EXPERIMENTAL TELEVISION CENTER LTD.
164 COURT ST.
BINGHAMTON NEW YORK 13901
607-723-9509

# Computer-Based Video Synthesizer System

This report summarizes the results of the research and development of a computer controlled video image processing and synthesizing device, a project supported in part by the National Endowment for the Arts and the New York State Council on the Arts.

Dr. Donald McArthur, Systems Design
Mr. Walter Wright, Software Development
Mr. Richard Brewster, Systems Construction

October 1977

The Experimental Television Center Ltd. is a not-for-profit educational corporation supported in part by the New York State Council on the Arts and the National Endowment for the Arts.

EXPERIMENTAL TELEVISION CENTER LTD.
164 COURT ST.
BINGHAMTON NEW YORK 13901
607-723-9509

As video image generating, processing and synthesizing systems become increasingly sophisticated, the problem of achieving maximum control over these systems must be addressed. This is particularly relevant to situations in which videomakers work independently and frequently as individuals in the creation of their works. Many video systems provide such a large number of image making variables that manual adjustments within the parameters of each control necessary to obtain desired structures and sequences is not always possible. The artist must then compromise the image to the system. Video synthesizer systems offer an enormous potential for intricate image constructions, but without appropriate control systems the individual artist may not be able to take full advantage of the system to achieve with accuracy the structures desired. Microprocessor systems, it was believed, could provide the necessary control for precision of image structuring if these computer systems could be completely dedicated to the processes of visual art making and be made usable by artists in direct ways.

The primary aim of the Computer-Based Video Synthesizer project was the research and development of such systems with capacity for direct use by artists in the production of independent works. A major design consideration in both hardware and software development concerned the establishment of a holistic system directly related to the requirements of individuals working in the electronic arts and usable by them in their personal work. It was considered important to reduce, as much as possible, the reliance by artists on outside technical support in the production processes because of the difficulties of communication and interpretation and the intimacy of the creative situation. The initial planning for this project began in 1975 with conferences at the Experimental Television Center involving Mr. Ralph Hocking, Mr. Walter Wright, Dr. Donald McArthur and Mr. Richard Brewster of Binghamton, New York and Steina and Woody Vasulka of Buffalo, New York. With support from the National Endowment for the Arts and the New York State Council on the Arts, the project was completed in the Fall of 1977; the resulting system is now operational at the Experimental Television Center and is available for use by artists through the production program at the Center.

An important philosophical consideration throughout the project concerned a humanistic approach to the design and utilization of computer and video systems technologies; one of the initial stages of this project involved the development of methodologies which would guide the construction of complex tools and systems dedicated to the needs of visual artists. To achieve this end, it was essential that artists, programmers and engineers work together in all aspects of the project, each group communicating from its own unique perspective. Artists helped to articulate and define the types of controls which they felt were important in image making. Engineers and programmers frequently introduced image making devices and control methods which had not previously been available; the structural and compositional potentials of these components were completely unexplored. In the design and construction of the hardware there were a number of specific objectives. A flexible and versatile system was important in order

to provide artists with as many options as possible for image generation, processing and control. The present system is modular in design and permits the inclusion or exclusion of discrete components in the assembly of a system specifically tailored to meet the individual requirements of a particular artist or project. Modular and standardized design also allows for the future research and development of new components and the modification of existing hardware and software all of which can be incorporated into the present system with a minimum of effort. The system is capable of interfacing with many video and computer components an increasing number of which are owned by or accessible to individual artists and small arts organizations. For example, the system at the Center is compatible with the system of Steina and Woody Vasulka, and exchanges of software and hardware are possible between Buffalo and Binghamton. Video production requires a fairly powerful microprocessor system which is capable of efficiently handling the large amount of information necessary in the generation and control of image structures. The needs for a powerful system, also low in cost, had to be weighed against the factor of complexity since the system was to be used by individual artists the majority of whom had little or no prior experience with computer hardware or software. The 16 bit system as it was designed and constructed met the criteria of low cost, high power and ease of operation. The hardware made use of commercially available components as much as possible in the interests of efficiency of operation and construction and ease of duplication of the system by artists and arts groups. Many specific components and interfaces, however, had to be designed and constructed specifically for this project since they were either not available commercially or were too costly; many of the commercial components which were available did not meet the specific requirements determined by the nature of the application of the system.

The software development for this project also emphasized a humanistic approach to the use of microprocessor and video systems by artists. The goal of the software research was the development of an interactive language usable by artists. This language had to be understandable to artists so that they could address the computer directly, using language and concepts derived from the visual arts, without the necessity of translation into high level computer languages. The language had to be responsive to the needs of artists, enabling them to manipulate discrete elements of design and compositional structures. Further, it had to allow the artist to intervene at any point in the construction of the composition so details of compositional configurations as well as whole sequences could be easily altered. Precision was felt to be critical; the artist had to be able to develop and score the composition, store, run and edit it in a manner which insured its accuracy and repeatability. It was felt that the language should also provide for the option of programmed randomness and operate in either structured or random modes or a predetermined combination of both modes.

The computer-based video synthesizer system which is now operational at the Center consists of two sub-systems, the microprocessor and the video system and their interface. The computer section consists of a 16 bit DEC LSI-11 microprocessor, teletype and printer, dual floppy disk and 20K of memory. Components designed and constructed specifically for this project include the parallel interface, buffer memory, module to element bus, element bus, digital to analog

converters, analog to digital converters and real time input. The video system includes a four channel analog colorizer with keyers, a 50 point switching matrix, spatial and intensity digitizer and a voltage control bank. The video system is modular in design; each of these components was researched, designed and constructed over a period of four years under the research and development program at the Center. Each of the video components may be combined with any other to form a system tailored to individual requirements; the video system may be operated manually or placed under computer control. This design consideration allows a maximum flexibility with a limited amount of equipment, permitting the same components to serve a variety of artists with different systems needs and experiences. Hardware design also permits manual interruption of computer processes at any point through the use of analog to digital converters and real time input. This feature allows the artist more complete control over all elements of the image and its temporal structures. Changes in composition may occur by software reprogramming or direct manual interactions by the artist or a combination of these techniques.

A more detailed description of the hardware aspect of this project is presented in the papers by Dr. Donald McArthur. Section I A provides an orientation to the system architecture. Section I B is a paper written from a transcript of a presentation by McArthur in Buffalo, New York in March 1977 for the 'Design/ Electronic Arts' conference supported by the National Endowment for the Arts and the New York State Council on the Arts and sponsored by Media Study/Buffalo and the Center for Media Study, State University of New York at Buffalo. This presentation by McArthur was based directly on the research McArthur had done for the Computer-Based Video Synthesizer project.

The aim of the software aspect of the project was the development of an interactive language which uses concepts and vocabulary derived from the visual arts. It was anticipated that this approach would make the computer based video system accessible to a much larger number of artists than would a system which depends on the presence of a programmer to interpret the ideas and images of an artist into a computer language. Before any except the most rudimentary of programming could be developed, analysis of the fundamental elements in the composition of single images as well as their temporal structures had to be conducted. Identification and definition of these elements and the parameters of change within each variable were the initial steps. Within single images, discounting the time function, elements which were chosen included color field variables such as hue, saturation, chroma and intensity, form and shape variables including type of shape, position and frequency, texture and density. Each element has parameters of change which involve the temporal aspects of video. The methods of change involve problems of duration and sequencing with references to rhythmic structures.

As is noted in the papers by Wright, the software research is still in its initial stages and further explorations are necessary before the interactive language is fully functional. Several programs have been developed, one of which is analyzed in Wright's paper, section II A, which represents an incomplete stage of the

language. Section II B is a transcript of a presentation by Wright at the 'Design/ Electronic Arts' conference in Buffalo in 1977; these materials are based directly on the research Wright had done for this project.

The computer based video synthesizer system is now operational at the Experimental Television Center in Binghamton, New York and is available to artists under the production program. As a greater number of artists utilize this system, each artist will be encouraged to articulate ways in which the system can be made more responsive. The results of this project have already indicated several important avenues for continued research, among them further and continued software development and the publication and dissemination of the results of the research to date. The computer based video system can serve as a model system; publication of research results will allow the duplication and modification of the system by other individual artists and arts organizations. Although the research to date has been specific to video, microprocessor systems are useful tools in many of the visual and performing arts, and a publication of this nature would assist many individuals from a variety of fields. A complete set of documentation has already been prepared; the next phase of this project, for which the Center is seeking support, includes the publication of these materials, including detailed schematic documentation. This publication will also include more theoretical papers, approaching the system and its applications from the points of view of aesthetics, physics, electronics and video and microprocessor technology. The aim of this publication is to provide specific and detailed information to permit duplication of the system and also introduce conceptual frameworks from which to view the electronic arts.

EXPERIMENTAL TELEVISION CENTER LTD.
164 COURT ST.
BINGHAMTON NEW YORK 13901
607-723-9509

# A Computer-Based Video Synthesizer System

## An Orientation to Hardware Architecture

Dr. Donald McArthur

Experimental Television Center Ltd.
Binghamton, New York

July 1977

This project is supported in part by the National Endowment for the Arts and the New York State Council on the Arts.

# NEA Report

## I. Introduction

As science advances, with the resulting advances in technology, we have new tools and new capabilities which influence our world in many ways. This new technology not only influences the traditional art forms but also produces new forms of art. The development of high speed electronic components and circuits, the cathode ray tube, the video camera, and inexpensive video tape recorders enabled the development of video art. The development of small but powerful computers now allows systems to be developed which can give the video artist a new dimension of control over the video image. With a computer-based video synthesizer (CBVS), one can generate a sequence of images while controlling each individual image with detail and precision that is many orders of magnitude greater than is possible with manual control.

The ability to control the dynamics of the image is useful to the artist only if the system is capable of generating the image in real time. With this requirement in mind, the natural choice of devices for converting electrical signals to visual images is the conventional video system. This choice also gives the capability of recording the video compositions with a conventional video tape recorder and of broadcasting to a large audience through existing network systems.

There are basically two modes of operation of the system: interactive compositional mode and automatic production mode. In the compositional mode, the artist can enter programs and parameters through the keyboard, observe the resulting sequence of images, and then modify parameters through either the keyboard or a real time input and thus build up a data set for a complete piece. At each stage of the composition process the data set, representing all the aesthetic decisions made by the artist, is stored in the computer. When the composition is finished the system will operate in the automatic production mode generating the final video signal in real time with no intervention by the artist. The artist may also choose to use a combination of these two modes in an interactive performance or allow an audience to interact with the system operating automatically. The system is structured so that all of these variations can be accomodated by appropriate programming.

The system may be operated as a generating synthesizer which produces a video signal entirely from internal signals or as a processing synthesizer which utilizes signals of external origin such as a video camera. Either of these two types of operations is carried out by a configuration of element modules, each of which performs a class of functions, with the specific function during one frame being determined by the control parameters received from the computer.

Since the computer functions only to generate the parameters which govern the behavior of the synthesizer modules, a video signal will be generated without operation of the computer. The system will simply repeat the frame until the parameters are changed. Thus the artist may choose to stop the computer in which case he is able to examine a single frame, or he may alter the program so that a given sequence is displayed very slowly or repeated very rapidly.

## II. System Structure

The CBVS consists of two parts: the computer section shown in the lower section of figure 1 and the video section shown in the upper section of the figure. Both sections operate simultaneously and independently, communicating through the buffer memory which has a capacity of 1,024 = $2^{10}$ 16 bit words. Each of these words is either a picture element, a number which controls some function of the video section and determines some aspect of one field of the video image, or it is a picture feature, a number determined by the video section and may depend on an external signal such as a video camera signal. The buffer memory is connected to the computer bus through a 16 bit parallel interface which is structured in such a way that each word in the buffer memory is addressable and may be read or written in exactly the same way as words in the main computer memory. This memory-mapped I/O system simplifies the software which controls the buffer memory. In order to update an element such as a control D/A, the computer must execute an instruction which stores the new value in the location corresponding to that element.

During the active scan time, the control computer reads features from the buffer memory and generates elements for the following field and stores them in the buffer memory. During the vertical blanking interval, information is transferred through the element bus from the buffer memory to the element modules or from the feature modules to the buffer memory. The designation of a particular area of the buffer memory as an element or feature is under program control. During the transfer between the buffer memory and the element bus, the computer is locked out of the buffer memory. On completion of the transfer, the interface generates a vectored interrupt which requests the computer to generate parameters for the next field.

The computer system consists of: a DEC LSI-11 microprocessor which has a 16 bit word length and an instruction execution time of about 7 microseconds; Teletype Keyboard and printer connected through a serial interface; 20 K of dynamic memory; a dual drive floppy disk system with a capacity of 256,256 bytes per diskette. An additional serial interface is also available for connecting through a modem to other computer systems. The entire system is dedicated to the synthesizer system.

The overall timing is determined by a 9.7552434 MHz clock which is phase locked to the subcarrier (3.579545 MHz). This frequency is chosen to insure a coherent subcarrier and to divide the active portion of the scan line into 512 pixels. The red, green, and blue signals are generated independently, and the chroma encoding is done with analog circuits; thus there is no advantage to following the common practice of making the pixel rate an integer multiple of the subcarrier frequency. With this clock frequency, a full nine bit word is used to define the horizontal position on the active portion of the raster. Figure 2 shows the X and Y wave forms. The X-Y module generates twenty bits of timing information (ten bits for horizontal, including the blanking period, and ten for the line count). This module also generates sync, drive, burst flag, and the transfer request $\overline{TR}$ signal which controls the timing of the buffer memory.

Timing details of the interface and buffer memory are shown in figure 3. The transfer request $\overline{TR}$ goes low at the beginning of vertical blanking initiating an arbitration for access to the buffer memory. If the computer is accessing the buffer memory, the current bus cycle is completed, then $\overline{READY}$ goes high, and the buffer memory controller cycles through memory making the required element and feature transfers. When this is completed, $\overline{READY}$ goes low, control of the memory is returned to the computer, and an interrupt is generated requesting data for the following field. As indicated in the diagram, during the Nth field, the computer is generating data for the N+1th field.

The timings of the signals on the element bus are indicated in figure 4. During the transfer, the memory controller generates: the addresses $A_{\overline{CME}}$ -$A_9$ ; the clock signals $\overline{\phi_1}$, $\overline{\phi_2}$, and $\overline{\phi_3}$ ; and the status signals $\overline{CME}$ indicating a transfer from the memory to an element and $\overline{CFM}$ indicating a transfer from a feature module to the memory. The signals $\overline{ETF}$ and $\overline{FTE}$ are generated by the synthesizer modules and initiate a controller Element/Feature mode change. The three phase clock system is used to control modules which have the structure shown in figure 5. Functions which use data from the computer during the vertical blanking interval are disabled when the buffer memory accesses that particular element by a signal generated using $\overline{\phi_1}$. This allows access to the buffer memory during $\overline{\phi_2}$. The third clock, $\overline{\phi_3}$ generates a memory write signal.

Time delays in the digital processing modules could produce errors and shifts of the image to the right. This is prevented by deskewing the output of each with a latch clocked by the master clock (9.755 MHz.). Compensation for the resulting 102.5 nSec. delay in each module is provided by starting the X count at the beginning of the horizontal blanking interval rather than at the end. An additional shift to the right or left is then achieved by adding (mod 512) a constant supplied by the computer. The default value of this constant is 404 + number of elements.

## III. Element and Feature Modules

The structure described above supports a variety of element and feature modules which may be chosen and configured according to the tastes of the artist. Our experience indicates that a large amount of work can be produced with a relatively small number of elements in a standard configuration. Whenever possible, a new element added to the system is configured in such a way that if the control word is set equal to zero it has no effect on the system. Thus a minimum amount of reprogramming is required following system expansion.

Two general classes of modules have been developed: digital and hybrid. The hybrid elements are: high-speed D/A converters used for generating the red, green, and blue video signals which are converted to NTSC format in the standard way; low-speed D/A converters used for generating control voltages, field-by-field controllable, used to operate existing voltage controlled analog image processing systems such as keyers, raster manipulaters, etc. Another hybrid element is the analog video switching matrix. Four bits of one control word are used to select one of sixteen inputs for one output.

Digital processing elements include: constant; X + constant; Y + constant; twelve-channel sixteen-line demultiplexer with output complement; and four-channel four-bit by sixty-four word memory.

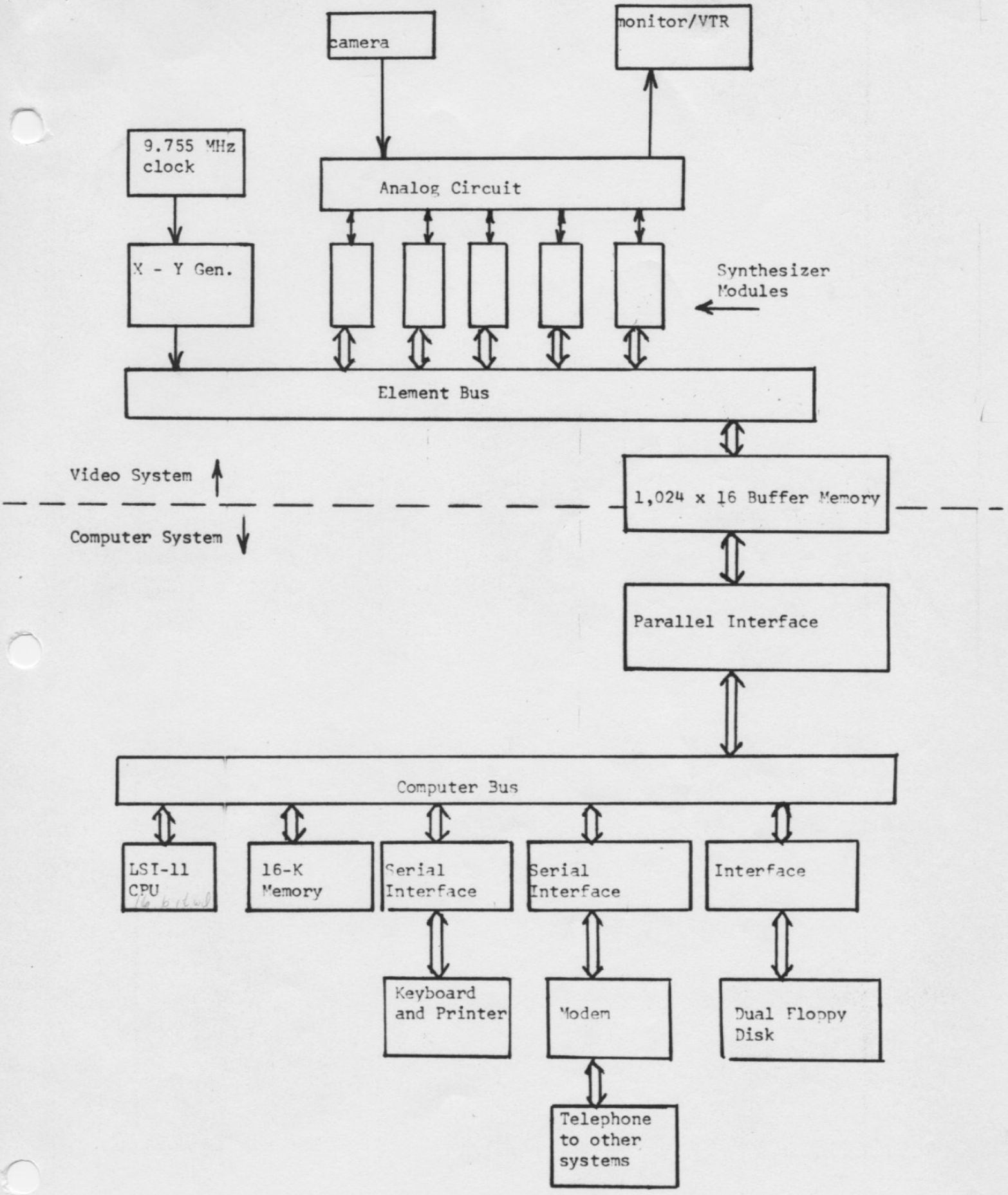

camera
monitor/VTR
9.755 MHz clock
Analog Circuit
X - Y Gen.
Synthesizer Modules
Element Bus
Video System
Computer System
1,024 x 16 Buffer Memory
Parallel Interface
Computer Bus
LSI-11 CPU
16-K Memory
Serial Interface
Serial Interface
Interface
Keyboard and Printer
Modem
Dual Floppy Disk
Telephone to other systems

Figure 1

## X and Y Half Cycle Durations and Wave Forms

$x_1$ 102.5 nSec.

