ILLINOIS--UNIVERSITY AT URBANA-CHAMPAIGN--DEPT. OF COMPUTER SCIENCE

REPORT

UIUCDCS-R-74-630

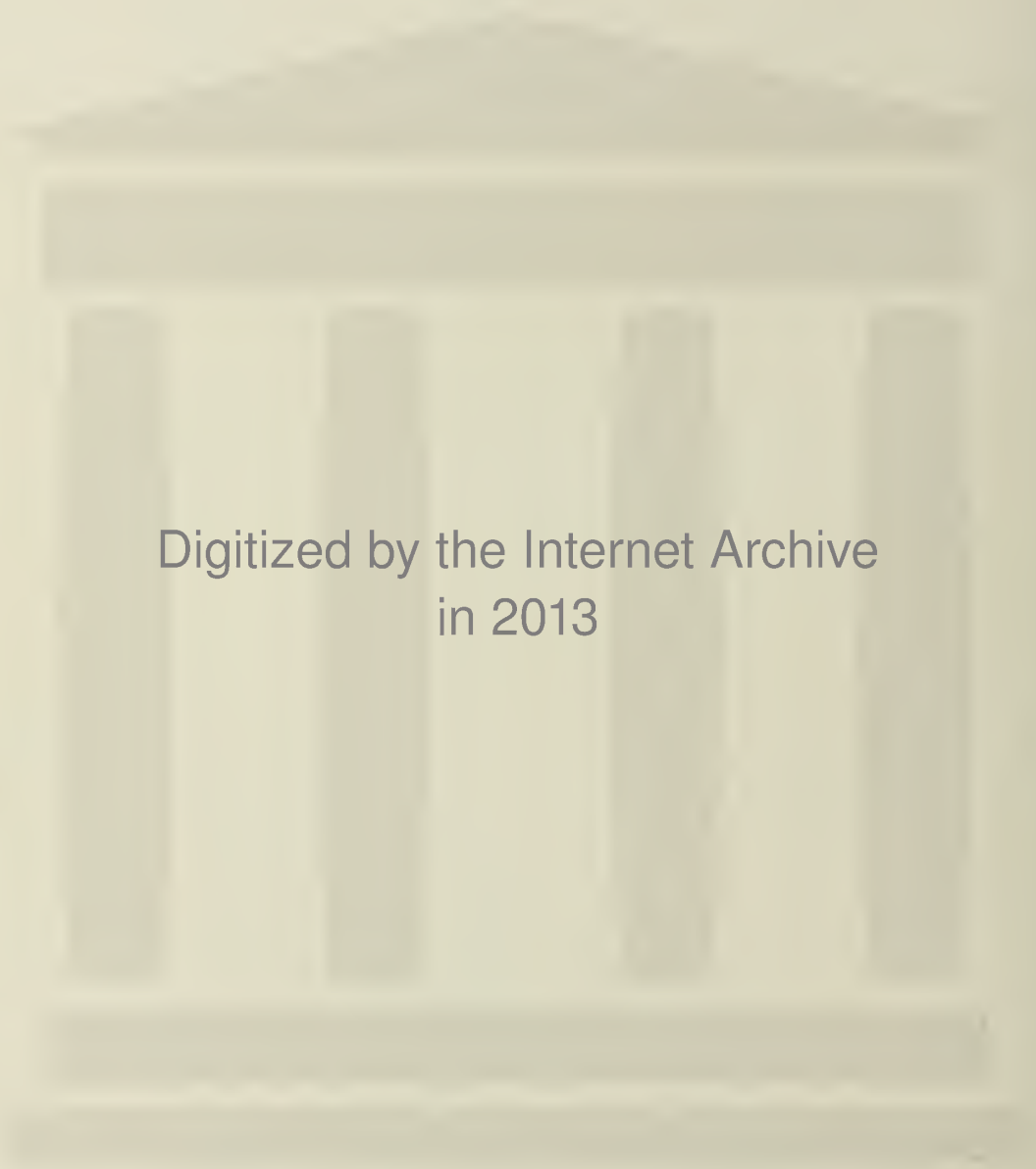
ERGODIC: COMPUTING WITH A COMBINATION OF STOCHASTIC AND BUNDLE PROCESSING

by

JAMES R. CUTLER

March, 1974

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS



UIUCDCS-R-74-630

ERGODIC*: COMPUTING WITH A COMBINATION OF STOCHASTIC AND BUNDLE PROCESSING

by

JAMES R. CUTLER

March, 1974

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

*Supported in part by the Office of Naval Research (ONR) under grant No. N000 14-67-A-0305-0007.

ACKNOWLEDGMENT

The author wishes to thank his advisor, Professor W. J. Poppelbaum, for suggesting this topic and for his support and guidance.

He will also like to thank Frank Serio and his staff for fabricating the circuit cards, Bert Semmelink for his technical skills, Evelyn Huxhold for typing the rough drafts, Mark Goebel and his staff for the drawings in this report, and Dennis Reed for the publication of this report.

Special thanks is due to his wife, Joyce, for her encouragement and perseverance.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

I. INTRODUCTION

This paper describes a stochastic computer called ERGODIC proposed by Professor Poppelbaum. In ERGODIC numbers are represented in a bundle of wires (called an ergodic bundle) by two methods. First, a number is represented by the number of wires in the bundle that are energized at any instant of time. (A wire in the bundle may be in one of two states: energized or de-energized). Also, the same number is represented by the number of times any particular wire is energized over a certain period of time: This redundancy allows one to check a bundle for errors.

ERGODIC is made up of three units: the generators, the arithmetic unit, and the processor. A generator converts a number into an ergodic bundle. Two ergodic bundles are then inputted into the arithmetic unit, which performs an arithmetic operation (multiplication, addition or subtraction). The output of the arithmetic unit is also an ergodic bundle. This bundle then goes to a processor which determines the number represented, and checks each wire in the bundle for any discrepancies from this answer.

II. BACKGROUND OF STOCHASTIC PROCESSING

The basis for all stochastic processing is quite simple:[5] Numbers are represented as probabilities of appearance in a random sequence. Two such sequences are then "multiplied" by merely inputting them into an AND gate. Stochastic processing, as developed at the Information Engineering Laboratory at the University of Illinois under the guidance of Professor Poppelbaum, progressed through several stages. The RPS (Random Pulse Sequence)-System was the first attempt at stochastic processing.[1] In this system the sequence consists of pulses occurring at random times and having random length and random height. The average duty cycle of this sequence is measured to obtain the data. Some digital logic was used but overlapping pulses cause problems. The SRPS (Synchronous Random Pulse Sequence)-System immediately followed.[1] In this scheme the pulses of the sequence occur (or do not occur) during predetermined time slots and have fixed length and height. The data is obtained by counting the number of pulses in a given number of time slots. Digital logic is used exclusively.

The next design resolved the difficulty of representing numbers other than those from 0 to 1 (the range of numbers that can be directly interpreted as probabilities): The Remapped SRPS-System accomplished the representation of numbers in the range of -1 to +1 by mapping them into the range for probabilities (0 to 1).[3] Then followed the Two Wire Remapped SRPS-System, which is able to represent all numbers: Each number is expressed as the ratio of a numerator and a denominator, the range for both the numerator and the denominator being from -1 to +1. The numerator and the denominator are both mapped into probabilities (0 to 1) as above.

A new flavor of probabilistic processing came in when the concept of bundle processing was suggested: A number of wires made up a bundle and each wire could be in one of two states.[2,6] Probabilities can now be assessed by

taking the number of wires in a given state, divided by the total number of wires in the bundle. The advantage of this concept is that it is failsoft; i.e. if a wire is cut or broken, the number that is represented is still approximately the same.

In a final step, the marriage of bundle processing and stochastic processing has led to ERGODIC. The property of an ergodic bundle is that each wire in the bundle has a sequence of pulses with a probability (time average!) identical to that of the bundle at any given instant (space average!) Comparing time and space averages now obviously leads to conclusions about the validity of the representation.

III. BASICS OF ERGODIC

A. Overall View of ERGODIC

There are three major parts of ERGODIC: 1) the ergodic generators, 2) the arithmetic unit and 3) the processor. Figure 1 shows the block diagram of these units. An ergodic generator takes a number within a certain range (-1 to +1 in steps of $\frac{1}{32}$) and represents this number in the ergodic bundle. The arithmetic unit takes two such bundles and performs arithmetic operations (addition, subtraction, and multiplication). The processor obtains a result from the resultant bundle of the arithmetic unit and also checks the bundle for any defective wires. A further description of each of these parts follows.

B. Number Representation

An ergodic bundle consists of 64 wires (this choice seems arbitrary but it turns out that implementation of the system is simplified. See section on implementation of ERGODIC). It is obvious that every number in the form:

$$\frac{N}{64} \qquad \text{where } N = 0,1,\ldots,64$$

may be represented in the bundle. But since the arithmetic operation of subtraction is going to be performed, it seems logical to be able to represent negative numbers by remapping; i.e. $y = f(x)$ where $0 \leq x \leq 1$ and y includes negative numbers. The range of y and the definition of the function, f, are determined by the arithmetic unit; the criterion being to keep the circuitry of the arithmetic unit as simple as possible.[4] The range of y was chosen to be from -1 to +1 and the function, f, was given by: $y = f(x) = 2x - 1$. Figure 2 shows the plot of the function, f.

C. Generation of an Ergodic Bundle

As stated before, an ergodic bundle is one in which the average number of pulses which occur over a certain period of time (time average) is equal to the

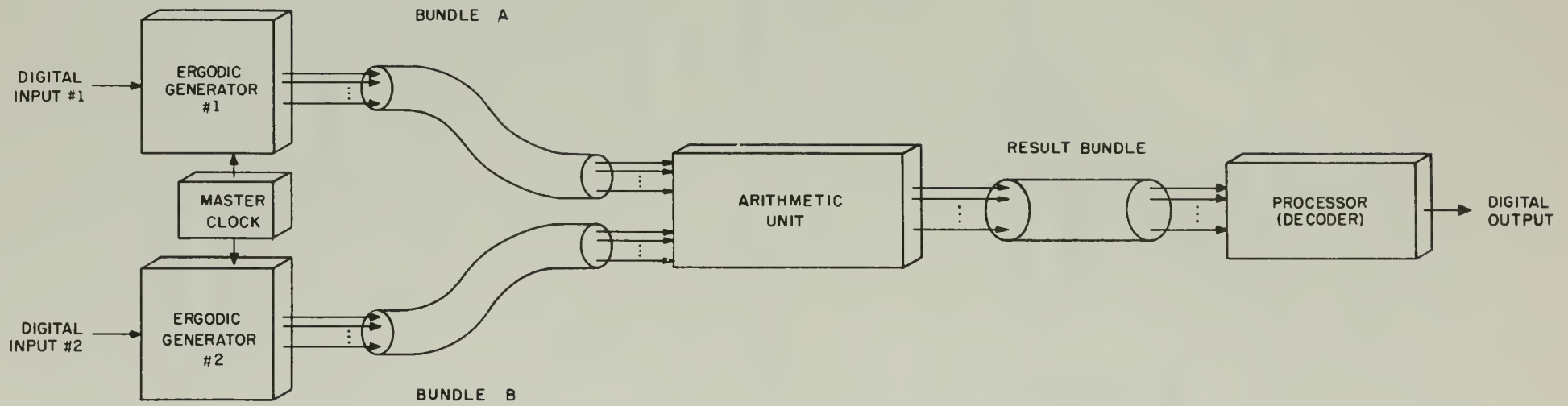


Figure 1. ERGODIC Block Diagram

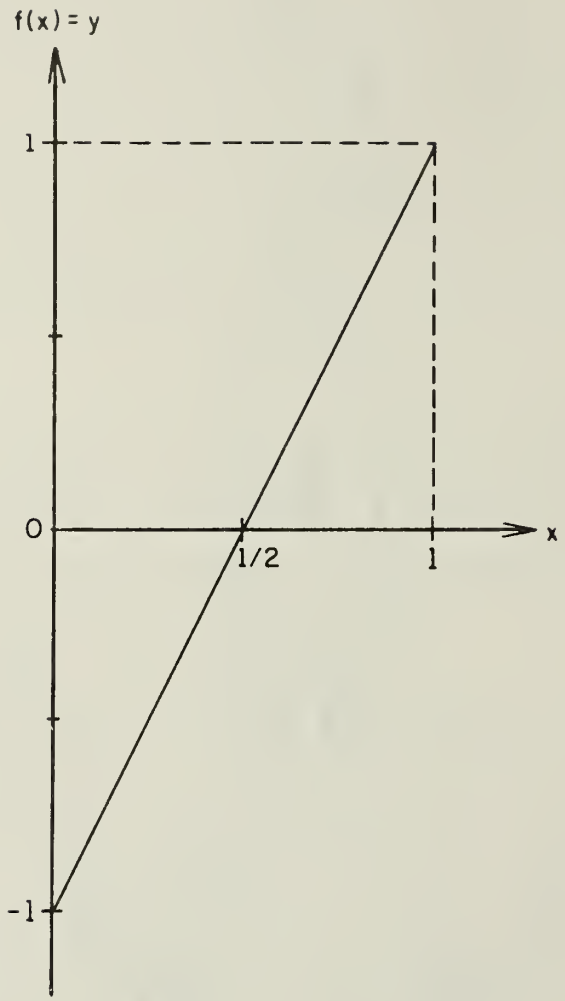


Figure 2. Plot of the function, f(x)

average number of wires within the bundle which are energized at any instant of time (space average). The requirements necessary for this part of the system are:

1) the two ergodic generators should be independent of each other. This property is necessary in order to obtain the correct answer from the arithmetic unit: It can be shown that the time sequences must not be correlated!

2) the circuitry of a generator should be as simple as possible.

The actual method of generating the ergodic bundle is described in a later section.

D. Arithmetic Operations

The basic concept for the arithmetic unit is that one wire from each bundle goes into a sub-arithmetic unit which performs a particular arithmetic operation. Thus, the arithmetic unit requires as many arithmetic sub-units as there are wires in a bundle. The outputs of all of the arithmetic sub-units then make up the resultant bundle. The block diagram of this unit is shown in Figure 3.

The most important requirement for the arithmetic sub-unit is that it should be kept as simple as possible. An advantage to stochastic computing is that the arithmetic unit is simplified and requires only a single gate for each arithmetic operation.[4,5] Because of the number representation chosen, an OR gate is used for addition, an Exclusive OR gate is used for subtraction, and an Exclusive OR gate is used for multiplication (assuming that positive numbers are inputted). An AND gate is also needed; for example, when two negative numbers are added. Figure 4 shows the circuit diagram of an arithmetic sub-unit.

Note that if all of the wires of the bundle go through inverters and the space average is x, the result is (1-x). This corresponds, of course, to taking

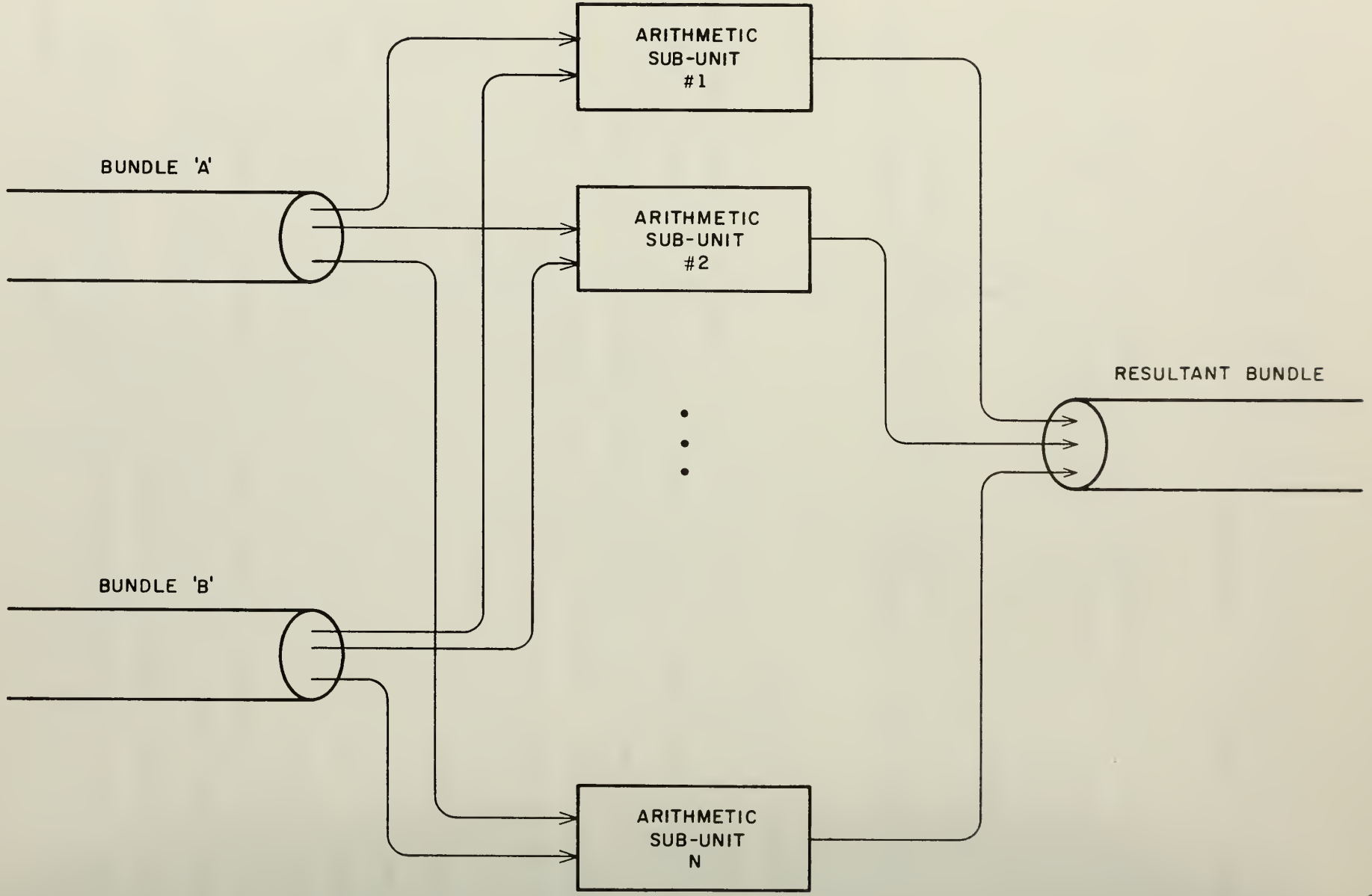


Figure 3. Block Diagram of the Arithmetic Unit

(N = no. of wires in a bundle)

