector number will indicate which entry in that table. The vector tables hold the characters which when sent to the graphics terminal will cause the proper vector to be drawn. The short vector table holds two characters per vector and the long vector table holds four. If the element is oriented vertically, the data is taken from a second pair of vector tables which essentially have the x and y coordinates of the vectors interchanged. This method of data storage, while seeming complicated, reduces the total amount of storage necessary for all

TABLE V

LIST OF FUNCTION FOR CIRCUITS

FUNCTION	DESCRIPTION
F0	Attention
F1	Resistor
F2	Capacitor
F3	Right/Up Diode
F4	Left/Down Diode
F5	Wire
F6	Inductor
F7	Battery
F8	Minus Battery
F9	Source
F10	Node
F11	Restart
F1F2	Transistor-PNP
F1F3	Transistor-NPN
F1F4	Redefine
F1F5	Corner Point
F1F6	Delete Element
F1F7	Delete Node
F1F8	Load Circuit
F1F9	Save Circuit
F2F3	Redraw

WORD	0 ... 11
1	COUNT
2	DATA
3	DATA
4	DATA

TYPICAL ELEMENT
DEFINITION TABLE

0 ... 3		7 ... 11			
VECTOR 3	T	VECTOR 2	T	VECTOR 1	T

DATA WORD

DECODE OF VECTOR FIELD

VECTOR i ΔX, ΔY	
000	RETURN
001	0,1
010	0,-1
011	1,0
100	1,1
101	1,-1
110	-1,1
111	-1,-1

SHORT VECTOR

T = 0

VECTOR i	ΔX, ΔY
000	3,0
001	2,-2
010	-2,-2
011	-2,2
100	0,-2
101	0,2
110	2,-2 INV
111	2,2

LONG VECTOR

T = 1

Figure 10. CIRCUITS Data Structure.

of the elements. For example, to draw all of the two terminal elements would require 44 short vectors and 57 long vectors. Since each short vector requires two characters and each long vector requires four characters to be transmitted to the ARDS, it would require 316 words of memory to store this data. Add to this the ability to rotate the vectors and 632 words of memory are required for the two terminal elements. The method used in CIRCUITS requires 59 words for the element definition tables, 47 words for the normal and rotated vector tables, and 102 words for the routine to do the translations, for a total of 208 words for the same elements.

The next subsection handles three terminal elements or transistors. These are drawn in a manner similar to the two terminal devices except that they are fixed in orientation and that only the base node is specified. When the user requests a transistor, via the function keyboard, it will be drawn with the base connected to the current node. In the process of drawing the transistor, two new nodes will be created, one at the collector, and one at the emitter. Upon exit from this section the emitter node will be the current node.

The last subsection handles the special functions. The first of these is RESTART. This function will erase the screen, reinitialize the program and dictionary, and allow the user to begin drawing a circuit diagram. REDRAW will redraw the circuit on the screen from the dictionary. DELETE ELEMENT will delete the element between the last node and current node from the dictionary. It will automatically resequence all elements with the same name and a higher number. DELETE NODE will delete the current node from the dictionary

if there are no elements connected to it. Also, all nodes higher than the deleted one will be resequenced. REDEFINE will redefine the referenced node as the current node without displaying the node symbol or changing the dictionary entry. SAVE will punch the dictionary description on paper tape so that is may be used at a later time. LOAD will read a paper tape produced by the SAVE function and construct a dictionary for that circuit. The circuit will then be drawn on the screen and the user may then add to it, modify it, or call for an analysis. CORNER will place a node on the screen at the intersection of two grid lines extending horizontally from the last node and vertically from the current node. ATTENTION will stop all processing and return control to the user. This is particularly useful when an error has been noticed during a REDRAW sequence. Table V lists the various functions recognized by CIRCUITS and their associated function codes.

The third major section of CIRCUITS is the dictionary handling routine. This section will make entries into the dictionary from data supplied to it by the node and element processors, or modify the entries upon request of the delete processors[14].