$x_2$ 205 nSec.

$x_3$ 410 nSec.

$x_4$ 820 nSec.

$x_5$ 1.64 $\mu$Sec.

$x_6$ 3.28 $\mu$Sec.

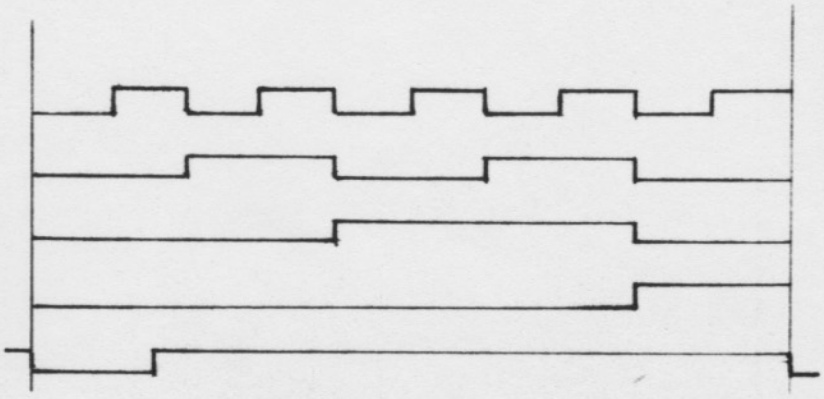
$x_7$ 6.56 $\mu$Sec.

$x_8$ 13.12 $\mu$Sec.

$x_9$ 26.24 $\mu$Sec.

$x_{10}$ 52.48 $\mu$Sec.

Horizontal Blanking



$y_1$ 16.66 mSec.

Field Index

$y_2$ 63.5 $\mu$Sec.

$y_3$ 127 $\mu$Sec.

$y_4$ 254 $\mu$Sec.

$y_5$ 508 $\mu$Sec.

$y_6$ 1.01 mSec.

$y_7$ 2.03 mSec.

$y_8$ 4.06 mSec.

$y_9$ 8.13 mSec.

$y_{10}$ 16.26 mSec.

Vertical Blanking

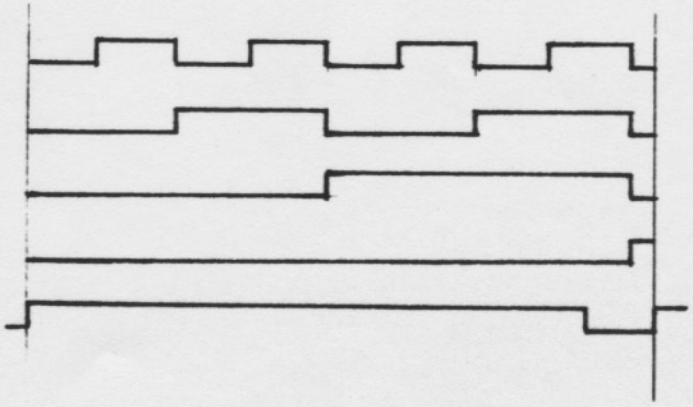
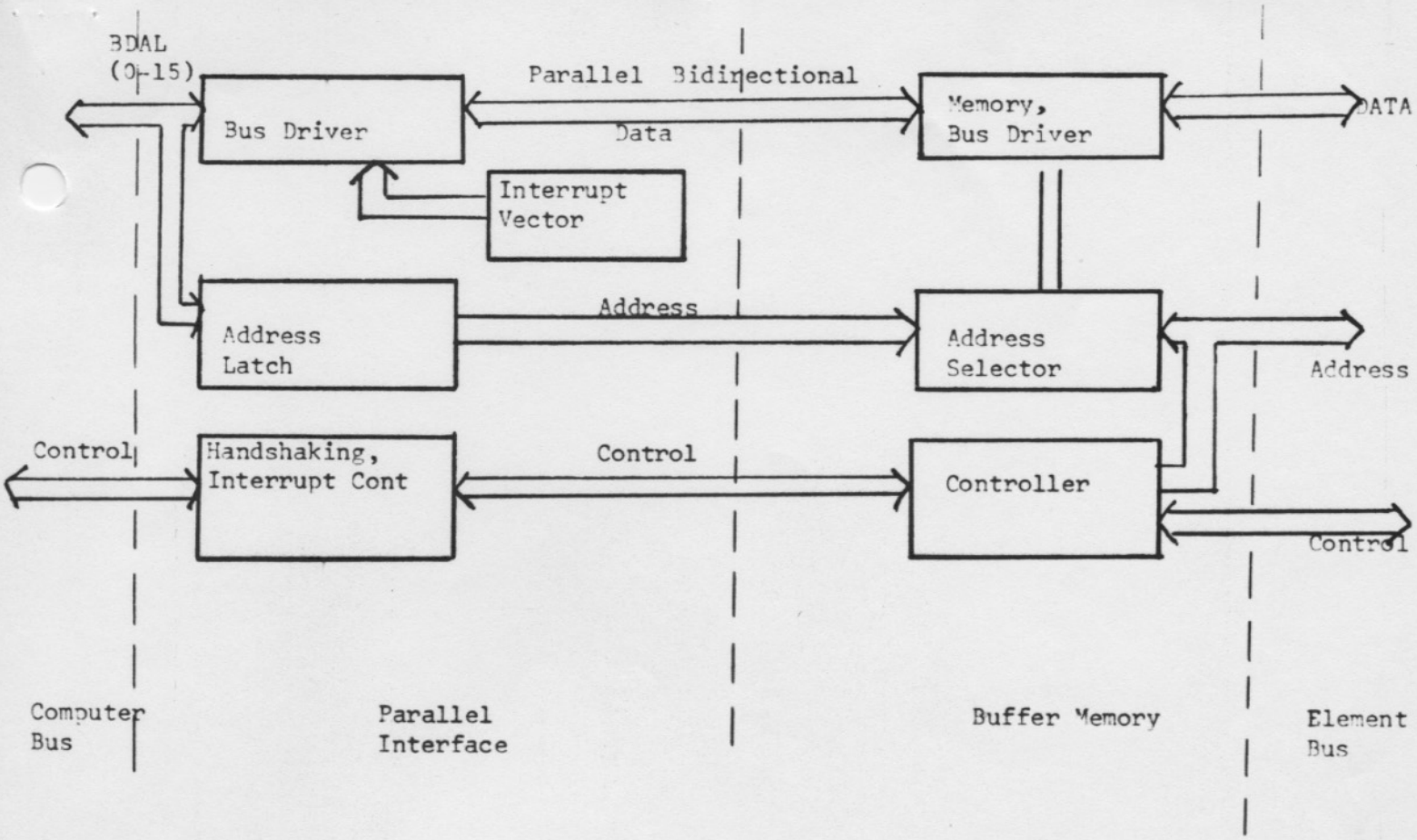


Figure 2


BDAL
(0-15)
Bus Driver
Parallel Bidirectional
Data
Memory,
Bus Driver
DATA
Interrupt
Vector
Address
Latch
Address
Address
Selector
Address
Control
Handshaking,
Interrupt Cont
Control
Controller
Control
Computer
Bus
Parallel
Interface
Buffer Memory
Element
Bus

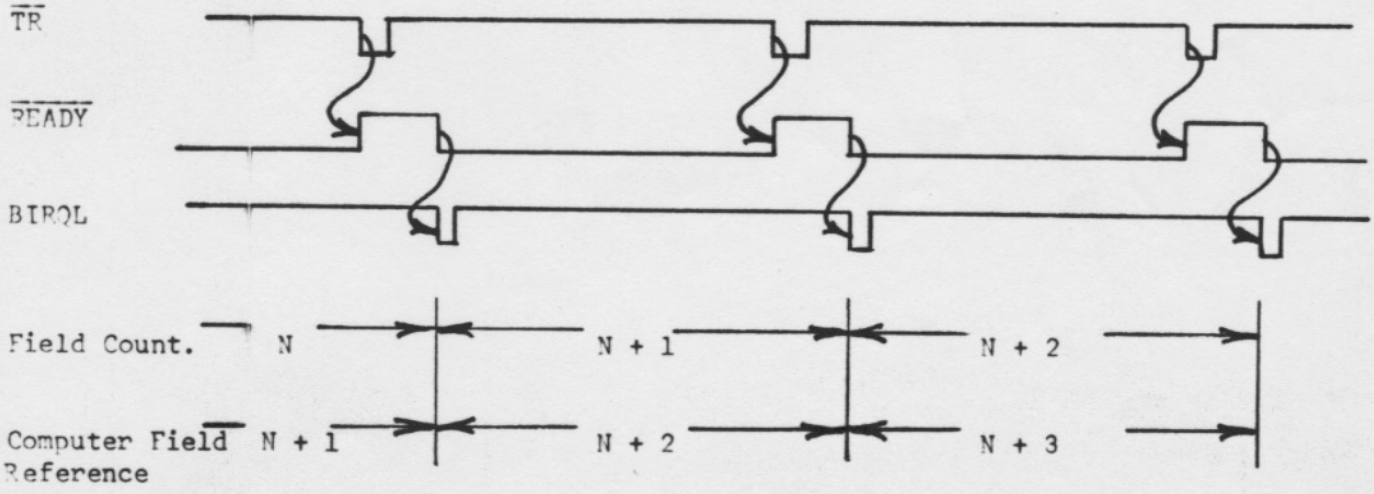

TR
READY
BIRQL
Field Count.
N
N + 1
N + 2
Computer Field
Reference
N + 1
N + 2
N + 3

Figure 3

Element Bus

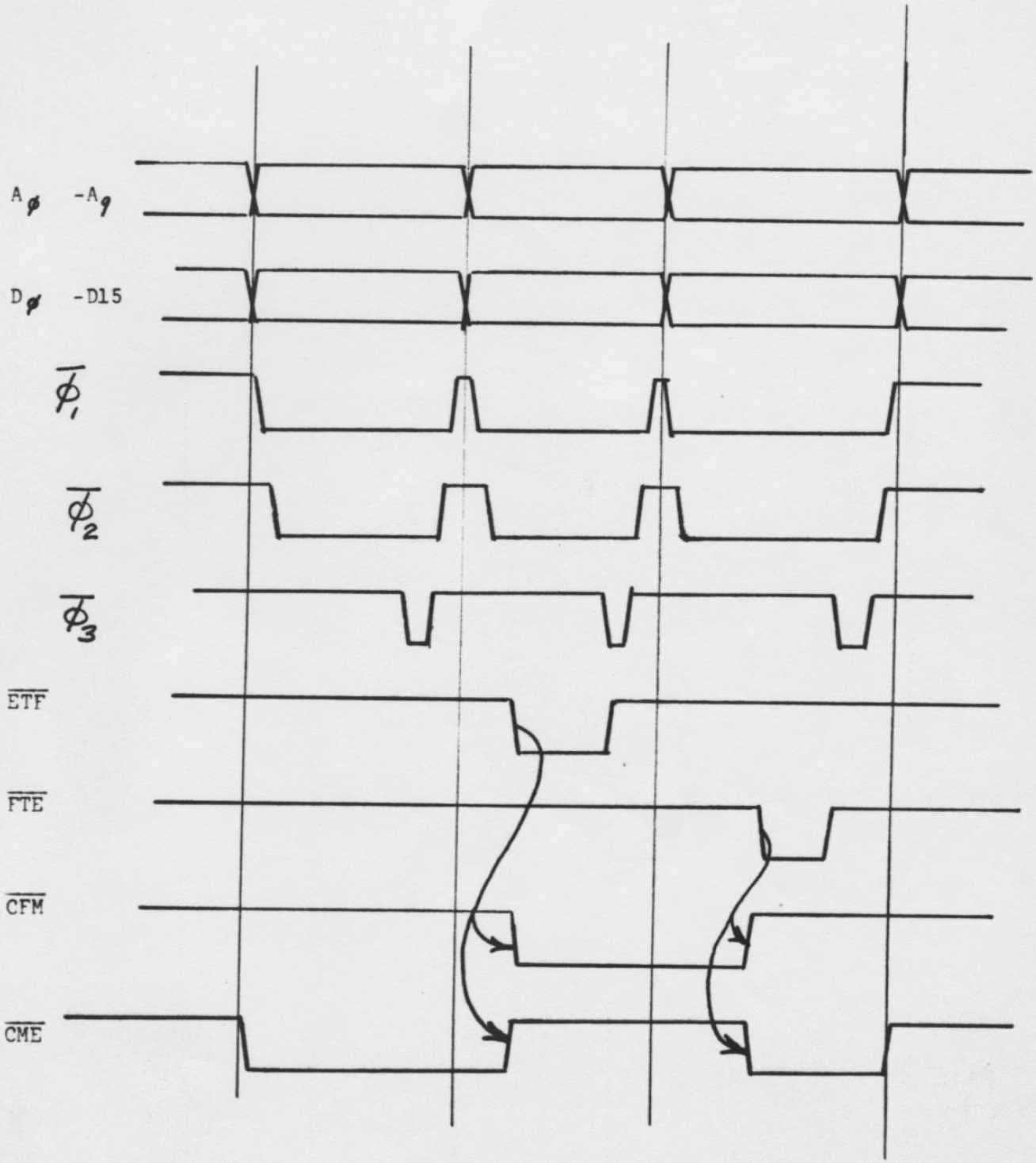



Figure 4

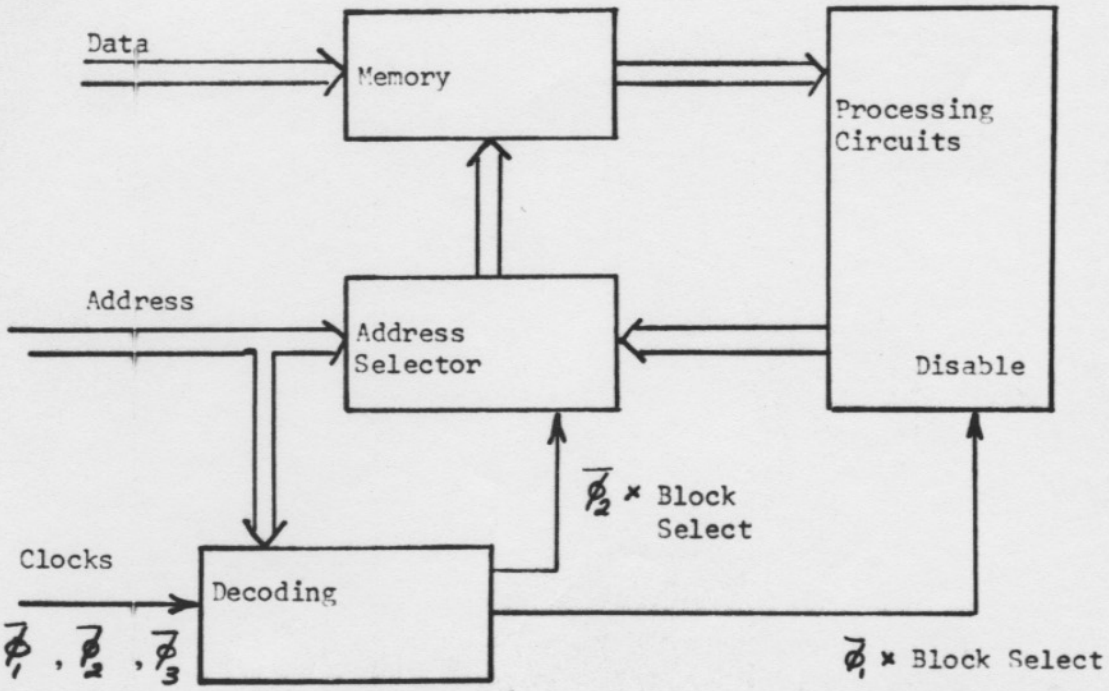

Data
Memory
Processing
Circuits
Address
Address
Selector
Disable
$\overline{\phi_2}$ × Block
Select
Clocks
Decoding
$\overline{\phi_1}$ , $\overline{\phi_2}$ , $\overline{\phi_3}$
$\overline{\phi_1}$ × Block Select

Figure 5

EXPERIMENTAL TELEVISION CENTER LTD.
164 COURT ST.
BINGHAMTON NEW YORK 13901
607-723-9509

# A Computer-Based Video Synthesizer System

A paper presented at the 'Design/Electronic Arts' conference in March 1977.

Dr. Donald McArthur

Experimental Television Center Ltd.
Binghamton, New York

This project is supported in part by the National Endowment for the Arts and the New York State Council on the Arts.

# I. Introduction

As science advances, with the resulting advances in technology, we have new tools and new capabilities which influence our world in many ways. This new technology not only influences the traditional art forms but also produces new forms of art. The development of high speed electronic components and circuits, the cathode ray tube, the video camera, and inexpensive video tape recorders enabled the development of video art. Advances in integrated circuit design and fabrication techniques have led to the development of small but powerful computer systems which can be utilized by the video artist to achieve a new dimension of control over the video image. With a computer-based video synthesizer (CBVS), one can generate a sequence of images while controlling each individual image with detail and precision that is many orders of magnitude greater than is possible with manual control.

The ability to control the dynamics of the image is especially useful to the artist if the system is capable of generating the image in real time. With this requirement in mind, the natural choice of devices for converting electrical signals to visual images is the conventional video system. This choice also gives the capability of recording the video compositions with a conventional video tape recorder and of broadcasting to a large audience through existing network systems.

There are basically two modes of operation of the system: interactive-compositional mode and automatic-production mode. In the compositional mode, the artist can enter programs and parameters through the

keyboard, observe the resulting sequence of images, and then modify parameters through either the keyboard or a real time input and thus build up a data set for a complete piece. The data set, representing all the aesthetic decisions made by the artist, is stored in the computer at each stage of the composition. When the composition is finished the system will operate in the automatic-production mode generating the final video signal in real time with no intervention by the artist. The artist may also choose to use a combination of these two modes in an interactive performance or to allow an audience to interact with the system operating automatically. The system is structured so that all of these variations can be accomodated by appropriate programming.

The system may be operated as a generating synthesizer which produces a video signal entirely from internal signals or as a processing synthesizer which utilizes video signals of external origin such as a camera. Either of these two types of operations is carried out by a configuration of elements modules, each of which performs a class of functions, with the specific function during one frame being determined by the control parameters recieved from the computer.

Since the computer functions only to generate the parameters which govern the behavior of the synthesizer modules, a video signal will be generated without operation of the computer. If the computer is stopped, the system will simply repeat the current frame until the parameters are changed. Thus the artist may choose to stop the computer and examine a single frame, or he may alter the program so that a given sequence is displayed very slowly or repeated very rapidly.

## II. General Design Considerations

The NTSC video format is shown in figure 1. The time interval represented by one line is the time of one horizontal sweep or 1/15,734 sec. = 63.5 microseconds. The number by each line indicates the number of times that each format is repeated. There are 483 active scan lines and forty-two lines of vertical blanking making a total of 525 lines to compose one frame (two fields). The first line (a) consists of the horizontal sync pulse, which is five microseconds long occuring during a 11 microsecond blanking interval, followed by color burst and the picture information which last 52.6 microseconds. The next line (b) shows the first equalization pulse which is followed by five more and the beginning of vertical sync (lines c,d). Vertical sync starts in the middle of the line for this field and at the beginning of the line for the next field (line m). Lines e,f,g, and h show the completion of vertical sync with serrations and six more equalization pulses. Vertical blanking is shown in lines i and j. The second field starts in line j. Line k shows the 241 lines which interlace with the first field to complete the frame. Finally there are 6 more equalization pulses before (line l) and after (line n) vertical blanking (line m). Line o shows twelve more lines of vertical blanking. This sequence is repeated 30 times per second making a total of 108,000 frames per hour.

The portion of the video signal which carries intensity information is a continuously varying voltage and may be analyzed into components of different frequencies: low frequency components correspond to course structures in the image, and high frequency components correspond to fine

structures. Although the video signal may, in general, have an arbitrary shape, a monitor will reproduce an image which corresponds to only a limited frequency range. This range is called the bandwidth. Thus components of the video signal which are outside the bandwidth of the monitor will not be visible. In a system any device which limits the bandwidth of the video signal will degrade the fine structure or spatial resolution of the image.

