ESD-TR-69-141

Semiannual Technical Summary

Graphics

31 May 1969

Prepared for the Advanced Research Projects Agency
under Electronic Systems Division Contract AF 19(628)-5167 by

Lincoln Laboratory

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Lexington, Massachusetts

AD0689782

The work reported in this document was performed at Lincoln Laboratory, a center for research operated by Massachusetts Institute of Technology; this work was supported by the U.S. Advanced Research Projects Agency of the Department of Defense under Air Force Contract AF 19(628)-5167 (ARPA Order 691).

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

LINCOLN LABORATORY

GRAPHICS

SEMIANNUAL TECHNICAL SUMMARY REPORT
TO THE
ADVANCED RESEARCH PROJECTS AGENCY

1 DECEMBER 1968 – 31 MAY 1969

ISSUED 24 JUNE 1969

LEXINGTON MASSACHUSETTS

SUMMARY

A relocating assembler and a linking loader have been programmed for the TX-2. To providc more convenient graphical facilities, a new Extended Graphic Subsystem has been written. Recent display hardware additions to TX-2 have included a color display, storage tubes, and a commercial ARDS console.

Application program work on semiconductor mask design has continued, with current work at an 80 gate/chip level. Initial exploratory work on a design data system has been performed. Application of the LX-1 microprocessor to vocoder and display control problems has been studied. Experimental systems for Interactive Computer-Mediated Animation and Real-Time Input saving have been completed. Performance measurements on the APEX system are being conducted and the data analyzed. A stand-alone circuit-testing terminal with a small computer is under development.

Accepted for the Air Force
Franklin C. Hudson
Chief, Lincoln Laboratory Office

CONTENTS

GLOSSARY

ALGOL	A high-level algebraic problem-solving language
AMBIT/G	A graphical programming language for manipulation of directed graphs
APEX	TX-2 time-sharing executive
ARDS	Advanced Remote Display System
BCPL	Basic Combined Programming Language – an intermediate level language for computer programming
DAP	Display Assembly Program
EGS	Extended Graphic Subsystem
LEAP	Language for Expressing Associative Procedures – an ALGOL-like TX-2 programming language
LSI	Large-Scale Integrated circuit technology
LX-1	A prototype microprocessor being constructed at Lincoln Laboratory
ML	A microprogramming assembly language
ROM	Read-only memory
TAP	TX-2 Assembly Program

GRAPHICS

I. LANGUAGES AND SYSTEMS

A. Assembler and Loader

An integrated relocatable code system has finally been created for the TX-2 comprising the BCPL compiler, a new relocatable assembler, and a linking loader. The output of the BCPL compiler is a text file of assembly code similar in form to any created by hand. The assembly code then passes to the new TAP (TX-2 Assembly Program) assembler which produces a relative binary file containing binary code, relocation bits, and symbolic linkage information. The linking loader combines relative binary files to produce executable TX-2 code.

This new system has been used to create both the BCPL compiler (written itself in BCPL) and the EGS described below. In addition, the TAP assembler with modifications serves as an assembler on TX-2 for the 16-bit testing terminal computer.

B. Extended Graphic Subsystem (EGS)

1. Introduction

During the past year at Lincoln Laboratory, a number of interactive graphic programs have been written on TX-2. These programs included the mask program, a logic input program, AMBIT/G, and an event simulator. All had one significant feature in common – provision for the users to draw a picture into the computer using the Sylvania Data Tablet and the character recognizer. A "windowing" capability is not available in the TX-2 hardware and is not provided by the display executive in APEX. Each programmer was required to provide his own windowing facility to allow users to input pictures larger than 10 inches square. While this task is not difficult to accomplish with LEAP, the duplication of effort is unnecessary. The EGS is designed to provide this service to user programs under APEX.

2. EGS

EGS is a subsystem under APEX and runs in user mode. The EGS contains five sections: a linkage and data structure package, a Display Assembly Program (DAP) interpreter, a clipper program, a display data generator, and a user-specified package of graphic entity definitions.

a. Entity Definitions

The definitions of graphic entities are procedures written in DAP by the user of EGS for his specific application. A simple procedure would take one point parameter and produce display data for that point. A more complicated procedure may take two point parameters and generate the data which describe an arrow, or a ruler. The user may even write a procedure which takes N points and generates the data which describe all straight lines connecting any two points.

b. DAP

These procedures are coded by the application programmer in DAP. The EGS system includes an interpreter to translate DAP. All data values in DAP are points (i.e., X, Y coordinate pairs). The simulated machine interpreting DAP contains 28 general registers for temporary storage (R0 → R27) and several special registers. There is also a stack for passing parameters. The format of a standard DAP instruction is OPCODE REGISTER.

Of the four DAP opcodes which generate display data, two generate data for vectors and two for characters. They are:

DRAW TO A POINT	DPT
DRAW BY DELTAS	DDL
CHARACTERS	CHAR
CHARACTERS CLIPPED	CHARC

c. Automatic Windowing

All position information from the DAP interpreter program passes through the clipper program. The clipper provides the automatic windowing facility for EGS.

All parameters passed to the DAP entity definition procedures are in terms of "floor" coordinates (raw position data). These coordinates are integers with the range ∓ 377777 ($\mp 130000_{10}$). The raw position data are translated and scaled by a user-specified offset and scale factor. These new position data are then clipped to lie within a user-specified window on the scope face. The windowing information which controls the actions of the clipper is contained in a 4-element array.

d. Picture Generation

Two types of calls can be made on EGS to generate a picture on the scope:

GENERATE
DISPLAY

GENERATE:– This call takes a list of parameters, pushes them onto the parameter stack, and then activates the DAP procedure specified in the call. The activation of any DAP procedure will, in general, process a number of parameters from the parameter stack and generate display data which will be added to the output buffer.

DISPLAY:– This call takes two Integer parameters: an item number, and a group number. All display data currently in the output buffer are passed to the APEX display executive as the specified item in the specified group. This call does not clear the output buffer and, in fact, may be followed by more GENERATE and DISPLAY calls. Each GENERATE call will add more data to the buffer. Each subsequent DISPLAY call will display the data accumulated in the output buffer.

The DAP regeneration program is activated by a REGENERATE call to EGS. This call takes a list of parameters containing the new values of the clipping array. The execution of this call performs two functions: first, it replaces the old values of the clipping array with the new ones; second, it regenerates the picture on the scope to reflect the changes in the clipping array. A REGENERATE call may be made at any time.

e. Other Calls on EGS

There are three other calls on EGS currently available: DELETE, INIT, and CLEAN.

A DELETE call takes the same parameters as DISPLAY: an item number, and a group number. The action of this call is to delete the specified item from the sequence 64 display structure and the DAP regeneration program. An item number of zero will cause all items in the specified group to be deleted and will remove the group from the sequence 64 display structure.

INIT initializes EGS and sets up an ephemeral file for EGS to use for impures. CLEAN also initializes EGS, but it differs from INIT in that it assumes the ephemeral file to be already set up.