The dictionary is actually composed of two tables, the NODE TABLE and the ELEMENT TABLE. The node table illustrated in Figure 11 contains seven entries for each node. The first word is the node number, and the second and third words are the node's x and y coordinates respectively. The next four words are reserved for the element names and numbers which are connected to this node. If a fifth entry is attempted under this node, an error condition will arise. The element table consists of four words for each element. The first

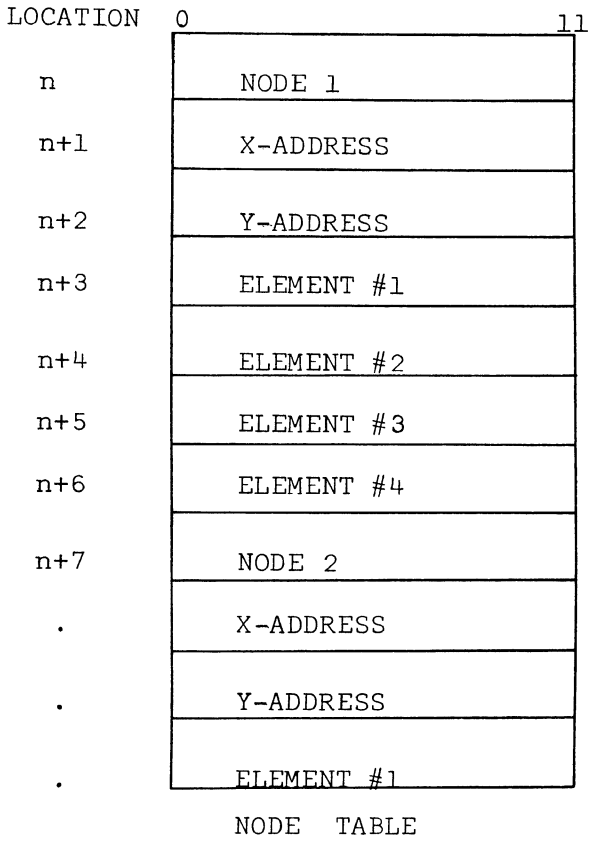


Figure 11. Dictionary for CIRCUITS.

is the name and number of the element, and the next word is the address of the value of the element in character form. The next two locations in the element table are the first and second notes connected to the element. Since transistors are completely specified by their base node, these two entries would show the same node. Given these two tables, the entire circuit can be reconstructed in either tabular form for an analysis program, or redrawn in graphical form.

The last section of CIRCUITS is the analysis routines. When the user requests an analysis of his circuit, he will be queried as to the analysis program name. A dictionary search will then be performed to insure that there are no elements which are unacceptable to that program. Assuming no such elements, the values for the elements will be requested and placed in the value table. Once this is complete, the user may request that a tabular description of the circuit be displayed or transmitted to the central processor.

The output from the analysis program will be directed to the graphics terminal for display. Once the output is complete, the user may modify his circuit by changing element values, or modify the topology of the circuit, and then call for another analysis.

V. AN EXAMPLE PROBLEM

This next section will present the steps necessary to draw a circuit on the ARDS. Figure 12 will be used as a typical circuit that a user may wish to construct for study.

The first step that is taken is to initialize the program by entering function number 11 (F 11) through the function keyboard. This will erase the screen and draw a dotted line across the lower portion of the screen. The user should refrain from entering into this area since error messages will be displayed here.

Using the mouse and the middle button (set point), the user would now point to the location on the screen where node #1 should appear. Since most analysis programs consider the lowest numbered node to be GROUND, this node should be entered first. Next F10 is entered and the terminal will respond with the symbol for a node and its number on the screen. Next the user points to the location of node #2 and enters this via the mouse and function keys. Now entering F9, the terminal will respond with the symbol for a signal source between nodes one and two. Next node #3 is entered, and then F1 is entered specifying a resistor. A resistor will then be drawn from node two to three. Notice that only one new node was entered; the element is always drawn from the last node to the current node. After drawing the resistor, node #3 will become the last node. This same process continues until R2 and B1 have been drawn.