A video signal may be represented by a finite set of samples. These samples may be stored, eg. in a TBC or frame buffer, and then used to reproduce the video signal. In this process some of the information may be lost, but the lost information corresponds to high frequency components or fine structure of the image. If the sampling rate is sufficiently high, the lost information will be outside the bandwidth of the monitor, and no impairment of the image will occur. The minimum theoretical sampling rate which will retain the video information within a given bandwidth is called the Nyquist rate and is equal to twice the maximum frequency. Thus for broadcast quality, a minimum of 4.2 MHz. x 2 x 52 $\mu$Sec. = 437 samples per line are required.

Each of the samples may be represented by a binary number with B digits. In general, a binary number with B digits has $2^B$ discrete values, and a B-digit representation corresponds to $2^B$ discrete gray levels. Since the video signal may have any one of a continuous set of values, an error is introduced when it is represented by a discrete number. Figure 2 shows the correspondence between binary and decimal numbers, the correspondence between a continuous video signal and its discrete representation, and the resulting error. This error is called the quantization error, and its root-mean-square

value is given by $\sqrt{\int_0^1 \left(\frac{1}{2} - \frac{1}{2^B} t\right)^2 dt} = \frac{1}{\sqrt{12}} \frac{1}{2^B}$.

If B is sufficiently large, the spacing between the discrete gray levels will be small compared to the intrinsic noise of the system and will not be visible on a monitor. The minimum acceptable value of B may be estimated by considering the quantization signal-to-noise ratio which is given by

$$S/N = 20 \log_{10} \left\{ \sqrt{12}\ 2^B \right\} = (6.02\ B + 10.79)\ \text{db}.$$

The signal-to-noise ratio for a one-half inch VTR is about 40 db, so about five binary digits are required for a comparable quality (monochrome) image.

For the analysis of equipment requirements for generating a video signal, we may utilize the measure of information given by the mathematical theory of communication. If a message occurs with probability P, then the amount of information (measured in bits) is

$$I = \log_2 (1/P).$$

Thus, the answer to a question which can be answered by yes or no with equal probability carries the quantity of information of

$$\log_2 1/(1/2) = \log_2 2 = 1 \text{ bit}.$$

Since a binary digit has two possible values, zero and one, each may carry at most one bit of information. Thus there are 5 x 437 x 483 = 1,055,355 bits of information in one frame of monochrome video signal (assuming thirty-two gray levels and 4.2 MHz. bandwidth). For generating a color signal, one may represent each of the red, green, and blue primary signals with a five bit word specified at time intervals of 100 nSec; this corresponds to an information rate of 150,000,000 bits per second. Other examples of amounts

of information are typewriter keystroke, about six bits, and one typewritten page (double spaced), 9,000 bits. The rate at which the human brain can process information has been estimated to be about forty bits per second.

The mathematical theory of communication introduces another concept, redundancy, which is useful in analyzing a video synthesizer system. A message which contains N bits of information may be coded in a way that uses more than N binary digits. For example, the message pair (yes/no) may be coded as (111/000) using three binary digits instead of one per message. In this case the coding is redundant. The pattern shown in figure 3 consists of 16 x 16 = 256 squares, each of which is either all black or all white. Thus by representing white by one and black by zero, any pattern of this format can be represented by a code consisting of 256 binary digits. If all possible patterns are allowed, then one pattern carries 256 bits of information. A circuit which generates a video signal corresponding to this pattern codes the message into a form with 1,055,355 binary digits (assuming the spatial and intensity resolution as above). The redundancy of the video signal is further increased if the set of possible patterns is restricted by requiring that the total pattern is built of sixteen 4 x 4 blocks each being one of the following: all black, all white, black and white as in the upper left hand corner, or the latter with black and white interchanged. Then two binary digits code the choice for each block, and since there are sixteen blocks, thirty-two bits of information are contained in the entire pattern.

The disparity between the rate at which the human brain can process information, which limits the rate that an artist can manipulate controls of a video synthesizer, and the rate that information must be generated in

order to produce a video signal clearly shows that the synthesizer must be structured in such a way to exploit redundancy in the video signal. A microprocessor typically requires 2 to 10 $\mu$Sec. to execute a single instruction; thus it cannot possibly be used to generate a point every 100 nS as required for a video signal in real time. On the other hand, one video field last 1/60 = 16.67 mS., a duration in which several thousand instructions can be executed. Thus a microprocessor is capable of generating signals according to a complex algorithm utilizing information supplied by the artist to produce control signals for high-speed special-purpose devices at a field-by-field rate.

Thus we are led to the hierarchical structure shown in Figure 4. Typical channel capacities are shown for each interconnection. The synthesizer consists of high speed special purpose circuits which generate a video signal with a character determined by the control signals supplied by the computer. The control signals fix the behavior of these circuits for an entire field. The computer takes information supplied by the artist, information defining the composition as a whole, and from it determines the control parameters required by the synthesizer for each field. Thus a gradual change in some picture parameters can be specified by the artist by a small set of numbers. In the simplest case only two numbers are required, the frame count of the first and last frame of the sequence. The computer utilizes this information to calculate the corresponding picture parameters for each field; thus it may produce several thousand control values.

## III. System Structure

The CBVS consists of two parts: the computer section shown in the lower section of figure 5 and the video section shown in the upper section of the figure. Both sections operate simultaneously and independently, communicating through the buffer memory which has a capacity of 1,024 sixteen-bit words. Each of these words is either a picture element, a number which controls some function of the video section and determines some aspect of one field of the video image, or it is a picture feature, a number determined by the video section and may depend on an external signal such as a video camera signal. The buffer memory is connected to the computer bus through a sixteen-bit parallel interface which is structured in such a way that each word in the buffer memory is addressable and may be read or written in exactly the same way as words in the main computer memory. This memory-mapped I/O system simplifies the software which controls the buffer memory. In order to update an element such as a control D/A, the computer must execute an instruction which stores the new value in the location corresponding to that element.

During the active scan time, the control computer reads features from the buffer memory and generates elements for the following field and stores them in the buffer memory. During the vertical blanking interval, information is transferred through the element bus from the buffer memory to the element modules or from the feature modules to the buffer memory. The designation of a particular area of the buffer memory as an element or feature is under program control. During the transfer between the

buffer memory and the element bus, the computer is locked out of the buffer memory. On completion of the transfer, the interface generates a vectored interrupt which requests the computer to generate parameters for the next field.

The computer system consists of a DEC LSI-11 microprocessor which has a sixteen-bit word length and an instruction execution time of about 7 microseconds, a Teletype Keyboard and printer connected through a serial interface, 20 K of dynamic memory, and a dual floppy disk system with a capacity of 256,256 bytes per diskette. An additional serial interface is also available for connecting through a modem to other computer systems. The entire system is dedicated to the synthesizer system.

The overall timing is determined by a 9.7552434 MHz clock which is phase locked to the subcarrier (3.579545 MHz). This frequency is chosen to insure a coherent subcarrier and to divide the active portion of the scan line into 512 pixels. The red, green, and blue signals are generated independently, and the chroma encoding is done with analog circuits; thus there is no advantage to following the common practice of making the pixel rate an integer multiple of the subcarrier frequency. With this clock frequency, a full nine bit word is used to define the horizontal position on the active portion of the raster. Figure 6 shows the X and Y wave forms. The X-Y module generates twenty bits of timing information (ten bits for horizontal, including the blanking period, and ten for the line count). This module also generates sync, drive, burst flag, and the transfer request $\overline{TR}$ signal which controls the timing of the buffer memory.

Timing details of the interface and buffer memory are shown in figure

7. The transfer request $\overline{TR}$ goes low at the beginning of vertical blanking initiating an arbitration for access to the buffer memory. If the computer is accessing the buffer memory, the current bus cycle is completed, then $\overline{READY}$ goes high, and the buffer memory controller cycles through memory making the required element and feature transfers through the element bus. When this is completed, $\overline{READY}$ goes low, control of the memory is returned to the computer, and an interrupt is generated requesting data for the following field. As indicated in the diagram, during the Nth field the computer is generating data for the N+1th field.

The timings of the signals on the element bus are indicated in figure 8. During the transfer, the memory controller generates the addresses, $A_0$-$A_9$, the clock signals $\overline{\phi_1}$, $\overline{\phi_2}$, and $\overline{\phi_3}$, and the status signals $\overline{CME}$ indicating a transfer from the memory to an element and $\overline{CFM}$ indicating a transfer from a feature module to the memory. The signals $\overline{ETF}$ and $\overline{FTE}$ are generated by the synthesizer modules and initiate a controller Element/Feature mode change. The three phase clock system is used to control modules which have the structure shown in figure 9. Functions which use data from the computer during the vertical blanking interval are disabled when the buffer memory accesses that particular element by a signal generated using $\overline{\phi_1}$. This allows access to the buffer memory during $\overline{\phi_2}$. The third clock $\overline{\phi_3}$ generates a memory write signal.

Time delays in the digital processing modules could produce errors and shifts of the image to the right. This is prevented by deskewing the output of each with a latch clocked by the master clock (9.755 MHz). Compensation for the resulting 102.5 nSec. delay in each module is provided

by starting the X count at the beginning of the horizontal blanking interval rather than at the end. An additional shift to the right or left is then achieved by adding (mod 512) a constant supplied by the computer. The default value of this constant is 404 + number of elements.

## IV. Element and Feature Modules

The structure described above supports a variety of element and feature modules which may be chosen and configured according to the taste of the artist. Our experience indicates that a large amount of work can be produced with a relatively small number of elements in a standard configuration. Whenever possible a new element added to the system is configured in such a way that if the control word is set equal to zero it has no effect on the system. Thus a minimum amount of reprogramming is required following system expansion.

Two general classes of modules have been developed, digital and hybrid. The hybrid elements are high-speed digital-to-analog converters used for generating the red, green, and blue video signals which are converted to NTSC format in the standard way and low-speed D/A converters used for generating control voltages, field-by-field controllable, used to operate existing voltage-controlled analog image processing systems such as keyers, raster manipulaters, etc. Another hybrid element is the analog video switching matrix. Four bits of one control word are used to select one of sixteen inputs for one output.

Digital processing elements include: constant, X + constant, Y + constant, twelve-channel sixteen-line demultiplexer with output complement, and four-channel four-bit by sixty-four word memory.

One of the possible ways of interconnecting digital element modules is shown schemetically in figure 10. With this arrangement the X and Y signals are processed by the three sequences $(A_1A_2A_3A_4)$, $(B_1B_2B_3B_4)$,

and ($C_1 C_2 C_3 C_4$) to produce three different patterns or textures. These three signals are then combined by element D to produce a composite image. Finally, the digital-to-analog converter produces an analog video signal. The form of the video signal generated by this system depends on both the choice of configuration of modules and on the control parameters supplied by the computer. Since the control parameters are constant during one frame, the video signal may be represented by an equation of the form

$$V(t) = f\left[X(t), Y(t), E_1(t), \ldots\ldots E_n(t)\right]$$

where V(t) is the video signal which varies with time t. The structure of V(t) and of the resulting image is determined by the function f which is defined by the configuration of element modules. The time dependence of the video signal is shown explicitly; during one field, only X and Y change while the control values $E_1(t)$, $E_2(t)$, $E_3(t)$,....$E_n(t)$ are held constant. The time dependence of the video signal has been divided into two classes: firstly, the variation from one field and the next is determined by the computer through the element values, and secondly, the variation during one field is determined by the element modules through the X and Y signals for fixed element values.

## V. System Operation

The power and versatility of this system may be seen by considering an application to an extremely simplified version of the system consisting of a pair of camera signals which are mixed by a voltage controlled mixer. With this system, only one element, a low-speed digital-to-analog converter, utilizes the element bus, and it generates a control voltage which determines the mixer operation for each frame. A flow chart of a program and an example of a data set are shown in figure 11. To use this system, the artist only needs to specify the numerical values in the data set.

The data set in this example corresponds to the mixer selecting the first camera until frame number 600, then fading to the second camera until frame 700, holding the second camera until frame 2000, then fading back to the first camera by frame number 2200. The values 8 and 4 in the parameter column determine the rate at which the fade takes place, and the numbers in the service routine column label indicate the selected service routine: a 1 for no change, a 2 for increasing the control voltage by P units, and a 3 for decreasing the control voltage by P units.

After the artist has stored these twelve numbers in the computer memory, the program may be started, and the computer will initialize the system and go to a background program where it waits for an interrupt. After every other interrupt the frame count is increased by one and compared with the frame count entry in the data set. Then the appropriate service routine (SR) is selected, and the element value is increased or decreased as required. Finally, the computer returns to the background

program and waits for the next interrupt. This process is repeated for every field thus generating the sequence of control voltages and fading from one camera to the other.

The artist can observe the resulting sequence of images and then make changes in the data set to achieve the desired result. This technique may be extended to more complex systems involving several elements and feature modules with corresponding programs and data sets. Thus the artist can produce complex sequences with precise control of each frame. When the composition is finished, the system will automatically generate the video signal which may be displayed on a monitor or recorded on a VTR.

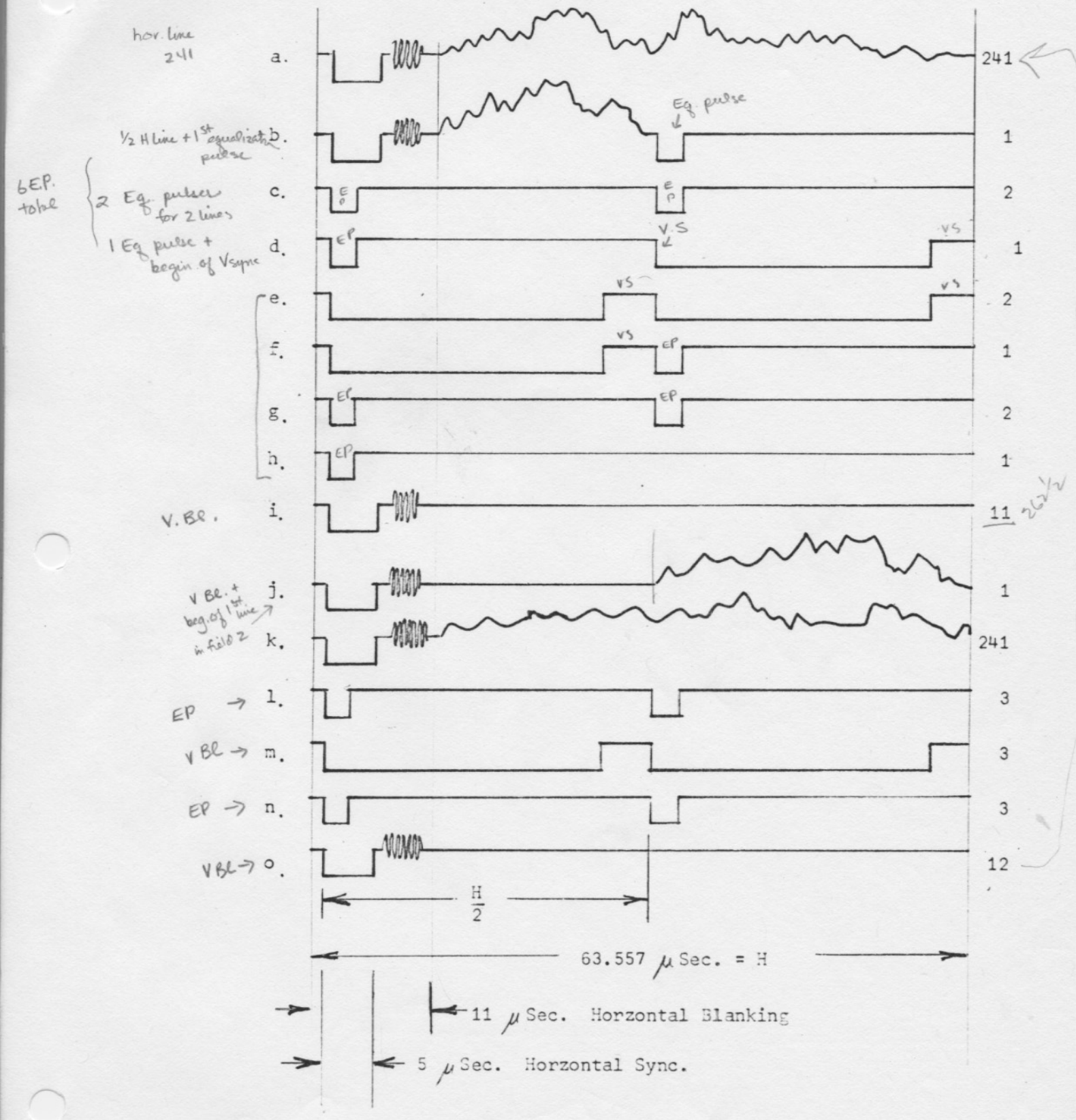
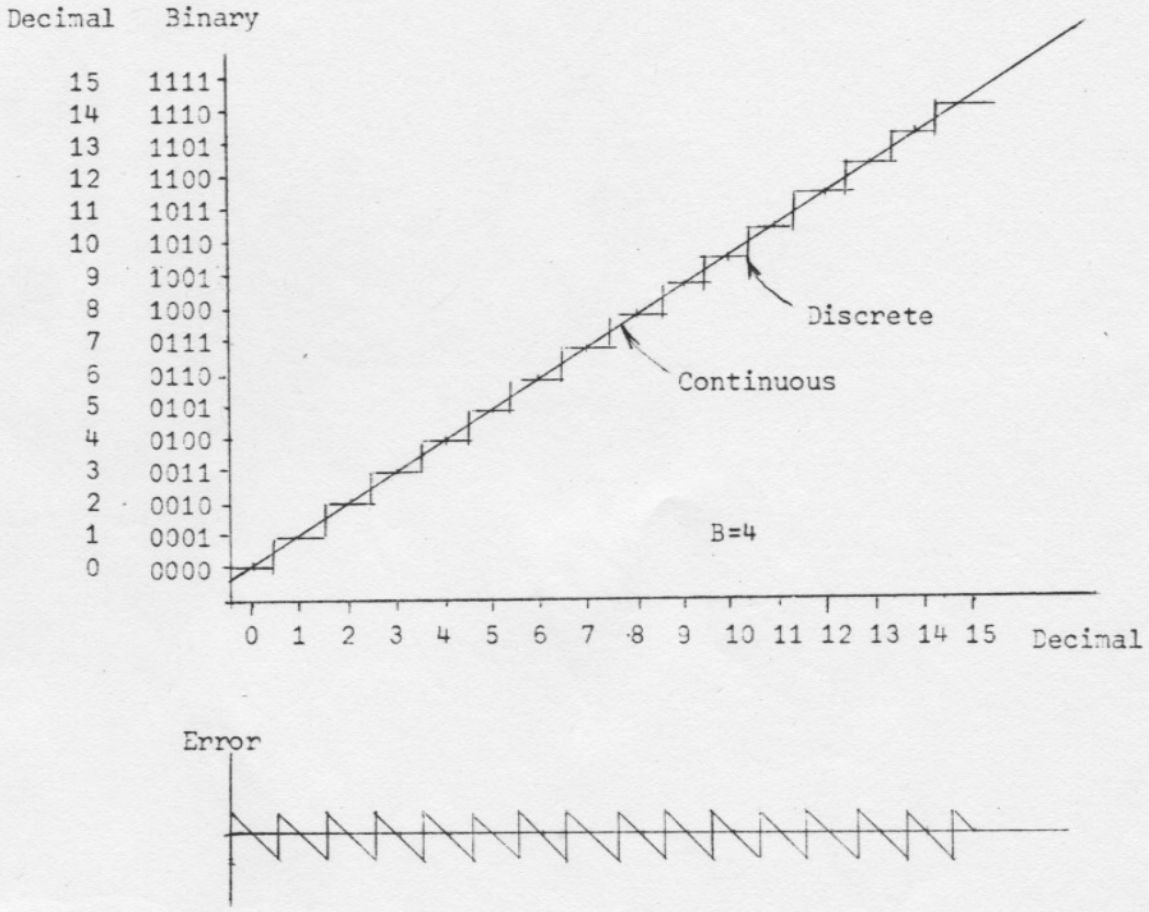

hor. line 241
a.
241
Eq. pulse
½ H line + 1st equalization pulse
b.
1
6 E.P. total
2 Eq. pulses for 2 lines
c.
2
1 Eq. pulse + begin of Vsync
d.
V.S
1
e.
2
f.
1
g.
2
h.
1
V. Bl.
i.
11
V Bl. + beg. of 1st line in field 2
j.
1
k.
241
EP →
l.
3
V Bl →
m.
3
EP →
n.
3
V Bl →
o.
12
H/2
63.557 μ Sec. = H
11 μ Sec. Horzontal Blanking
5 μ Sec. Horzontal Sync.