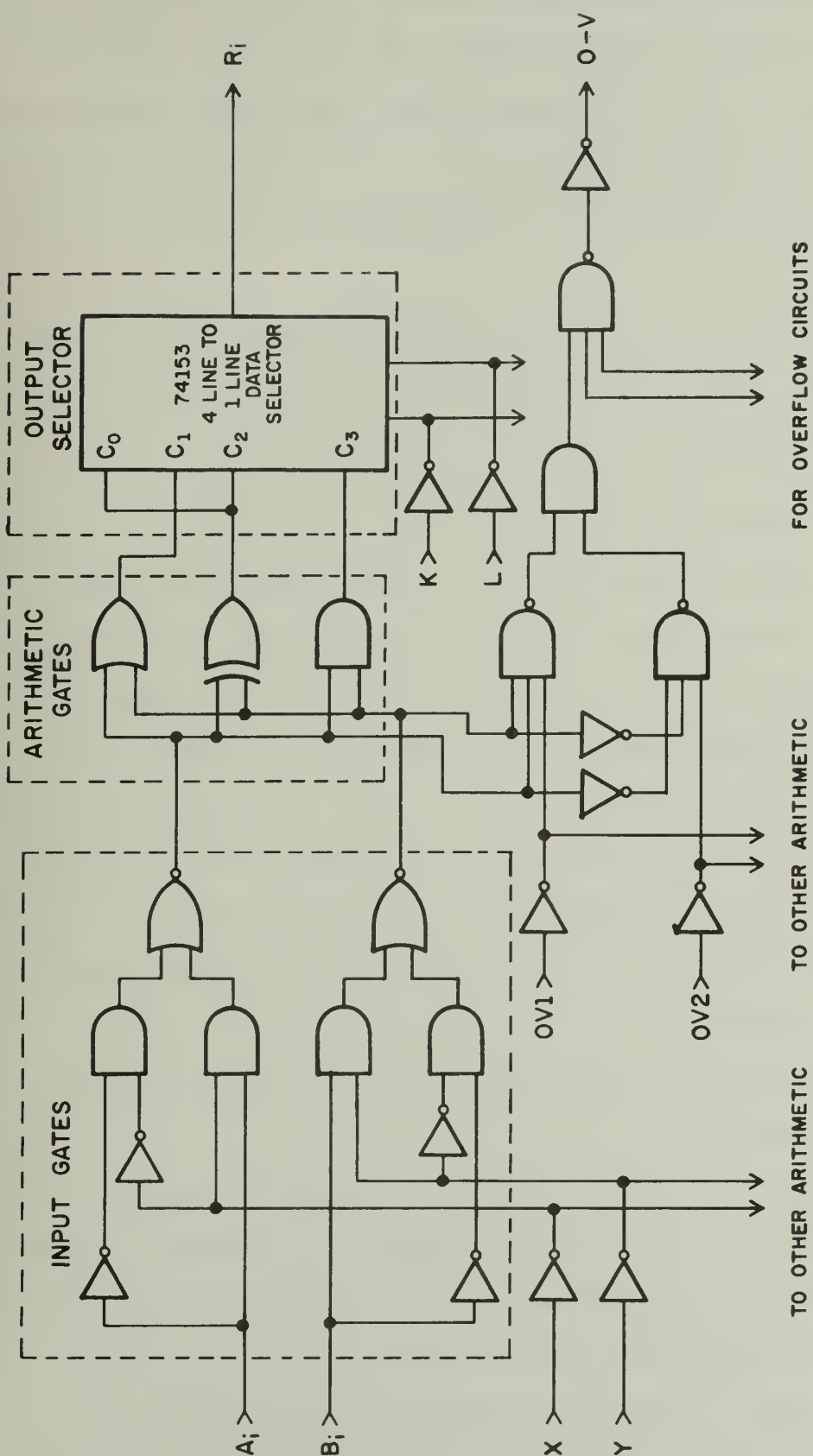


Figure 4. Circuit Diagram of an Arithmetic Sub-Unit

the negative of the represented number, y [where y = 2x-1]. Thus, the use of inverters in the arithmetic sub-unit changes the sign of the represented number.

The arithmetic unit also determines whether overflow has occurred; i.e., a result greater than +1 or less than -1. In this case, a light will go on and the result will be either +1 or -1 depending upon which is closer to the actual result.

Not shown in Figure 3 is the arithmetic control, which will be discussed in a later section. From this control come the signals energizing the lines labeled X, Y, OV1 and OV2.

E. Processing of Ergodic Bundles

Because the space average and the time average of an ergodic bundle are equal, there is a redundancy that may be used to check for errors and malfunctions in the bundles. For example, if one wire of the bundle happens to break, the space average is approximately correct and the time averages of the remaining wires must still coincide! A processor can be constructed which determines the proper number that is represented in the bundle and also compares the time average of each wire with this number to determine the broken or defective wires. Note that as long as one wire remains unbroken, the number represented in the bundle is determined!

Many algorithms for determining this number and defective wires exist. The one chosen is explained later.

The circuitry used to accomplish the above tasks is all digital. Counters are used to calculate the time and space averages. Thus, the time required to obtain these averages, T, will be some multiple of the number of wires in the bundle, N, times the period of the master clock, τ.

$$T = nN\tau \quad \text{(where n = non-zero positive integer)}$$

For example, if the master clock has a frequency of 2.5 MHz and the bundle has

64 wires, the time required is 25.6n μsec. Note that to obtain the space average the minimum time is 25.6 μsec, but in order to obtain a more accurate time average, n should be made large (> 10).

IV. IMPLEMENTATION OF ERGODIC

A. The ERGODIC Generator Unit

This unit receives a 7-bit digital number and represents this in the ergodic bundle. This digital number uses the signed absolute value system to represent numbers between -1 to +1; i.e.

$$b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0 \equiv (-1)^{b_6+1} \sum_{i=0}^{5} b_i\ 2^{i-5}$$

(Note that it is possible to represent numbers less than -1 and greater than +1 but the circuitry merely interprets those numbers as -1 and +1, respectively). After inputting the desired number, it is transformed into a fraction between 0 to 1 by the mapping shown in Figure 2. This fraction then determines how many wires are energized in the ergodic bundle. For example: Let Y be the number to be represented, X be the fraction represented in the bundle and N be the number of wires to be energized.

Suppose $Y = -5/32$

then $X = \frac{1}{2}(Y+1) = 27/64$

and $N = 64\ X = 27$

Now the question arises: How does one transform the bundle into an ergodic bundle, i.e. a bundle where the time average equals the space average? Obviously, if each wire has an identical pulse sequence, the bundle is not ergodic. The trick is to redistribute the energized wires from time slot to time slot: This guarantees that the time average of the pulse sequence is constantly equal to the number represented by the bundle! Practically we use an N-bit shift register, with n bits at 1. If the shift register is connected so that it is circular, the number of bits at 1 will remain at n. Thus, the space average at any instant is $\frac{n}{N}$. If any bit of the shift register is considered, the time average of that bit being at 1 is $\frac{n}{N}$, if the sample period is $N\tau$ or a multiple of $N\tau$

($\frac{1}{\tau}$ is the frequency of the shift register-shifts/sec). Thus, a N-bit circular shift register is a simple ergodic generator!

The problem with an N-bit circular shift register is that the sequence repeats itself after N time slots. Thus the randomness of the sequence is determined by how many wires there are in a bundle. This problem is resolved in the arithmetic unit.

Since the N-bit circular shift register is by far the simplest ergodic encoder, the decision was made to use it in ERGODIC. The block diagram showing this arrangement for generator is shown in Figure 5. A discussion about ergodic generators follows this section.

B. The Arithmetic Unit

This unit performs the arithmetic operations of addition, subtraction and multiplication. This section will describe the methods and circuitry needed to complete the above operations. First, addition and subtraction will be described and then multiplication.

In bundle processing, addition is accomplished by ORing two wires--each coming from the two different input bundles--together: The proper result will be obtained providing that both wires are not energized together. In other words, the two bundles must be <u>disjoint</u>. Subtraction is accomplished by EXCLUSIVE ORing two wires together: The result is correct if the energized wires of the subtracted bundle overlaps the energized wires of the other bundle. These two conditions are contradictory, and it appears that subtraction and addition cannot both occur in the same system...

This problem is solved by a unit called a <u>randomizer</u>. The block diagram of the arithmetic unit with the randomizers included is shown in Figure 6. A detailed diagram of a randomizer is shown in Figure 7. The randomizers essentially rearrange the data of the bundle according to information of the bundle

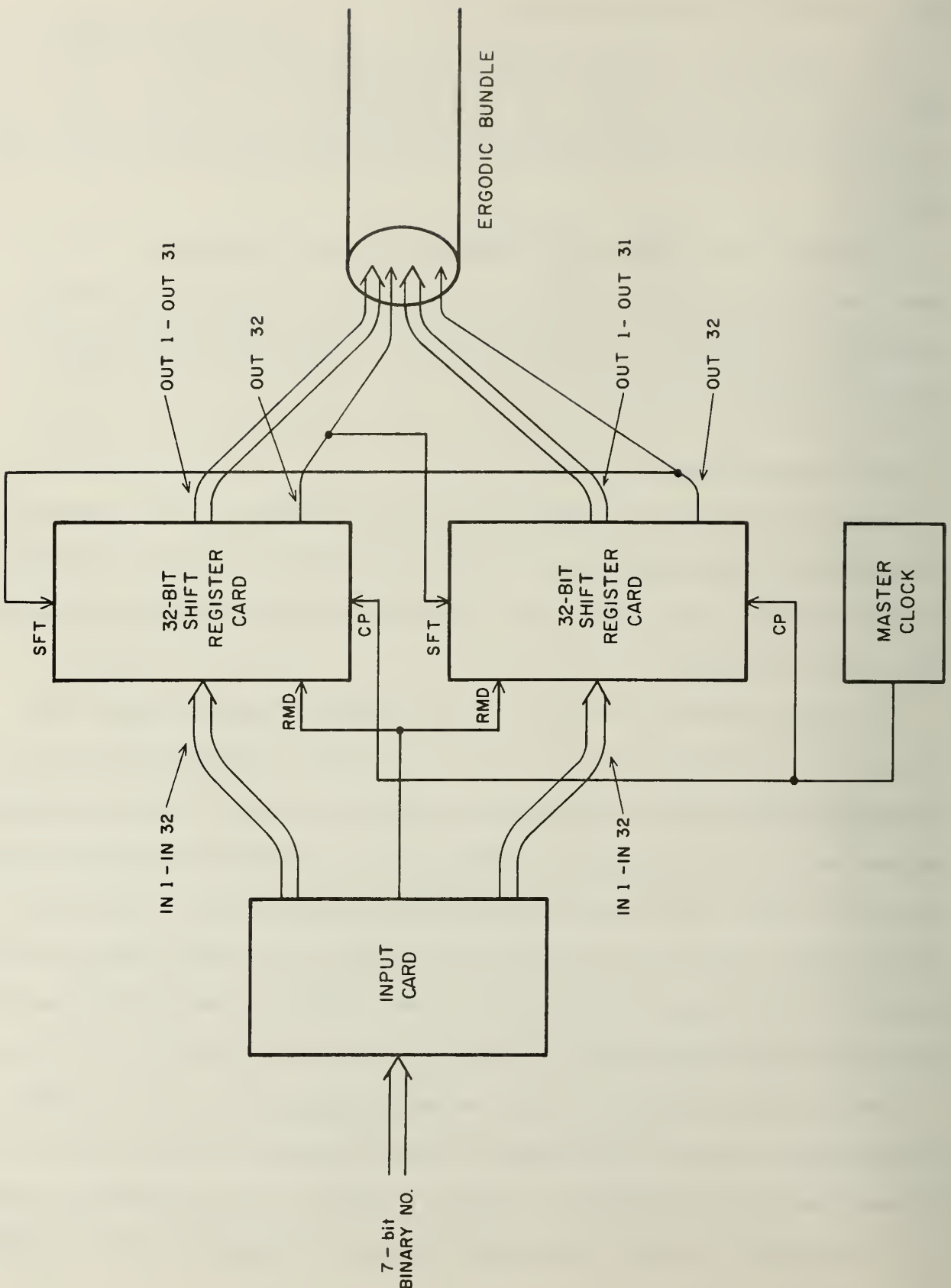


Figure 5. Block Diagram of an Ergodic Generation

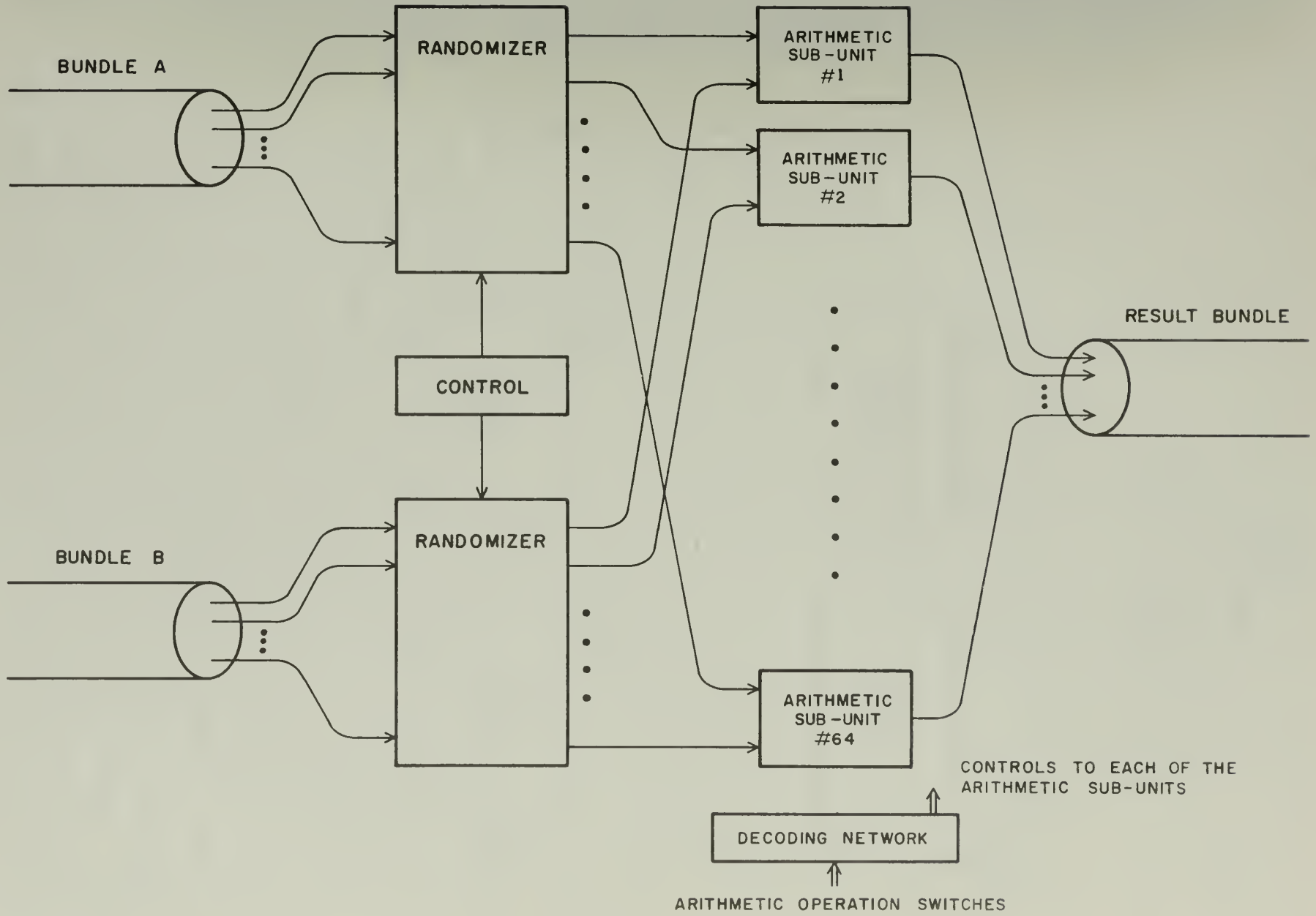


Figure 6. Block Diagram of Arithmetic Unit

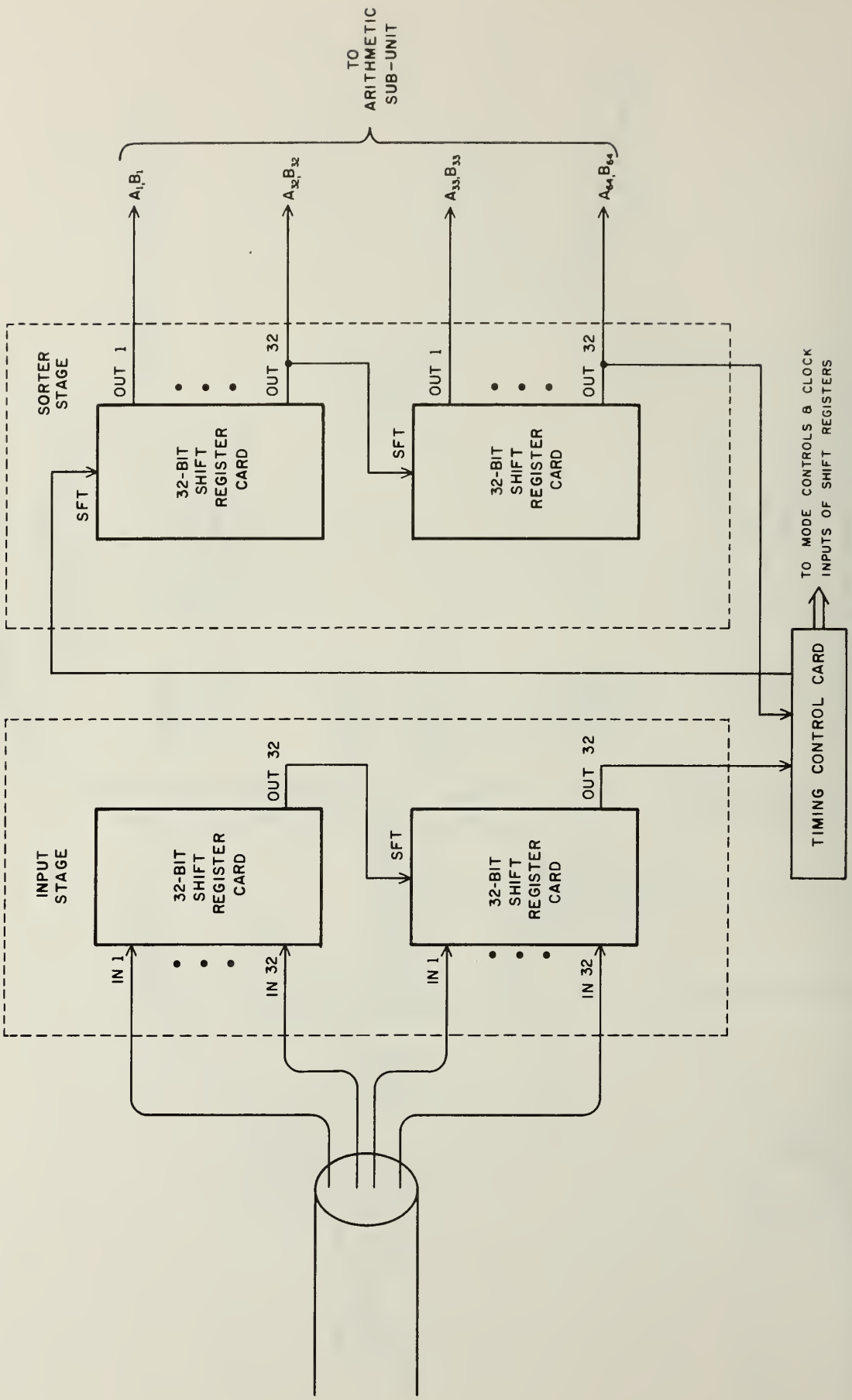


Figure 7. Block Diagram of a Randomizer

and the arithmetic operation to be performed. Thus, during subtraction the randomizers causes the bundles to overlap and during addition it causes the bundles to be disjoint. The signal names, A_i and B_i where $1 \leq i \leq 64$, indicate where these wires are connected with respect to the corresponding lines in the arithmetic sub-unit shown in Figure 4.