C. Display Hardware

1. Color Display

Experiments with a two-layer red-green phosphor in a standard TX-2 display tube are under way. The color is switched by changing the accelerating potential of the CRT; circuitry for accomplishing a rapid change of 5 kV has been built and operated. The change in accelerating potential necessitates an accompanying change in the deflection sensitivity, and circuits for this have been developed.

The color tube has been tested on TX-2 with the mask layout program. Changes to support this display were temporarily made in the executive program. An important function in the mask program is the selection of a set of components as command operands. At present, selection of components is indicated by an X displayed on top of each designated part. The color display was used to replace this indicator with a color shift. The basic display is green, and selected components change to red. Initial results appear promising and further experiments will be conducted.

2. ARDS Terminal

Provision has been made for the connection of an ARDS terminal to TX-2. The ARDS was established as another console and placed into regular service. User acceptance has been encouraging, with the rapid display of text a motivating factor. A public program accessible from LEAP has been written which allows pictures to be drawn by programs.

3. Storage Tubes at TX-2 Consoles

Storage-tube displays have been added to the TX-2 consoles and are used by various programs. Several noninteractive graphical output programs (e.g., data plotting) have been modified to utilize this hardware. The storing of a reference picture on the storage scope by the mask program has also been found useful. In addition, a storage display has been placed in the Transistor lab in conjunction with the TIC (Testing Integrated Circuits) terminal. Parameter plots can be viewed immediately after measurements are made on actual circuit components.

4. Character Generator

A new character generator is under development using the high-speed decoders produced for the conic generator. This character generator is a direct replacement for the commercial one already installed, and its improved features include flexible character specification in a read-only memory, and variable speed generation required for efficient utilization with both refreshed and storage scopes.

II. GRAPHICS AND APPLICATIONS

A. Semiconductor Mask Design

Mask design on the TX-2 continues at a steady pace. Completed mask designs used to fabricate circuits include:

(1) A new version of the read-only memory,

(2) A simple 3-input AND gate design for the collector diffused isolation (CDI) technology,

(3) Beam-lead and face-down bonding masks for use in the heat dissipation studies,

(4) A fabrication process evaluation chip,

(5) Masks used in the face-up bonding investigations,

(6) A basic 80-gate array which will have custom metallization applied to it in order to implement a wide range of functions.

Several custom metallizations are in the final design stages for the 80-gate array. These include a circuit to provide data on gate fanout vs gate time delay and circuits for the microprocessor's adder, register, and multiplier function.

Microphotographs of the actual timing gate chain chip, previously designed on the TX-2, are found in Figs. 1 and 2.

A portion of the support of the Semiconductor Mask design software has entered the "fine tuning" stage. Recent software modifications have not changed the program's input/output characteristics but have improved the utilization of the TX-2's cpu, memory, and drum memory. For example, the program was modified to use integer representations of the mask component coordinates as opposed to real number representations. This change, not affecting the program's capabilities, accelerated the program by 30 percent, primarily because the TX-2 has no floating-point hardware; this change typifies a major portion of the support. The measurements for changes were done using the system measuring hardware on the TX-2.

Other software support includes new methods of generating CalComp plots of the designed masks and a means of documenting what components are where on a mask set. The mask design software has been factored into separate entities representing application-dependent and application-independent information. The application-dependent information represents component definitions for a particular circuit fabrication method such as bipolar, CDI, or MOS; the application-independent information represents the program's control and structural skeleton. These entities are kept as distinct text files and are merged to form the desired mask program. This setup allows the application-independent information to be kept in one place, as opposed to being replicated in several different places, and insures that any application program has the most current copy of the control section.

This factoring has spawned other uses of the control and structural skeleton of the mask design software. For example, an application program for designing a Naval ship's compartments was constructed by defining the appropriate application-dependent information. This program, which required four man hours for construction, was described and demonstrated for the Department of the Navy, Naval Ship Systems Command.

The "production" use of the mask design software has recently required 2 hours a day during which it is necessary to insure maximum response to a mask designer's requests. As an experiment during these hours, APEX has been conditioned to examine the status of console 1 (the mask user) in determining its scheduling strategy. Status is determined by the position of a switch in the system configuration register. If console 1 "is king," then it receives all the cpu and memory that it asks for and can use. Console 1's programs and data are brought into memory from the drum and given free reign over the cpu. As a direct result, other programs and data are removed from main memory and put onto the drum.

When console 1 has completed its computation, the other consoles are scheduled in the normal mode. Typically, these other consoles' programs and data are on the drum. Console 1's programs and data must be swapped out (they occupy at least 60 percent of memory) and other programs and data swapped in. This is a very slow process. Console 1 can wait up to 5 seconds before its programs and data are available in main memory. This wait is a fatiguing one for the mask designer and is not tolerated. Thus, in this instance, a critical TX-2 resource is its main memory rather than its cpu.

B. Design Data System

A trial implementation of a design data system has been created on TX-2. This work is patterned after the system developed at Stanford Research Institute but with a heavy bias toward the local style of graphical interaction. The ultimate aim is to create a computer-based aid for recording and manipulating the complex data involved in a system design. Work to date has produced a program which can be used to explore the kind of interactive assistance needed. The program is written in LEAP and is coupled to the input saving facility also described in this report.

1. Data Concepts

Information about a design is contained in a collection of pictures, each containing any number of boxes and lines. These components are normally used to construct some kind of a diagram (Fig. 3). Arbitrary selections of text can be placed in a box or attached to a line. Routines adjust display of text to the confines of a box. Pictures can be viewed with normal zoom and offset controls. Each picture has a name and can have a descriptive title.

Connection between related pictures is provided by several mechanisms. A box may be an entry point to a subordinate picture and, if so, has a "P" at its upper left corner. Making an appropriate drawn command on such a box causes the link to be traversed and the subordinate picture to be displayed. Links by name to pictures are also possible. Either the first word of text in a box or the word following "(SEE" in the text can be used as a picture name for cross referencing. Appropriate commands drawn on top of boxes will invoke any of the three linking mechanisms. A linking command to a nonexistent picture creates a new blank picture appropriately connected into the picture structure.

The box linkages provide a hierarchy of pictures descending from an initial picture. An arbitrary fanout is available at any level since any number of boxes can be included in that picture and used as entries for subordinate pictures. It is possible to restructure these links and make two separate boxes (perhaps in different pictures) have the same subordinate picture. Thus, the tree hierarchy can in principle be turned into arbitrary connections. However, the box linkage mechanism is intended to provide hierarchical ordering to pictures. In contrast, the name linkage mechanism provides arbitrary links across the picture structure at will.

2. Program Operation

Commands to this program are mainly given by drawn marks on the Sylvania tablet and fall into several broad categories:

(a) Picture Element Manipulation: Commands to make, move and delete boxes, lines, and text and to control picture viewing (zoom, etc.)

(b) Structure Linking: Commands to move the display to selected pictures in the structure and to modify the structure

(c) Storage Scope Control: Commands to place various pictures or picture parts on the console storage scope for reference purposes

(d) Administrative: Commands to save or read in a structure from disk or to go to the symbol trainer, etc.