Figure 1


Decimal
Binary
15 1111
14 1110
13 1101
12 1100
11 1011
10 1010
9 1001
8 1000
7 0111
6 0110
5 0101
4 0100
3 0011
2 0010
1 0001
0 0000
Discrete
Continuous
B=4
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Decimal


Error

Figure 2

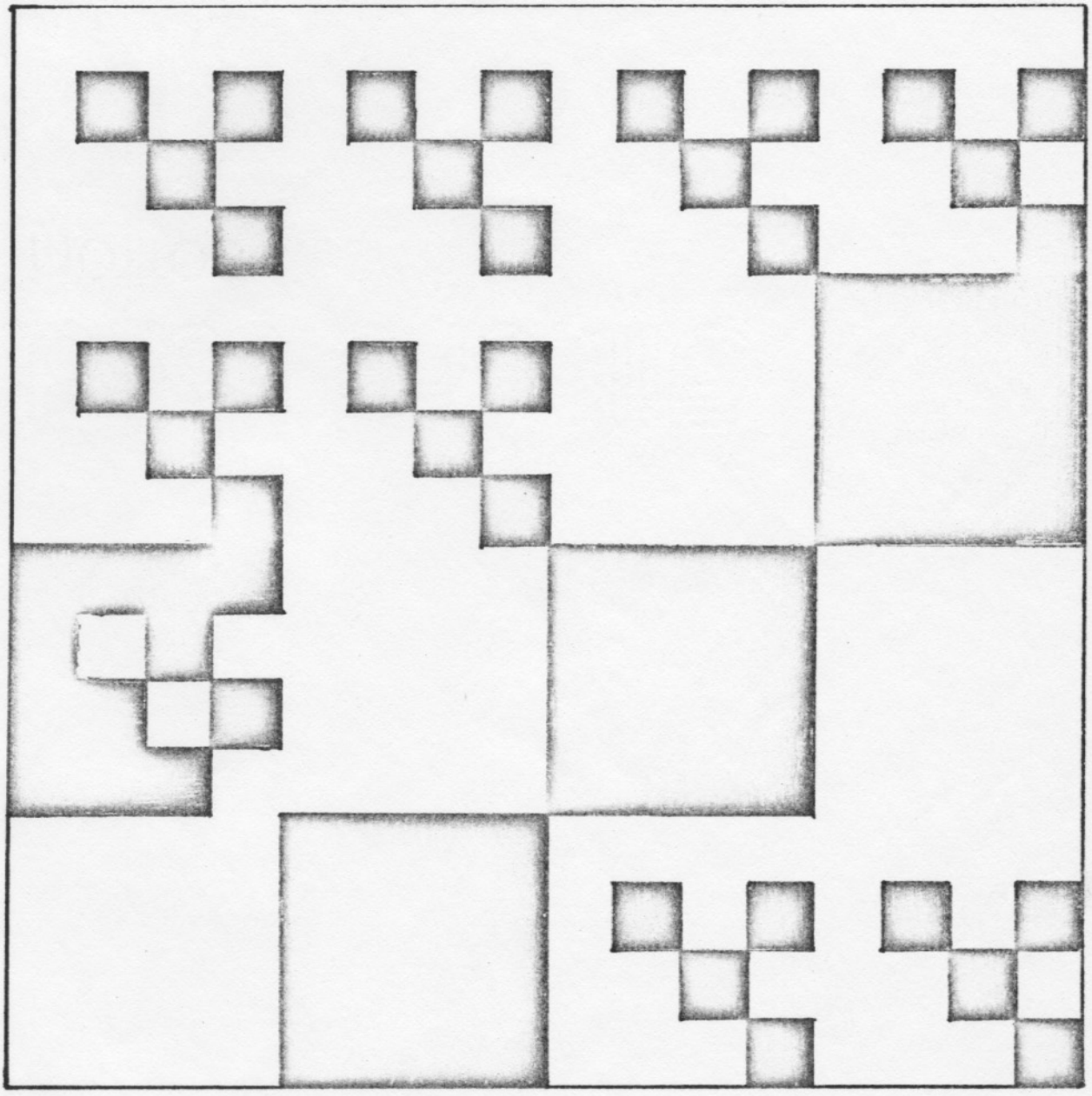


Figure 3

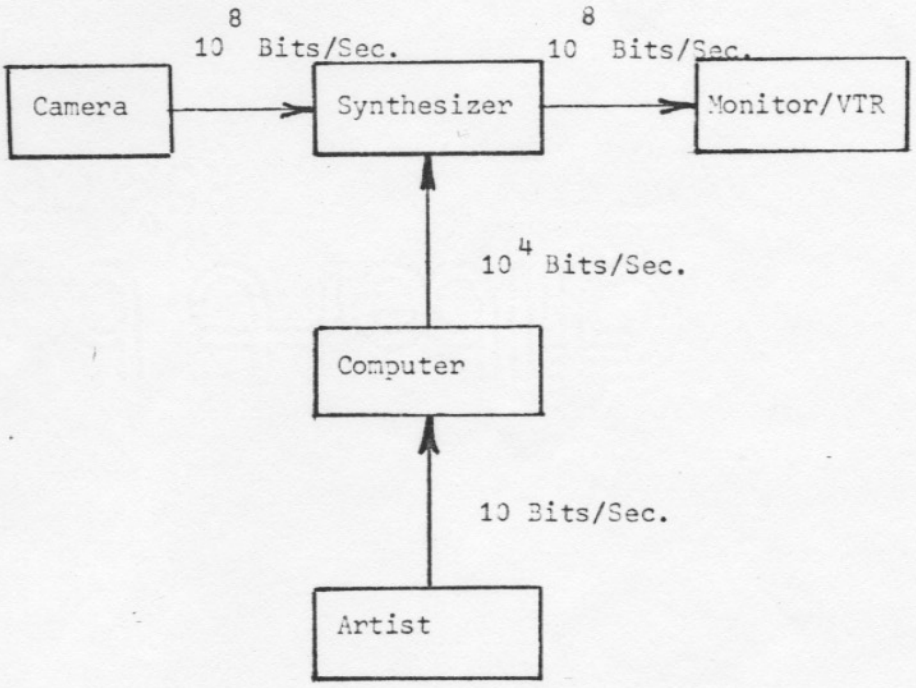

$10^8$ Bits/Sec.
$10^8$ Bits/Sec.
Camera
Synthesizer
Monitor/VTR
$10^4$ Bits/Sec.
Computer
10 Bits/Sec.
Artist

Figure 4

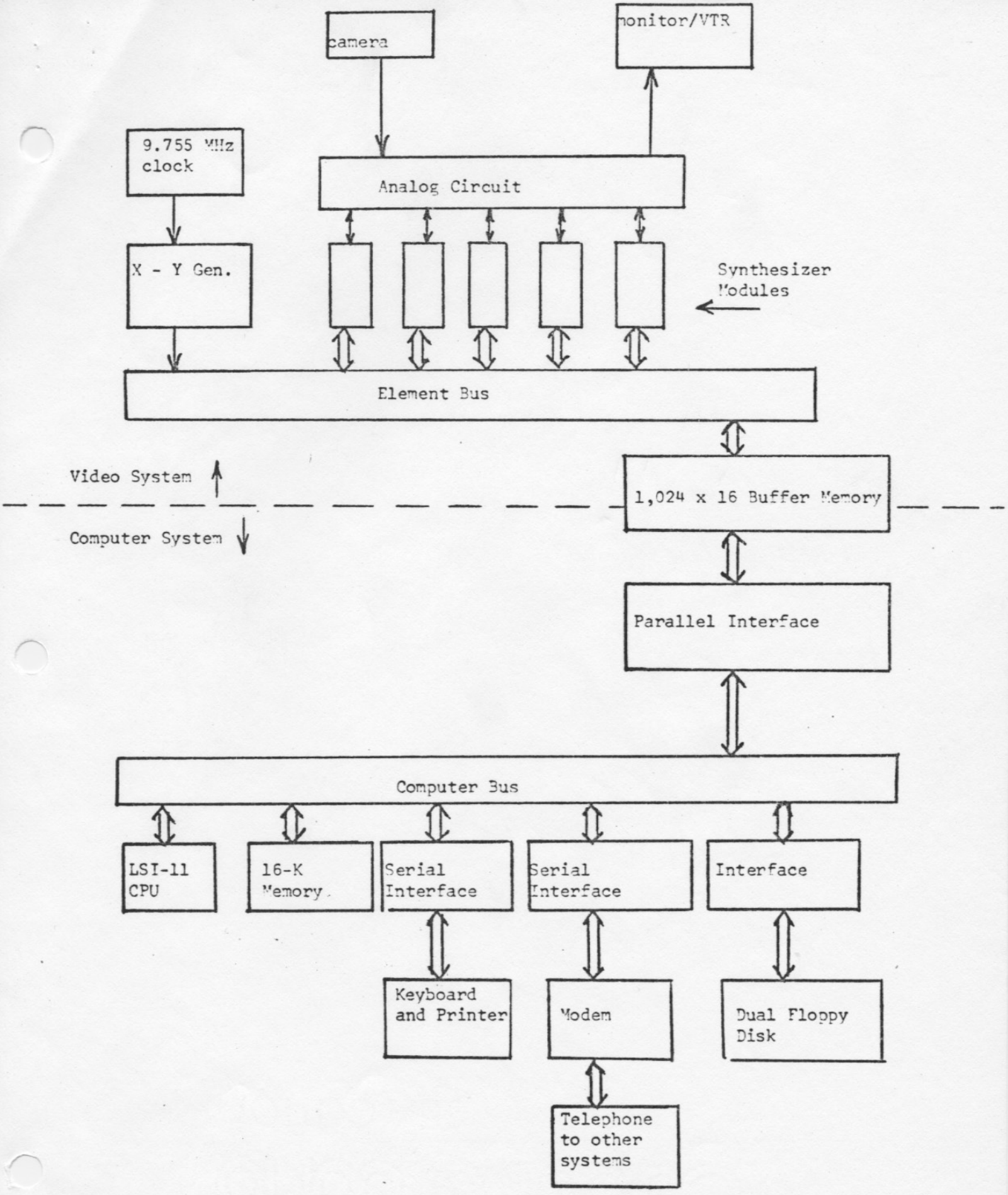

camera
monitor/VTR
9.755 MHz clock
Analog Circuit
X - Y Gen.
Synthesizer Modules
Element Bus
Video System
Computer System
1,024 x 16 Buffer Memory
Parallel Interface
Computer Bus
LSI-11 CPU
16-K Memory
Serial Interface
Serial Interface
Interface
Keyboard and Printer
Modem
Dual Floppy Disk
Telephone to other systems

Figure 5

## X and Y Half Cycle Durations and Wave Forms

$x_1$ 102.5 nSec.

$x_2$ 205 nSec.

$x_3$ 410 nSec.

$x_4$ 820 nSec.

$x_5$ 1.64 $\mu$Sec.

$x_6$ 3.28 $\mu$Sec.

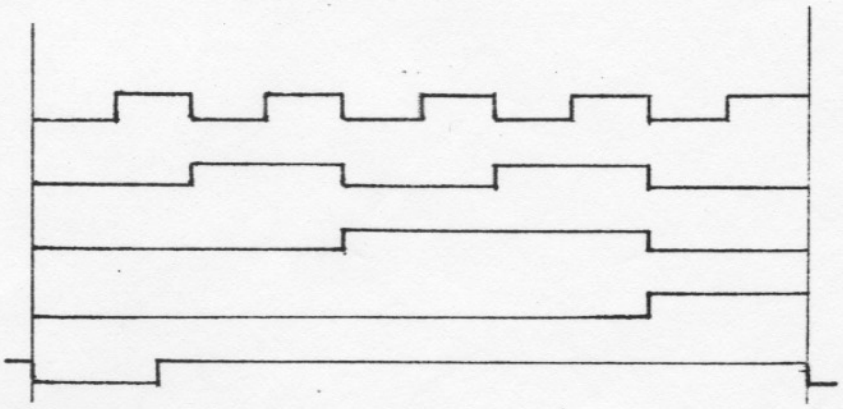
$x_7$ 6.56 $\mu$Sec.

$x_8$ 13.12 $\mu$Sec.

$x_9$ 26.24 $\mu$Sec.

$x_{10}$ 52.48 $\mu$Sec.

Horizontal Blanking



$y_1$ 16.66 mSec.

Field Index

$y_2$ 63.5 $\mu$Sec.

$y_3$ 127 $\mu$Sec.

$y_4$ 254 $\mu$Sec.

$y_5$ 508 $\mu$Sec.

$y_6$ 1.01 mSec.

$y_7$ 2.03 mSec.

$y_8$ 4.06 mSec.

$y_9$ 8.13 mSec.

$y_{10}$ 16.26 mSec.

Vertical Blanking

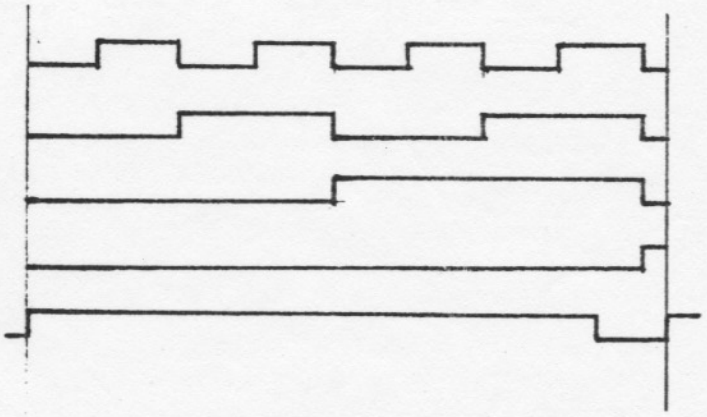
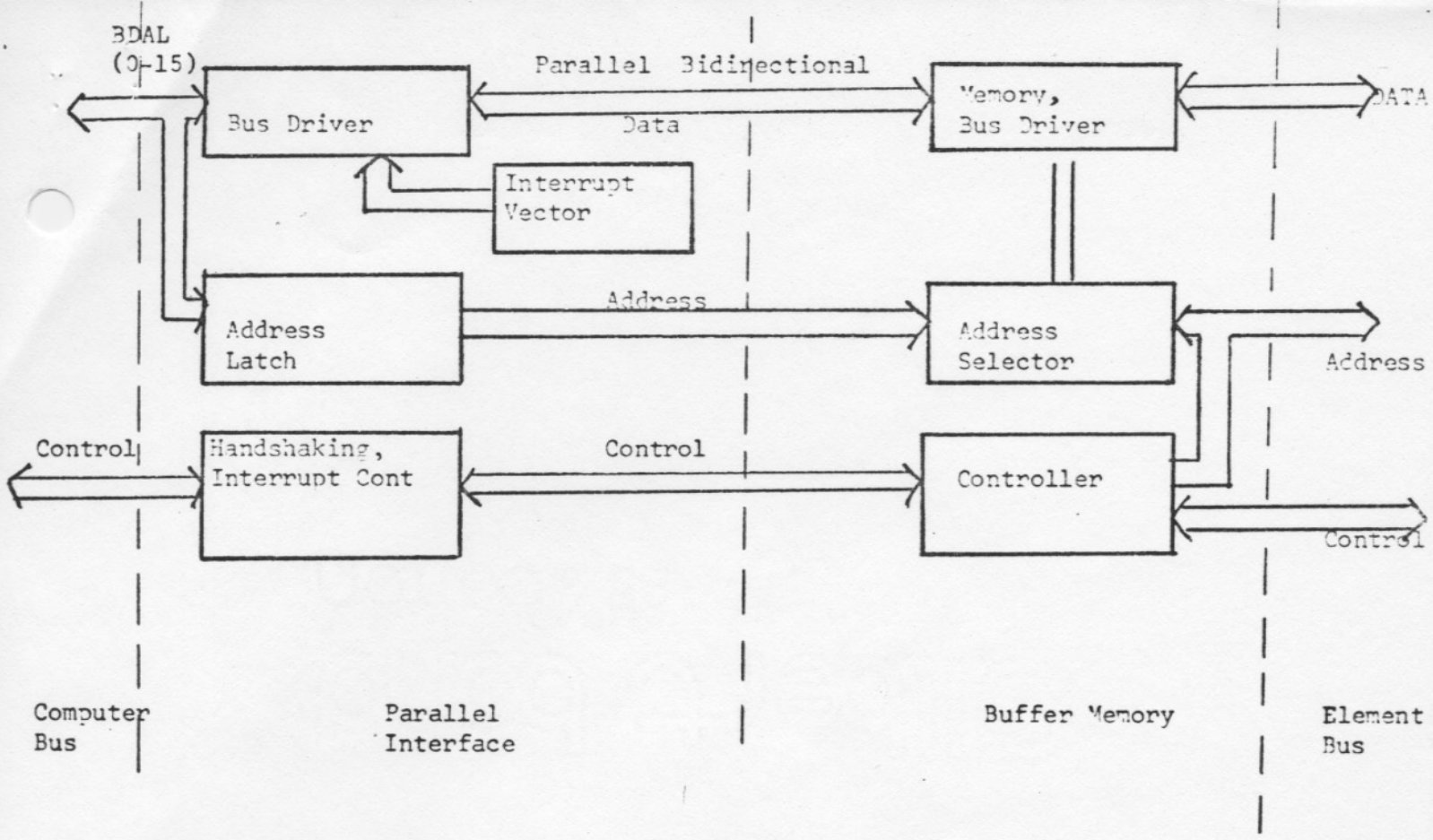


Figure 6


BDAL
(0-15)
Parallel Bidirectional
Data
Bus Driver
Memory,
Bus Driver
DATA
Interrupt
Vector
Address
Address
Latch
Address
Selector
Address
Control
Handshaking,
Interrupt Cont
Control
Controller
Control
Computer
Bus
Parallel
Interface
Buffer Memory
Element
Bus

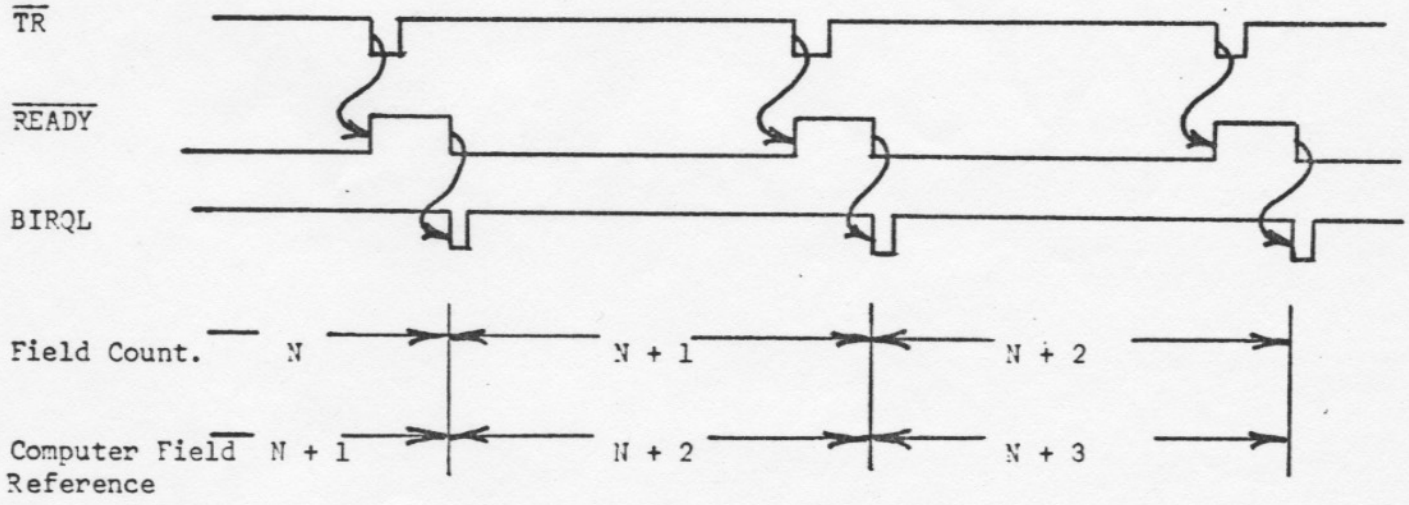

TR
READY
BIRQL
Field Count.
N
N + 1
N + 2
Computer Field
Reference
N + 1
N + 2
N + 3