The essential pieces of information are the signs of the two numbers represented that are being inputted into the arithmetic unit. The randomizer makes it possible to look at just one bit to determine the sign! In Figure 7 the input stage stores a particular space average (which is identical to the time average) of the ergodic bundle. The sorter stage then arranges the 1's and 0's of this stored space average so that all of the 1's are together at one end and the 0's are together at the other end. Since the 32nd bit determines a space average of greater than or equal to 1/2 (32/64) [which represents the number, 0; $Y = 2X-1 = 2(\frac{1}{2}) - 1 = 0$], the 32nd bit determines whether the represented number, Y, is positive or negative. Now by knowing the signs of the represented numbers and the arithmetic operation, it may be determined whether the two bundle should be disjoint or overlapping.

The loading of the sorting register is now determined: For bundle A the 1's are loaded from top to bottom, i.e. the 1's are located from A_1 to A_x and the 0's are located A_{x+1} to A_{64}, where x is the number of 1's. For bundle B and the arithmetic operation of addition, the 1's are loaded starting at B_{32} and going toward B_1. If these bits become filled, then B_{64} through B_{33} are used. For the operation of subtraction, bundle B is loaded so that B_{33} through B_{64} is filled first, and then B_1 through B_{32}. These loading directions are shown in Figure 8.

It should be mentioned that the rearranged 1's and 0's of the sorter stage take on a new significance. Suppose the bundle A is taken as an example. If the represented number is positive, the first 32 bits (A_1 - A_{32}) will be 1's and the next 32 bits may or may not be 1's depending on the number. Then the first 32 bits

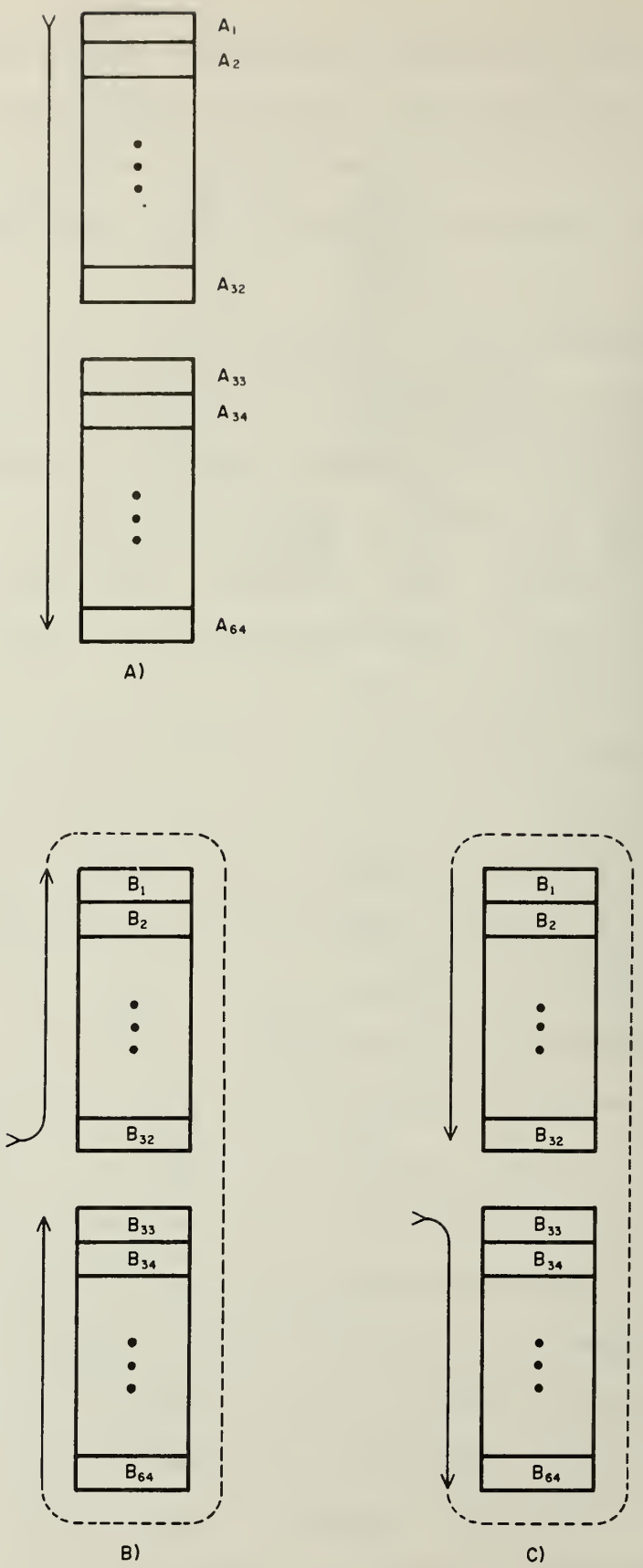


Figure 8. Loading Direction of the Sorter Stage of the Randomizer
a) shows loading direction of bundle A,
b) & c) shows loading direction of bundle B for addition and subtraction operations, respectively

may be considered the sign portion of the bundle and the last 32 bits may be considered the number portion. That is, if two positive numbers are being added, the sign portion of the result bundle should be kept intact but the number portion should show the sum of the two number portions of the input bundles. On the other hand, if the represented number is negative, the first 32 bits ($A_1 - A_{32}$) may or may not be 1's, depending on the number. The last 32 bits will be 0's. Thus the last 32 bits now indicate the sign portion and the first 32 bits represent the number portion. This reversal makes it necessary to change the loading direction of the sorter stage of bundle B.

After the sorting stages impress the proper configuration onto the bundle, it is necessary to load the sign portion of the bundles with the proper level, so that the result bundle has the proper sign and the overflow circuitry works properly. For example, in the case of adding two positive numbers, one of the sign portions of the bundles must return to 0, since the overflow circuitry would indicate a false overflow. Figure 9 shows the timing diagram of the entire cycle of a randomizer. Table I indicates this loading procedure as well as the gate necessary and overflow enable controls for given signs of the represented numbers and the arithmetic operation. All that is necessary to make the resulting bundle ergodic is that the two sorter stages be connected in a circular fashion and move in synchronism.

The arithmetic operation of multiplication is a different story: The necessary condition for this operation is that each bundle must be independent of one another. Note that rearranging is not necessary (as in the subtraction and addition operations). Suppose that the two sorter stages are loaded independently of one another (these stages are moving at the same shifting speed). This independence is not enough, since--once loaded--one bit of the sorter stage of one randomizer will forever look at the same bit of the sorter stage of

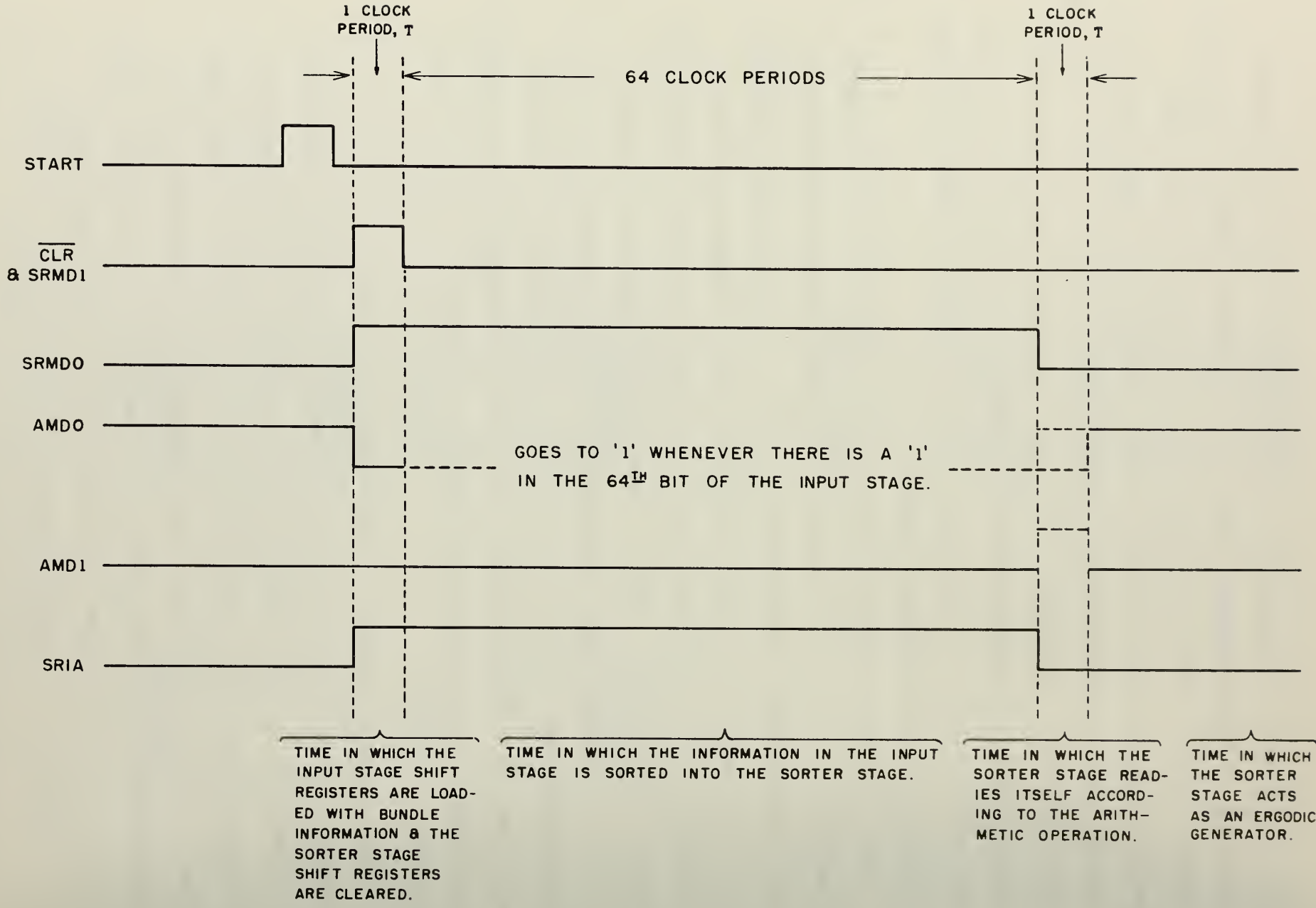


Figure 9. Timing Diagram of a Randomizer

TABLE I. List of Arithmetic Operations

	Multiplication	Addition	Subtraction	
			A - B	B - A
A & B both positive	$R_i = \overline{A}_i \oplus B_i$ Overflow disables (One sorter stage is moving 64 shifts faster than the other).	$R_i = A_i + B_i$ Overflow if $A_i \cdot B_i = 1$ (first 32 bits of A are first loaded with 0's).	$R_i = A_i \oplus B_i$ Overflow disabled (last 32 bits of B are first loaded with 0's).	$R_i = A_i \oplus B_i$ Overflow disabled (first 32 bits of A are first loaded with 0's).
A negative & B positive		$R_i = \overline{A}_i \oplus B_i$ Overflow disabled (last 32 bits of A are first loaded with 1's).	$R_i = A_i \cdot \overline{B}_i$ Overflow if $\overline{A}_i \cdot B_i = 1$ (last 32 bits of B are first loaded with 0's).	$R_i = \overline{A}_i + B_i$ Overflow if $\overline{A}_i \cdot B_i = 1$ (last 32 bits of A are first loaded with 1's).
A positive & B negative		$R_i = A_i \oplus \overline{B}_i$ Overflow disabled (last 32 bits of B are first loaded with 1's).	$R_i = A_i + \overline{B}_i$ Overflow if $A_i \cdot \overline{B}_i = 1$ (first 32 bits of B are first loaded with 1's).	$R_i = \overline{A}_i \cdot B_i$ Overflow if $A_i \cdot \overline{B}_i = 1$ (first 32 bits of A are first loaded with 0's).
A & B both negative		$R_i = A_i \cdot B_i$ Overflow if $A_i \cdot B_i = 1$ (last 32 bits of A are first loaded with 1's).	$R_i = \overline{A}_i \oplus \overline{B}_i$ Overflow disabled (last 32 bits of A are first loaded with 1's).	$R_i = \overline{A}_1 \oplus \overline{B}_1$ Overflow disabled (first 32 bits of B are first loaded with 1's).

(The subscript, i, indicates the respective bits of the inputs and the output. $1 \leq i \leq 64$).

the other randomizer. Thus, the result bundle will indicate a fraction anywhere between $1-x_1-x_2$ and min (x_1,x_2) where x_1 and x_2 are the fractions indicated in the input bundles and min $(\cdot,\cdot)$ is the minimum value of the two numbers. The corresponding represented numbers are y_1 and y_2 ($y_1 = 2x_1-1$) and, therefore, the desired result bundle should indicate

$$y_1\ y_2 = (2x_1-1)\ (2x_2-1) = 4x_1\ x_2 - 2(x_1+x_2) + 1$$

It may be shown that $y_1\ y_2$ lies between $1-x_1-x_2$ and min (x_1,x_2). Thus, by chance the appropriate result may be obtained.

The answer to this problem is to shift one sorter stage 64 times faster than the other, instead of moving them in unison. This difference in shifting speeds results in each bit in one sorter stage being able to respond to each of the bits of the other sorter stage. Now the result bundle obtains the proper time average answer, x, where

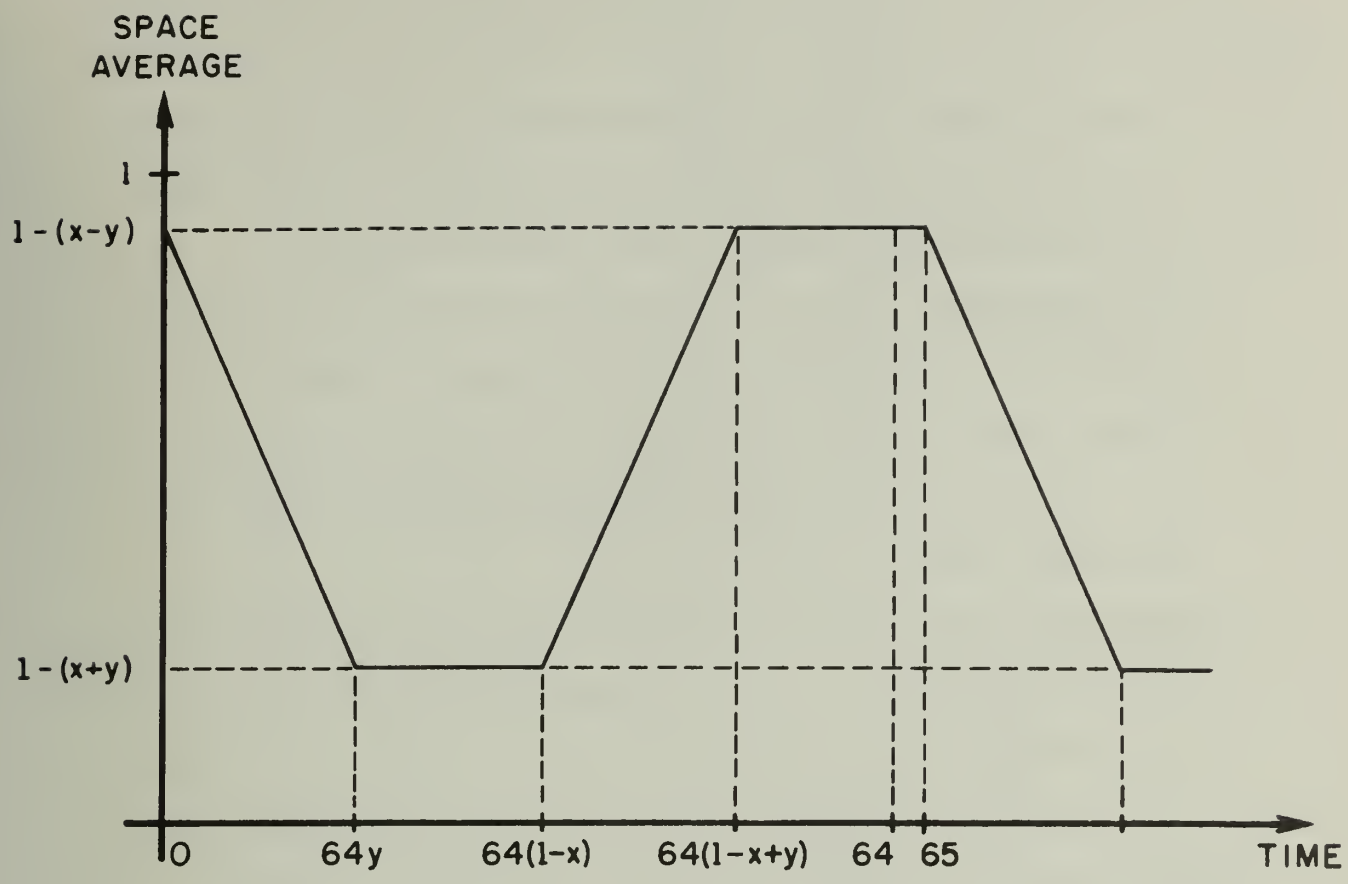
$$x = x_1\ x_2 + (1-x_1)\ (1-x_2) = 2x_1\ x_2 - (x_1+x_2) + 1$$

and $$y = 2x-1 = 4x_1\ x_2 - 2(x_1+x_2) + 1$$

(This answer is obtained after 64x64 shifts). Unfortunately, this shift differential does not solve the problem completely. The space average now does not remain constantly equal to the time average. But the average of the space average is equal to the time average. This phenomenon is shown in Figure 10.

C. Processing Unit

This unit has two functions. First, it receives the result bundle and determines the represented number of that bundle. Second, it compares each wire's time average with an overall average. The overall average will determine the represented number and may be determined by a number of methods. The purpose of all checks is to see if a wire of the bundle has been destroyed or if an arithmetic sub-unit associated with that wire is not functioning properly.