Trial use of this system has just begun to obtain experience with the various features. As with any program of this type, it is a difficult task to discover just what kinds of computer help are really useful and which initial "good ideas" never get used.

A critical and as yet unsolved problem is that of indexing. When working with this system, an overview of the structure created is necessary. Schemes for obtaining a list of the existing pictures are available, but more are needed.

C. Microprocessor Applications

1. A Microprogrammed Digital Vocoder

a. Introduction

The feasibility of using the microprocessor under development by Group 23 as a digital vocoder analyzer and/or synthesizer has been investigated. An implementation of the microprocessor as an analyzer only (with no pitch detection) was microcoded and simulated. Hardware modifications to the microprocessor to enable real-time operation are described below.

b. Short Functional Description of the Microprocessor LX-1

Two versions of the microprocessor are contemplated. The first, called the prototype, will have an 80-nsec control memory cycle time. The second version, using LSI technology, will have a control memory cycle time which is five times faster.

The LX-1 microprocessor is described in a memo by G. D. Hornbuckle,* an excerpt of which follows:

> "The three major components of LX-1 are a 64 bit × 256 word control memory (sometimes referred to as the read-only memory or ROM), a set of sixteen 16-bit registers, and several function units which perform operations on the data in the registers. Each of the registers is connected to three buses called A, B, and D (see Fig. 4). The A and B buses are the data paths from the registers to the function units, and the D bus is the data path from the function units to the registers. Control signals come, for the most part, from the control memory."

*G. D. Hornbuckle, "LX-1 Reference Manual," private communication.

There is also a writeable 16-bit × 128-word scratchpad memory, accessible in one control memory cycle time (80 nsec). Thus, some example operations on the LX-1 are

```
A·BUS/3–B·BUS→D·BUS
```

where A·BUS, B·BUS, and C·BUS are register numbers, and /3 implies a 3-bit right shift

```
λ14→D·BUS
```

i.e., read the value of scratchpad location 14 onto the D bus.

Branching in LX-1 is achieved by setting and testing six special bits, denoted

C	carry bit
Z	zero bit ($Z = 1 \Rightarrow$ word is zero)
H	high-order bit
L	low-order bit
S	bit 1
F	overflow

Thus,

```
A·BUS + B·BUS→D·BUS, Z, F
```

will cause the Z and F bits to be set according to the result of the addition;

```
LOOP(1) A·BUS→D·BUS, C/LOOP(C)
LOOP(0)
```

will cause a branch to the label according to the value of C.

c. Short Functional Description of the Digital Vocoder

The vocoder is described in papers by W. M. Anderson, Jr.,* and by T. Bially.† A short outline of the analyzer is given here (quoted from T. Bially, "Structure of a Digital Channel Vocoder").

> "The spectrum analyzer portion of the vocoder consists of thirty-two filters of the form shown in Fig. 5. Input speech is sampled at a ten kilohertz rate (once every 100 μsec) and is simultaneously modulated by two reference sinusoids whose frequency is equal to the center frequency of the filter in question. Fifty successive samples at the output of each modulator are summed and added to the sum of the fifty output samples. Thus at time 99T the signal at point x in Fig. 5 has the value
>
> $$\sum_{k=0}^{99} S(kT) \cos n w_0 kT$$
>
> and that at point y is

* W. M. Anderson, Jr., "Specification of a Digital Vocoder System," presented at Acoustical Society of America, Cleveland, 19-22 November 1968.

† T. Bially, "Structure of a Digital Channel Vocoder," to be presented at IEEE International Conference on Communications, Boulder, Colorado, 9-11 June 1969.

$$\sum_{k=0}^{99} S(kT) \sin nw_0kT$$

The square root of the sum of the squares of these two values is computed at point z:

$$\sqrt{\left(\sum_{k=0}^{99} S(kT) \cos nw_0kT\right)^2 + \left(\sum_{k=0}^{99} S(kT) \sin nw_0kT\right)^2}$$

$$= \left| \sum_{k=0}^{99} S(kT)e^{jn_0kT} \right|$$

Every five milliseconds then, the signal at z is numerically equal to the magnitude of the nw_0 component of the discrete Fourier transform of the preceding ten milliseconds of S(kT). . . . Four successive outputs are averaged to yield the final output once every 20 milliseconds. The thirty-two analyzer channels are spaced one hundred cycles wide. The center frequencies are 100, 200, . . . , 3200 Hz."

Thus, every twenty milliseconds, we have thirty-two values corresponding to the channel signals for the current interval.

"These are encoded into a total of thirty two bits by a combination of logarithmic quantization, averaging of adjacent channel values at the high frequency end of the spectrum, and a linear (Hadamard) transformation which removes some of the inter-channel correlation. The spectral parameters thus require $32/20 \times 10^3$ or 1550 bits per second to transmit."

d. Description of the Microprogrammed Vocoder

Structure:— The vocoder was simulated using the LX-1 simulator on TX-2.* The simulator takes input samples from an APEX file and returns the vocoded speech to that file. A 20-cycle delay was assumed from the time a main memory read or write request is issued, until the request is completed. However, since main memory operations are overlapped with computation, the effective delay is probably three cycles.

Except for input/output, and input buffer maintenance, the vocoder simulator microcode is identical to that of a real implementation. The I/O operations, however, do not add significant overhead to the computations; thus, the values obtained from simulation are reasonable.

There are three main loops in the code:

LOOP 1 This calculation is done every 100 μsec (if real time) and involves taking the discrete Fourier transform

*Semiannual Technical Summary to the Advanced Research Projects Agency on Graphics, Lincoln Laboratory, M. I. T. (30 November 1968), DDC 679991.

of the current sample at 32 frequencies (100 to 3200 cps) and taking the running sum over the last 50 samples.

LOOP 2 Every 50 samples (every 5 msec), the sum is dumped and, for each channel, it is added to the previous 5-msec sum. We thus have, every 5 msec, the sum over the last 100 samples. The magnitude of the spectrum of each channel is calculated, and the running sum over the last four spectra is calculated.

LOOP 3 Every fourth spectral measurement (every 20 msec), the logarithm of the sum of the spectra is calculated, and the 32 channels are compressed to 16 by appropriate linear combinations of the logarithms. Finally, the sixteen results are multiplied by a Hadamard matrix to decorrelate the channels, and the channels are appropriately quantified to yield a total of 32 bits every 20 msec.

Further details on each of the loops follows.

LOOP 1:— In the current implementation, this loop is approximately 50 microinstructions long, plus 25 control memory words where the sine and cosine table is stored. The calculation of the current sample is interleaved with reading in of the next sample from control memory, thus the latter effectively takes no time. Only one-quarter of both the sine and the cosine waves is stored. The calculation to discover which table value to use and whether to complement is approximately 18 microinstructions long and takes from 9 to 18 cycles, depending on the values used. The average is 13 cycles. At this point, we have loaded the appropriate values for the sine and cosine in a register. These must be separated, multiplied by the sample value, and the result added to the running sum. This section is approximately 19 microinstructions long and takes from 17 to 44 cycles, depending on the values used. The average is 30 cycles. The calculation for LOOP 1 is thus 45 cycles/channel, or 1440 cycles/sample, i.e., 115 μsec, assuming 80-nsec cycle time. The worst case is around 160 μsec; the best case is around 80 μsec.