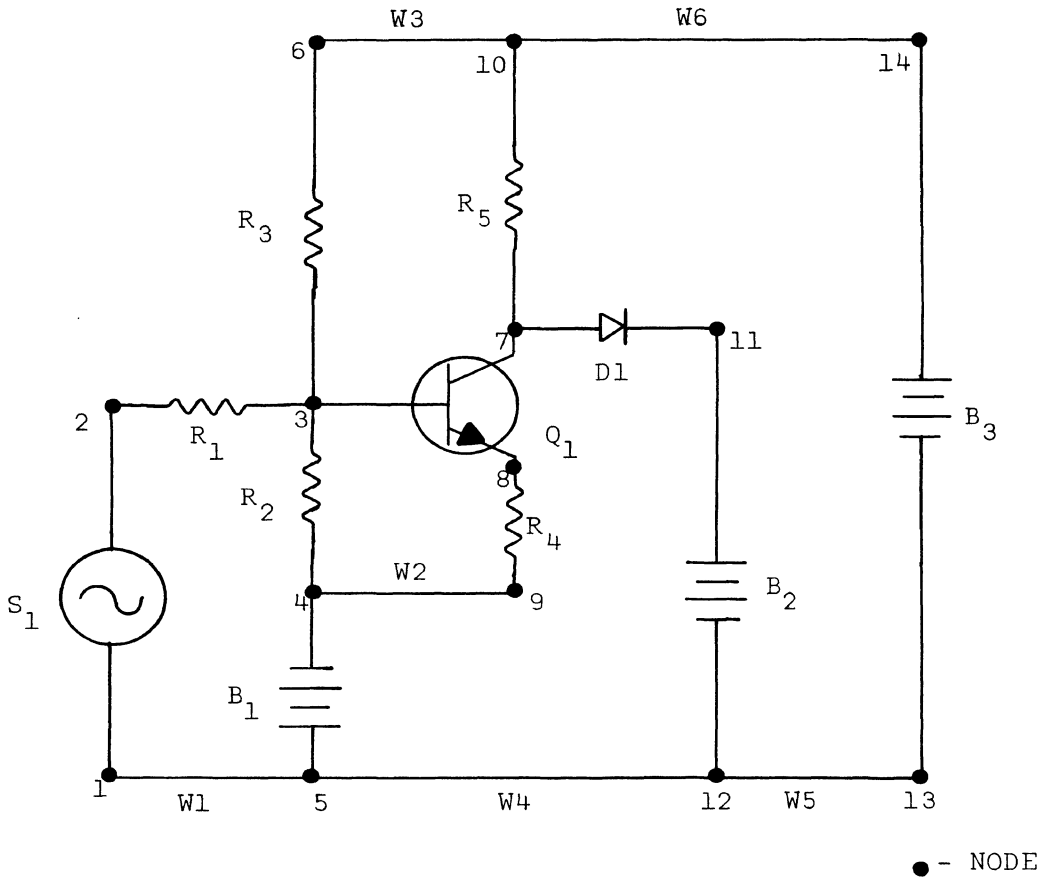


Figure 12. An Example Circuit.

At this point, node #5 is the last node and we wish to draw a wire to node one. By pointing the mouse to node one and entering F14, node one is redefined as the current node but no output is generated. After entering F5, a wire will be drawn between node one and node 5. Using F14 again, node three is redefined as the current node and then the user enters node six and resistor #3. Redefining node three again, the user enters F13 and a NPN transistor is drawn on the screen and two new nodes are created, node seven at the collector, and node eight at the emitter. Node eight is also the current node after the transistor is drawn. Entering node nine and F1 will draw a resistor from the emitter to node nine. Redefining node four and entering F5 will draw W2.