Figure 7

Element Bus

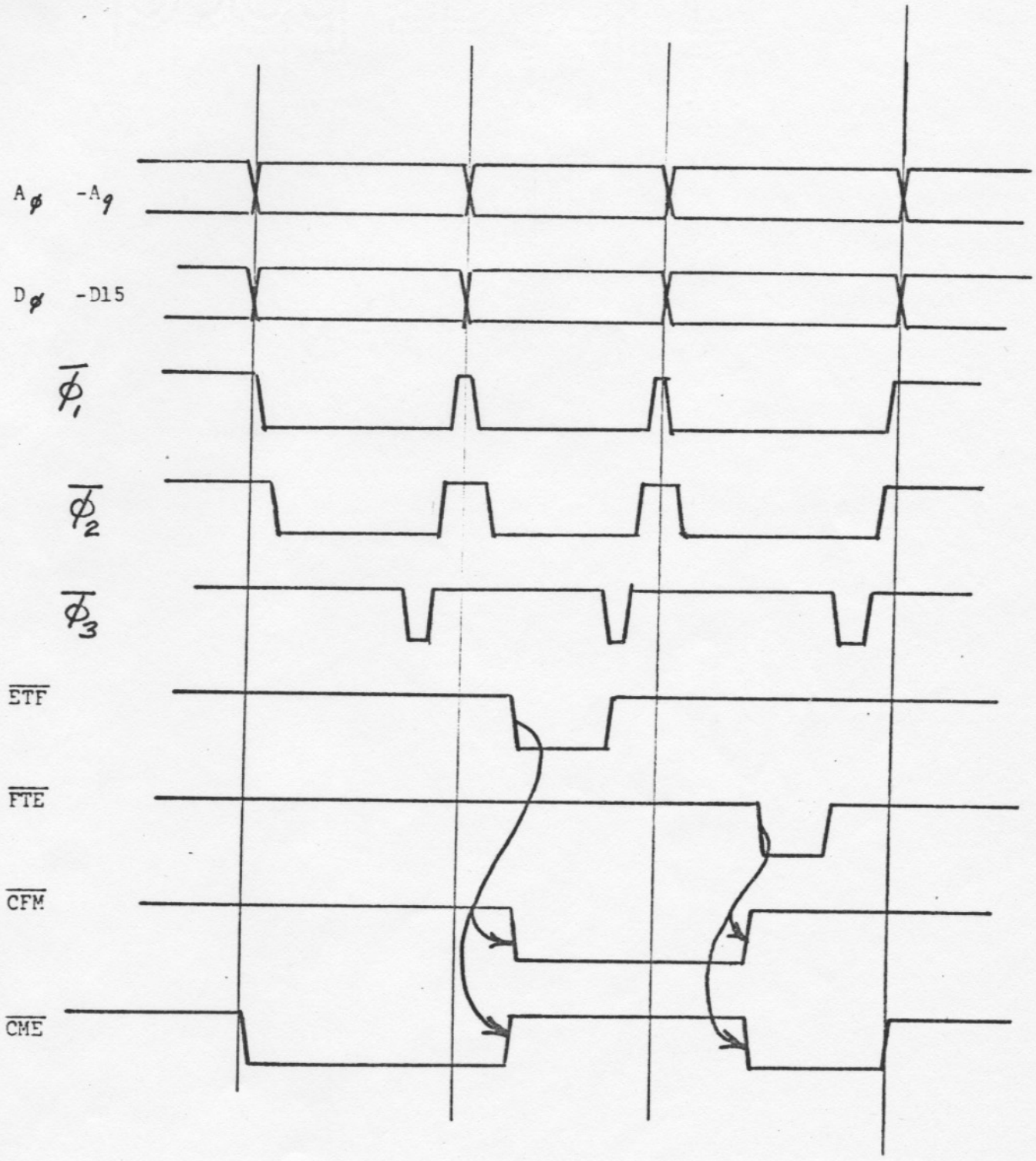



Figure 8

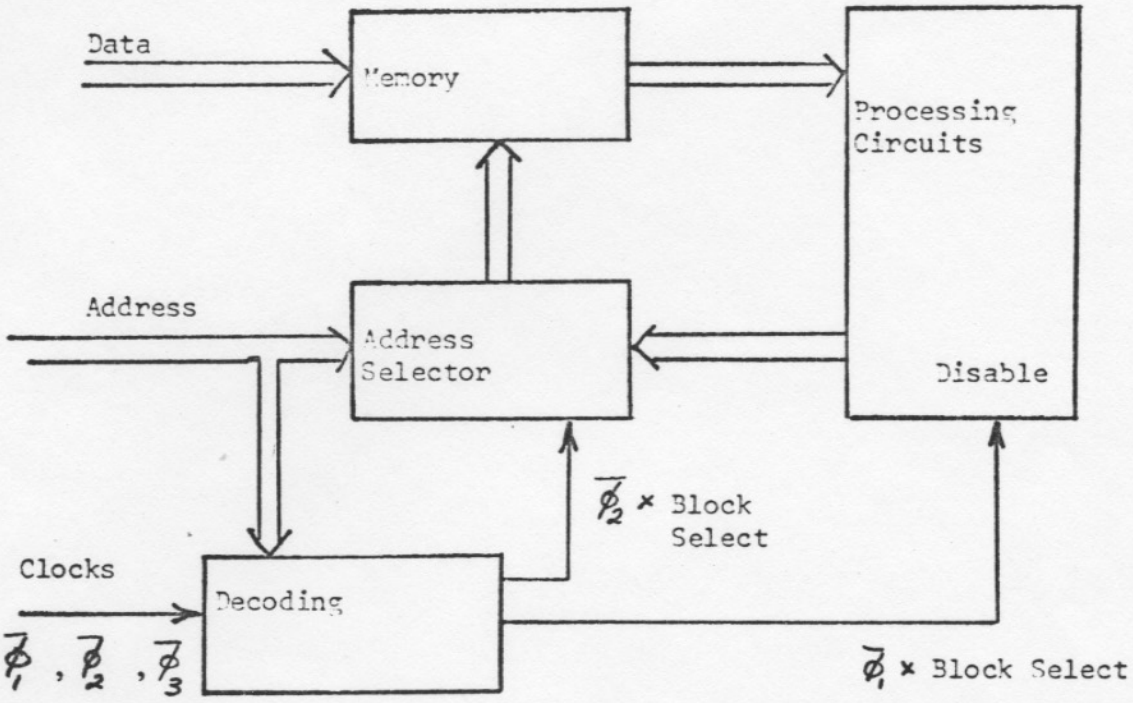

Data
Memory
Processing Circuits
Address
Address Selector
Disable
$\overline{\phi_2}$ × Block Select
Clocks
Decoding
$\overline{\phi_1}$ , $\overline{\phi_2}$ , $\overline{\phi_3}$
$\overline{\phi_1}$ × Block Select

Figure 9

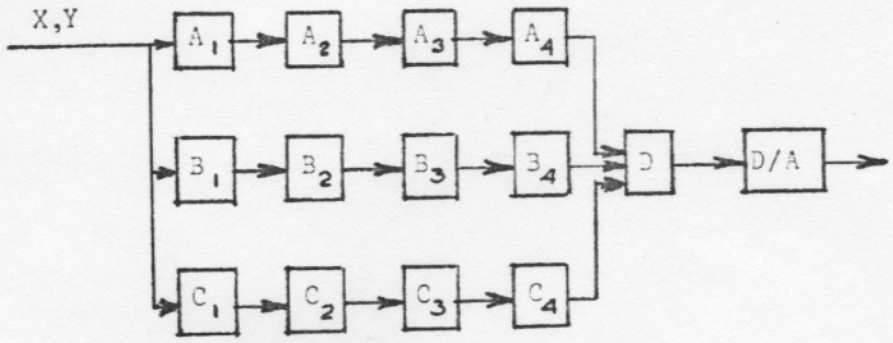
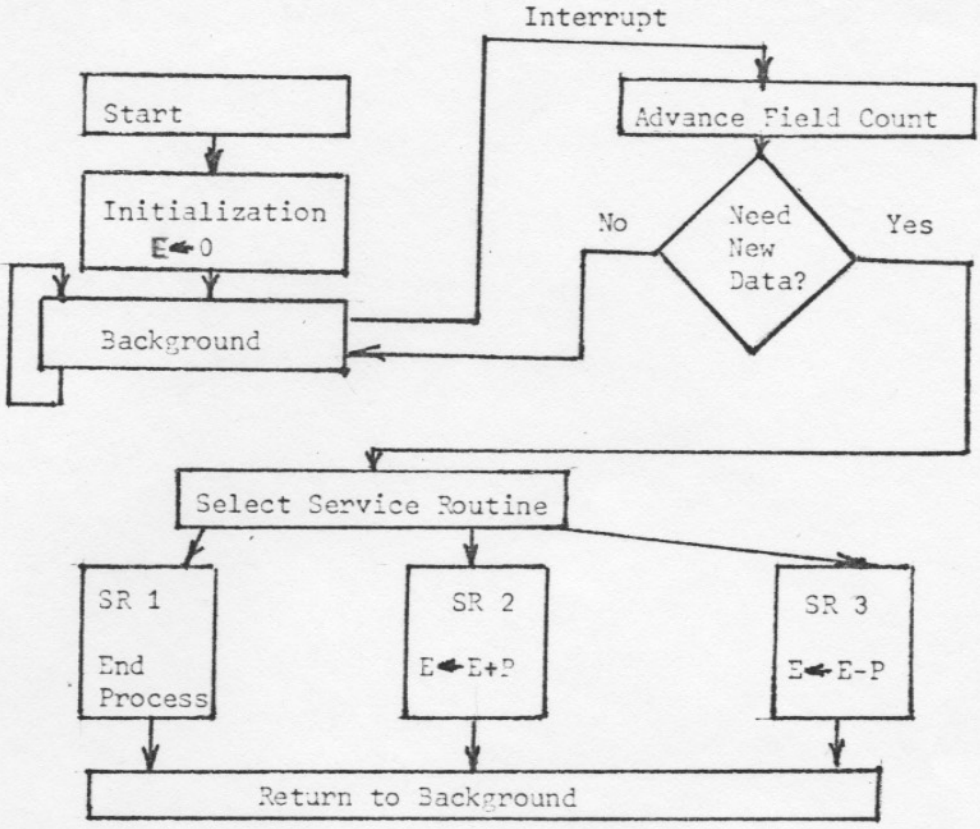



Figure 10




Data Set

| Frame count | Service Routine Lable | Parameter P |
|---|---|---|
| 600 | 2 | 8 |
| 700 | 1 | 0 |
| 2000 | 3 | 4 |
| 2200 | 1 | 0 |

Figure 11

EXPERIMENTAL TELEVISION CENTER LTD.
164 COURT ST.
BINGHAMTON NEW YORK 13901
607-723-9509

# A Computer-Based Video Synthesizer System

## Software

Walter Wright

Experimental Television Center Ltd.
Binghamton, New York

July 1977

This project is supported in part by the National Endowment for the Arts and the New York State Council on the Arts.

EXPERIMENTAL TELEVISION CENTER LTD.
164 COURT ST.
BINGHAMTON NEW YORK 13901
607-723-9509

# A Computer-Based Video Synthesizer System

## Software

Walter Wright

Experimental Television Center Ltd.
Binghamton, New York

July 1977

This project is supported in part by the National Endowment for the Arts and the New York State Council on the Arts.

## Development Program

The first program was developed for Woody Vasulka who uses an LSI-11 microcomputer interfaced to video synthesis modules including digital to analog converters,(D/A's), analog to digital converters (A/D's), Don McArthur's modules described elsewhere in this report and George Brown's multiple level keyer.

The D/A's and A/D's are controlled through four words in memory as follows:

1. status word - LEWSTA at location 167770 (8)
2. output word - LEWOUT at location 167772 (8)
3. input word - LEWIN at location 167774 (8)
4. channel address - LEWCHA at location 167776 (8)

McArthur's modules are controlled through the buffer memory which appears as normal memory to the program. Any location in buffer memory can be read in or written to, and finally arithmetic and logic operations can be performed thereupon. This technique of "memory-mapped I/O" makes the programmer's life much easier and besides it's quick - important because all modules must be updated in less than 1/60 sed. Control words for McArthur's modules are located in the upper reaches of memory as follows.

## NEA Development Program

Next the system macros are invoked with the following statements:

```
BEGIN:          .MCALL  ..V2..,.REGDEF,.EXIT
                ..V2..
                .REGDEF
```

The label BEGIN: is used by the linking loader to identify the entry point for the main program. This is done using this statement at the end of the program:

```
    .END  BEGIN
```

The ..V2.. macro identifies the monitor system being used by the LSI-11. The .REGDEF macro defines the LSI-11's internal registers using two character mnemonics as follows -

1) R0 - general purpose register 0
2) R1 - general purpose register 1
3) R2 - general purpose register 2
4) R3 - general purpose register 3
5) R4 - general purpose register 4
6) R5 - general purpose register 5
7) PC - program counter, register 7 ← address of next instruction to be executed
8) SP - stack pointer, register 6 ← location of last entry on stack

Now we're ready to initialize the D/A's which is accomplished thus -

```
1)              MOV             #100000, @# LEWOUT
2)              MOV             #10,R0
3)  BGNI:       DEC             R0
```

NEA Development Program

```
4)          MOV          R0,@# LEWCHAz
5)          TST          R0
6)          BNE          BGN1
```

The first line of code moves the octal number 100000 to the output word in memory which controls the D/A's. This causes the D/A to output a constant 0V (+10Vv= 177700 & -10V = 0). The prefix # defines a real number, and the prefix @# defines an address (a location in memory). However the data transfer is not consummated until the D/A channel is addressed through the channel address word. The are 8 D/A channels 0 - 7. Therefore we set register 0 to the octal number 10 = decimal 8 (line 2). Then we count down register 0 with a loop (lines 3,5 & 6) and at the same time enable the D/A's by moving the contents of register 0 to the channel address word (line 4).

And we initialize the buffer memory -

```
1)          MOV     #D0NOUT_, R0
2)          CLR     (R0) +
3)          CLR     (R0) +
4)          CLR     (R0) +
5)          CLR     (R0) +
6)          MOV     # JEFOUT , R0
7)          CLR     (R0) +
8)          CLR     (R0) +
9)          CLR     (R0) +
```

## NEA Development Program

This code uses the auto-increment mode of addressing (register) +. Line 1 moves the octal number 171040 into register 0. Then we clear that memory location and add + 2 to register 0 which now points to the next word in memory (lines 2-5). This sets the red, green and blue 16?1 select channels to black and the inversion register to normal (non-inverting). Similiarly the ALU's are set to pass red, green and blue respectively.

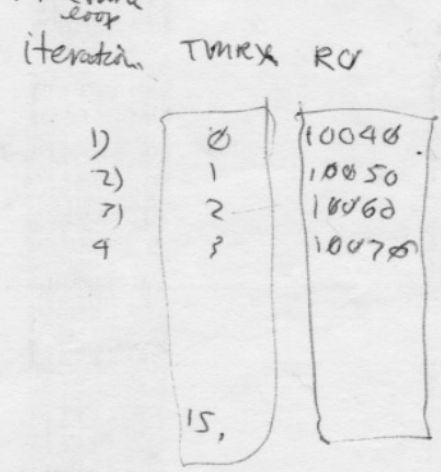
The maximun number of data buffers is set -

```
MOVB    #20 , TMRY
```

That is, the program tolerates no greater than octal 20 = 16 decimal buffers. This fact is recorded in the byte labelled TMRY.

Each data buffer is associated with four parameter words and these 64 words (4x16) are kept in the parameter buffer PBUF. We initialize this buffer as follows -

```
1)        MOV     #PBUF , R0
2)        SUB     #10 , R0
3)  BGN2: CMPB    TMRX , TMRY
4)        BPL     TMR
5)        INCB    TMRX
6)        ADD     #10 , R0
7)        CLR     (R0)
8)        MOV     #1 , 2(R0)
9)        MOVB    TMRX , R1
10)       DEC     R1
```



NEA Development Program

```
11)          SWAB    R1
12)  ~~ADD~~  ADD     #DBUF , R1
13)          MOV     R1 , 4 (R0)
14)          CLR     6(R0)
15)          BR      BGN2
16   PBUF:   .= .+200  ~~+200~~
```

Again we use a loop; we set register 0 to the location of PBUF (lines 1 and 2). Note PBUF is created by causing the program counter (.) to skip over 64 words of memory (line 16). The loop is controlled by TMRX andT TMRY. TMRX counts up to the maximum number of data buffers, then a branch to the next block of code is executed (lines 3,4,5 and 15). The four parameterswords are -

1) timing counter — R0
2) timing interval — 2(R0)
3) pointer to DBUF — 4(R0)
4) data — 6(R0)

The first word is cleared (line 7). The timing interval is set to a single field (line 8). Next address of the data buffer is calculated and put in the third word (lines 9-13). There are 16 data buffers each containing 128 words. Therefore the pointer is set initially as follows -

pointer = #DBUF + (256 * (TMRX - 1))

This formula is coded from right to left.

NEA Development Program

In line 9 TMRX is moved into register 1, the decrement instruction in line 10 subtracts 1 from the register; the swap byte instruction in line 11 effectively multiplies the register by 256 (equivalent to 8 left shifts or multiplication by $2^8$); DBUF is added to register in line 12; and finally in line 13 the result is stored in the parameter buffer using the indexed addressing mode (6(R0) the contents of register 0 plus the index 6 produce the effective address).

From here we go to the timing routine (TMR). This routine enables the 1/60 sec interrupt, and every 1/60 sec polls the parameter buffer checking for time outs (timing counter equal timing interval). If a data buffer times out a branch to the next block of code is executed.

The buffer memory transfers data to the modules during the vertical interval between each field of video. Then the buffer memory generates an interrupt telling the computer to get working on data for the next field. This interrupt is enabled or disabled with the status word (@# DONSTA). If the status word equals 1 the interrupt is enabled; if 0 the interrupt is disabled. So much for the buffer memory - the LSI - 11 handles interrupts thus. The computer interrupts its normal flow of operations and as a precaution pushes the current program counter (PC or register 6) and the program status word (PSW) onto the stack. The stack pointer (SP) is decremented by 4. Then the computer goes to a predeter-

mined location in memory (in this case @# 170) and uses the contents of this location as the new program counter (PC). Execution begins anew from the location pointed to by @# 170. Usually this is an interrupt service routine, however I have taken a shortcut as explained below.

```
1)  1)  TMR:    MOV     #TMRI , @# 170
    2)          CLRB    TMRX
    3)          INC     @# DONSTA
    4)          BR      .
    5)  TMRI:   CLR     @# DONSTA
    6)          ADD     4 , SP
```

In line 1 we prepare for the inevitable interrupt by loading location 170 with the location #TMRI; the location where we will resume execution. Next the fuffer counter (TMRX) is cleared and the interrupt is enabled (LI lines 2 and 3). Now we wait for the interrupt by executing the branch instruction on line 4. Following the interrupt we return to line 5 and disable further interrupts by clearing the status word in the buffer memory. Then in line 6 we do some housekeeping, restoring the stack pointer (SP).