LOOP 2:— This section is approximately 30 microinstructions long and involves summing the value of the previous 5-msec sum to the current one. The previous values must be read in from external memory overlapped. Finally, the magnitude of the spectrum is taken.

The magnitude taker is 7 microinstructions long and takes from 4 to 7 cycles. Setup for taking the magnitude is 6 cycles. Taking the running sum and testing for conditions is another 6 cycles. Thus, approximately 18 cycles/channel are executed, i.e., 576 cycles every 5 msec, or 48 μsec. This calculation thus adds approximately 1 μsec/sample, if the input samples are buffered.

LOOP 3:— This section is approximately 120 microinstructions long. The log taker is 15 microinstructions long and takes 18 cycles. The channel compressor is 40 microinstructions long and takes approximately 100 cycles.

Calculating the Hadamard coefficients is 22 microinstructions long and takes approximately 20 cycles/coefficient. The matrix multiplication itself takes 6 cycles/element including setup

256 coefficients × ~30 cycles ⟹ 7680 cycles/spectrum

Quantization is 40 microinstructions long and takes ~100 cycles. Thus, the total is 18 × 32 cycles for log, 100 cycles for compressor, 7680 cycles for Hadamard, and 100 cycles for quantization, i.e., 8500 cycles or 680 μsec every 20 msec. If the input samples are buffered, this averages out to approximately 4 μsec/sample.

Discussion:– All times and cycle values are approximate and are on a per-sample basis, i.e., the input is appropriately buffered:

	Cycles	Prototype 80 nsec/cycle	LSI 15 nsec/cycle
LOOP 1	1500	120 μsec	25 μsec
LOOP 2	15	1 μsec	250 nsec
LOOP 3	50	4 μsec	750 nsec
Overall	1565	125 μsec	26 μsec

The above implementation shows that, with the anticipated microprocessor prototype structure, the analysis of input speech can be done in under twice real time. However, two factors have not been taken into account: the maintenance of the input buffer, and the section of the analyzer dealing with pitch detection.

It can also be seen that LOOPS 2 and 3, even though they are long, add at most a 10-percent overhead to the overall calculation if a 200-sample input buffer is assumed. In order to achieve a real-time implementation, therefore, LOOP 1 must be optimized.

Optimization of LOOP 1:– Two improvements can be made to LOOP 1 by adding extra hardware.

(a) Instead of storing one-quarter of the sine wave only in the read-only memory, a full cycle could be stored (100 registers), thus making the decoding for the table lookup much faster. The table would have to be 12 bits × 100 words long and be accessible in one cycle time. Lookup would then take 4 to 6 cycles, instead of 9 to 18 cycles. With these values, LOOP 1 would take 55 μsec at best, and 128 μsec at worst. Thus, this change alone is not sufficient.

(b) A hardware multiplier could be added as a function box. Assuming it could multiply in one cycle, the two multiplies and adds by the sine and cosine would take 10 cycles. With this change alone (i.e., with only a quarter sine cycle stored), LOOP 1 would take 55 μsec at best, and 80 μsec at worst. Thus, this change alone is sufficient to insure real time if the maintenance of the input and output buffers and pitch detectors could be separate hardware.

If both these changes are made, LOOP 1 would take around 40 μsec, leaving ample time for LOOPS 2 and 3.

Conclusion:– For a real-time implementation, the current microprocessor prototype is sufficient if

(a) a 200-sample (slow) input buffer memory is added, and

(b) a fast multiplier is added.

A 100-word read-only sine-cosine table would help but would be insufficient alone to insure real-time operation.

e. Other Possible Hardware Modifications

Instead of the 128-word scratchpad memory, where all temporary results are stored, dynamic shift registers could be used to store the accumulations of sines and cosines, as well as to store the 50-sample sum delays, as suggested by T. Bially.

f. Conclusions

The LX-1 microprocessor prototype can be used as a vocoder analyzer, with some modifications. The LSI version, which should be five times faster, could certainly be used with no hardware modifications other than the input buffer, and, indeed, might be fast enough to be used in half-duplex mode for both analysis and synthesis.

2. Microprocessor as a Display Processor

Work is currently in progress on applications of the LX-1 microprocessor as a display controller. The speed of LX-1 and its very flexible I/O characteristics make it very suitable for this application.

Two system structures are currently being considered. One is a remote terminal serving several scope consoles, with a disk, perhaps, for bulk storage. The other is a system using refreshed scopes running partly from the central computer's memory and making calls to display subroutines stored in the LX-1's local memory. The objective of this second system is to reduce the load on the central computer memory due to cycle stealing. In both system organizations, the microprocessor would perform automatic windowing.

D. Interactive Computer-Mediated Animation*

A Ph.D. thesis[†] has been submitted to the Electrical Engineering Department at M.I.T.; the abstract is quoted below:

> "The use of interactive computer graphics in the construction of animated visual displays is investigated.
>
> "The dissertation presents a process called <u>interactive computer-mediated animation</u>, in which dynamic displays are constructed by utilizing direct console commands, algorithms, free-hand sketches, and real-time actions. The resulting "movie" can then be immediately viewed and altered.
>
> "The dissertation also describes a special kind of interactive computer-mediated animation that exploits the potentialities of direct graphical interaction. The animator may sketch and refine (1) static images to be used as components of individual frames of the movie, and (2) static and dynamic images that represent dynamic behavior, that is, movement and rhythm. Because these latter pictures drive algorithms to generate dynamic displays, the process is called <u>picture-driven animation</u>.

---

*A portion of this work was supported by Project MAC at M.I.T.

†R. M. Baecker, "Interactive Computer-Mediated Animation," Ph.D. Thesis, Department of Electrical Engineering, M.I.T. (June 1969).

"Each representation of movement and rhythm determines critical parameters of a sequence of frames. Thus, with a single sketch or action that generates or modifies the representation, the animator can exercise dynamic control over an entire interval of the movie. One natural way to do this is by mimicking in real time a movement or a rhythm, using a stylus or a push-button.

"These concepts are supported by experience with three special-purpose picture-driven animation systems which have been implemented and used on the M.I.T. Lincoln Laboratory TX-2 computer.

"The dissertation also presents an outline of the proposed design of a multi-purpose, open-ended, interactive Animation and Picture Processing Language. APPL is a conversational language which accepts direct sketches, direct console commands, and algorithms that control interactive dynamic displays.

"Solutions are presented for the following problems: How can the system be structured so that the command set can easily be augmented by the animator? How can movie time be represented in the language, and how does the choice of representation interact with the flow of program and system control? What computational data structure can facilitate the modeling of sequential and hierarchic structures of pictures and dynamic data? How can we provide a rich picture description capability in the language? How can we facilitate the construction of programs which describe the user's interaction with the system?

"APPL programs are included to demonstrate that the language can be gracefully used to construct dynamic displays, to build system tools that aid the construction process, and to implement special-purpose interactive computer-mediated animation systems."