In order to create node ten at the intersection of two grid lines from node six and node seven, the user should first redefine node six and then redefine node seven. Now, if F15 is entered, a node will be created horizontally from the first redefined node (node six) and vertically from the last redefined node (node seven). If F5 is entered, W3 will be drawn. The rest of the elements and nodes can now be drawn in a similar manner.

As an example of the deletion capabilities, assume that when the diode was drawn F4 had been entered instead of F3. This would have resulted in a diode pointing to the left. To correct this, nodes seven and eleven are both redefined and F16 is entered. This will delete the diode from the dictionary but not the screen. To see the circuit without the element enter F23. Now redefining nodes seven and eleven and entering F3 will cause the proper diode to be drawn.

Once the display is complete, entering F19 will cause the dictionary which describes this circuit to be placed on paper tape. This allows the user to redraw this circuit on the screen at a later time merely by placing the tape in the reader and entering F18. Once the dictionary is constructed from the tape, the circuit will be drawn on the screen and control returned to the user. He may then use any of the functions to change or add to the circuit.

Tables VI and VII illustrate the dictionary for this circuit.

TABLE VI

NODE TABLE* FOR CIRCUIT SHOWN IN FIGURE 12

LOCATION	NODE	X-ADD.	Y-ADD.	ELEMENT #1	#2	#3	#4
n	N 01	-200	-100	S 01	W 01	-	-
n+7	N 02	-200	150	S 01	R 01	-	-
.	N 03	-100	150	R 01	R 02	R 03	Q 01
.	N 04	-100	0	R 02	A 01	W 02	-
.	N 05	-100	-100	A 01	W 01	W 04	-
	N 06	-100	300	R 03	W 03	-	-
	N 07	-20	200	Q 01	R 05	D 01	-
	N 08	-20	100	Q 01	R 04	-	-
	N 09	-20	0	R 04	W 02	-	-
	N 10	-20	300	R 05	W 03	W 06	-
	N 11	80	200	D 01	B 02	-	-
	N 12	80	-100	B 02	W 04	W 05	-
	N 13	180	-100	W 05	B 03	-	-
n+91	N 14	180	300	B 03	W 06	-	-

*Refer to Figure 11 for Detailed Memory Map.

TABLE VII

ELEMENT TABLE* FOR CIRCUIT SHOWN IN FIGURE 12

LOCATION	NAME	NUMBER	VALUE ADD.	MODE #1	MODE #2
m	S	01	-	01	02
m+4	R	01	-	02	03
.	R	02	-	03	04
.	A	01	-	04	05
.	W	01	-	05	01
	R	03	-	03	06
	Q	01	-	03	03
	R	04	-	08	08
	W	02	-	09	02
	R	05	-	07	10
	W	03	-	10	06
	D	01	-	07	11
	B	02	-	11	12
	W	04	-	12	05
	W	05	-	12	13
	B	03	-	13	14
m+64	W	06	-	14	10

*Refer to Figure 11 for Detailed Memory Map.

BIBLIOGRAPHY

1. M. T. Cook (ed.), "Interactive Graphics in Data Processing," IBM Systems Journal, vol. 7, 1968.

2. R. F. Crall and J. H. Tracey, "Operation of the SCC 650 Computer," Dept. of Electrical Engineering, University of Missouri-Rolla, Technical Bulletin CRL68.1, August 1968.

3. Advanced Remote Display Station (ARDS 100A) Reference Manual, Computer Displays, Inc.

4. R. D. Parslow, R. W. Prowse, and R. E. Green (ed.), Computer Graphics Techniques and Applications, London, Plenum Press, 1969.

5. D. B. Brick and E. N. Chase, "Interactive CRT Display Terminals," Modern Data, vol. 3, July 1970, pp. 60-68.

6. A. E. Brenner and P. deBruyne, "A Sonic Pen: A Digital Stylus System," IEEE Transactions on Computers, vol. C-19, June 1970, pp. 546-548.

7. S. Pardee, P. E. Rosenfeld, and P. G. Dowd, "G101-A Remote Time Sharing Terminal with Graphical OUtput Capabilities," available from the IEEE Computer Group Repository #R-70-162.