We are now ready to poll the data buffers -

```
1)          MOV     #PBUF , R0
```

NEA Development Program

```
 2)          SUB     #10 , R0
 3)  TMR2:   CMPB    TMRX , TMRY
 4)          BPL     TMR
 5)          INCB    TMRX
 6)          ADD     #10 , R0
 7)          MOVB    TMRX , R 2
 8)          DEC     R2
 9)          ADD     #EBUF , R2
10)          TSTB    (R2)
11)          BEQ     TMR2
12)          INC     (R0)
13)          CMP     (R0)
14)          BLE     TMR2
15)  TMR3:   CLR     (R0)
16)          JSR     PC , INT
17)          BR      TMR2
18)  TMRX:   .BYTE   0
19)  TMRY:   .BYTE   0
```

Again we have a loop similiar to the loop used to initialize the parameter buffer. Lines 1 and 2 load register 0 with #PBUF - 8. In line 3 the counter (TMRX initially 0) and the number of buffers (TMRY) are compared. Assuming all the buffers were checked we branch back to wait for

the next interrupt (line 4). Otherwise we increment register 0 by 8 (line 6) and check the enable buffer (lines 7 to 10). If the buffer is disabled (the contents of location #EBUF + (TMRX - 1) equal 0) we branch back to TMR2 (line 11). If the buffer is enabled the timing counter is incremented (line 12) and compared with the timing interval (line 13). If the counter is less than or equal the interval we branch back to TMR2 (line 14).OtOtherwise we clear the timing counter and jump to the interpreter routine (lines 15 and 16). Upon returning from the interpreter, (line 17), we branch back to TMR2 completing the timing routine. Lines 18 and 19 reserve space im memory for the buffer counter (TMRX) and the nmbber of buffers (TMRY).

## NEA - DEVELOPMENT PROGRAM

The interpreter reads a command word from the data buffer and uses this word to create a special jump subroutine instruction. The subroutine in turn executes the command reading additional data words from the buffer as required.

```
 1) INT:   MOV   4(R0) , R1
 2)        MOV   (R1)+ , R2
 3)        ASL   R2
 4)        ADD   #JBUF , R2
 5)        MOV   (R2) , R2
 6)        SUB   #INTI , R2
 7)        MOV   R2 , INTI-2
 8)        CLR   R5
 9)        JSR   PC , EXIT
10) INTI: MOV   R1 , 4(R0)
11)        TST   R5
12)        BEQ   INT
13)        RTS   PC
```

Remember that register 0 contains the address of the first of the four parameter words controlling the data buffer. In line 1 the data pointer (4(R0)) is moved to register 1. Then the command word ((R1)+) is moved from the data buffer to register 2; and the data pointer in auto-incremented

(line 2). The jump subroutine through the program counter instruction (line 9) is decoded by the assembler as two words - 004767 , XXXXXX. The first three digits of the first word (004) indicate a JSR instruction. The fourth digit (7) indicates that register 7 (PC) will be the linkage pointer. The fifth and sixth digit represent the destination, the fifth digit specifies the index addressing mode and the sixth digit indicates that the index value follows the instruction. The index value plus the program counter equals the destination address. In lines 3 - 6 the index value is calculated using these formulae -

1) index = subroutine entry pt - # INTI

2) subroutine entry pt = # JBUF + (2 *Command word)

The index value is moved to location INTI - 2 (line 7). Register 5 is a done flag set following the output command, it is cleared initially (line 8). The jump subroutine instruction is executed (line 9), the program executes the appropriate subroutine, and returns to restore the data buffer pointer (line 10). The done flag (R5) is tested (line 11); if zero the program branches back and reads the next command word (line 12), or returns to the timing routine (line 13).

A cross reference table (JBUF) follows the interpreter. The entry points for the subroutines are stored sequentially and are accessed with the command word. A summary of the function of each subroutine follows -

NEA Development Program

SUB00 - (00) sets the timing interval (second word in the parameter list) equal to the next word in the data buffer.

```
MOV     (R1) + , 2 (R0)
```

SUB01 - (01) adds the next word in the data buffer to the timing interval.

```
ADD     (R1) + , 2 (R0)
```

SUB02 - (02) subtracts the next word in the data buffer from the timing interval.

```
SUB     (R1) + , 2(R0)
```

SUB03 - (03) complements the timing interval, equivalent to $177777_8$ - timing interval.

```
COM     2(R0)
```

SUB04 - (04) shifts the timing interval to the right, the most significant bit (bit 15) is cleared, equivalent to timing interval/2.

```
CLR
ROR     2 (R0)
```

SUB05 - (05) shifts the timing interval to the left, the least significant bit (bit 0) is cleared, equivalent to 2* timing interval.

```
CLR
ROL     2 (R0)
```

Command words 06 and 07 are not used, therefore they are cross referenced to the error routine ERR in JBUF.

SUB 10 - (10) sets the data word (fourth word in the parameter list) equal to the next word in the data buffer.

NEA Development Program

```
                MOV     (R1) +, 6 (R0)
```

SUB 11 - (11) increments the data word, equivalent to data word + 1.

```
                INC     6(R0)
```

SUB 12 - (12) decrements the data word, equivalent to data word - 1.

```
                DEC     6 (R0)
```

SUB 13 - (13) adds the next word in the data buffer to the data word.

```
                ADD     (R1) +, 6 (R0)
```

SUB 14 - (14) subtracts the next word in the data buffer to the data word.

```
                SUB     (R1) +, 6 (R0)
```

SUB 15 - (15) complements the data word, equivalent to $177777_8$ - data word.

```
                COM     6 (R0)
```

SUB 16 - (16) shifts the data word to the right, the most significant bit (bit 15) is cleared, equivalent to data word/2.

bit n becomes bit n-1

bit 0 dropped

```
                CLC
                ROR     6 (R0)
```

SUB 17 - (17) shifts the data word to the left, the least significant bit (bit 0) is cleared, equivalent to 2* data word.

bit n becomes bit n+1

bit 15 dropped

```
                CLC
                ROL     6(R0)
```

NEA Development Program

SUB 20 - (20) rotates the data word to the right, shifts the bits right, the least significant bit (bit 0) is rotated around to become the most significant bit (bit 15).

15 -0

bit 0 becomes bit 15

```
MOV     6 (R0) , R2
ROR     R2
ROR     6 (R0)
```

SUB 21 - (21) rotates the data word to the left, shifts the bits left, the most significant bit (bit 15) is rotated around to become the most significant bit (bit 0).

15 0

bit 15 becomes bit 0

```
MOV     6 (R0) , R2
ROL     R2
ROL     6 (R0)
```

SUB 22 - (22) takes the next word in the data buffer and clears each bit in the data word which corresponds to a set bit in the former, equivalent to - data word = ~next word in buffer $\wedge$ data word

NEA Development Program

eg. next word in buffer 0 000 001 010 011 100

data word 0 000 001 101 100 111

data word 0 000 000 101 100 011

BIC (R1) +, 6 (R0)

SUB 23 - (23) takes the next word in the data buffer and sets the corresponding bits in the data word, equivalent to - data word + next word in buffer $\vee$ data word.

eg. next word in buffer 0 000 001 010 011 100

data word 0 000 001 001 001 001

data word 0 000 001 011 011 101

BIS (R1) +, 6 (R0)

SUB 24 - (24) takes the next word in the data buffer and exclusive OR's ($\veebar$) it with the data word.

eg. next word in buffer 0 000 001 010 011 100

data word 0 000 001 001 001 001

data word 0 000 000 011 010 101

MOV (r1) +, R2

XOR R2 , 6 (R0)

Command words 25, 26 and 27 are not used, therefore they are cross-referenced to the error routine ERR in JBUF.

SUB 30 - (30) calls the input routine and sets the data word equal to input data (in register 2).

JSR PC , IN

MOV R2 , 6 (R0)

NEA Development Program

SUB 30 - (30) calls the input routine and sets the data word equal to input data (in register 2).

```
JSR     PC , IN
MOV     R2 , 6 (R0)
```

SUB 31 - (31) calls the input routine and adds the input data to the data word.

```
JSR     PC , IN
ADD     R2, 6 (R0)
```

SUB 32 - (32) calls the input routine and subtracts the input data from the data word.

```
JSR     PC , IN
SUB     R2 , 6 (R0)
```

SUB 33 - (33) calls the input routine and clears each bit in the data word as in SUB 22.

```
JSR     PC , IN
BIC     R2 , 6 (R0)
```

SUB 34 - (34) calls the input routine and sets each bit in the data word as in SUB 23.

```
JSR     PC , IN
BIS     R2 , 6 (R0)
```

SUB 35 - (35) calls the input routine and exclusive OR's the input data with the data word as in SUB 24.

```
JSR     PC , IN
XOR     R2 , 6 (R0)
```

NEA Development Program

Command words 36 and 37 are not used, therefore they are cross-referenced to the error routine ERR in JBUF.

LOOP - (40) this subroutine uses the next three words in the data buffer to create a repeating loop in the data buffer. The three words are -

1) a counter, incremented each repetition

2) maximum number of repetitions

3) (counter) to the top of the loop

Each time a loop command (40) is encountered in the data buffer, the loop subroutine first compares the counter with the maximum number of repetitions (line 1). If the counter is less than the maximum number the counter is incremented, the pointer to DBUF (third word in the parameter list) is updated with the pointer to the top of the loop, and return to the interpreter (lines 3-5). If the counter is equal to or greater than the counter we branch to LOOP 1 (line 2), clear the counter (line 6), step the data buffer pointer (line 7), and return to the interpreter (line 8).

```
1)  LOOP:   CMP     (R1) , 2 (R1)
2)          BPL     LOOP 1
3)          INC     (R1)
4)          MOV     4 (R1) , R1
5)          RTS     PC
6)  LOOP 1: CLR     (R1)
7)          ADD     #6 , R1
```

```
8)          RTS     PC
```

Command words 41 - 45 are not used, therefore they are cross-referenced to the error routine ERR in JBUF. The error routine is in reality the exit routine -

EXIT - (47) this subroutine is invoked overtly by command word 47 and covertly by 06,07,25,26,27,36,37,41,42,43,44 and 45. It ends the program in a relatively painless manner and returns control to the system monitor using the .EXIT macro.

```
.EXIT
```

The input subroutine services these fourteen input devices -

1 - 8) 16 word data tables defined by user

9 - 12) analog to digital converters

13) real time interface

14) random number generator

The first part of the input routine retrieves data from the tables (input devices 1 - 8)

```
1)  IN:     MOV     (R1) + , R2
2)          CMP     R2 , #11
3)          BPL     IN1
4)          MOV     (R1) +, R3
5)          DEC     R2
6)          ASL     R2
7)          ASL     R2
```

```
 8)          ASL     R2
 9)          ASL     R2
10)          DEC     R3
11)          ASL     R3
12)          ADD     R3 , R2
13)          ADD     #TABLES , R2
14)          MOV     (R2) , R2
15)          RTS     PC
```

In line 1 the input device number is transferred from the data buffer to register 1, and the buffer pointer incremented. If the device number is greater than 8 branch to IN1 (lines 2 and 3). If not move the table entry number to register 2 and calculate the location of the data (lines 4 - 13) as follows -

location = # TABLES + 2 * (entry number - 1) + 16 * (device number - 1).

Finally register 2 transforms itself into the requested data (line 14) and we return to the calling subroutine (line 15).

The second part of the input routine services the analog to digital converters (input devices 9 - 12) -

```
 1)  IN :    CMP     R2 , # 15
 2)          BPL     IN2
 3)          SUB     # 11 , R2
 4)          MOV     R2 , @# LEWCHA
```

```
5)          MOV     @# LEWIN , R2
6)          RTS     PC
```

Again we test the device number. If greater than 12 branch to IN2 (lines 1 and 2). The channel address is calculated and moved to the control word LEWCHA (lines 3 and 4). The data appears at the input word LEWIN and is tramsferred to register 2 (line 5). We return to the calling subroutine (line 6).

The third part of the input routine service Don McArthur's real time interface (a register loaded from the outside world using toggle switches, input device 13).

```
1)  IN2:    CMP     R2 , # 16
2)          BPL     IN3
3)          MOV     @# DONIN , R2
4)          RTS     PC
```

A model of the efficiency of memory mapped I/O; but first we test the device number. If greater than 13 branch to IN3 (lines 1 and 2). In a single of code the data is transferred to register 2 (line 3) and we return to the calling subroutine. Good work Don!

The final section of the input routine is a random number generator of sorts (input device 14) -

NEA Development Program

```
 1)  IN3:    CMP     R2 , # 17
 2)          BPL     IN4
 3)          MOV     TEMP , R2
 4)          CLC
 5)          ROL     TEMP
 6)          BCC     RND1
 7)          INC     R2
 8)  RND1:   ROL     TEMP + 2
 9)          BCC     RND2
10)          INC     R2
11)  RND2:   ROL     TEMP + 4
12)          BCC     RND3
13)          INC     R2
14)  RND3:   ROL     TEMP + 6
15)          BCC     RND4
16)          INC     R2
17)  RND4:   COM     R2
18)          ADD     R2 , TEMP
19)          MOV     TEMP , R2
20)  IN4:    RTS     PC
21)  TEMP:   .WORD   0 , 0 , 0 , 0
```

Test the device number, if greater than 14 return to the calling program via IN4 (lines 1, 2 and 20). Now we perform a left shift on TEMP (a giant 64 bit word). This is done in 4 steps (of 16 bits) through the

carry register (1 bit).

```
64        48       47       32        31        16        15        0
   TEMP+6            TEMP+4             TEMP+2              TEMP
c3             c3                 c2                  c1
```

TEMP = TEMP + (-1) * (TEMP + C4 + C3 + C2 + C1)

The initial value of TEMP is stored in register 2 and the carry register cleared (lines 3 and 4). Now the 4 shifts are executed and the resultant carrys added to register 2 (lines 5 - 16). We wrap it up (lines 17 and 18), move the low order 16 bits to register 2 (line 19), and return to where we came from (line 20). Space for the TEMP is created with the .WORD macro (line 21).

The output subroutine services these fifteen output devices -

1-8) digital to analog converters
9) red 16:1 select channels
10) green 16:1 select channels
11) blue 16:1 select channels
12) inversion register
13) red ALU (arithmetic logic unit)
14) green ALU
15) blue ALU

Through an unaccountable mental lapse on my part, the data buffers correspond

directly to the output devices - data buffers 1-8 control the A/D's, data buffer 9 controls the red 16:1 select channels, and so on. The first part of the output routine controls the A/D's -

```
1)  OUT:    CMPB    TMRX , # 11
2)          BPL     OUT 1
3)          MOVB    TMRX , R2
4)          DEC     R2
5)          MOV     R2 , @# LEWCHA
6)          MOV     6 (R0) , @# LEWOUT
7)          INC     R5
8)          RTS     PC
```

If the buffer number is greater than 8 branch to OUT1 (lines 1 and 2). If not calculate the channel address and move it to the control word LEWCHA (lines 3 - 5). Next move the data to the output word LEWOUT, set the done flag (register 5), and return to the calling program (lines 6-8).

The second part of the routine controls Don McArthur's 16:1 selects and inversion register -

```
1)  OUT1:   CMPB    TMRX , # 15
2)          BPL     OUT2
3)          MOVB    TMRX , R2
4)          SUB     # 11 , R2
5)          ASL     R2
```

NEA Development Program

```
6)          ADD     # DONOUT , R2
7)          MOV     6 (R0) , (R2)
8)          INC     R5
9)          RTS     PC
```

If the buffer number is greater than 12 branch to OUT2 (lines 1 and 2). If not calculate the output address (lines 3 - 6) -

output address = # DONOUT + 2 * (TMRX - 9)

Finally we transfer the data word to output address, set the done flag line, and return to the calling program (lines 7 - 9).

NEA Development Program

Part three of the routine is similiar; it controls Jeff Shier's arithmetic logic units -

```
1)  OUT2:   CMPB    TMRX , #20
2)          BPL     OUT3
3)          MOVB    TMRX , R2
4)          SUB     # 15 , R2
5)          ASL     R2
6)          ADD     # JEFOUT , R2
7)          MOV     6 (R0) , (R2)
8)  OUT3:   INC     R5
9)          RTS     PC
```

If the buffer number is greater than 15, game over, we return to the calling program via OUT3 (lines 1,2,8 and 9). If not calculate the output address (lines 3 and 5 ) -

output address = # JEFOUT + 2 * (TMRX-13)

Finally we output the data word, set the done flag, and return (lines 7-9).

## Data Buffers and Programming Techniques

In the program areas of memory are reserved for data buffers including -

1) EBUF - the enable buffer indicates whether the data buffer is active.

2) PBUF - the parameter list used by the timer and interpreter to access the data buffers.

3) TABLES - used to store prepared data, accessed with the input subroutine (IN).

4) DBUF 1 - 15 - sixteen data buffers containing sequences of command words which control the available input/output devices on the system.

EBUF

PBUF

DBUF 1

## Data Buffers and Programming Techniques

These data buffers become a seperate program which is linked to the main program by the system loader before execution. First we establish the globals identifying the labels common to both the main program and the data program -

```
.GLOBAL     TABLES, EBUF, DBUF
```

There are eight tables of sixteen words (8 X 16 = 128). The following sequence of code will reserve memory for the tables.

```
1)  TABLES:
2)  TBL1:
3)          . = TABLES + 20
4)  TBL2:
5)          . = TABLES + 40
6)  TBL3:
7)          . = TABLES + 60
8)  TBL4:
9)          . = TABLES + 100
10) TBL5:
11)         . = TABLES + 120
12) TBL6:
13)         . = TABLES + 140
14) TBL7:
15)         . = TABLES + 160
```

```
16)  TBL8:
17)          . = TABLES + 200
```

Note the first two labes are synonymous (TABLES and TABL1 lines 1 and 2) for convenience. After each table heading (TBL1, TBL2, etc) a block of sixteen words is reserved by setting the program counter (.) to the next heading or label (line 3, etc).

Tables are filled in as illustrated in the following example -

```
1)  TBL1:   .WORD   104210
2)          .WORD   177777
3)          .WORD   167356
4)          .WORD   156735
5)          .WORD   146314
6)          .WORD   135673
7)          .WORD   125252
8)          .WORD   114631
9)          .WORD   73567
10)         .WORD   63146
11)         .WORD   52525
12)         .WORD   42104
13)         .WORD   31463
14)         .WORD   21042
15)         .WORD   10421
16)         .WORD   0
```

## Data Buffers and Programming Techniques

This table contains the simplest bar patterns available on Don McArthur's 16:1 select modules.

line 1) represents a solid field

line 2) two horizontal bars

line 3) four horizontal bars

line 4) eight horizontal bars

line 5) sixteen horizontal bars

line 6) thirty-two horizontal bars

line 7) sixty-four horizontal bars

line 8) one hundred and twenty-eight horizontal bars

line 9) two vertical bars

line 10) four vertical bars

line 11) eight vertical bars

line 12) sixteen vertical bars

line 13) thirty-two vertical bars

line 14) sixty-four vertical bars

line 15) one hundred and twenty-eight vertical bars

line 16) two hundred and fifty-six vertical bars

Other tables are useful - shaded bar patterns, crosshatch patterns, and masks for example.

Following the tables is the enable buffer (EBUF), a short buffer of sixteen bytes (eight words) set 0 for an inactive buffer, and 1 for an active buffer.

## Data Buffers and Programming Techniques

```
1)  EBUF:   .BYTE   0,0,0,0,0,0,0,0
2)          .BYTE   1,1,1,1,0,0,0,0
3)          . = EBUF + 10
```

In the example only buffers 9,10,11 and 12 are active and the remainder inactive. The block of eight words is created (lines 1 and 2) and the program counter set to the next label (line 3).

Now we reserve memory for the sixteen data buffers as follows -

```
1)  DBUF:
2)  DBUF1:
3)          . = DBUF + 400
4)  DBUF2:
5)          . = DBUF + 1000
6)  DBUF3:
7)          . = DBUF + 1400
8)  DBUF4:
9)          . = DBUF + 2000
10) DBUF5:
11)         . = DBUF + 2400
12) DBUF6:
13)         . = DBUF + 3000
14) DBUF7:
15)         . = DBUF + 3400
16) DBUF8:
```

```
17)             . = DBUF + 4000
18)     DBUF9:
19)             . = DBUF + 4400
20)     DBUF10:
21)             . = DBUF + 5000
22)     DBUF11:
23)             . = DBUF + 5400
24)     DBUF12:
25)             . = DBUF + 6000
26)     DBUF13:
27)             . = DBUF + 6400
28)     DBUF14:
29)             . = DBUF + 7000
30)     DBUF15:
31)             . = DBUF + 7400
32)     DBUF16:
33)             . = DBUF + 1000
34)             . END TABLES
```

Again the first two labels (DBUF and DBUF1, lines 1 and 2) are synonymous. After each buffer heading (DBUF1, DBUF2, etc.) a block of one hundred and twenty-eight words is reserved by setting the program counter (.) to the next heading or label (line 3, etc.).