Programs implementing selected portions of the concepts developed have been created on the TX-2. The initial portion of an animated film explaining this approach to animation has been created. Figures 6(a) and (b) show two different scenes from a short cartoon clip from this film, and Figs. 7(a) and (b) show pictures which define and drive the animation actions of the cartoon.

A more detailed summary may be found in a paper by R. M. Baecker.*

E. Storing of Graphical Inputs

Exploratory work on the storing and later reuse of interactive graphical inputs has been reported in a thesis.† An initial implementation of this capability has been programmed for the TX-2. Application programs which use the input replay mechanism can now be written even though this first implementation is somewhat inefficient. It is possible from an application

* R. M. Baecker, "Picture Driven Animation," Proc. 1969 Spring Joint Computer Conference, Boston, Massachusetts.

† E. L. Thomas, "The Storing and Reuse of Real-Time Graphical Inputs," MS Thesis, Department of Electrical Engineering, M.I.T. (June 1969).

program to cause the storing of sequences of on-line interactive inputs, to observe and modify the stored input stream, and to play back a stored stream to the program as though the inputs were coming directly from the console. The implementation is particularly oriented toward the data tablet as the principal console input device.

The contents of a stored input stream can be observed in several ways:

(1) As a text representation showing condensed information about each component of input

(2) As a graphical representation of the input shown in its proper place on the display screen

(3) As a sequence of inputs viewed in a movie-like manner one after the other.

The condensed text representation is used for deletions and insertions by positioning a cursor to the appropriate item. New input can be added by drawing directly; however, the lack of a program display picture context makes this not a generally useful mode of input extension. Deletion of extra inputs is easily accomplished and often necessary to make a useful stored input set.

On-line input commands can be inserted in the stream and, when encountered by the playback system, will temporarily halt the playback until a live input is entered. This input is used by the application program, then the playback of recorded inputs continues.

Figures 8 through 11 show pictures of the recording and playback system in action within the design data program.

Initial usage of this kind of an input saving appears promising, but further work and experimentation will be needed to determine the real potential of this facility.

F. System Performance Measurement

A system performance measurement facility has been developed for the APEX time-sharing system.

When activated, this facility records the occurrences of specified events in the running of APEX. Events are recorded by means of strategically placed subroutine calls to a program which buffers the information and writes it out on secondary storage for later analysis.

In order to measure scheduling efficiency, the following events have been specified: a user moving from the inactive to the active queue (e.g., completion of typing a command), a user being put on the air, the breaking of a user before completion of a job and returning him to the active queue (e.g., in-out fault, time slice expired), and a user completing a job and returning to the inactive state. With each event, real-time clock value, number of core pages used, etc., are also recorded.

The two plots shown in Figs. 12(a) and (b) were obtained by analyzing the scheduling events occurring during one hour of APEX activity and by plotting the data using the TX-2 Lincoln Reckoner facilities. The plots represent waiting time (i.e., time from first entering the active queue to completion of the job), in the horizontal axis, vs the amount of time actually spent executing the user's instructions, in the vertical axis. In the ideal case, all points would lie on the diagonal, indicating that all of a user's waiting time was spent executing his instructions. The units in the plots are numbers of microseconds; to avoid bunching toward the low end, log-log (base 10) plots were taken.

Figure 12(a) covers all the jobs. The cluster of points at about 10 to 50 msec has been determined to be the heavy usage of the scope text editor; each movement of the cursor and each insertion or deletion of a character is an individual job. The proximity to the diagonal of this cluster indicates the comparative efficiency with which the scope text editor interactions arc scheduled. Elsewhere on the plot can be seen jobs running for other periods of time; some of the jobs clearly required a proportionately large amount of waiting timc in comparison with execution time. Points at the lower left-hand corner of the plot are far from the diagonal due to the relatively large amount of overhead in scheduling a job that runs only hundreds of microseconds.

Figurc 12(b) covers only the jobs which required running and waiting times in excess of 100 msec.

G. Computer-Aided Testing

An SEL810A computer with 8k of 16-bit core storage and a 225k word-swapping drum has been delivered for use as a controller for automated circuit testing. Through a combination of hardware modifications and software routines, a demand paging memory organization has been implemented and is working. The user program runs as though it had available 128 pages of 512 memory words each. Of these 128 pages, 14 at most will physically be in core storage; the others will be resident on the swapping drum. If the program attempts to reference a page that is not in core, the modified hardware will trap the reference, an interrupt routine will load the desired page into core from the drum, and the user program will be restarted. When no free pages are left in core, space is made available by unloading filled pages back onto the drum. The operation of this paging system is transparent to the user program.

A fixed-priority multiprogrammed executive has been designed to run within this virtual memory system and is currently being implemented. Up to twelve processes can be run beneath the executive on a fixed-priority basis. The approach is different from that of a general-purposc time-sharing system in that all processes operate in the same 128-page address space rather than each having its own address space. The processes will consist of a user program, several I/O handlers, and a normally dormant interrogation and debugging package. In the event of system failure, control will be passed to the interrogation package. Normally, control will be with the user program and its evoked I/O routines. The process runner will attempt to give control to the highest priority process that is runnable. In the event of a missing page, lower priority proccsses will be freeloaded when possible until the page has been retrieved from the drum. Processes and interrupt routines are able to wake up other processes or to block themselves pending later wakeup. An aggressive paging strategy attempts to remove from core pages belonging to blocked processes.

Except for the inner core of the executive, consisting of the process runner, the interrupt routines, and the swapping routines, all coding is being done in BCPL, and is being compiled into SEL binary files on the TX-2. The executive core, the total length of which is less than one core page, is written in assembly level language. This executive, along with the 128-word page table and a few hundred words of global variables, is the only part of the system frozen in core.

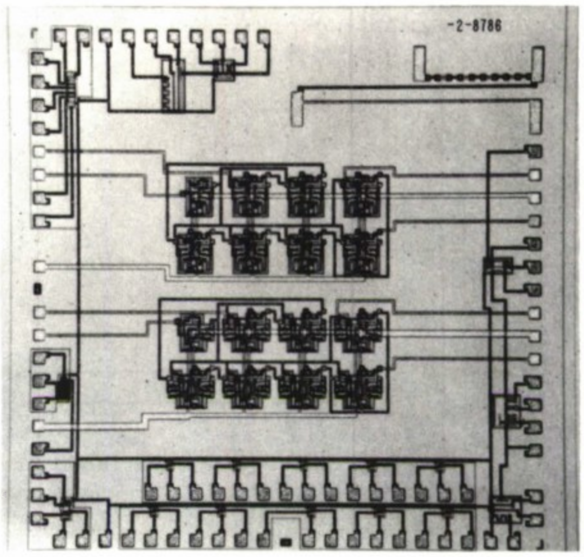


Fig. 1. Microphotograph of on entire chip contoining o test circuit for timing.