(ASSUMING THAT $x>y$, $x+y \leq 1$ WHERE x AND y ARE THE INPUT BUNDLE FRACTIONS.)

AVERAGE OF SPACE AVERAGE =

$$\frac{1}{64}\left[\sum_{i=0}^{64y}\left[1-(x-y)-\frac{i}{32}\right] + 64(1-x-y)^2\right.$$

$$\left.+\sum_{i=1}^{64y}\left[1-x-y+\frac{i}{32}\right] + 64(x-y)(1-x+y)\right]$$

$$= 2xy-(x+y)+1$$

Figure 10. Space Average of the Result Bundle for Multiplication

The method used to determine the represented number will depend upon the particular use and environment of the system. One method is to form the space average of the bundle and to use it as an approximation. We then would obtain the time average for each wire and update the approximation with each time average, using an appropriate algorithm. Such algorithms might be:

1) A new approximate answer is determined by the mean of the approximate answer and the time average. A defective wire is defined by its time average being outside a certain range of the approximate answer. In a case of a defective wire the new approximate answer is simply the approximate answer.

2) A new approximate answer is determined by the time average. A defective wire is defined by having its time average equal to zero unless the approximate answer is already a zero. If the wire is defective the new approximate answer is the same as the old approximate answer.

The first algorithm may best be used in an environment in which the arithmetic unit is unreliable (high temperature) or result bundle in a noisy area (high electro-magnetic field)! The second algorithm may be best used in an environment in which the wires of the result bundle have a high likelihood of being cut (manufacturing plant)...

Since this system is located in a safe and ideal area (the laboratory), any algorithm that follows the guidelines above would be suitable. However, it was decided that the second algorithm would be best for demonstration purposes; i.e., the cutting of a wire in the bundle without the result changing. Figure 11 shows the block diagram of the processor using that particular algorithm.

One may ask about the usefulness of the space average. In this case, it is not used because it takes just as long to calculate it as it takes to calculate a time average. But this algorithm could easily take advantage of the space average without complicating the circuitry too much. One must also keep in mind

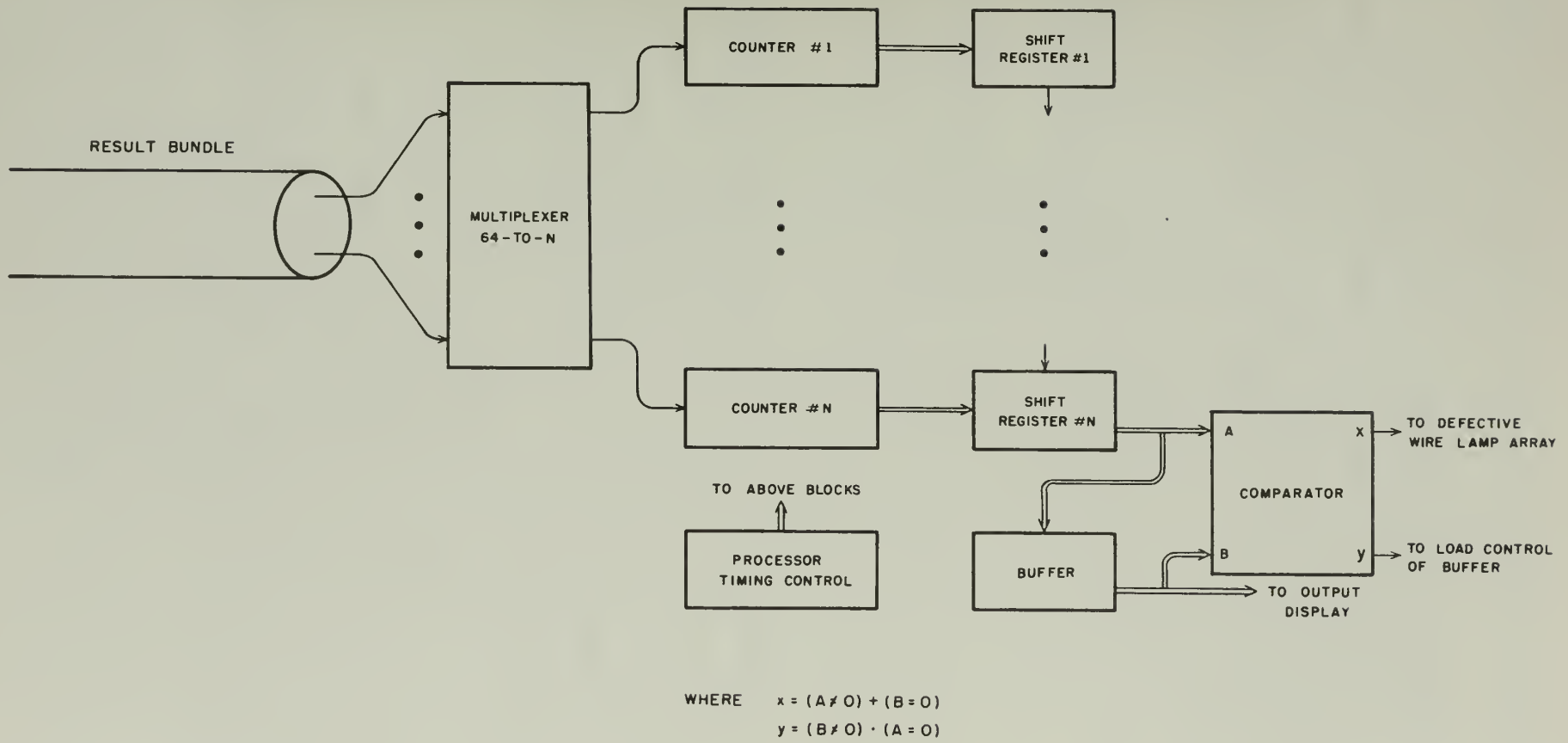


Figure 11. Block Diagram of the Processor

the problem of the space average of the result bundle when the arithmetic unit is multiplying.

The actual circuitry of ERGODIC has four time average processors. The multiplexer shown in Figure 11 is formed of four 16- to -1 multiplexers. By using this arrangement, we save one-fourth of the time required to check the time averages of all of the wires in the bundle. This amount of time, T, (to check all of the wires) is:

$$T = 16n\tau$$

where n = number of shifts that each counter looks at one particular wire

and τ = time period of one count pulse

In this case, $n = 64 \times 64 = 4096$ and $\tau = 400$ ns, then $T \simeq 26.2$ ms. The average time, $\bar{t}$, to determine the time average of a wire is:

$$\bar{t} = \frac{1}{64} T \simeq 410 \text{ μs.}$$

Note that n may be made smaller but the result obtained when the arithmetic operation of multiplication is being performed will be inaccurate. The processing unit may be speeded up only by increasing the counter speed (decreasing τ). The minimum value for τ is approximately 40 ns. Thus, the minimum values for T and $\bar{t}$ with the accuracy being retained are:

$$T \simeq 2.6 \text{ ms.}$$

$$\text{and } \bar{t} \simeq 41 \text{ μs.}$$

Of course, if speed is more vital than accuracy, these times, T and $\bar{t}$, may be decreased more. Note here that when the arithmetic operations of addition and subtraction are being performed, the accuracy is maintained when $n = 64$. Thus, the values for T and $\bar{t}$ are (assuming that $\tau = 400$ ns.):

$$T \simeq 410 \text{ μs.}$$

$$\text{and } \bar{t} = 6.4 \text{ μs.}$$

D. Overall View

One thing that has not been pointed out so far is that all three units--the Generators, Arithmetic Unit, and the Processor--are independent of each other and connected only by the bundles which are the transmitters of the information. As a consequence, these units may be located wherever it is convenient or necessary. Since this information is redundant throughout the bundle, there is no problem with losing this information by the destruction or cutting of part of the bundle. Thus, this system would be ideal for a dangerous environment where space is at a minimum; i.e., a submarine or a space ship.

V. IMPROVEMENTS AND SUGGESTIONS

There are several improvements one could think of: 1) the elimination of the randomizers in the arithmetic unit by making the ergodic generators more random, 2) the increase of the number of types of arithmetic operations performed by the arithmetic unit, 3) the improvement of ability of the system to do complicated calculations by increasing the number of arithmetic units and the introduction of a memory.

By using an ergodic generator that is more random (one that repeats only after a large number time slots), it is possible to eliminate the randomizers located in the arithmetic unit. Suppose that there exists a random sequence generator for each of the N wires in the bundle: If each of the N generators presents the same time average, and the space average is approximately equal to the time average, the bundle will be ergodic. [By approximately we mean the space average, $\overline{S}$, to be within a small range, Δ, of the time average, $\overline{T}$: $S = T + E$ where $|\varepsilon| \leq \Delta < 1$]. It is speculated that the randomness of this generator is directly related to the magnitude of the range, Δ. The circuitry should still be kept at a minimum and thus, N-random generators would not be feasible and probably not necessary since they are not independent. (Because of the space average, these generators must be dependent upon one another). A more reasonable solution would be to have a random generator for a time average and a random generator that would then determine the space average. Suppose that one random generator fills an N-bit shift register, so that the last N outputs of this generator are stored in the shift register. The other generator could now determine how many shifts the shift register will take in the next n shifts ($n < N$). An N-bit buffer is connected to the shift register and is loaded every τ sec. Thus, the speed of this "ergodic generator" is $\frac{1}{\tau}$ shifts/sec. Figure 12 shows the block diagram of this generator.

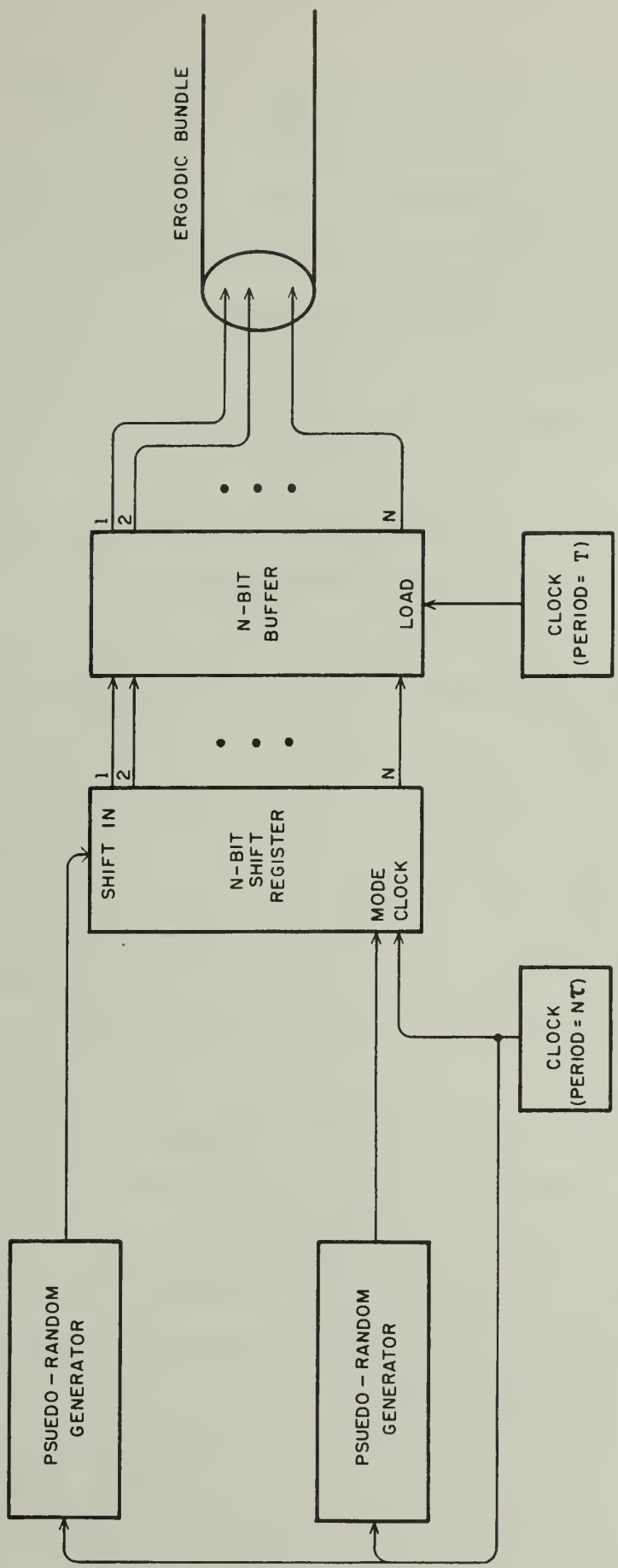


Figure 12. Block Diagram of Revised Ergodic Generator

There is one important arithmetic operation that is not performed in the arithmetic unit: Division. Obviously, it would be a necessary feature in a useful system. There are two methods to perform division: One described in Professor Poppelbaum's book and another described in Esch's thesis. The problem is the retention of the property of ergodicity.

These two improvements leads to the last one: the ability of a system is perform complicated calculations. This ability may be accomplished by two methods:

1) Have more than one arithmetic unit in which these units are connected in a universal manner or in a variable manner.

2) Have one arithmetic unit and a storage unit.

In the first method the calculation would be done in an array and the number of arithmetic units would be greater or equal to the number of operations in this calculation. An arithmetic unit would be able to perform a wide selection of arithmetic operations (being also able to pass the input information to the output without changing it). The important criterion for the arithmetic unit is its size and simplicity. Thus, the elimination of the randomizers becomes necessary. Figure 13 shows how several arithmetic units may be connected together.

In the second method the calculation would be done in one arithmetic unit but the result would be stored in a memory. Since the result would be random, it may be necessary to convert the result into a binary number and use an ordinary computer memory or perhaps it would be feasible to store a large portion of the random result in a shift register. At any rate this system would be the analog of a modern digital computer.

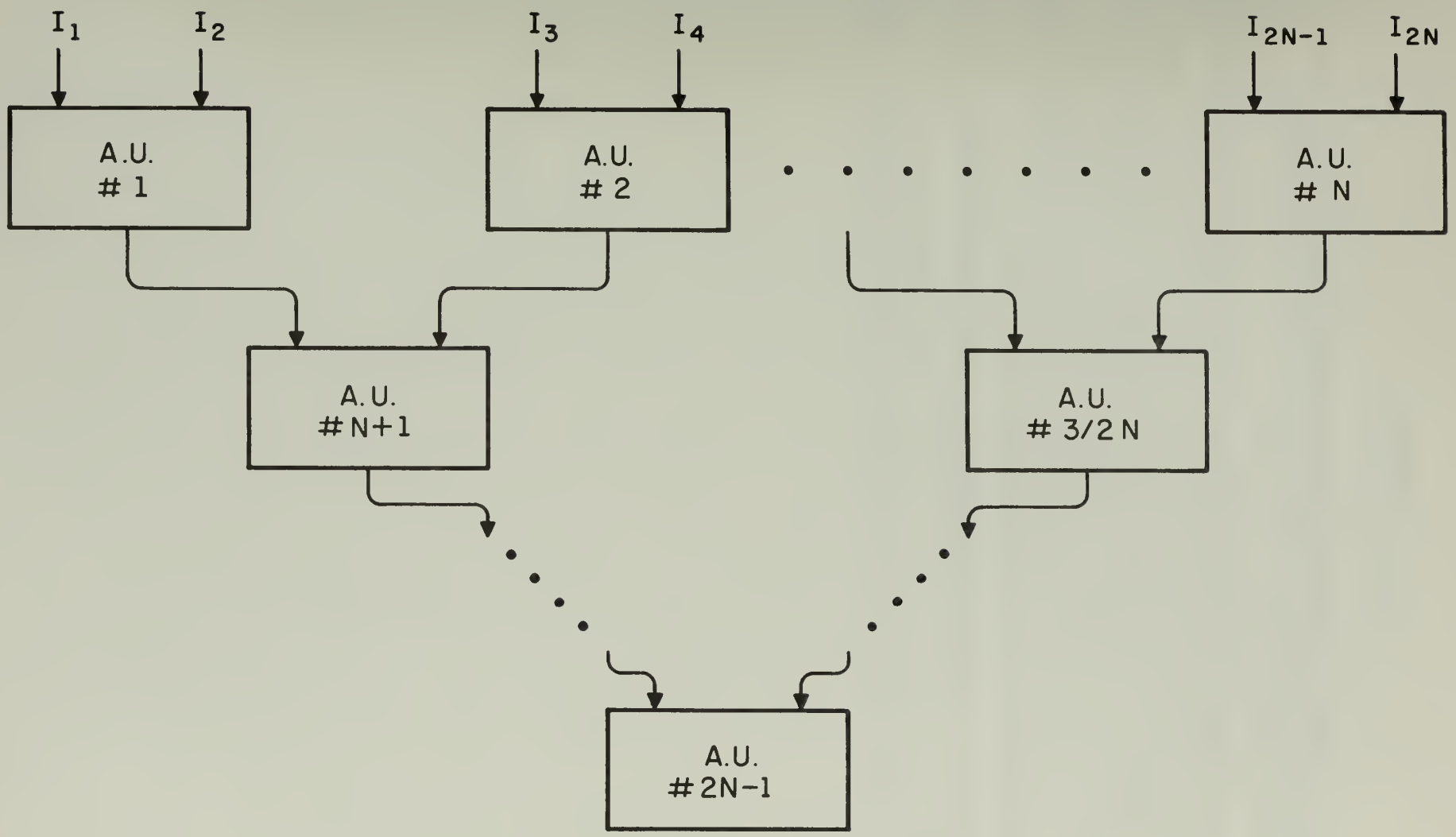


Figure 13. Block Diagram of the Interconnections of many Arithmetic Units (A.U.)

LIST OF REFERENCES

1. Afuso, C., "Analog Computation with Random Pulse Sequences," Department of Computer Science Report No. 255, University of Illinois, Urbana, Illinois, February, 1968.

2. Coombes, D., "Sabuma - Safe Bundle Machine," Department of Computer Science Report No. 412, University of Illinois, Urbana, Illinois, August, 1970.

3. Esch, J., "Rascel - A Programmable Analog Computer Based on a Regular Array of Stochastic Computing Element Logic," Department of Computer Science Report No. 332, University of Illinois, Urbana, Illinois, June, 1969.

4. Gaines, B. R., "Stochastic Computer Systems," Advances in Information Systems and Science, Ed. by Tou, J. T., Plenum Press, 1969.

5. Poppelbaum, W. J., Computer Hardware Theory, The Macmillan Company, New York, 1972.

6. Ring, D., "BUM - Bundle Processing Machine," Department of Computer Science Report No. 353, University of Illinois, Urbana, Illinois, October, 1969.

APPENDIX

This section is devoted to the circuit diagrams that make up Ergodic and their interconnections. It is arranged so that the interconnections of cards for the generaters are shown first and then the circuit diagram of these cards are shown. Following this part comes the same arrangement for the arithmetic unit and the processor in that order.