An example of a real data buffer follows -

## Data Buffers and Programming Techniques

```
1)  DBUF9:  .WORD   0 , 60.
2)          .WORD   10 , 31020
3)          .WORD   46
4)  L901:   .WORD   13 , 10421
5)          .WORD   46
6)          .WORD   40 , 0 , 777 , L901
7)          .WORD   47
```

The data buffer is filled with a sequence of command words used by the main program to control, in this case, the McArthur's red 16:1 select module. First the timing interval is set to 1 sec. (60 fields, line 1). The command word is 0, the interval is coded as 60. the period indicating a decimal (rather than octal) number. The command 10 sets the data equal to the octal number 31020 (line 2). Finally a 46 causes the data to be transferred to the buffer memory. The main program goes on to the next buffer and will not return to this buffer for another 60 interrupts or 1 sec. When it does return (to line 4) it adds the octal number 10421 to the data and transfers the sum to the buffer memory (line 5). Again the main program returns after 1 sec. I returns (to line 6) and finds a loop command - 40. Initially the counter is 0, the number of times through the loop will be 777 octal, and the data buffer pointer will be set back to L901. The main program will repeat lines 4 - 6, 777 octal times and then expire (line 7).

Review of command codes (in JBUF) -

00 - 0 , N ; set the timing interval (first word in PBUF)

## Data Buffers and Programming Techniques

interval = N where $0 < N < 200000_8$

- the interval is the number of fields the main program waits before returning to the data buffer for the next command.

01 - 1 , N ; add to the timing interval

interval = interval + N

02 - 2 , N ; subtract from the timing interval

interval = interval - N

03 - 3 ; complement the timing interval

interval = interval $\not\wedge$ $177777_8$

04 - 4 ; shift the timing interval right

interval = interval $\div$ 2

an interval of 1 sec becomes ½ sec.

05 - 5 ; shift the timing interval left

interval = interval * 2

an interval of 1 sec becomes 2 sec.

10 - 10 , N ; set the data word (fourthe word in PBUF)

data = N where $-1 < N < 200000_8$

11 - 11 ; increment the data word

data = data + 1 note: $177777_8$ + 1 = 0

12 - 12 ; decrement the data word

## Data Buffers and Programming Techniques

data = data - 1  note: 0 - 1 = $177777_8$

13 - 13 , N ; add to the data word

data = data + N

if $177777_8$ - data < N then

data = N -

14 - 14 , N ; subtract from the data word

data = data - N

if N > data

data = $\overline{2^{16}}$ (N - data)

15 - 15 ; complement the data word

data = data $\forall$ $177777_8$

16 - 16 ; shift the data word right

data = data $\div$ 2

17 - 17 ; shift the data word left

data = data * 2

20 - 20 ; rotate the data word right

15 0

bit N becomes bit N-1

bit 0 becomes bit 15

21 - 21 ; rotate the data word left

15 0

bit N becomes bit N+1

bit 15 becomes bit 0

## Data Buffers and Programming Techniques

22 - 22 , N ; bit clear, data word with N

data = data $\wedge \overline{N}$

old data 0 110 101 011 010 111 - 065327 (8)

N 0 100 001 101 100 010 - 041542 (8)

new data 0 010 100 010 010 101 - 024225 (8)

23 - 23 , N ; bit set, data word with N

data = data $\vee$ N

old data 0 110 101 011 010 111

N 0 100 001 101 100 010

new data 0 110 101 111 110 111 -065767 (8)

24 - 24 , N ; XOR , data word with N

data = data $\veebar$ N

old data 0 110 101 011 010 111

N 0 100 001 101 100 010

new data 0 010 100 110 110 101 - 024665 (8)

30 - 30 , N1 , N2 ; get data with N1 = 1 to 8 (device number)

N2 = 1 to 16

- register 2 becomes the value contained in table N1 , entry N2.

- with N1 = 9 to 12 (device number)

-register 2 becomes the value sensed by A/D N1.

Data Buffers and Programming Techniques

- - with N1 = 13

- register 2 becomes the value sensed by the real time interface (device number 13).

- with N1 = 14

- register 2 is set by the random number generator

- note if N1 = 9 to 14 then N2 is not used, and the command takes the form - 30 , N1

31 - 31 , N1 , N2 ; get new data and add to old data

combines commands 30 and 13

32 - 32 , N1 , N2 ; get new data and subtract from old data

combines commands 30 and 14

33 - 33 , N1, N2 ; get new data and bit clear with old data

see commands 30 and 22

34 - 34 , N1 , N2 ; get new data and bit set with old data

see commands 30 and 23

35 - 35 , N1 , N2 ; get new data and XOR with old data

combines commands 30 and 24.

40 - 40 , N1 , N2 , LABEL ; loop command

- the program is set to repeat a sequence of commands where -

N1 = 0 , used as a counter by program

N2 = 0 - 177777 , number of repetitions

LABEL , pointer to top of loop

- example of single loop -

Data Buffers and Programming Techniques

1) LABEL1: command

2) command

3) 40 , 0 , 100. , LABEL 1

- example of nested loops -

1) LABEL1: command

2) LABEL2: command

3) command

4) 40 , 0 , 100. , LABEL2

5) 40 , 0 , 100. , LABEL1

- example of multiple loops -

1) LABEL1: command

2) command

3) 40 , 0 , 100. , LABEL1

4) command

5) command

6) 40 , 0 , 100. , LABEL1

46 - 46 ; output command

the data word contained in the parameter list is transferred to the buffer memory and the main goes on to the next data buffer.

47 - 47 ; the exit command, the end, finis.

## Data Buffers and Programming Techniques

Now for some simple (minded) examples of programming techniques. The easiest devices to program are the D/A converters (output devices 1 - 8) which translate a number into a control voltage as follows -

1777XX = + 10V

1000XX = 0V

0XX = - 10V

XX - low order bits 0 - 5 not used

### A Simple Ramp

```
1)          0 , 60
2)          10 , 0
3)          46
4)  L101:   13 , 100
5)          46
6)          40 , 0 , 1776 , L101
```

+ 110V

0V

-10V

$t_0 = o$ $t_1 = 1024$ seconds

Data Buffers and Programming Techniques

$\Delta t$ = 1 sec

$\Delta V$ = 20/1024 V

duration = 1024 seconds

amplitude = 20V pp around 0V

In line 1 we set the timing interval to 60 fields or 1 sec. We set the D/A to -10V (line 2) and output this value to the D/A (line 3). Now we construct a loop (lines 4 to 6). The label L101 sets the top of the loop, the commands to be repeated are add $100_8$ to the data and output the new value to the D/A. This is repeated $1776_8$ times.

A simple method for understanding a loop is shown in this table -

| # repetitions | old data | new data |
|---|---|---|
| 1 | 0 | $0 + 100_8 = 100_8$ |
| 2 | $100_8$ | $100_8 + 100_8 = 200_8$ |
| 3 | $200_8$ | $200_8 + 100_8 = 300_8$ |
| 4 | $300_8$ | $300_8 + 100_8 = 400_8$ |
| 5 | $400_8$ | $400_8 + 100_8 = 500_8$ |
| 6 | $500_8$ | $500_8 + 100_8 = 600_8$ |
| 7 | $600_8$ | $600_8 + 100_8 = 700_8$ |
| 8 | $700_8$ | $700_8 + 100_8 = 1000_8$ |

## Data Buffers and Programming Techniques

### A Repeating Sawtooth

```
1)          0 , 1
2)  L101:   10 , 0
3)          46
4)  L102:   13 , 10000
5)          46
6)          40 , 0 , 17 , L102
7)          40 , 0 , 10000. , L101
```

+ 10V

0V

- 10V

$t_0 = 0$  $t_1 = 16$ fields

$\Delta t = 1$ field

$\Delta V = 1.25V$

frequency - approx 4 Hz

amplitude - 20V pp

This could be a negative going sawtooth -

```
1)          0 , 1
2)  L101:   10 , 177700
3)          46
```

```
4)   L102:   14 , 1000
5)           46
6)           40 , 0 , 17 , L102
7)           14 , 7700
8)           46
9)           40 , 0 , 10000. , L101
```

In both examples a pair of nested loops is used, loop 101 repeats the basic wave form 10,000 times (lines 2 - 9) and loop 102 builds the waveform (lines 4 - 6).

There is a simpler way of building a sawtooth which uses the wrap around feature of the CPU's arithmetic logic unit -

```
1)           0 , 1
2)           10 , 0
3)           46
4)   L101:   13 , 10000
5)           46
6)           40 , 0 , 20 , L101
7)           40 , 0 , 10000. , L101
```

This produces exactly the same waveform as the first example. On the sixteenth repetition we get $170000_8 + 10000_8 = 0$, which completes the inside loop. The outside loop remains the same.

## Data Buffers and Programming Techniques

### Repeating a V Triangle

```
1)          0 , 1
2)          10 , 0
3)          46
4)   L101:   13 , 10000
5)          46
6)          40 , 0 , 17 , L101
7)          13 , 7700
8)          46
9)          14 , 7700
10)         46
11)  L102:   14 , 10000
12)         46
13)         40 , 0 , 17 , L102
14)         40 , 0 , 1000., L101
```

+ 10V

0V

- 10V

$t_0 = 0$ $t_1 = 32$ fields

$t = 1$ field

$V = 1.25V$

frequency - approx. 2 Hz

amplitude - 20V pp.

Again the timing interval is set to 1 field and D/A converter set to 0V (lines 1 - 3). The outside loop (lines 4 - 14) repeats the waveform 1000 times. The first inside loop builds the positive going slope of the triangle (lines 4 - 6). Then, the peak of the triangle is formed (lines 7 - 10). The second inside loop builds the negative slope (lines 11 - 13).

## Data Buffers and Programming Techniques

### Making a Sine Wave

First examine this table of numbers

| | | |
|---|---|---|
| 1) | 0 | +100 |
| 2) | 100 | +200 |
| 3) | 300 | +400 |
| 4) | 700 | +1000 |
| 5) | 1700 | +2000 |
| 6) | 3700 | +4000 |
| 7) | 7700 | +10000 |
| 8) | 17700 | " |
| 9) | 27700 | " |
| 10) | 37700 | " |
| 11) | 47700 | " |
| 12) | 57700 | " |
| 13) | 67700 | " |
| 14) | 77700 | " |
| 15) | 107700 | " |
| 16) | 117700 | " |
| 17) | 127700 | " |
| 18) | 137700 | " |
| 19) | 147700 | " |
| 20) | 157700 | " |
| 21) | 167700 | +4000 |

```
22)  173700  +2000
23)  175700  +1000
24)  176700  +400
25)  177300  +200
26)  177500  +100
27)  177600
```

The table is coded as follows -

```
 1)        0,6.
 2)        10,0
 3)        46
 4)        13,100
 5)        46
 6)        13,200
 7)        46
 8)        13,400
 9)        46
10)        13,1000
11)        46
12)        13,2000
13)        46
14)        13,4000
15)        46
```

```
16)  1101:   13,10000
17)          46
18)          40,0,14.,1101
19)          13,4000
20)          46
21)          13,2000
22)          46
23)          13,1000
24)          46
25)          13,400
26)          46
27)          13,200
28)          46
29)          13,100
30)          46
```

+10v

0v

-10V

$t_0 = 0$ $t_1 = 156$ fields

t= 6 fields
V varies

This is too much work for a sine wave, improvements will be made.

At this point development stops and so does the report.

# Appendix A- LSI-11 Operation Codes

Legend

| | |
|---|---|
| B | = 0 for word/ 1 for byte |
| SS | = source field- 6 bits |
| DD | = destination field- 6 bits |
| R | = general register- 3 bits- 0 to 7 |
| XXX | = offset- 8 bits- +127 to -128 |
| N | = number- 3 bits |
| NN | = number- 6 bits |
| $\wedge$ | = AND |
| V | = inclusive OR |
| $\not\vee$ | = exclusive OR, XOR |
| $\sim$ | = NOT |
| s | = contents of source |
| d | = contents of destination |
| r | = contents of register |
| | = becomes |
| X | = relative address |
| % | = register definition |
| , | = concatenated with |
| N | = sign condition code, 1 bit |
| Z | = zero condition code, 1 bit |
| V | = overflow condition code, 1 bit |
| C | = carry condition code, 1 bit |

## Appendix A- LSI-11 Operation Codes

| Mnemonic | OpCode | Instruction | |
|---|---|---|---|
| CLR(B) | B050DD | clear | d←0 |
| COM(B) | B051DD | complement | d←∿d |
| INC(B) | B052DD | increment | d←d+1 |
| DEC(B) | B053DD | decrement | d←d-1 |
| NEG(B) | B054DD | negate | d← -d |
| TST(B) | B057DD | test | sets status bits |
| | | | |
| ROR(B) | B060DD | rotate right | →C,d |
| ROL(B) | B061DD | rotate left | C,d← |
| ASR(B) | B062DD | shift right | d-2 |
| ASL(B) | B063DD | shift left | 2*d |
| SWAB | 0003DD | swap bytes | |
| | | | |
| ADC(B) | B055DD | add carry | d+C |
| SBC(B) | B056DD | subtract carry | d-C |
| SXT | 0067DD | sign extend | 0 or -1 |
| | | | |
| MFPS | 1067DD | move byte from PS | d←PS |
| MTPS | 1064SS | move byte to PS | PS←d |
| | | | |
| MOV(B) | B1SSDD | move | d←S |
| CMP(B) | B2SSDD | compare | s-d, sets status bits |
| ADD | 06SSDD | add | d←s+d |
| SUB | 16SSDD | subtract | d←d-s |

## Appendix A- LSI-11 Operation Codes

| Mnemonic | OpCode | Instruction | |
|---|---|---|---|
| BIT(B) | B3SSDD | bit test | s∧d, sets status bits |
| BIC(B) | B4SSDD | bit clear | d← (~s)∧d |
| BIS(B) | B5SSDD | bit set | d← s∨d |
| XOR | 074RDD | XOR | d← r⊻d |
| MUL | 070RSS | multiply | r← r*s |
| DIV | 071RSS | divide | r← r÷s |
| ASH | 072RSS | arithmetic shift | |
| ASHC | 073RSS | shift combined | |
| FADD | 07500R | floating add | |
| FSUB | 07501R | floating subtract | |
| FMUL | 07502R | floating multiply | |
| FDIV | 07503R | floating divide | |
| BR | 000400 | branch unconditional | |
| BNE | 001000 | branch if ≠ 0, Z= 0 | |
| BEQ | 001400 | branch if = 0, Z= 1 | |
| BPL | 100000 | branch if plus, N= 0 | |
| BMI | 100400 | branch if minus, N= 1 | |
| BVC | 102000 | branch if overflow clear, V= 0 | |
| BVS | 102400 | branch if overflow set, V= 1 | |
| BCC | 103000 | branch if carry clear, C= 0 | |
| BCS | 103400 | branch if carry set, C= 1 | |

## Appendix A- LSI-11 Operation Codes

| Mnemonic | OpCode | Instruction |
|---|---|---|
| BGE | 002000 | branch if $\geq 0$, $N \oplus V = 0$ |
| BLT | 002400 | branch if $< 0$, $N \oplus V = 1$ |
| BGT | 003000 | branch if $> 0$, $Z \vee (N \oplus V) = 0$ |
| BLE | 003400 | branch if $\leq 0$, $Z \vee (n \oplus V) = 1$ |
| BHI | 101000 | branch if higher, $C \vee Z = 0$ |
| BLOS | 101400 | branch if lower or same, $C \vee Z = 1$ |
| BHIS | 103000 | branch if higher or same, $C = 0$ |
| BLO | 103400 | branch if lower, $C = 1$ |
| JMP | 0001DD | jump PC $\leftarrow$ d |
| JSR | 004RDD | jump subroutine |
| RTS | 00020R | return from subroutine |
| MARK | 0064NN | mark |
| SOB | 077RNN | subtract 1 & branch if $\neq 0$ |
| EMT | 104*** | emulator trap |
| TRAP | 104*** | trap |
| BPT | 000003 | breakpoint trap |
| IOT | 000004 | input/output trap |
| RTI | 000002 | return from interrupt |
| RTT | 000006 | return from interrupt, inhibit trap |

## Appendix A- LSI-11 Operation Codes

| Mnemonic | OpCode | Instruction | |
|---|---|---|---|
| HALT | 000000 | halt | |
| WAIT | 000001 | wait for interrupt | |
| RESET | 000005 | reset bus | |
| NOP | 000240 | no operation | |
| CLC | 000241 | clear C | C←0 |
| CLV | 000242 | clear V | V←0 |
| CLZ | 000244 | clear Z | Z←0 |
| CLN | 000250 | clear N | N←0 |
| CCC | 000257 | clear all | |
| SEC | 000261 | set C | C←1 |
| SEV | 000262 | set V | V←1 |
| SEZ | 000264 | set Z | Z←1 |
| SEN | 000270 | set N | N←1 |
| SCC | 000277 | set all | |

NEA Development Program

Summary

As obvious the program fails to satisfy the original design criteria. The program is not interactive. It is not concerned with graphic design of composition. It cannot reprogram itself in response to external stimula. However it's not a total loss - the basic groundwork is complete. The elements of the language outlined in Appendices A & B are still beyond the uniniated. But, from these elements a higher level language will be created. This new language will facilitate the dialogue between the artist and the program allowing him to create the images and sequences of images in a language he understands - a graphic design language.

The present program runs in batch mode. That is, the data must be prepared before the program is run. Then the main program and the data are linked, loaded, and finally processed. If the results are not quite as expected (the norm rather than the exception) then the whole process must be repeated - hardly instant gratification.

Again, this mode of operation is only temporary; real time interaction will be added by expanding the interpreter routine to include the ability to listen and talk back.

If the program listens and talks then it can learn. Combining the random number generator with a simple algorithms for analyzing images we can endow the program with a personality (or several personalities).

But what is the language spoken by the artist and the program? That's a question for continuing research.

## NEA Development Program

Proposed program development includes -

1. Adding a terminal input and output routine to the interpreter
2. Adding macro commands invoking command word sequences.
3. Adding a data buffer to output device cross-reference table.
4. Adding editing commands to modify data buffer contents in real time.
5. Adding condition branch commands.
6. Designing a higher level language based on the elements and attributes of graphic design.
7. Expanding the manual of programming techniques.
8. Creating a personality for the program - anthropomorphization of the program.

And finally I will attempt to keep up with the break-neck speed of our hardware development.

EXPERIMENTAL TELEVISION CENTER LTD.
164 COURT ST.
BINGHAMTON NEW YORK 13901
607-723-9509

# A Computer-Based Video Synthesizer System
## Software

A paper presented at the 'Design/Electronic Arts' conference in March 1977

Walter Wright

Experimental Television Center Ltd.
Binghamton, New York

This project is supported in part by the National Endowment for the Arts and the New York State Council on the Arts.

# A COMPUTER-BASED VIDEO SYNTHESIZER:
# PART II: SOFTWARE

Walter Wright

First, I'm not a physicist. And second, I'm going to try and get at software from a little different angle than the ordinary. So the title of this little section is supposed to be "Software for a Computer-Based Video Synthesizer." "Synthesizer" implies for me a collection of programmable to modules for processing and/or creating images, and of course sound. "Video synthesizer" in particular is that which then creates images. "Computer-based" implies that a computer programs these modules which together comprise the synthesizer. And "Software" means that somebody has to program the computer. So, beginning at the end of the process, I'd like to deal first with the image. The image is essentially the two-dimensional surface of the screen. It's also a wave form, containing both spatial and temporal information. In fact, it's interesting to note that a proportion of this wave form is not seen, and is not really part of the image at all. And in most systems it's stripped away and we forget about sync until the very end as a final parting gesture to the newly processed image sync gets pasted back on. There may be some possibilities for sync processing. I know that Woody has drift modules and Bill Etra has a horizontal centering-or-other control on the Rutt/Etra. So you can think about that. Returning to the image and the TV screen, again the image is a surface, a light-emitting surface. It's of rather low resolution approximately five hundred and twenty-five horizontal lines per frame and half of that per field. It resurrects itself every sixtieth of a second, that is for a field. And it's retrievable, therefore, we can record it, or we can create seeming movement. And I think it's important to remember that the image doesn't move anywhere. The tape moves. Instead the image is replaced over and over again. I think that's

interesting to remember. We had a lot of discussion about simulated motion, but here are other possibilities for the creation of sequences of images; for example, fast sequential switching.

andrelated areas that we've been getting into recently at the Experimental Television Center. Of course we can create the illusion of fluid motion, but as I said we can do a lot of other things as well. Different images related or unrelated by composition, impact, or whatever can be inter-leaved, as I described field by field, frame by frame, and multiples thereof. I mention this in passing because I feel in discussing software we must avoid the trap of imitating, borrowing, and misapplying techniques, theories and other trappings of related technologies. We must really find out what makes TV tick, you know. Just be aware of this dinosaur syndrome. I think there are useful similarities and comparisons that exist between TV and photography, TV and film, TV and electronic music, and amongst the whole gaggle of electronic arts in general.