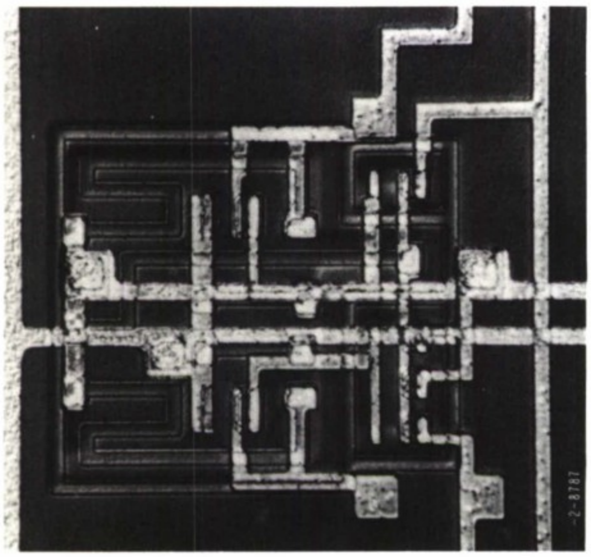


Fig. 2. Microphotogroph of o single gate in timing chip.

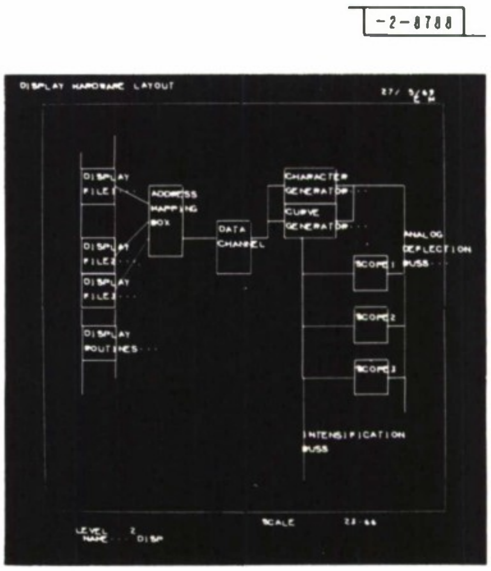


Fig. 3. Sample diogrom drawn with design data system.

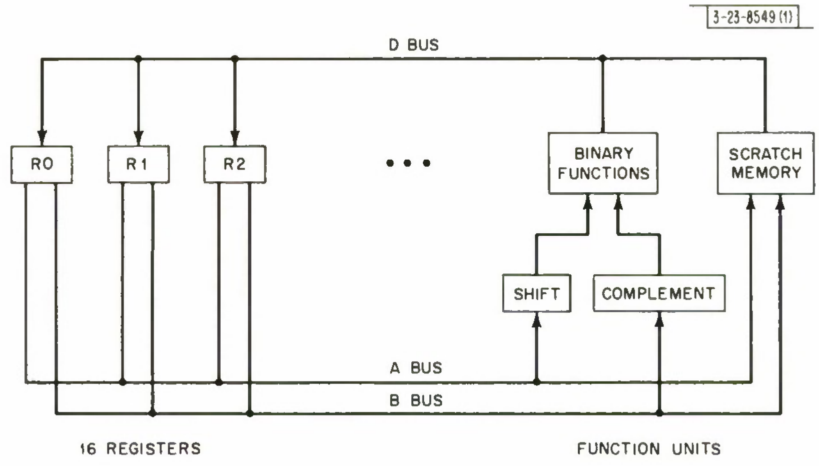


Fig. 4. LX-1 microprocessor bus organization.

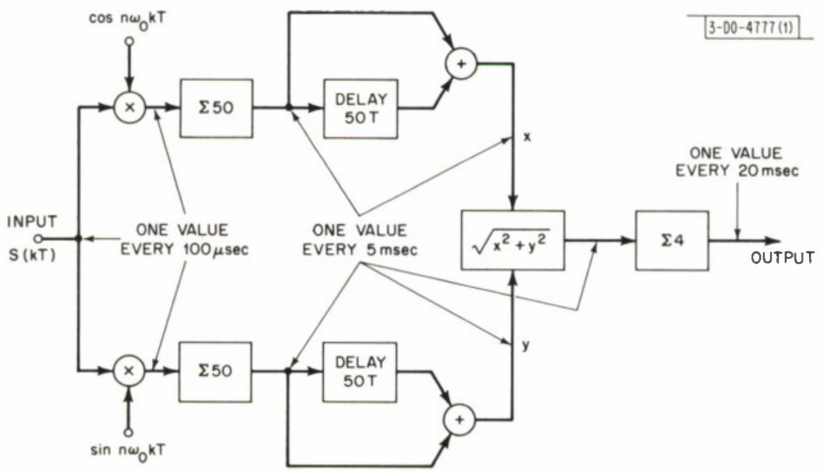


Fig. 5. One of 32 analyzer channels.

-2-8789

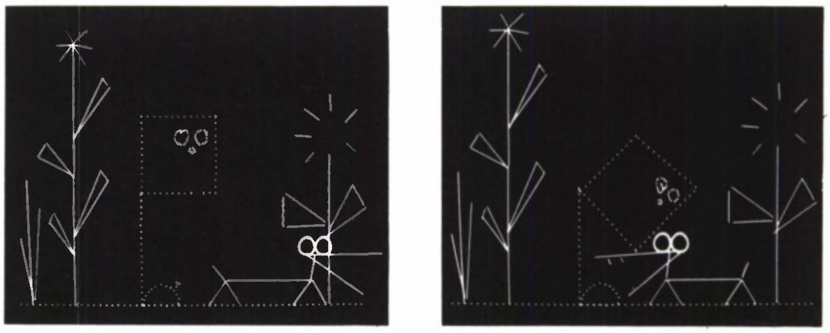
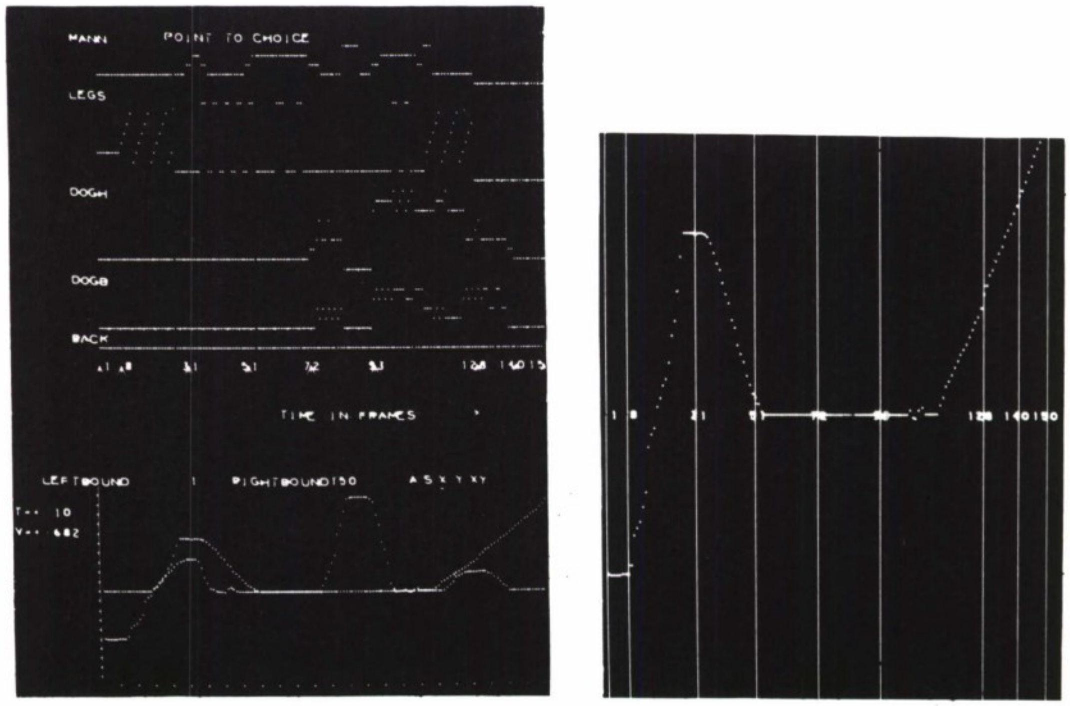