The notation of these diagrams should be described. Every line that goes in or out on a circuit card has a signal name and the connector pin number is listed below this name. The number inside the integrated circuit (I.C.) or gate indicates the package number and the numbers on lines going into or out of the I.C. or gate indicate the pin number of the I.C.

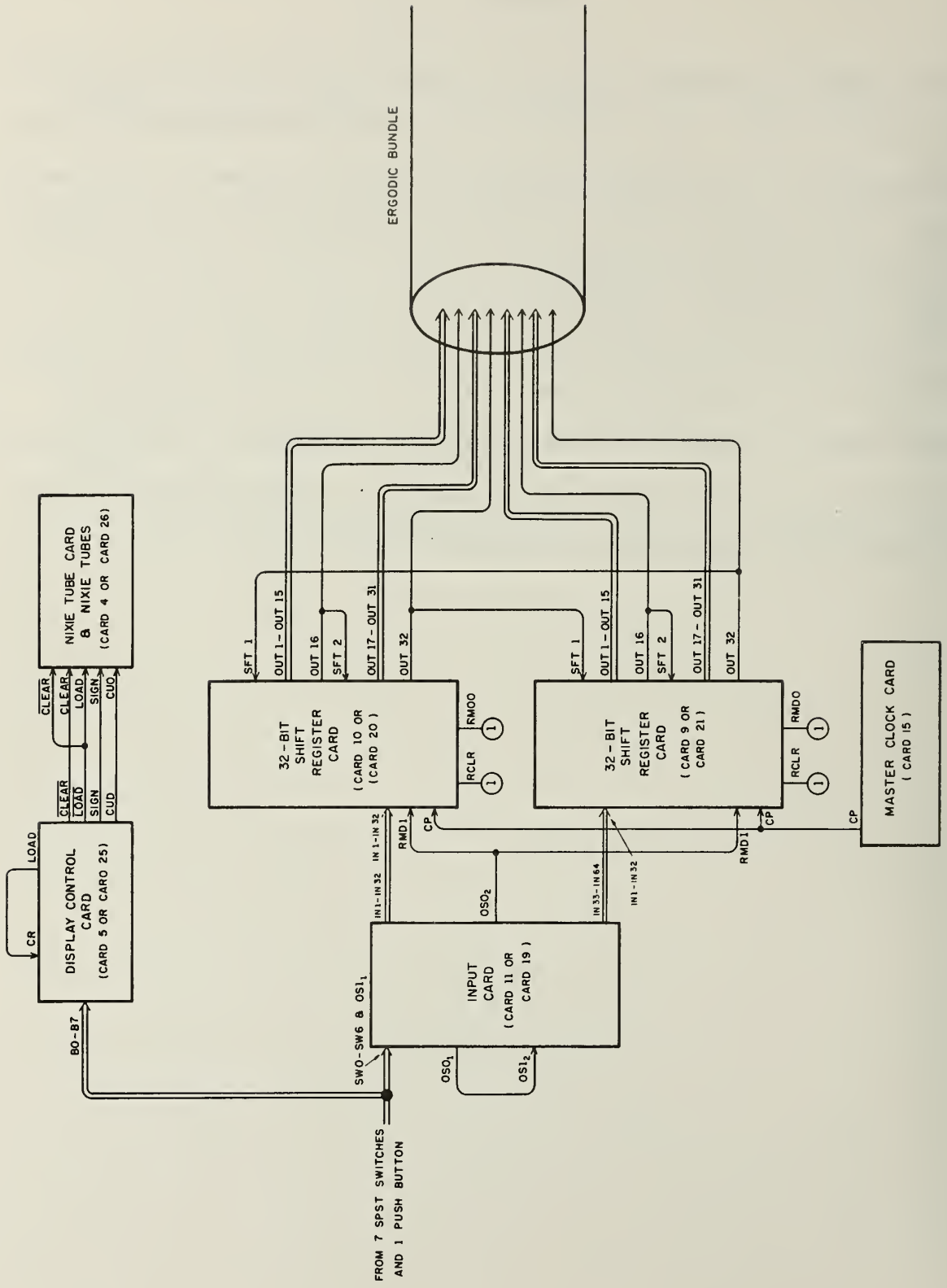


Figure A-1. Interconnections for an Ergodic Generator

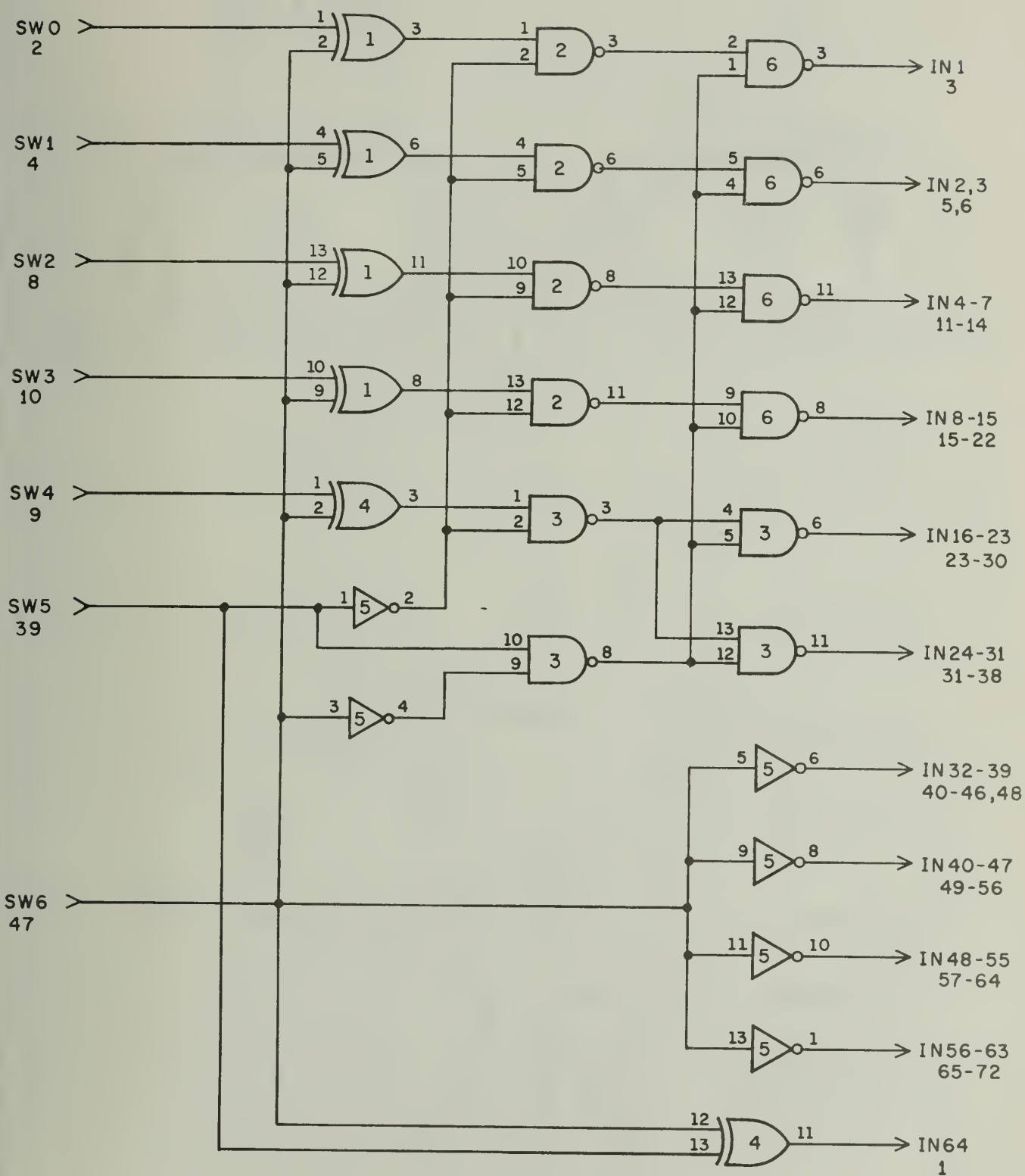