8. C. Machover, "The Intelligent Terminal," Pertinent Concepts in Computer Graphics (Proceedings of the Second University of Illinois Conference on Computer Graphics), ed. by M. Faimand and J. Nievergelt, University of Illinois Press, 1969, pp. 179-199.

9. D. Thornhill, J. Brackett and others, "Case Study in Interactive Programming: A Circuit Drawing and Editing Program for Use with a Storage Tube Display Terminal," presented at the Cybex Associate, Inc. Short Course on Computer Graphics (Minneapolis), June 1969.

10. G. I. Rhine, "ARDS Interface: Logic Diagrams," Dept. of Electrical Engineering, University of Missouri-Rolla, Technical Report CRL71.2, April 1971.

11. D. F. Dawson, F. F. Kou, and W. G. Magnuson, "Computer-Aided Design of Electronic Circuits a Users Viewpoint," Proceedings of the IEEE, vol. 55, pp. 146-1954, November 1967.

12. G. I. Rhine, "CIRCUITS: An Electronic Circuit Drawing Program," Dept. of Electrical Engineering, University of Missouri-Rolla, Technical Report CRL71.3, April 1971.

13. W. H. Sass, "Partitioning A Small Computing System for Computer-Aided Design," Software Age, vol. 5, pp. 10-12, February/March 1971.

14. R. Williams, "A Survey of Data Structures for Computer Graphics Systems," Computing Surveys, vol. 3, pp. 1-21, March 1971.

15. Y. Chu, "An ALGOL-Like Computer Design Language," Communications of the ACM, vol. 8, pp. 607-615, October 1965.

16. H. J. Pottinger and J. H. Tracey, "Formal Description of the SCC 650 Computer," Dept. of Electrical Engineering, University of Missouri-Rolla, Technical Report CRL68.1. August 1968.

17. G. I. Rhine and R. F. Crall, "Notes on the Operation of the SCC-650 Assembler," Dept. of Electrical Engineering, University of Missouri-Rolla, Technical Bulletin CRL69.3, December 1969.

VITA

George Irvin Rhine, Jr. was born on October 1, 1942, in Washington, D.C., where he received his primary and secondary education. He served in the United States Air Force for four years. During this period, he attended Anchorage Community College, Anchorage, Alaska. The remainder of his undergraduate training was at George Washington University, Washington, D.C. and the University of Missouri - Rolla. He received a Bachelor of Science in Electrical Engineering in January, 1970.

Since that time, he has been enrolled in the Graduate School of the University of Missouri - Rolla.

APPENDIX A

FORMAL DESCRIPTION OF THE ARDS-SCC 650 INTERFACE

This appendix presents a formal description of the ARDS-SCC 650 interface. The language used in this description is Computer Design Language postulated by Chu[15]. No attempt has been made to describe the computer or ARDS except where absolutely necessary for understanding of this description. A formal description of the SCC 650 computer should be referred to if a detailed description is desired[16].

All data directions in this description are with respect to the computer. RCV, for example, refers to the computer receiving data, and the ARDS transmitting.

Comment Begin The following flip-flops form the various control flip-flops and registers used in the interface. END

```
register    ACT;          $ Active F/F
            ARMFCN;       $ Arm Function F/F $
            ARMRCV;       $ RCV Interrupt Armed F/F $
            ARMXMIT;      $ XMIT Interrupt Armed F/F $
            CLK(1-4);     $ Clock Counter $
            ECHO;         $ Echo F/F $
            FCN1;         $ Function Control #1 $
            FCN2;         $ Function Control #2 $
            FF(1-11);     $ Function F/F's $
            IDL;          $ Idle F/F $
            IDRDY;        $ Interface Ready F/F $
            LOUT;         $ Line Out F/F $
            MODE;         $ Mode F/F $
            PRTY;         $ Parity Counter $
            SIP;          $ Shift In Progress F/F $
```

```
        SR(1-10);     $ Shift Register $
        SRF;          $ Shift Register Full $
        SINT;         $ XMIT Control F/F $
```