(a) (b)

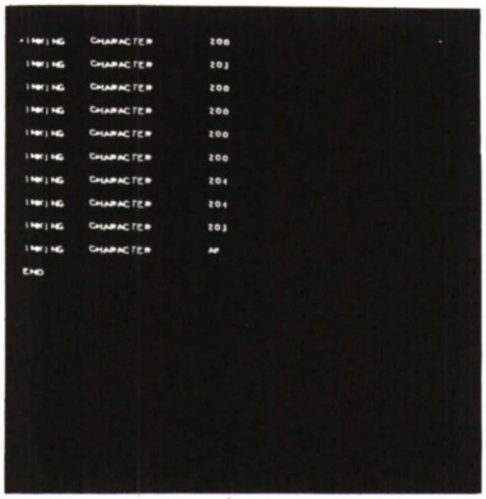
Fig. 6(a-b). Two scenes from a short cartoon clip.

-2-8791



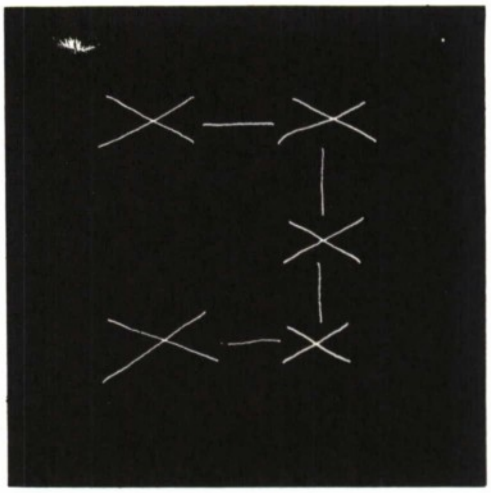
(a) (b)

Fig. 7(a-b). Pictures of data defining animation of cartoon clip.

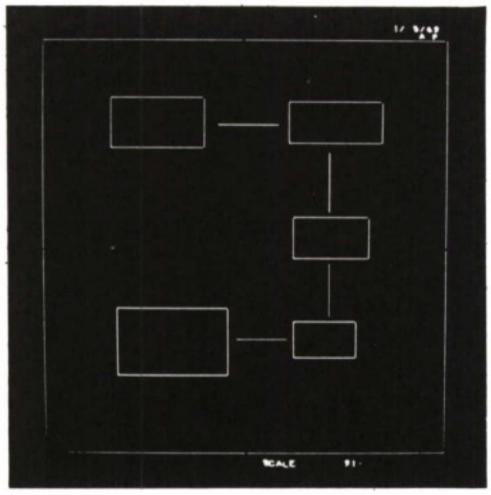


```
*INKING   CHARACTER   200
 INKING   CHARACTER   203
 INKING   CHARACTER   200
 INKING   CHARACTER   200
 INKING   CHARACTER   200
 INKING   CHARACTER   200
 INKING   CHARACTER   204
 INKING   CHARACTER   204
 INKING   CHARACTER   203
 INKING   CHARACTER   AP
END
```

(a)



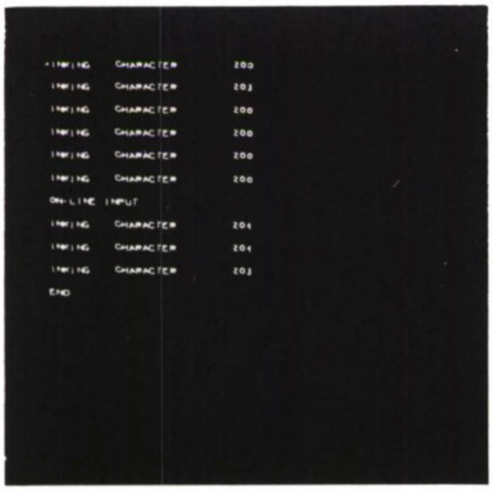
(b)



(c)

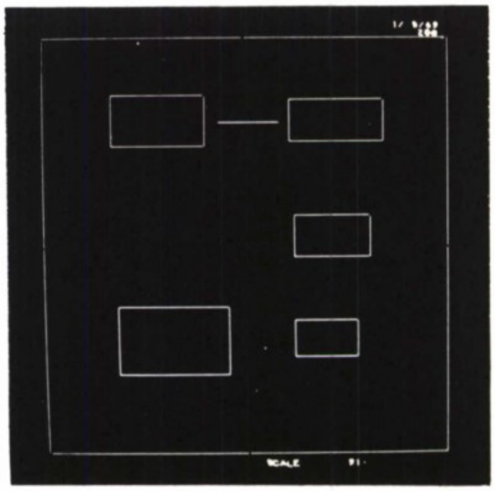
Fig. 8. (a) Typical stored input sequence; (b) display of each of inputs superimposed; and (c) effect of inputs in a particular program environment.

-2-8793



```
•INKING    CHARACTER        200
 INKING    CHARACTER        203
 INKING    CHARACTER        200
 INKING    CHARACTER        200
 INKING    CHARACTER        200
 INKING    CHARACTER        200
ON-LINE  INPUT
 INKING    CHARACTER        204
 INKING    CHARACTER        204
 INKING    CHARACTER        203
END
```

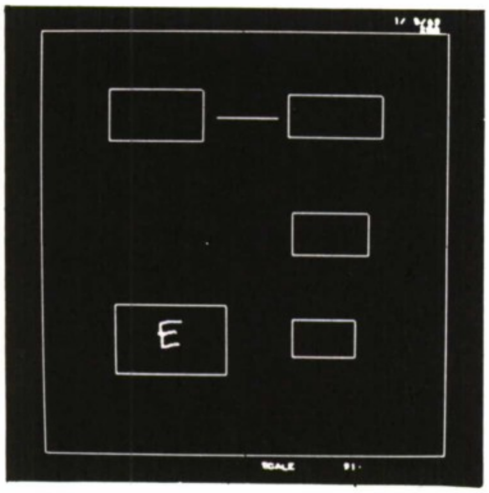
(a)



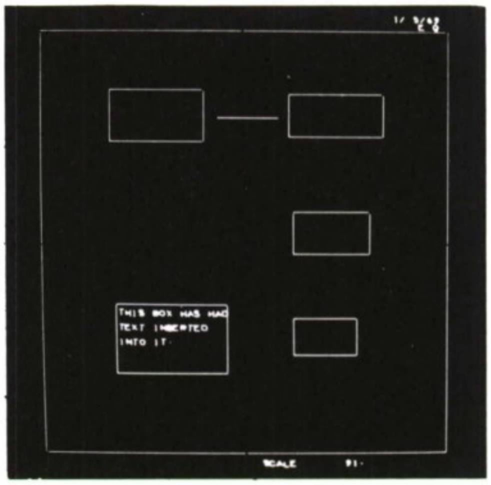
(b)

Fig. 9. (a) Input sequence of Fig. 8(a) modified to include an on-line input, and (b) replayed until occurrence of on-line input.