Figure A-2. Input Card

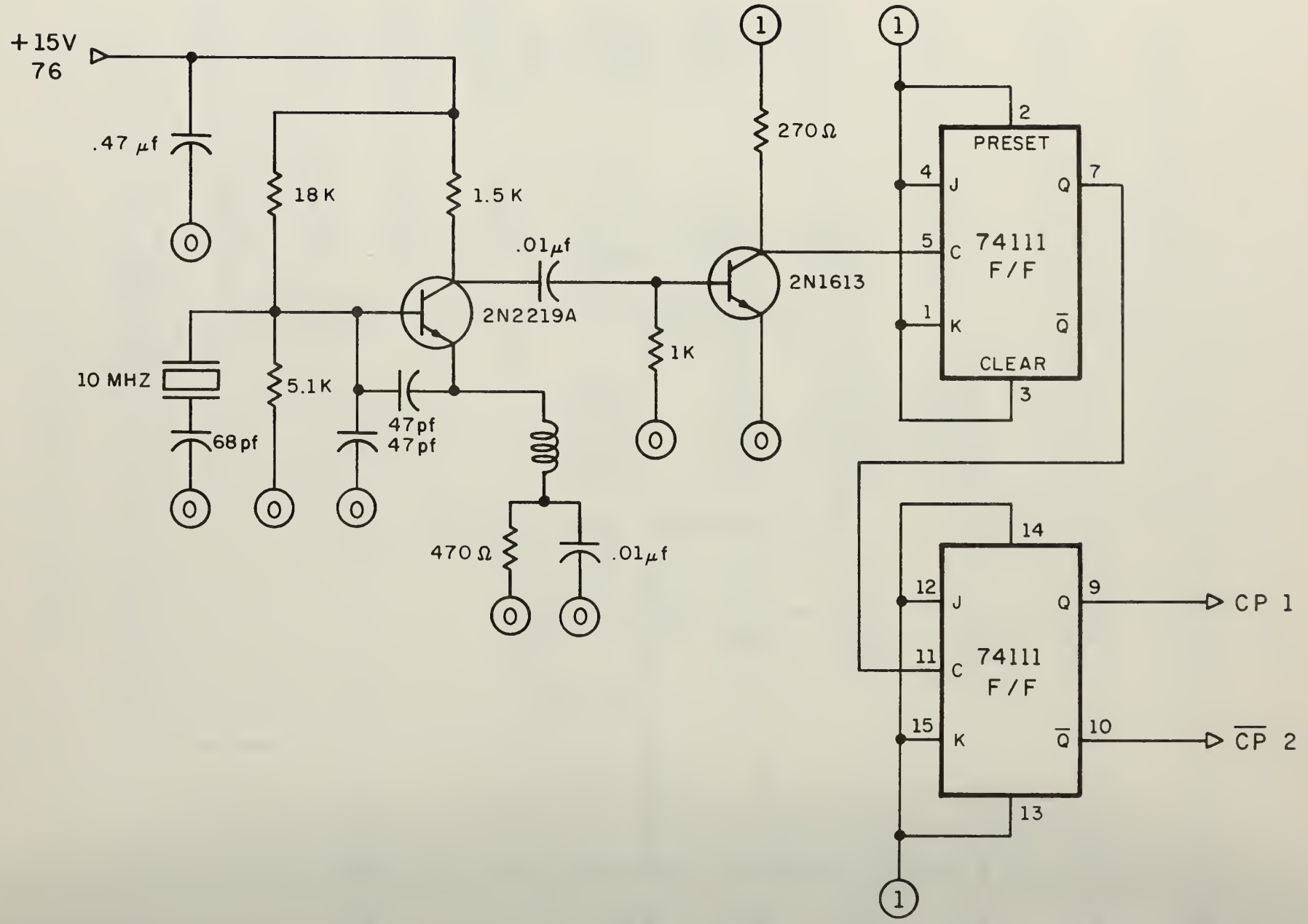


Figure A-3. Master Clock Card

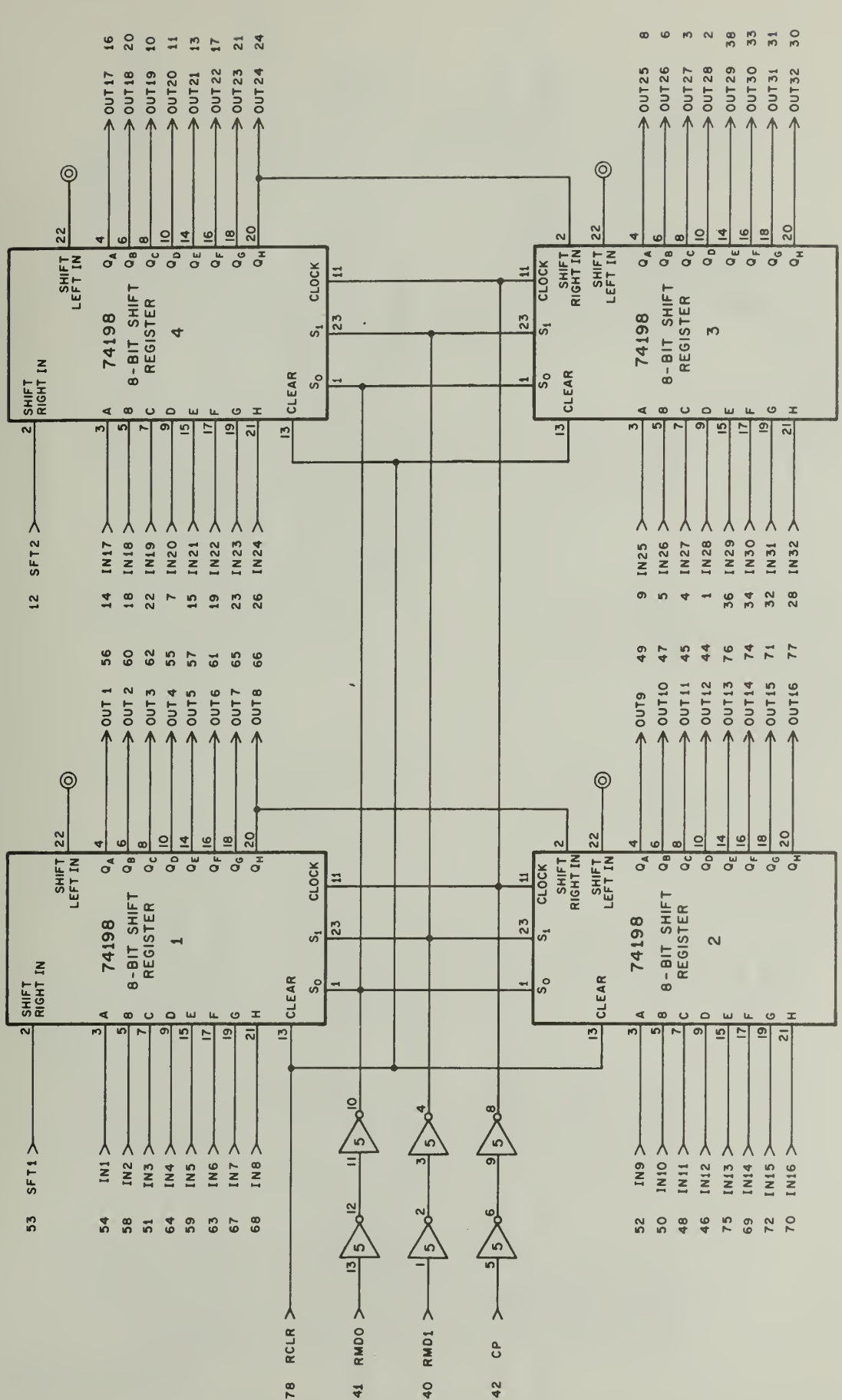


Figure A-4. 32-bit Shift Register Card

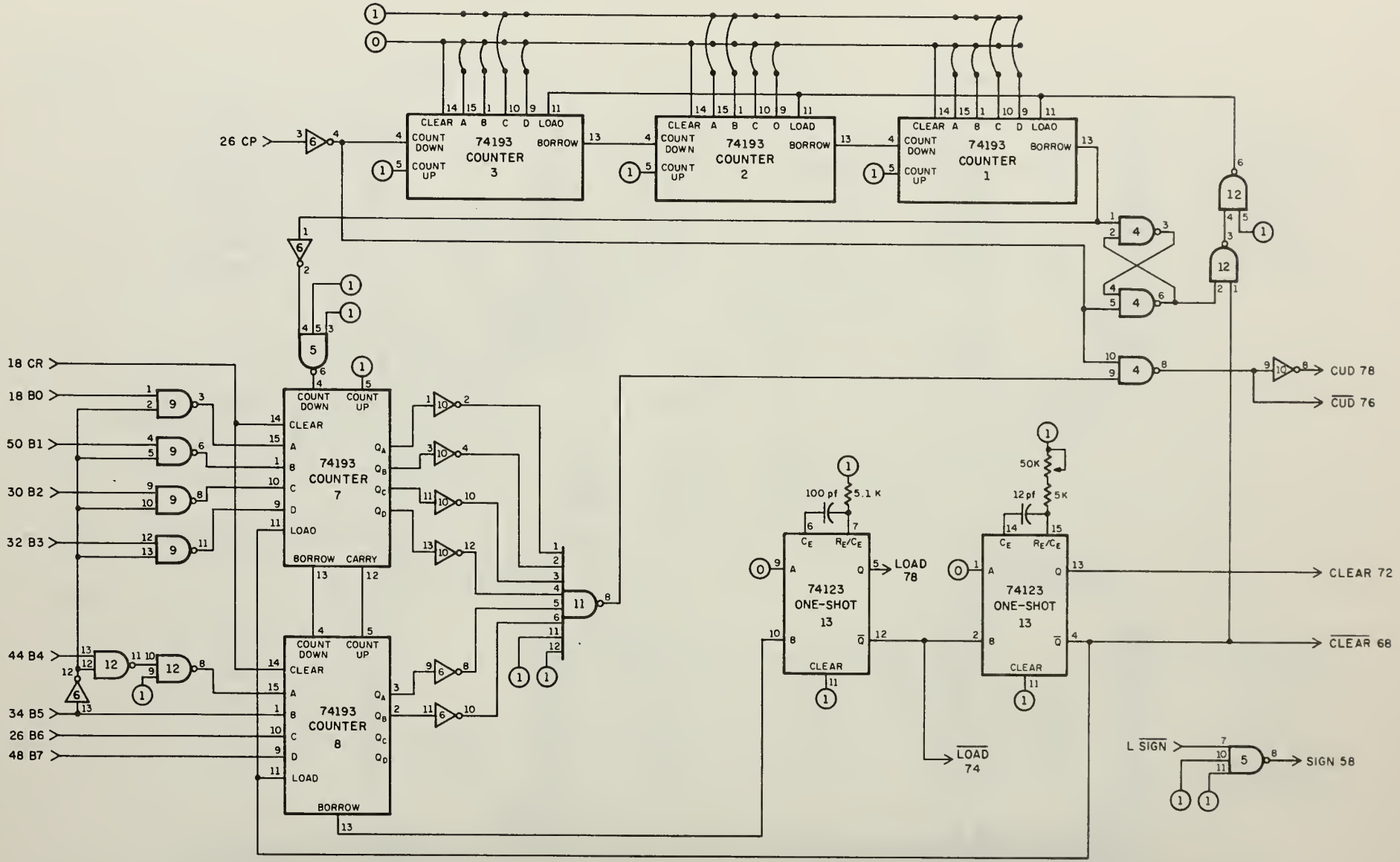


Figure A-5. Display Control Card

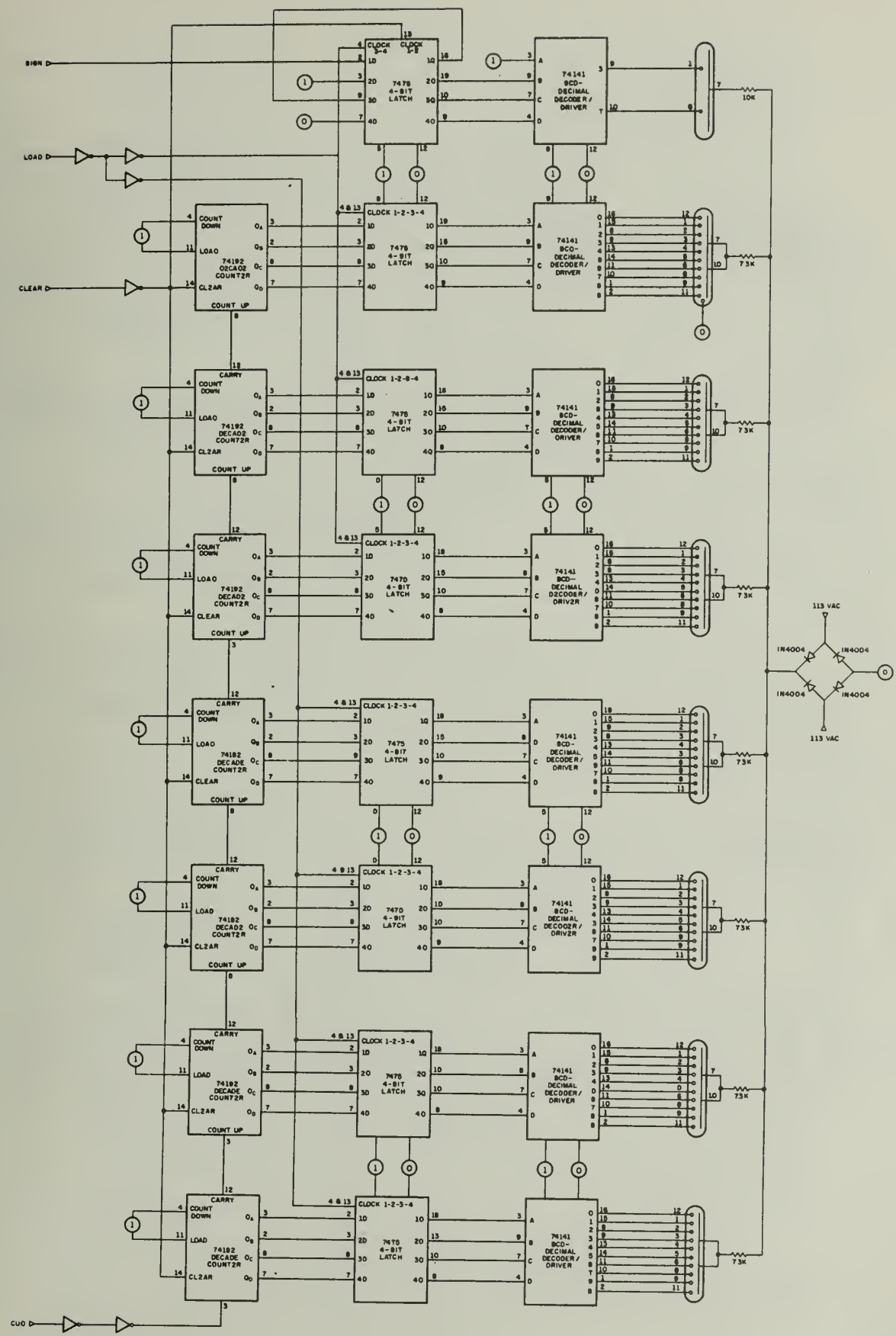


Figure A-6. Nixie Tube Card

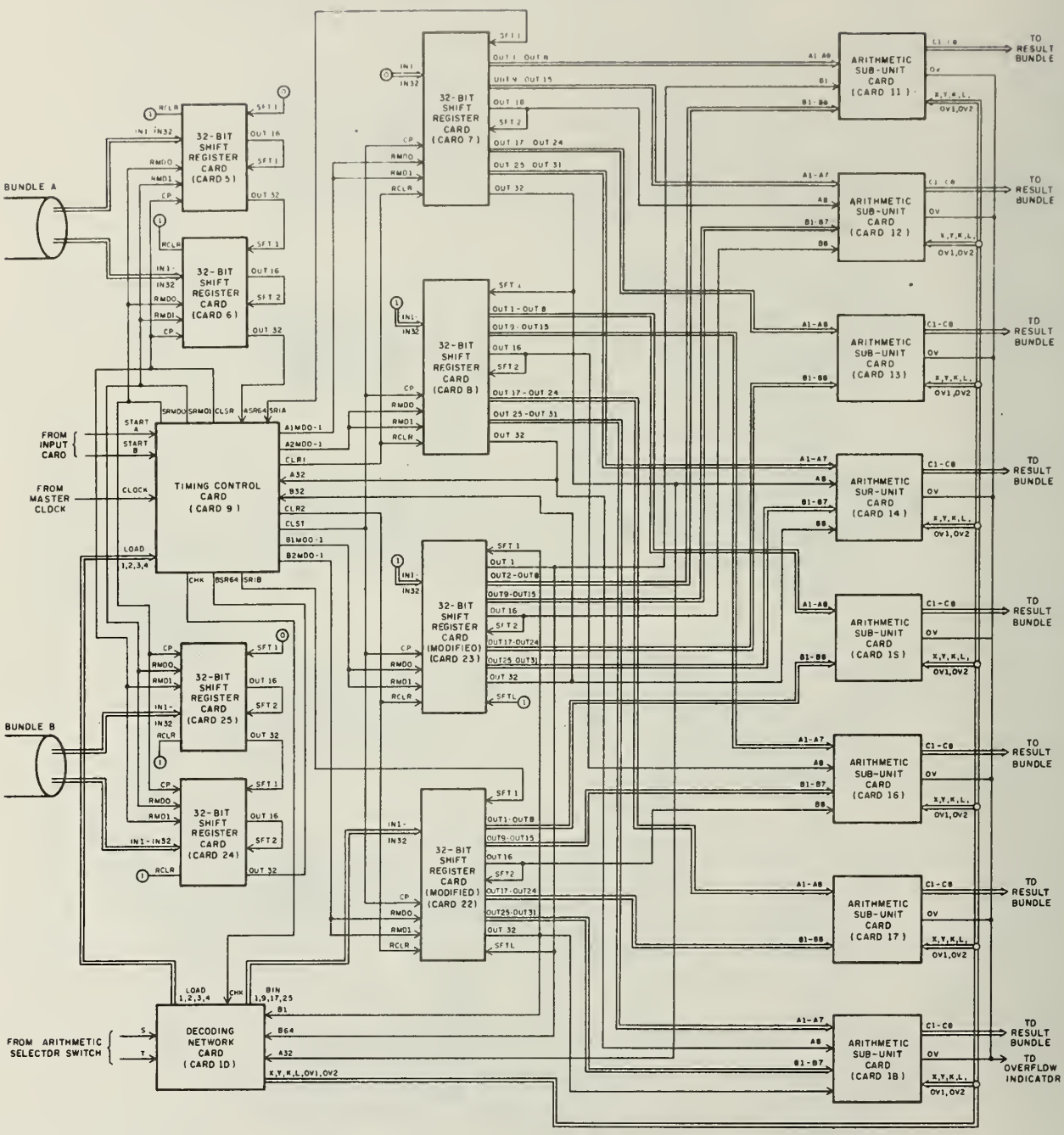


Figure A-7. Interconnections for the Arithmetic Unit

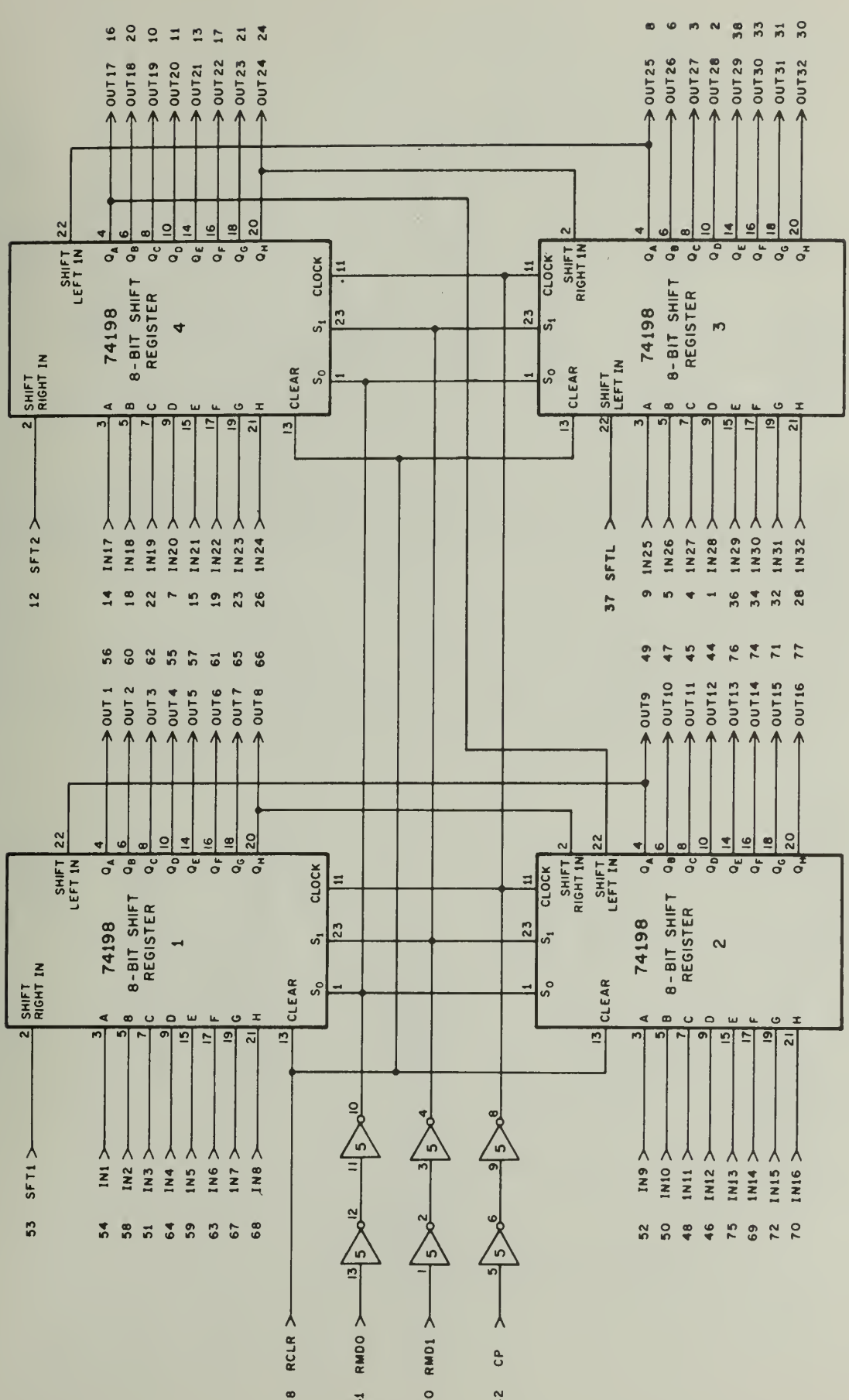


Figure A-8. 32-bit Shift Register Card (modified)