Comment Begin The following signal are logical combinations of other signals. END

```
Terminals ARDS=R7*R8*R9*R10*R11;          $ Device code '13 $
          CLF=CLRSW+MSCLR+DRDY*FTR*T6;    $ Clear Function $
          DSTARDS=DST*ARDS;               $ Display Status $
          DTATFR=IDL*SRF*TTARDS;          $ Data Transfer $
          EXUARDS=EXU*ARDS;
          FCN=FCN1*ARMFCN;                $ Function Active $
          FTR=FCN*TTARDS*SRF;             $ Function Transfer $
          MSCLR=STCLR+DTR+TMRARDS;        $ Master Clear $
          RCVINT=(IDL*SRF)+FCN;           $ Receive Interrupt Pending $
          RCV=MODE;                       $ REC Mode $
          SEL=IOP*R4*R5*R6                $ Select Command $
          SELARDS=SEL*ARDS;
          ST7=CLK1*CLK2*CLK3*CLK4;        $ Shift Time 7 $
          ST10=CLK1*CLK2*CLK3*CLK4;       $ Shift Time 10 $
          ST12=CLK1*CLK2*CLK3*CLK4;       $ Shift Time 12 $
          TFARDS=TFA*ARDS;                $ Transfer to ARDS $
          TMR=IOP*R4*R5*R6;               $ Terminate Command $
          TMRARDS=TMR*ARDS;
          TTARDS=TTA*ARDS;                $ Transfer from ARDS $
          XMIT=MODE;                      $ XMIT Mode $
          XTFR=XMIT*IDL*TFARDS;           $ XMIT Transfer $
```

Comment Begin The following signal are primary inputs from the ARDS and the computer.

The R-register is the CPU instruction register. The eight low order bits are inputs to the interface so that the I/O instructions and device codes can be decoded. The CPU is executing an I/O instruction when IOP is true. The CUP decodes three instructions, TTA, TFA, and DST, and the interface decodes the rest.

The computer will set DRDY=1 at T3 if IDRDY is true. This allows a "handshaking" to take place between the CPU and peripheral interface before any transfers are accomplished. DRDY is unconditionally cleared at T7. The accumulator outputs are available from T0 through T6. END

```
input       TTA,TFA,DST;        $ Computer I/O Commands $
            IOP;                $ Computer is executing an I/O
                                  Command $
            R(4-11);            $ CPU Instruction Register $
            A(0-11);            $ CPU Accumulator Outputs $
            T(0-7);             $ CPU Timing Pulses $
            DRDY;               $ CPU Ready F/F $
            STCLR;              $ Start-Clear Switch on CPU $
            DTR;                $ Data Terminal Ready from ARDS $
            LIN;                $ Data Line in From ARDS $
            CLRSW;              $ Clear Function Switch $
            TFRSW;              $ Transfer Function Switch $
            FSW(1-11);          $ Function Switches $
```

Comment Begin The following signals are primary outputs to the ARDS or computer.

Data will be gated into the accumulator at T5 if DRDY is true. EXTINT is the external interrupt signal to the computer. If this signal is true and the CPU interrupts are enabled, the CPU will be interrupted and control transferred to a service routine. If IOE is true, a flag will be set in the CPU indicating an error during an I/O sequence. This flag can be tested by the IOT command. END

```
Output        IN(0-11);            $ CPU Accumulator Inputs $
              LOUT;                $ ARDS Data Line $
              EXTINT;              $ External Interrupt to CPU $
              IOE;                 $ Error Signal to CPU $
```

Comment Begin The following subregisters are defined. END

```
subregister   DATA(1-7)=SR(2-8);   $ Data in Shift Register $
              PARITY=SR(9);        $ Shift Register Parity $
              START=SR(1);         $ Start Bit $
              STOP=SR(10);         $ Stop Bit $
```