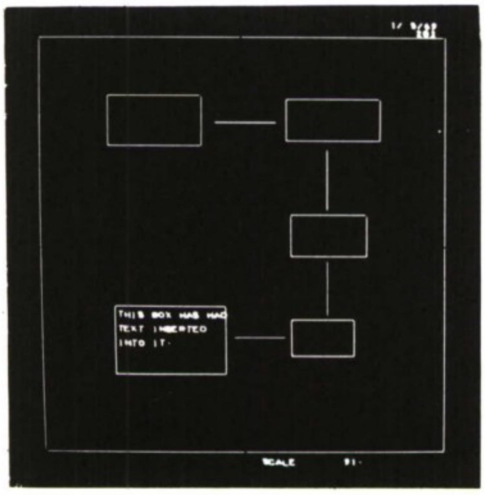
-2-8794



(a)



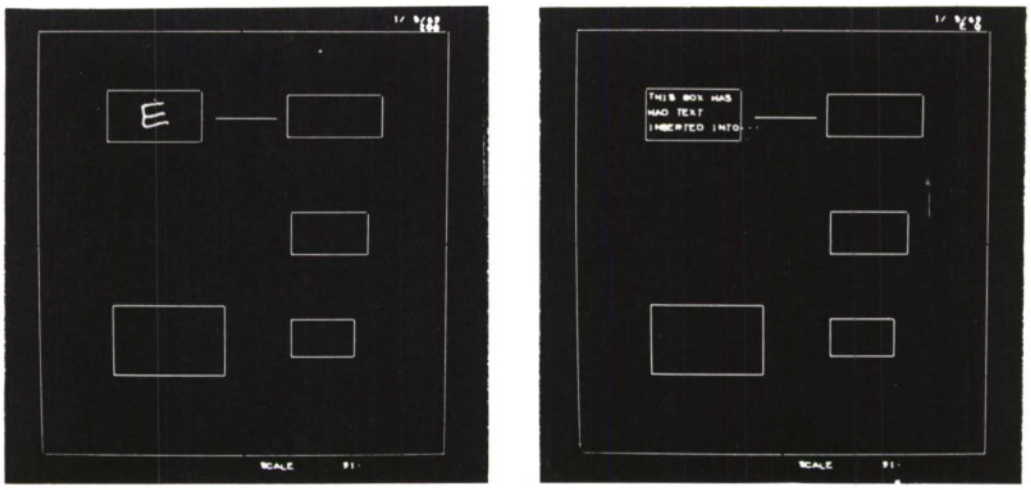
(b)



(c)

Fig. 10. (a) Inking character E used as an on-line input to select box into which text is to be inserted, (b-c) then rest of input sequence is replayed.

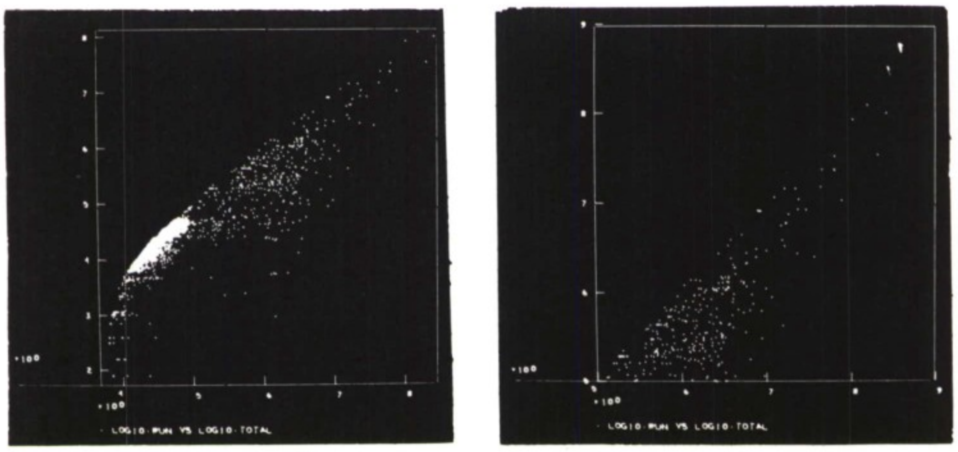
-2-8795



(a) (b)

Fig. 11(a-b). A different box is selected with on-line input.

-2-8796



(a) (b)

Fig. 12(a-b). Response performance of APEX system showing plots of job cpu time (vertical scale) vs total job time (horizontal scale).

DOCUMENT CONTROL DATA – R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)* Lincoln Laboratory, M.I.T.	2a. REPORT SECURITY CLASSIFICATION Unclassified 2b. GROUP None

3. REPORT TITLE

Semiannual Technical Summary Report to the Advanced Research Projects Agency on Graphics

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

Semiannual Technical Summary Report (1 December 1968 through 31 May 1969)

5. AUTHOR(S) *(Last name, first name, initial)*

Raffel, Jack I.

6. REPORT DATE 31 May 1969	7a. TOTAL NO. OF PAGES 28	7b. NO. OF REFS 7

8a. CONTRACT OR GRANT NO. AF 19 (628)-5167	9a. ORIGINATOR'S REPORT NUMBER(S) Semiannual Technical Summary for 31 May 1969
b. PROJECT NO. ARPA Order 691	
c.	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)* ESD-TR-69-141
d.	

10. AVAILABILITY/LIMITATION NOTICES

This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES None	12. SPONSORING MILITARY ACTIVITY Advanced Research Projects Agency, Department of Defense

13. ABSTRACT

A relocating assembler and a linking loader have been programmed for the TX-2. To provide more convenient graphical facilities, a new Extended Graphic Subsystem has been written. Recent display hardware additions to TX-2 have included a color display, storage tubes, and a commercial ARDS console.

Application program work on semiconductor mask design has continued, with current work at an 80 gate/chip level. Initial exploratory work on a design data system has been studied. Application of the LX-1 microprocessor to vocoder and display control problems has been studied. Experimental systems for Interactive Computer-Mediated Animation and Real-Time Input saving have been completed. Performance measurements on the APEX system are being conducted and the data analyzed. A stand-alone circuit-testing terminal with a small computer is under development.

14. KEY WORDS

graphical communication	time-sharing	display systems
TX-2	man-machine	programming languages