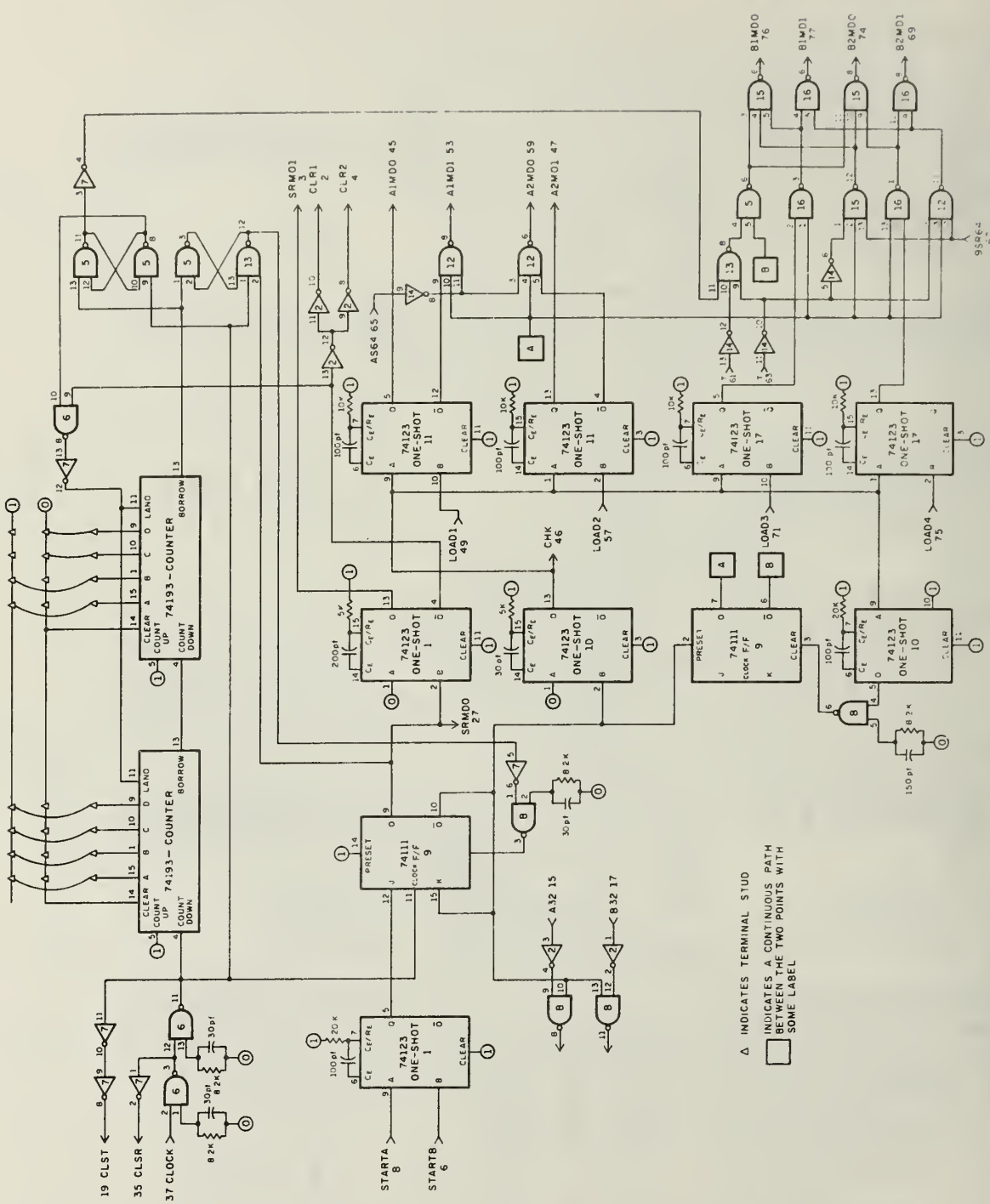


Figure A-9. Timing Control Card

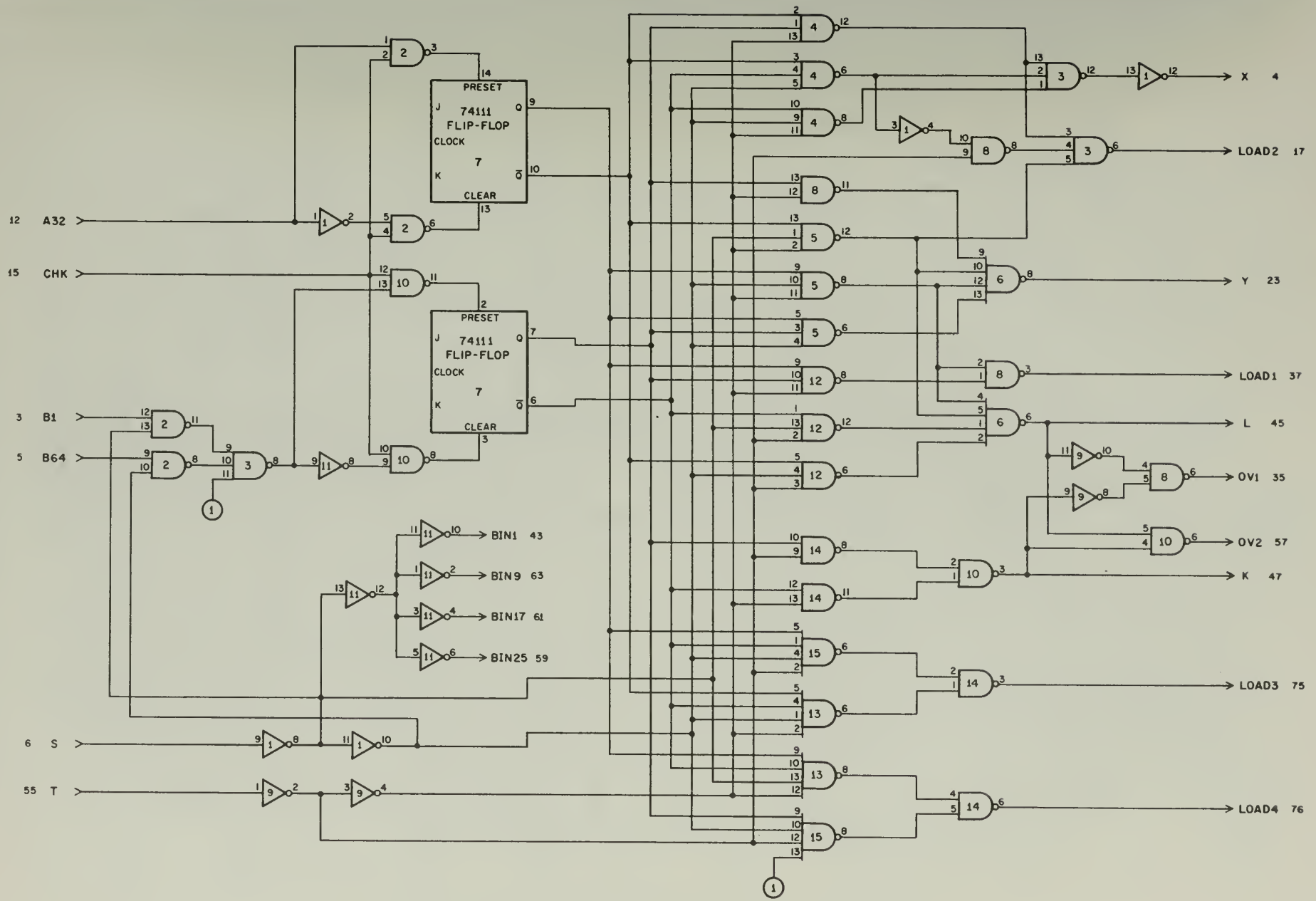


Figure A-10. Decoding Network Card

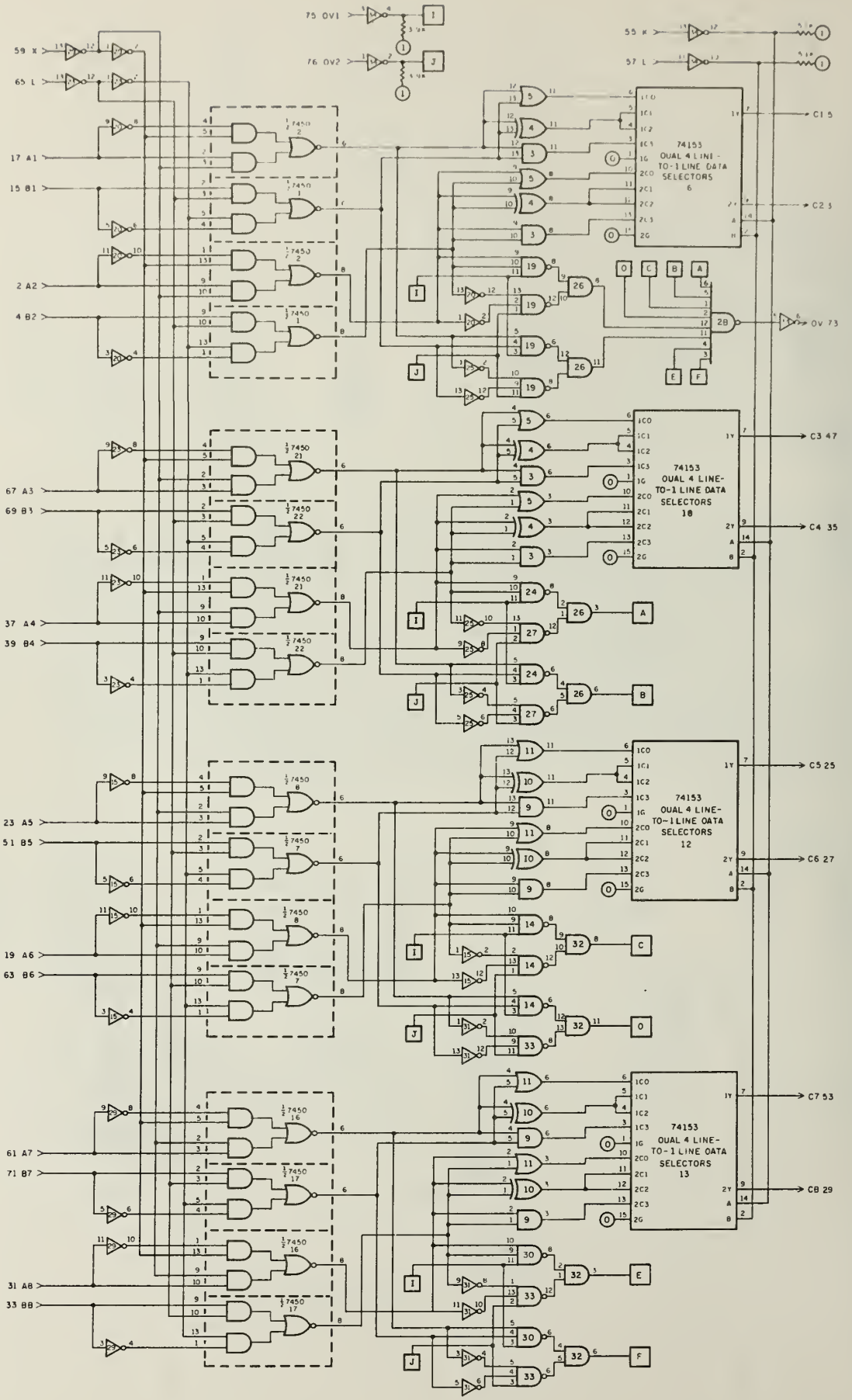


Figure A-11. Arithmetic Sub-unit Card

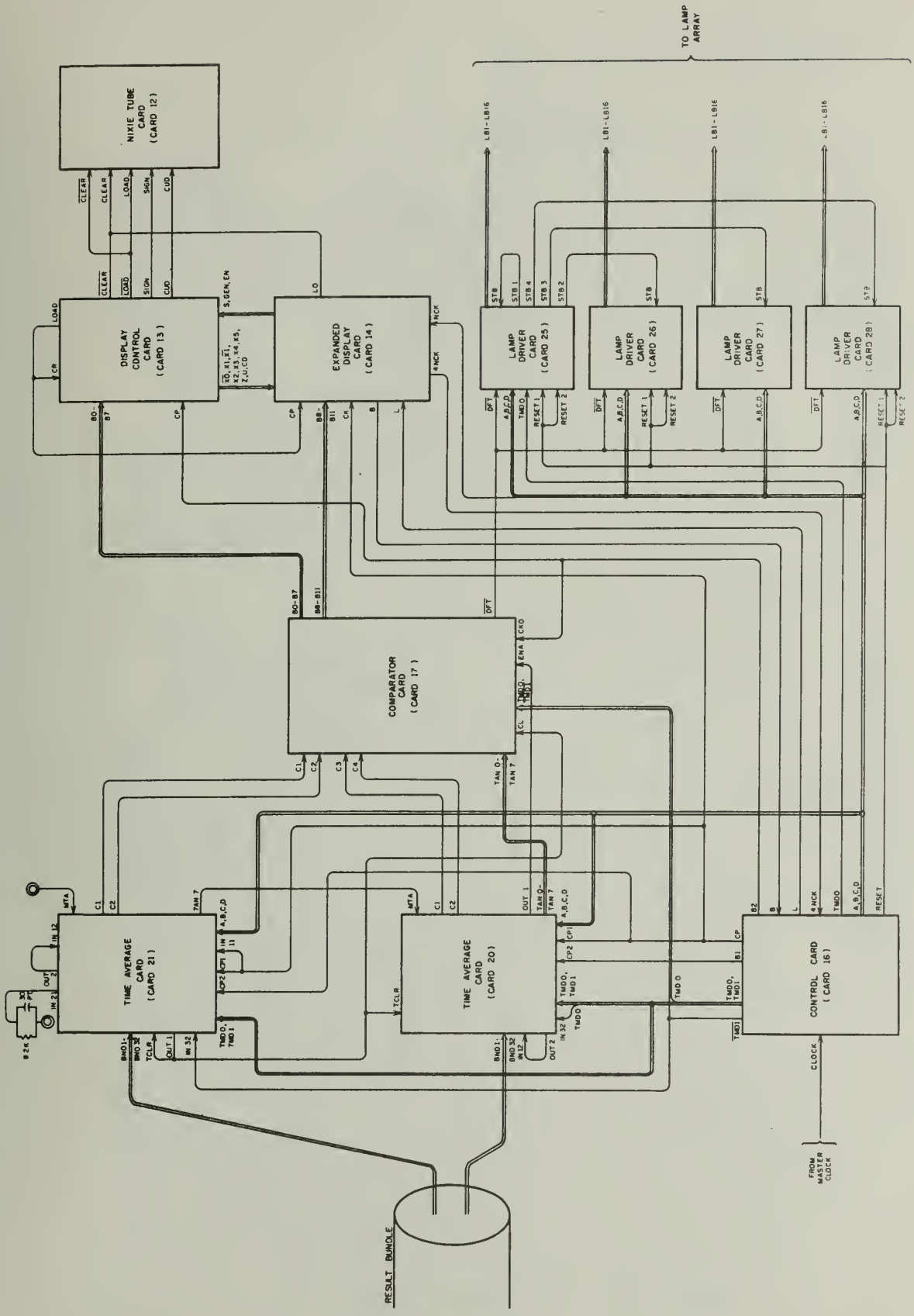


Figure A-12. Interconnections for the Processor

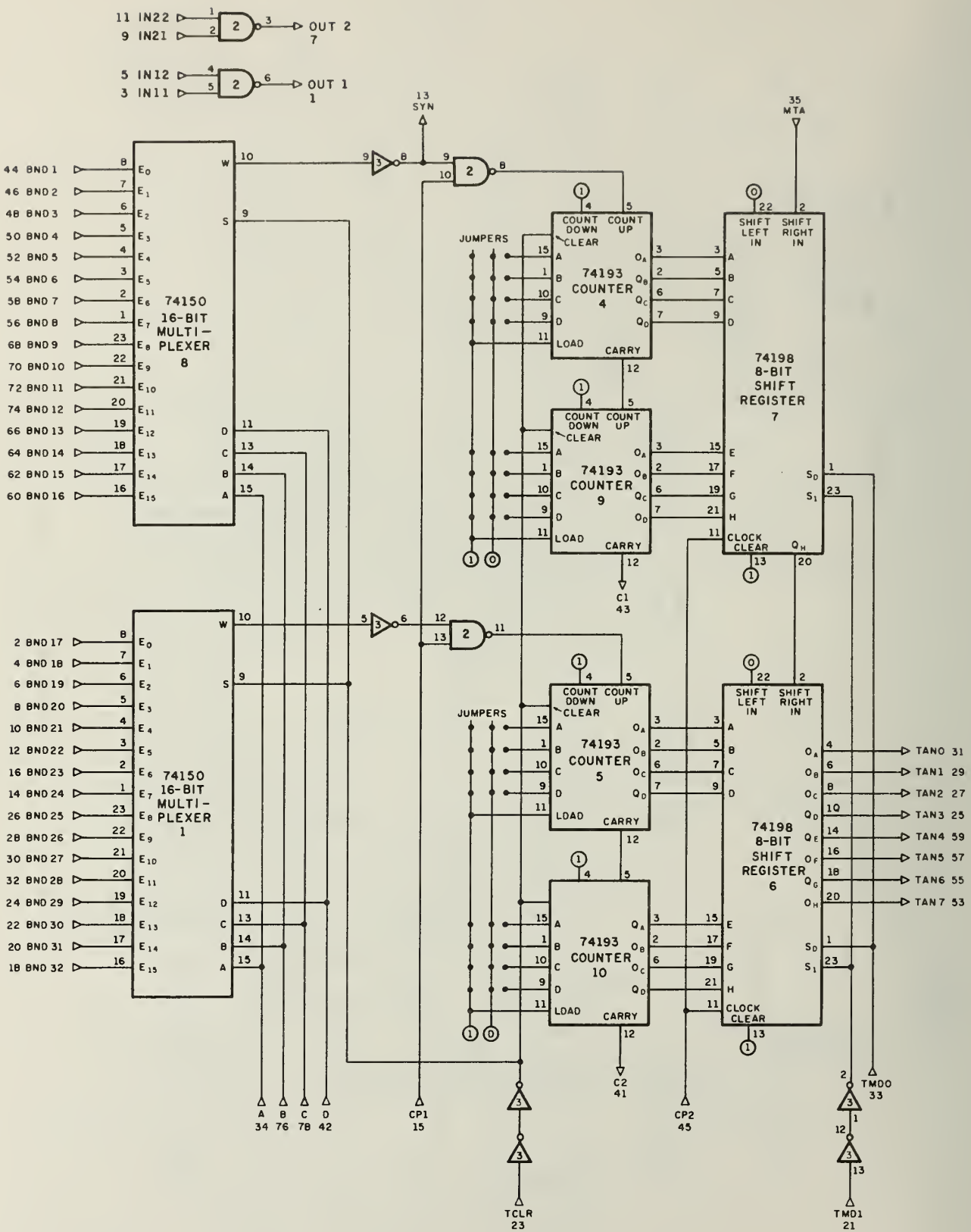


Figure A-13. Time Average Card

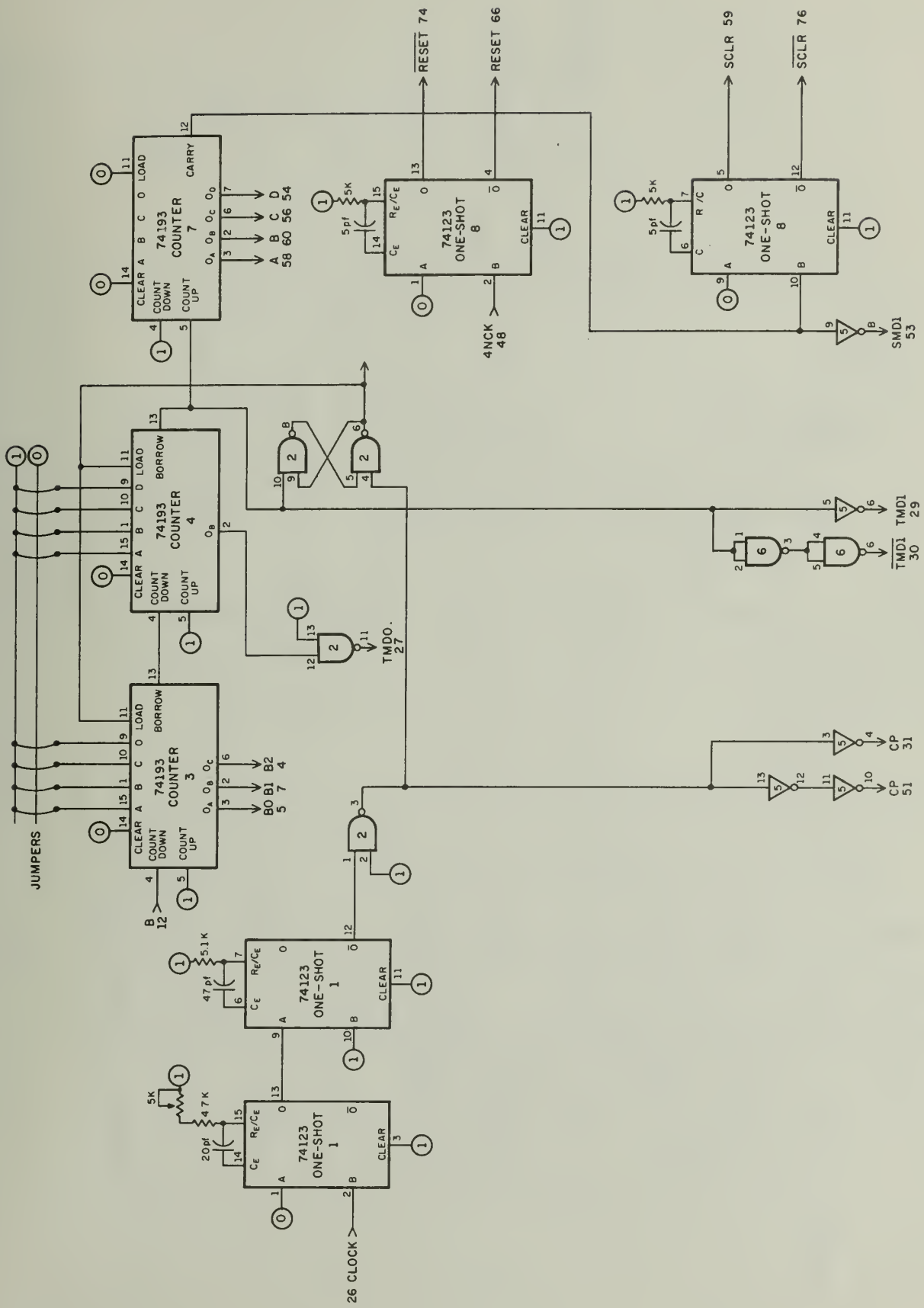


Figure A-14. Control Card

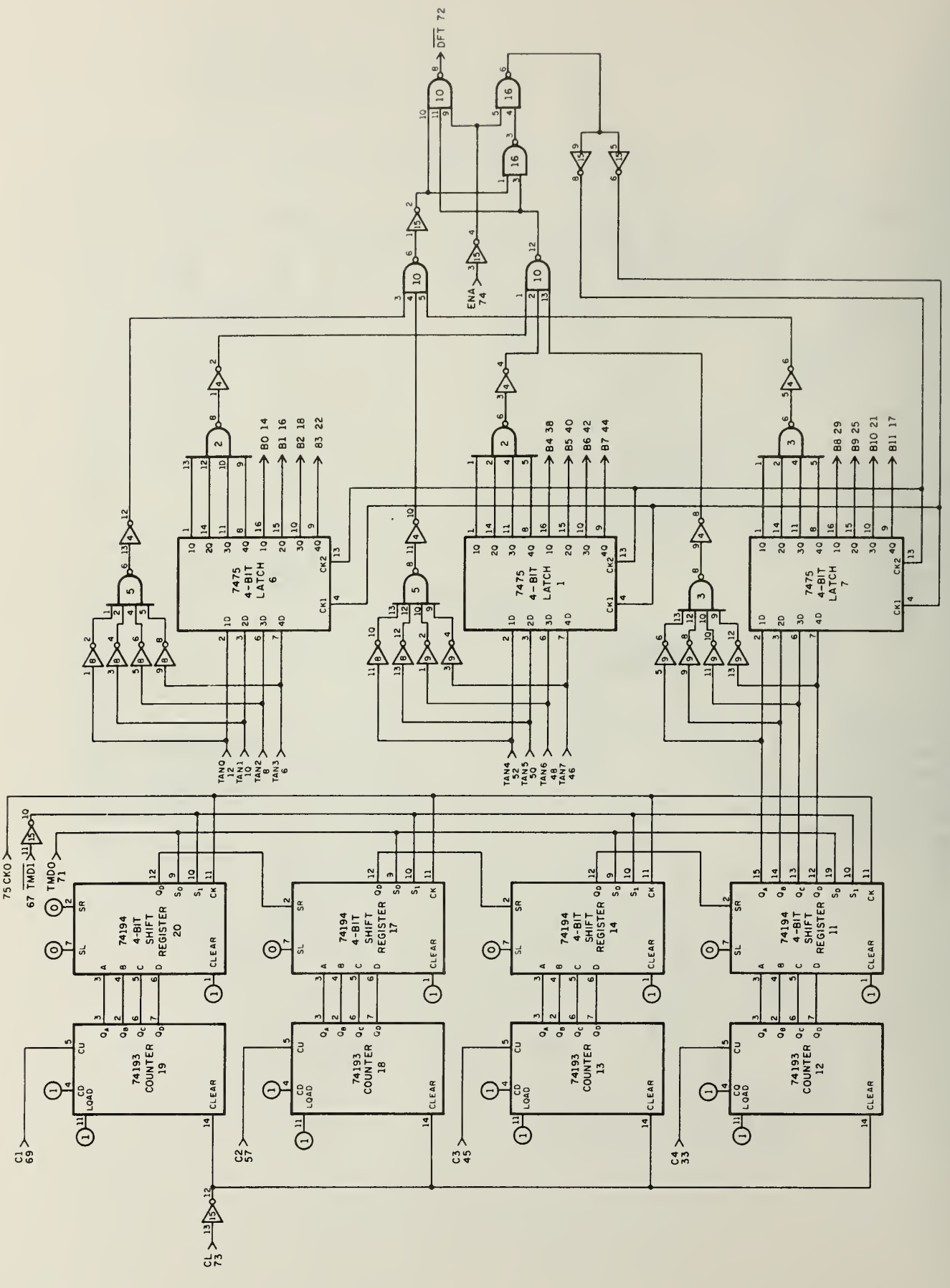


Figure A-15. Comparator Card

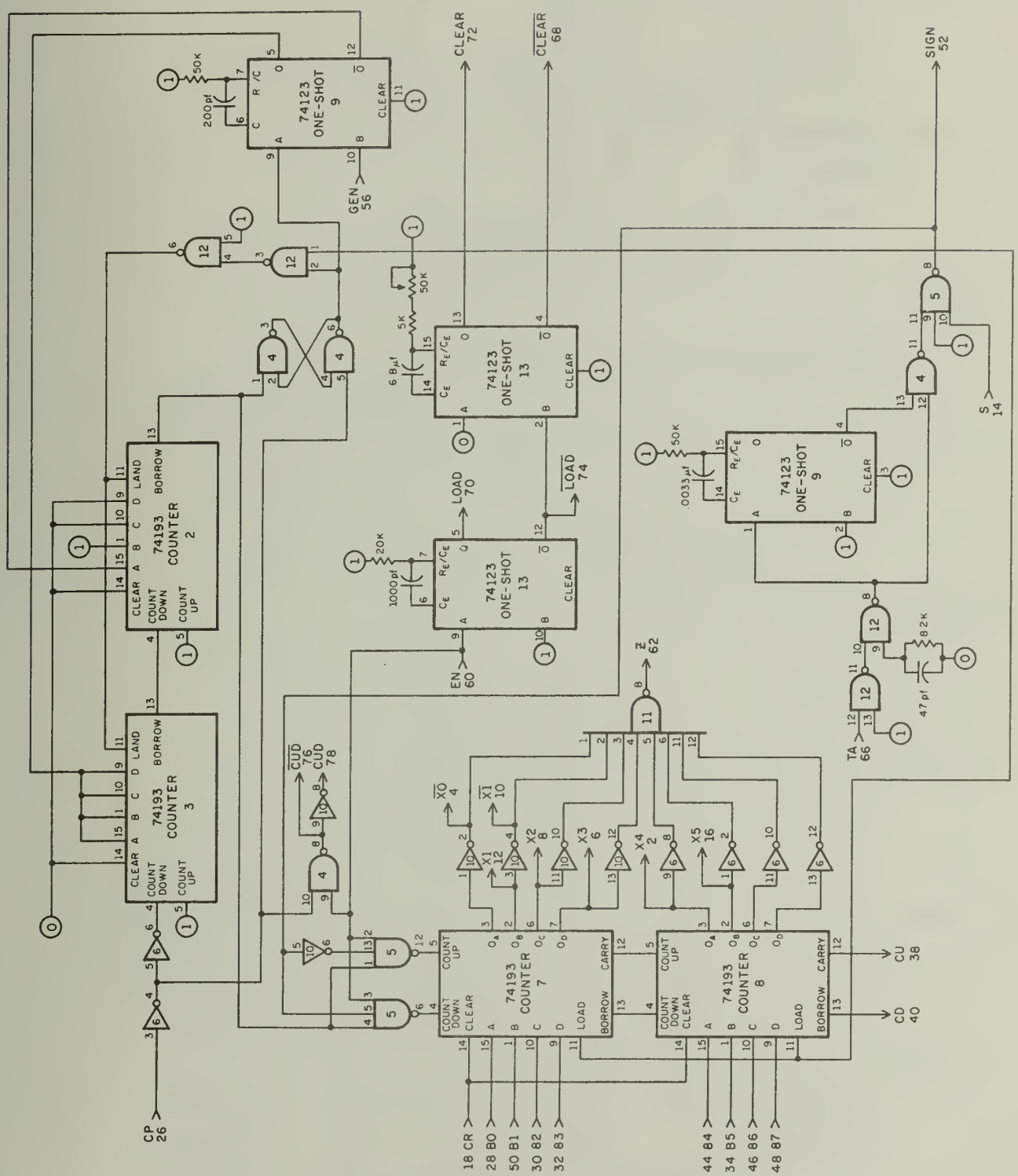


Figure A-16. Display Card

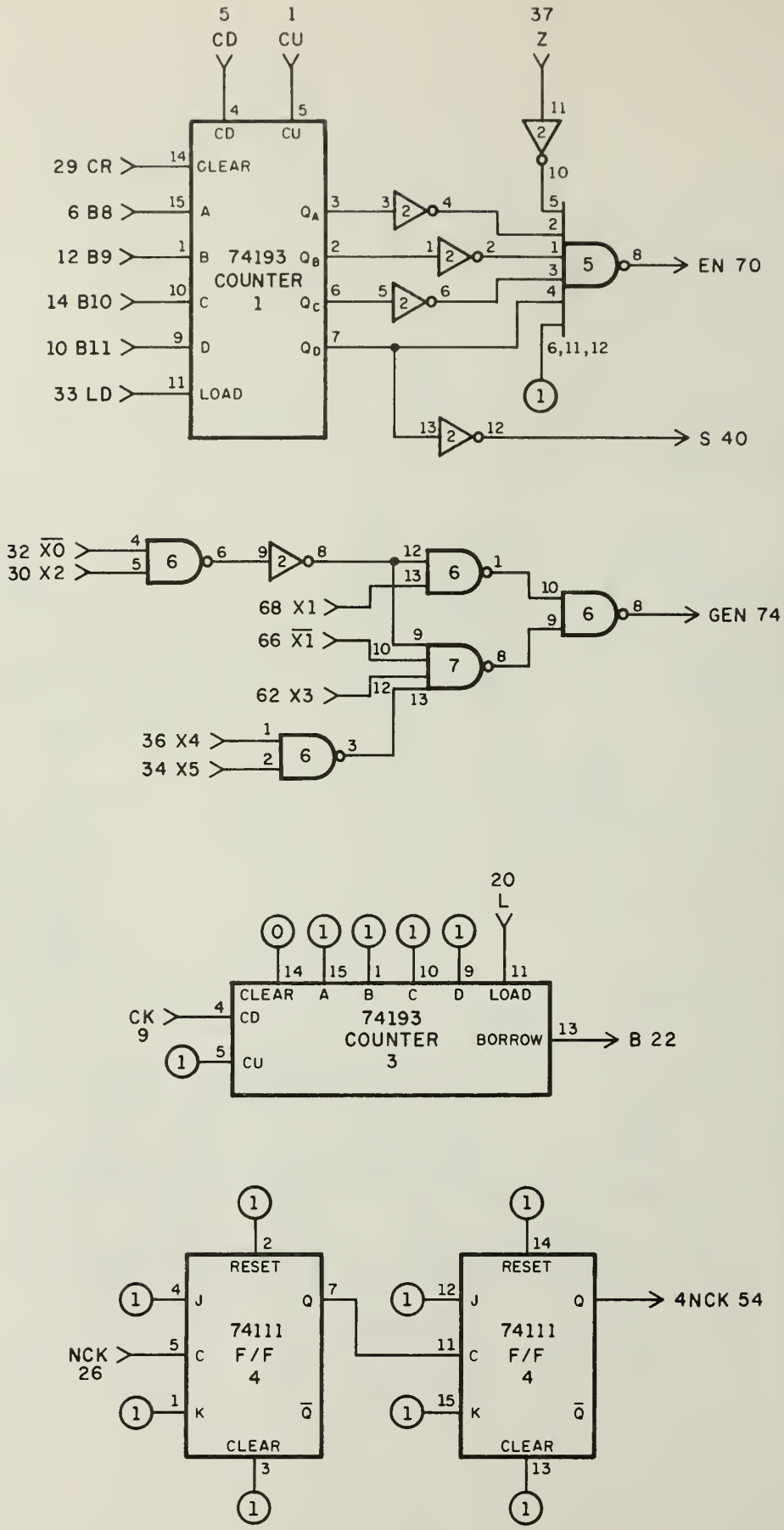


Figure A-17. Expanded Display Control Card

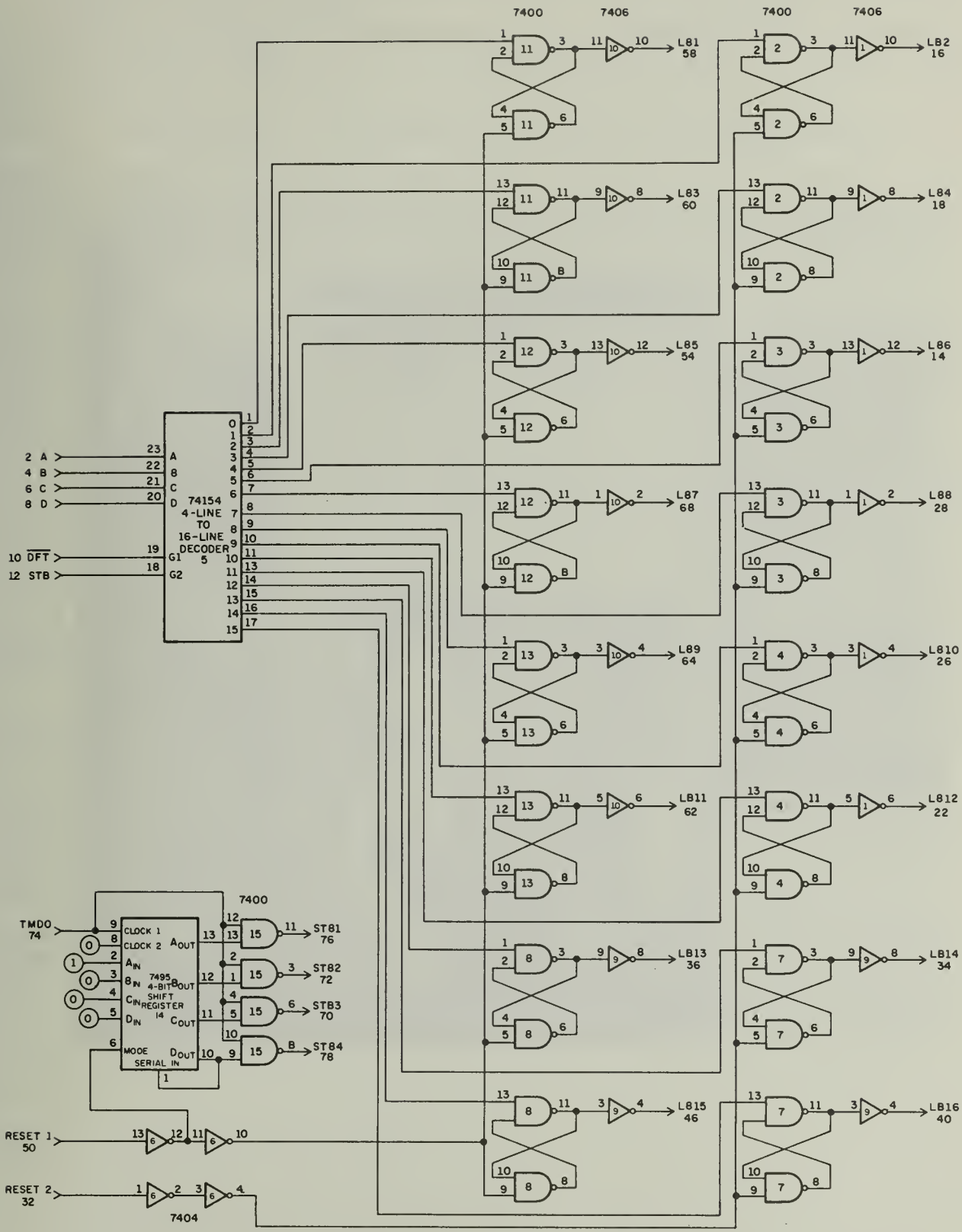


Figure A-18. Lamp Driver Card

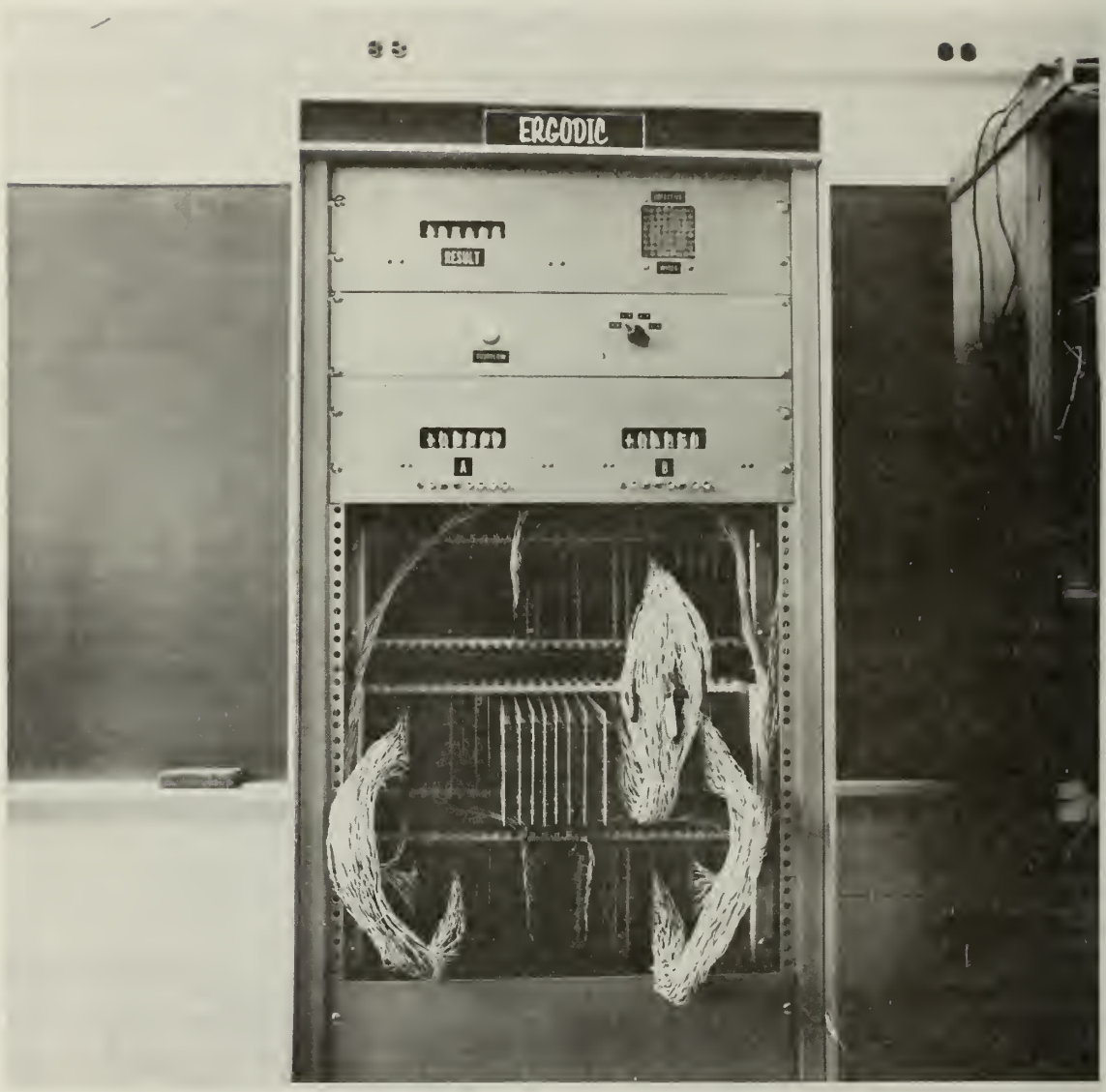


Figure A-19. Front View of ERGODIC

UNCLASSIFIED

Security Classification

DOCUMENT CONTROL DATA - R&D

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

1. ORIGINATING ACTIVITY *(Corporate author)*

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

2a. REPORT SECURITY CLASSIFICATION

Unclassified

2b GROUP

3. REPORT TITLE

ERGODIC: COMPUTING WITH A COMBINATION OF STOCHASTIC AND BUNDLE PROCESSING

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

Technical Report March, 1974

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

Cutler, James R.

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
March, 1974	60	6

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
N000 14-67-A-0305-0007	
b. PROJECT NO.	
c.	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*
d.	

10. AVAILABILITY/LIMITATION NOTICES

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Office of Naval Research 219 South Dearborn Street Chicago, Illinois 60604

13. ABSTRACT

A system called Ergodic which combines stochastic processing and bundle processing is described. Its background, theory, implementation are discussed in full as well as several suggestions for the improvement of this system. An appendix is included which shows all of the circuit cards used to implement Ergodic.