Comment Begin The following section describes the asynchronous mode of operation. END

```
MSCLR: ACT←0,ARMFCN←0, ARMREC←0,
       ARMXMIT←0, ECHO←1, FCN1←0, FF(1-11)←0,
       IDL←0, IDRDY←0, LOUT←1, MODE←0, PRTY←0,
       SIP←0, SR(1-10)←0, SRF←0, XINT←0; $ This
       initializes all the flip-flops$
```

```
        SIP : CLK(1-4)←0;      $ clears Clock Counter $
            : if FSW(i) then FF(i)←1; Sets Function F/F $
            : if DELFCN then DELFCN←0; $ 1-millisecond
                                        monostable $
            : if TFRSW then DELFCN←1;
        DTR : if SELARDS then ACT←1; $Sets Active $
IDL*ACT*RCV : if LIN←0 then SIP←1; $Starts Clock $
        SIP : CLK=CLK count 1; $ Counts Clock Pulses $
        SIP : SR=SR rshift 1; $ Shifts Register $
   SIP*XMIT : STOP←1, LOUT←START, if ST12 then SIP←0
              and SRF←1; $ SR fills with one's data goes
              into LOUT and stops when complete $
    SIP*RCV : STOP←LIN, if ST10 then SIP←0 and SRF←1;
              $ Data goes into SR and stops when complete $
    SIP*RCV : if ECHO then LOUT←LIN; $ ECHO's Data $

     Comment Begin  This section describes the interface in the
     transmit mode. END
ACT*T1 : if TFARDS*IDL then MODE←1; $ Sets Mode for XMIT $
ACT*T2 : if XTFR then IDRDY←1; $ Interface is Ready $
ACT*T3 : if XTFR then START←0, DATA(1-7)←0, PARITY←0,
         STOP←1, PRTY←0; $ Clears Shift Register $
ACT*T5 : if DRDY*XTFR then DATA(1-7)←A(5-11);
         $ Data goes into Shift Register $
ACT*T6 : if DRDY*XTFR then SIP←1; $ Starts Clock $
    T7 : IDRDY←0;
ACT*T7 : IDL = XMIT*SIP' + SRF*SIP'; $IDL = 1 when done $
```

```
      T1 : if XMIT*SIP then XINT←1; $ Shift complete $
      T7 : if XINT*SIP then MODE←0; $ Resets to RCV $
         : if TFARDS + TTARDS then XINT←0;
     ACT : if ARMXMIT*XINT*IDL then EXTINT←1; $ Interrupt $
XMIT*SIP : if START then PRTY←PRTY; $ Counts ONE's out $
XMIT*ST7 : SR(2)←PRTY; $ Ensures Even Parity $

    Comment Begin  The next section describes the interface in
    the receive mode. END

 ACT*T2 : if RCV*SRF*TTARDS then IDRDY←1; $Data is ready$
    ACT : if DRDY*TTARDS*SRF then IN(5-11)←DATA(1-7),
          IOE←PRTY; $Data to computer and set Error if
          Odd Parity $
     T6 : if DRDY*SRF*TTARDS then SRF←0, PRTY←0, START←0,
          STOP←1, DATA(1-7)←0; $Data has been transferred.
          Clear Shift Register and SRF $
     T7 : IDRDY←0;
    ACT : if ARMRCV*RCV*RCVINT then EXTINT←1; $ INTERRUPT $
SIP*RCV : if LIN then PRTY←PRTY; $ Counts ONE's in $

    Comment Begin  The next section describes the various modes
    that can be controlled by the use of the Execute Command. END
EXUARDS : if A8*A9 then ECHO←0;          $ Sets ECHO Off $
          if A8*A9 then ECHO←1;          $ Sets ECHO On $
EXUARDS : if A10*A11 then ARMXMIT←1;     $ Arms Xmit Interrupt $
          if A10*A11 then ARMREC←1;      $ Arms Rec Interrupt
```

```
        if A10*A11 then ARMXMIT←0,
        ARMREC←0;                          $ Disarms Interrupt System $
EXUARDS : if A6*A7 then ARMFCN←1;          $ Arms Function Keys $
        if A6*A7 then ARMFCN←0;            $ Disarms Function Keys $
```