I have recently discovered, or re-discovered, that the video image is subject to some of the basic rules of composition. Kind of a revelation, and not too obvious in broadcast TV. Anyway, the elements or the attributes of design, or whatever you want to call them, seem to apply. Think about an image in terms of a point or grain, having to do with the resolution and the surface material of the screen. The Trinitron monitor over there is a matrix of over 100,000 red, green, and blue dots. Think about the images made up of lines, like a raster line. Or on a manipulation system, is a set of lines that can form angles to each other. Think of an image as texture combining the above, texture derived from noise, or texture derived from a high speed oscillator module. Anyway, other design elements like area are defined by lines or by texture. Elemements such as value, luminance contrast, saturation, hue-all of these define color. Or, as some of the participants mentioned, RGB components plus luminance. These are some of the design elements of the image.

The field, which is one-sixtieth of a second long, contains two hundred and sixty-two and a half lines. Now that's kind of like the basic image unit, although Don uses the frame. You can, of course, group successions of fields to form larger units. Like the frame or whatever. And of course there are those things that result from the grouping together of points and lines and textures and fields.

Now let's look at groups of elements together in a field or frame. You can talk about things like density and balance and imbalance and symmetry and asymmetry, focal points, proportion, scale, depth, object/field relationships, pair and form and others that escame me.

I'm sure they're all there. And finally those considerations resulting from the succession of the fields or frames. Something that Laurie Spiegal was talking about--harmony, rhythm, counterpoint, translation, rotation, ex-and implosion, warping, bending, convolution, all of these apply to the succession of frames. This may seem like a peculiar way to approach the design of software, but I'll just let that hang for a minute and go on to micro-computers, or computers in general.

I think we are kind of very unfair to our computers ? ? maintaining dossiers on subversives. Amongst other things, this approach lacks, I think, imagination and creativity. It raises a difficult problem at this point in my writing, does a computer imagine or create? And, can it be used imaginatively and creatively? That's probably a better way to put it, and gets me out of the problem. Most of the software we've heard described kind of runs on high school mathematics or at some undergraduate level. And I don't think that a graduate degree, from my experience, has anything to do with imagination or creativity. And therefore I've looked elsewhere for my inspiration in programming. Several participants have mentioned that learning requires feedback. As in any relationship, the computer and the synthesizer must interact. They have to be able to talk to each other, and so on. Granted the

computer may never learn to be imaginative or creative, but there's one attribute of the artist that it can share, and that's unpredictability. Or more exactly, predictable unpredictability. Joel Chadake mentioned the use of a random number generator to create a situation in which the artist and the computer can improvise together. In the simplest case software is created which causes the computer to reprogram the synthesizer at predetermined intervals. In this case the computer responds to the clock, and away it goes. A good synthesizer I think should contain modules not only to synthesize or put together an image, but also modules to analyze and to break apart an image. Modules that provide information to the computer concerning, believe it or not, things like texture, value, scale, balance, symmetry, rhythm, etc. And this can be done in several ways. First by monitoring the programming of the synthesizer, that is the patching together of the modules or the route that's taken by the image through the synthesizer. And second by analyzing the image that's actually output by the synthesizer. Remember feedback? Now we can make the computer respond in a less willy-nilly manner than in the random generation scheme. In fact, the software might even be able to allow the artist to identify for the computer what is a good and what is a bad image. That's kind of overstating it; let's say what is a more desirable and what is a less desirable image or sequence of images. Or another way, what is a more probable or a less probable image or sequence of images? Finally, the computer could probably be programmed to reprogram itself. Think about that.

A word should be said about micro-processors. One of their big features is that they have reduced the cost ratio between the CPU, which is that central control unit, and the total system from about one-to-ten to about one-to-one hundred. So a twenty dollar chit means a two thousand dollar computer. Another ratio that I think is important to remember is that there's much more effort that goes into software development than hardware development. Something like ten-to one.

Anyway, since I got back to software somehow. I think the software must be concerned with composition, first of all. First must be the elements and the attributes of design. Second, the software must be capable of both analyzing and synthesizing images. Third, the software should have a mind of its own. It should be capable of reprogramming the synthesizer and reprogramming itself. Fourth, it must interact in real time with the artist. And fifth, it has to cost almost nothing.

I got this brochure from Carl Geiger and Rod Fountain of Synapse and it's kind of interesting because Rod has been working on a program for the Altair which has a lot of features in it--and is moving in the direction that I've described. The program is called HARPO. It allows one to generate control voltages that can run asynchronously with each other. It allows one to define these as a score and allows one to interact with your score in real time.

Now, here's a diagram (figure one) which describes a program that we have implemented for Woody Vasulka. The columns sticking up vertically are stacks of data which define what's going to happen in the way of generating control voltage, or they supply digital information to a synthesizer module. So it could be generating a control voltage, reading the intensity value at a point on the raster, whatever. For each device or each module in the system there's a unique data stack. Now below the data stacks you see a set of blocks that contain information regarding the arrow that's riding up and down the stack (which is a pointer telling me where I'm at in my data at the moment), the address of the data, the device I'm using, and a timing counter. The stacks can just operate in parallel under their own timing. The timing counter decides what the duration of a particular control function is going to be, and the counter just counts off frames and allows it to go on for that duration or length of time. The arrow or pointer points to a control word which will cause a branch to a sub-routine in turn will generate the information for module in

the synthesizer. The large arrow that goes around like a belt comes from the control module, which keeps track of all the housekeeping. And it is synchronized to the vertical drive pulse, in Woody's system it's a thirtieth of a second. It's not the field rate which is the smallest unit; its that or the frame rate. Every one thirtieth you get an interrupt and around it goes. The program goes through the little boxes at the bottom, decides if something current is supposed to be happening, and does it. If it needs data it locates the pointer, fishes the data out of the data stack and proceeds on its way. So the computer, then, can execute all of these things kind of asynchronously, in parallel. Here's a bubble diagram (figure two), let's deal with the little circles first. The control program is sitting in there in the middle. The other little circles also represent programs. I thought I'd just show this because it's not a hierarchical structure it's a network. Anyone of those programs can call anyone of the programs. And you can move on any path in within the network. So you can have circles that take care of synthesizer modules. You can have things that just operate internally, inside the computer, such as Library routines, transforming numbers, and of course the input and output routines. I wanted to show you a network and that will relate to the final diagram (figure three) here. Let's assume that one module does one transformation. If we lay out modules on two sides of the graph we have a Pinboard. Connect one to two, two to three, three to four by placing a pin at the appropriate cells in the matrix. Or what we can do is, we can define the way our system is going to behave by not indicating a definite yes or no, but by indicating a probability. In other words, if we have just executed or if we have just passed the image through module one, what is the probability that it's going to go to module two, three, four or five? We can say that it's very unlikely that it's going to go to module five, but it's highly likely that it's going to module two. After it's been to two we can say that it's very unlikely that it's going to go anywhere else but seven. And we can code this information on the matrix, and this will cause the program to do whatever

it has to do in terms of patching the system. This is what I meant by predictable unpredictability. We've gained a kind of control over our system, which is not totally predictable but can identify for us certain groups of images that will be more likely than others. So we'll go through a sequence of images which will be related. And you may not be able to predict exactly what each image will look like, but you will be able to predict certain properties of them. Say that certain of the modules always generated symetric images. I could, by making sure that those modules got connected together guarantee symmetry was maintained over a certain period of time. Now you can see that it's possible to fill in both sides of the matrix and talk about moving from module two to module one and from module seven back to module two. That doesn't have to be the same probability going one way or other in the system. Here's where the artist interacts most effectively with the system by controlling these probabilities in real time. In a longer composition what he would do is identify the probability of certain kinds of images at a certain point and change that probability over time to switch to another group of images as being more probable. Then you can apply this not only to the patching network of a synthesizer, but we could apply this to various other elements in the composition of the image, the performance of a particular module, the frequency range that an oscillator is going to go through, what frequency it's likely to start off at. You could use this approach in many ways. And it's an approach, again, which uses a small amount of data to generate a large effect over a long period of time. So it's very useful.

I want to mention a couple of other possibilities. Another technique I think could be used in programming is the idea of the conditional branches which are inherent in assembly languages. That is the "if" condition. If condition 'one' (density, balance, grey value, whatever you want to measure) is true then we're going to create condition 'two'. We're going to change a probability. We are going to alter a control parameter. We're going to take a

coffee break, whatever. Other types of conditional branches then that we can build from those are like not just the "if" "then", we can do the "if" "then" "else". And in most languages it's possible to nest these, although I know some people don't like loops. And you can do things like "if then", "ifthen", "if-then", "else", "else", "else". Anyway, I think that's all I have to say about software. Any questions?

QUESTION PERIOD

QUESTION: You've got elements in your probability matrix, but what do they represent? Do they represent keys or color values or . . .?

WRIGHT: Well, in that case I was just considering PATCHING. If you had like a switching matrix controlled point by point, you could patch one module into another and route the signal. So I was just saying, is it likely that after its having passed through a keyer, would it be likely in Dan Sandin's system to go to an adder/multiplier, and the answer is yes. The signal from the keyer would be likely to go there, that would be a good thing. But you may not want that at a certain point in the composition. You may not want the hard edge. You might want to do something else. So you would turn off the keyer by lowering its probability of being patched in.

QUESTION: But you're talking about manually controlling these effects, so it makes sense to switch them in this way.

WRIGHT: I was just assuming this switcher.

QUESTION: O.K. But you're talking about stuff--that this is the kind of design of what you're doing. Or is this--is this what is manifesting in these tapes you're showing?

WRIGHT: No, it doesn't have anything to do with the tapes I'm showing. The tapes don't have any computer in them at all.

QUESTION: Oh, O.K., so the tapes are feedback? In general?

WRIGHT: No, the tapes are background. It's kind of where I've been and an idea of where I want to go, you know, with using those kinds of systems. This is still the Jones colorizer.

QUESTION: Uh huh. O.K. Thank you.

WRIGHT: Tom.

TOM: It would seem to me a third graphic on the opaque project (figure three) that there's some correlation between what you're talking about and what Don spoke about earlier. And perhaps this is the product of your collaboration. Particularly, it would seem like you're anticipating the kind of software that would be used in a system such as the type that he outlined. Is that what--are you anticipating a way of programming this pattern generator that he's prototyping now?

WRIGHT: I think it's kind of a little bit even more than that. We're trying to approach the design of the hardware and software from the outside of the circle and work inwards. And so what I'm trying to do is develop some environment in which I'm going to design the software. I'm trying to get my head into a space where I can make it flexible enough that it's going to be able to deal not only with that but could be used for a frame buffer. It could be used for, you know, position to voltage. It could be used for other modules. You see, Don McArthur has tried to maintain this in the design of his hardware by providing that element buss, which you can plug anything into. So I've tried to make the software in such a way that it leaves the general processes defined but leaves open the number of data stacks used or the

actual kinds of device that are used. I can't predict exactly what's going to turn up.

TOM DEWITT: Well, I'm really confused now. You're talking about the element buss which is as I understood it part of the video portion of the hardware architecture. But this programming construct would be implemented in the computer side.

WRIGHT: Yes.

TOM: And judging from--I'm not sure that I fully comprehend his (Don McArthur's) presentation--it would appear that matrix switching that you're postulating would be very useful in the context of being able to generate a wide variety of patterns from these multiplexers that he . . .

WRIGHT: Right.

TOM: . . . that he uses. And so, you know, these--you would be able to quickly review for the observer all the different possibilities that he says that this system could produce. You could program an event that would show us these patterns, and it would just sort of evolve. We could just sit back and watch it. Is that something that will happen? As you move from outside the circle toward the center? Where it's real.

WRIGHT: I think so. I think what you're describing is kind of an ideal situation. And maybe the final remark here is that the area of concern in designing software has got to be the language between the artist and the computer. And I think there are a number of things that we have to watch out for here. First of all, this language must relate directly to the creation of images and generate the data therefrom. The language

must be able to direct the flow or the succession of fields, it must be able to generate individual fields, it must be able to identify the quality of an image or a group of images, it must be able to create new images, and must be able to create a score. That's not exactly what you said. But what you said is something like being able to use the system to look at a broad range of images and then perform a selection process to arrive at some kind of conclusions about what images you like and don't like. Is that right, Tom? You're saying that the system could be used that way.

TOM: Well, I think what Don McArthur is going to build is something we'll understand once we see it running.

WRIGHT: You can see part of it running now.

TOM: Are the Vasulkas using that kind of control system:

WRIGHT: Um hum. (Yes)

TOM: Is this under computer now?

WRIGHT: Um hum (Yes)

TOM: Yeah. So we see a lot of variety and I'm sure that there are moments when we want a reference back, and then juxtapose those with other moments that at this point are occurring at a more random programming than an artist might desire for some particular effects. So then we can go back into that particular point in time which might be kept track of on a

clock that we're watching and start juxtaposing certain patterns that the thing makes with each other based on some other idea that's in the head when we come in. Yeah, yeah-yeah, that's it.

WRIGHT: O.K., George. (Chaik in)

GEORGE: Yeah. I was struck by the correlation between your probabilistic state transition table and I'm not sure whether you're aware of it, but you should be. That's why I'm bringing it up. You should be made aware of it if you're not. Between that and at the very foundation of verbal language the connection between phonemes and morphemes. It's been--it was demonstrated by Selig Harris that morphemes, the basic meaning element of language, are related to phonemes, the basic sound elements of language, by a stochastic Markoff-like process, which is inherent in that table. It seems very exciting that you are building a machine that at least has a capability, the possibility, of beginning to play with the formation of visual morphemes, because of that correlation.

WRIGHT: That's the general idea. I got the idea from architectural design--locating rooms by a probability matrix. I think those matrixes are used in a number of areas and fields in programming. It's not--I presume they would occur in a lot of places, like your spiral. Any other questions?

GEORGE: (Yes)

QUESTION: I just want to make sure I understand...the way I gather it from what you've said is that you are using a digital machine, a computer of some sort, which is on a frame by frame basis creating patches between some number of modules, and is also at least in some instances, also providing control data, which are telling those modules how bright, how much color to cut out, etc. Is that the basic . . .

WRIGHT: Yeah, that was the example that I used.

QUESTION: O.K. The question is, how many modules are you really talking about? And in terms of this state diagram that was drawn, was projected before. What percentage or what, you know, fraction of total conceivable image space are we really talking about with a system like this. In other words, how versatile is it in terms of approaching the totality of all conceivable images that . . .

WRIGHT: I don't think we have the number of images equal to the, you know, number of atoms in the universe or whatever. There, I don't really want to answer that. I can simply point out how to arrive at that for one's particular situation. And it's usually a cost or an economic factor. You can only afford to build so many modules, and as Don McArthur says, he has some ideas how to choose effectively the right ones. And in terms of the program you have only so much memory space, and if you are using buffers you have so much mass memory space. So you can only do, you know, have a hundred of those data stacks or ten of those data stacks or whatever fits inside the box that you've got. That's I think the best answer that I could give. What I try to do is propose the design in such a way that it is upwardly expandable.

QUESTION: Well, do you get to a point in trying to upwardly expand the system where the number of modules that have to be built becomes excessive to the point where the approach of using analog processing modules begins to be very cost ineffective:

WRIGHT: Yeah, probably.

CHUCK (KENNEDY): (This question not quite audible about polling modules to find out if they're in use).

WRIGHT: Sure, the program I've got with Woody, I think I just set this up in the very beginning by entering which modules are in use and which are not. And then the polling order is simply, you know, from top to bottom. Yeah, it's quite conceivable to down and up modules interactively by just including a cross reference table.

QUESTION: (Again, inaudible--concerning difference between hardware and software).

WRIGHT: Yeah, there's this kind of grey area between software and hardware, you know, like, do you have a hardware modem or do you have a software modem. You know, things like that. The circuits that disappear into thin air. What Don McArthur was referring to is the fact that these circuits pop up in software now, you know. And if you've had to pay five to ten dollars for a micro-processor chip you may even see more of this happen.

LAURIE SPIEGAL: I had just thought of slightly clarifying something Tom DeWitt was getting at in his question. And that is, that is a basic orientation for this system. It sounded like it, when you were talking about it, that it's essentially oriented toward the production of a continuous creative process with fluctuating probabilities, etc. And Tom is asking about its potential as a system for a specific designed entity with fixed, predetermined relationships among things. And that these are in fact two different projects of composition which may be compatible within one system in various ways, and there may be many grey areas in between but that there may be fundamental logical differences between the system which is designed to have continuous

generation processes, and one which is designed to produce specific editable and selectable effects. And I was wondering how you felt about--if you understand that kind of polarity in terms of compositional method that Tom was getting at.

WRIGHT: I guess I didn't understand Tom's question too well. I think the first diagram which just shows those stacks of data, you know, parallel, that's kind of oriented towards, I forget whether it's the former or the latter, approach that you talked about. That's in a sense scorable, scriptable, previsualizable. In a sense those data stacks can be built up in absolute form. What is said to happen on frame one hundred and ninety-three will in fact happen to exactly that module at exactly, you know, that time. And then the second one that I put in was a chance to expand the total environment of the program. O.K.? So the interaction that the artist can have, I should think, would allow him to utilize both modes of operation.

LAURIE: Yeah.

WRIGHT: I think what Tom said is something even more interesting in the sense that you could put the machine into a mode where it will generate just a whole universe of images, totally unexpected, and you will just sit there and say, reject, thumbs-up, thumbs-down, thumbs-up. And it'll go back then and play from that set and start--and then you can analyze, if you have the right modules, why that set, what are the characteristics of that set and proceed from there. That's the kinds of interaction I would like to see in a program.

LAURIE: And then perhaps even a program above that could create counterpoint among the things you'd selected, and etc. Yeah. O.K.

QUESTION: (Inaudible--concerning software development).

WRIGHT: I don't know. I've--I can't really say--I was going to describe a few simple algorithms that I've used in graphic programming. You know, Cartesian coordinate type plotter programming, and generating either line images or point images in a frame. And then being able to perform analysis of that image. For instance, it's very easy to, say, propose an axis of symmetry and then test for it if you've got a matrix of points in Cartesian coordinates, you know? And then you can vary that axis or move it and test for symmetry again and see if it happens. If it doesn't you can force it. Or you can create asymmetry by picking up the points on the matrix or lines and transposing them and overlapping them, and proceeding from there to generate new images. I think that--I didn't want to get too specific in those kinds of algorithms. We don' have them yet. And but I've worked with them in related fields, and I think there's probably a whole contribution of algorithms that could come. I think--see, what I was really saying is that a lot of programmers just forget about the fact that they are in the end creating an image. Then they forget about the basic rules of composition and I think that's where the language should start. That's what I really meant to say. Yeah.

QUESTION: O.K. You have started out with a design approach here in which the video information that's coming off the tube is, has a fixed rate. A thirtieth of a second per frame. And you then figured out things like, well, how fast can a human put out information and therefore how much versatility can we cope with before it becomes impossible for the human to put out information rapidly enough to be able to twiddle more and more parameters. I just would like to ask, why that trade-off? Why the whole idea of going necessarily in real time and limiting the number of things, the number of aspects of an image which the

artist can control because of his limited motor and perhaps even cognitive thought processes? In other words, is that clear?

WRIGHT: No.

QUESTION: O.K. Why make the trade-off for real time in which the limiting factors then becomes the amount of information per unit time that the artist can put into the system to control the images, to control the way the thing is going. As opposed to working in less than real time with greater versatility and more time for the artist to put in more control information to more subtly modulate the images on a frame by frame basis.

WRIGHT: It (the software) works on both modes.