The system generates Ergodic bundles, performs arithmetic operations (multiplication, addition, and subtraction) on Ergodic bundles and then determines the information in the Ergodic bundle and also checks for failures in the system.

DD FORM 1 JAN 64 1473

UNCLASSIFIED

Security Classification

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R-74-630	2.	3. Recipient's Accession No.

4. Title and Subtitle

ERGODIC: COMPUTING WITH A COMBINATION OF STOCHASTIC AND BUNDLE PROCESSING

5. Report Date: March, 1974

6.

7. Author(s): James R. Cutler

8. Performing Organization Rept. No. UIUCDCS-R-74-630

9. Performing Organization Name and Address

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

10. Project/Task/Work Unit No.

11. Contract/Grant No. N000 14-67-A-0305-007

12. Sponsoring Organization Name and Address

Office of Naval Research
219 South Dearborn Street
Chicago, Illinois 60604

13. Type of Report & Period Covered: Technical

14.

15. Supplementary Notes

16. Abstracts

A system called Ergodic which combines stochastic processing and bundle processing is described. Its background, theory, implementation are discussed in full as well as several suggestions for the improvement of this system. An appendix is included which shows all of the circuit cards used to implement Ergodic.

The system generates Ergodic bundles, performs arithmetic operations (multiplication, addition, and subtraction) on Ergodic bundles and then determines the information in the Ergodic bundle and also checks for failures in the system.

17. Key Words and Document Analysis. 17a. Descriptors

Stochastic processing
Bundle processing
Ergodic bundle
Space average
Time average

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages
	20. Security Class (This Page) UNCLASSIFIED	22. Price