Comment Begin This section describes the operation of the Function Keyboard. END

```
      : if FSW(i) then FF(i)←1;            $ Sets Function F/F $
      : if CLRSW then CLF←1, FCN2←1;
  CLF : FCN1←0, FF(1-11)←0;                $Initializes Control $
TFRSW : if FCN2 then FCN1←1, FCN2←0;
      : if TFRSW*DELFCN then FCN2←1; $ Doesn't allow another
                                       transfer until switch is
                                       lifted $
  FCN : if SRF*TTARDS then IN(0)←1,
        IN(1-11)←FF(11-1);               $Function word goes to CPU $
  FCN : if SRF*TTARDS*DRDY*T6
        then CLF←1;                      $ Clears Control after
                                       data is transferred $
```

APPENDIX B

EXAMPLE INPUT/OUTPUT ROUTINES

The following sample routines are to illustrate means of communicating with the ARDS graphics terminal. They are written in SCC-650 Assembler[17] and it is assumed the reader is familiar with this language or some assembler language.

Example #1. This routine writes N characters stored starting at BUF then reads a character and jumps to GO.

```
SEL     '13       Select ARDS interface
DST     '13       Get status word
ANL     1         And with bit 1
SRA     0,G       Test if ACT = 1
HLT               No. halt. error
LDA     N         Get number
SRA     5         Convert to negative
STA     CNT       Store as count
LDA     BUF
STA     LOC       Pointer to Buf
LDA*    LOC       Get character
TFA     '13       Write
JMP     *+2       Ready
JMP     *-2       Not ready
MIN     LOC       LOC +1
MIN     CNT       CNT +1
JMP     -A        Not done yet
TTA     '13       Read character
JMP     *+2       Ready
JMP     *-2       Not ready yet
```

Example 1 Cont.

```
                JMP     GO          Continue
CNT             PAR     0           Count parameter
LOC             PAR     0           Location parameter
N               BSS     1           Number of characters
BUF             BSS     1           Location of characters
```

Example #2. This routine will select the graphics terminal and will enable the interrupt system and function keyboard.

```
                DST     '13         Read status
                ANL     4           AND bit 9
                SRA     0,G         Skip if DTR =1
                HLT                 Error, Halt
                SEL     '13         Select ARDS
                LDA     CW1         Control word 1
                EXU     '13         Arm XMIT interrupt and
                                     function keys
                LDA     CW2         Control word 2
                EXU     '13         Arm REC interrupt
                JMP     GO          Continue
CW1             PAR     '23         A6=0, A7=1, A10=1, A11=1
CW2             PAR     1           A11=1, A10=0
```

Example #3. This routine is an interrupt service routine. Control will be transferred here if there is an external interrupt. The routine will test to see if the ARDS caused the interrupt and if

not will go to SEV2. If the ARDS was the interrupting device, a test will be made to determine if the character at OUT should be transmitted to the ARDS or if input is waiting. If input is waiting, it will be stored at CHAR or FCN depending on whether it is a character or function word.

```
SERVICE     BSS     2           Linkage area
            STA     SA          Save accumulator
            DST     '13         Read status
            AND     *+1         AND with A1=1
            PAR     '2000       To check interrupt
            SRA     0,G         Skip if ARDS
            JMP     SEV2        Not ARDS
            LDA     OUT         Get character
            TFA     '13         Try and write it
            JMP     END         It went
            STA     OUT         Did not go, put it back
            TTA     '13         Read input
            SRA     0,N         Skip if negative
            JMP     CH
            STA     FCN         Store as Function
            JMP     END
CH          STA     CHAR        Store as character
END         LDA     SA          Restore accumulator
            CLI                 Clear interrupt
            JRT     SERVICE     Return to program
SA          PAR     0           Accumulator save area
```