1972

Decimal-base binary logic (DBBL) adders and registers

James Oliver Bondi

Department:

DECIMAL-BASE BINARY LOGIC (DBBL) ADDERS AND REGISTERS

BY

JAMES OLIVER BONDI, 1949-

A THESIS

Presented to the Faculty of the Graduate School of the

UNIVERSITY OF MISSOURI-ROLLA

In Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

1972

Approved by

Paul D. Stigall (Advisor) James A. Tracy

James J. Higgins

ABSTRACT

This paper discusses a new type of base n adder and storage register. This new type of logic is called "n-base binary logic", or NBBL. The NBBL system is compared and contrasted with the Post base n system (a type of n-valued logic), the binary-coded base n system, and the straight binary system. The main purpose of this paper is to show that a decimal, or base 10, system can have some important inherent advantages over a binary system, such as greater daily operational efficiency. Furthermore, it is shown that a "decimal-base binary logic" system, or DBBL system, has inherent advantages over the Post and binary-coded decimal systems. A cost analysis of the DBBL system relative to the straight binary system is performed and several circuit realizations for general NBBL adders and storage registers are shown. Two of the storage register realizations are SCR models that the author has actually built and thoroughly tested.

ACKNOWLEDGEMENT

The author is very grateful to Paul D. Stigall for his continued advice and assistance in preparing this manuscript. The author also wishes to thank James H. Tracey for his helpful suggestions and comments and to thank Scott P. Stager for his willing assistance.

TABLE OF CONTENTS

Table of Contents (continued)

LIST OF ILLUSTRATIONS

List of Illustrations (Continued)

I. INTRODUCTION

In most of today's digital computers it is necessary to convert all input-output data between the decimal, human world and the binary, machine world. On large machines employing high-level languages, this necessary conversion is usually done automatically by sophisticated software and/or hardware conversion systems. On minicomputers, however, these automatic conversion routines often may not exist or may not be practical to use in the very small specific-task-oriented programs in which minicomputers are often applied. Thus, a minicomputer programmer may often find himself bothered with the tedious task of having to write conversion routines or having to actually use pencil and paper to convert his machine input-output data. Even on large machines having the automatic conversion routines, this conversion must still be done. So whether some sophisticated automatic routine does it or the programmer does it, time, and thus, money, must be spent somewhere along the line doing this conversion. Besides losing this conversion time, the binary machine user, if his system uses software conversion, must bear the ever-increasing cost of such software.

Obviously, machines that operate in the decimal world, just as humans do, would eliminate these problems. Of course, decimal machines, such as some of IBM's old BCD machines, have existed for quite some time. In general,

however, such machines have been shied away from, the excuse being that they were too expensive and too slow when compared to binary machines. This does not mean that the idea of decimal machines ever completely died. On the contrary, IBM's use of a faster, improved type of eight digit BCD adder in the recent IBM System/360 Model 195 indicates that the idea of decimal arithmetic machines is still quite alive even in cost-conscious industry [1].

This paper proposes a new type of adder and storage register made with "n-base binary logic" (NBBL). Furthermore, it is shown that a decimal mahcine using "decimal-base binary logic" (DBBL) adders and registers, although it would still cost more to build, can offer its user the important advantage of greater daily operational efficiency.

II. ADDERS

The adder circuitry in a digital computer can be designed to operate on numbers in any base. This section discusses straight binary, traditional binary-coded base n, Post base n, and NBBL adders.

A. Standard Binary Adder

The internal circuitry of a standard binary adder is of little importance in this paper. Therefore, the standard binary adder stage will be viewed as a "black box" containing a carry input from the previous stage, one input from each of the two digits to be added, one sum digit output, and a carry output to the next adder-stage (Fig. 1).

B. Binary-Coded Base N Adder

The binary-coded base n adder is traditionally a modified standard binary adder designed to operate on numbers in bases greater than two. The main difference between a binary-coded base n adder stage and a straight binary adder stage is that whereas the straight binary device has only one input line from each digit to be added and one sum output line, the binary-coded device has multiple input lines from each digit to be added and multiple sum output lines. These multiple lines represent a binary-coded form of the digits [2] (Fig. 2). Note that only one line is shown for the input and output

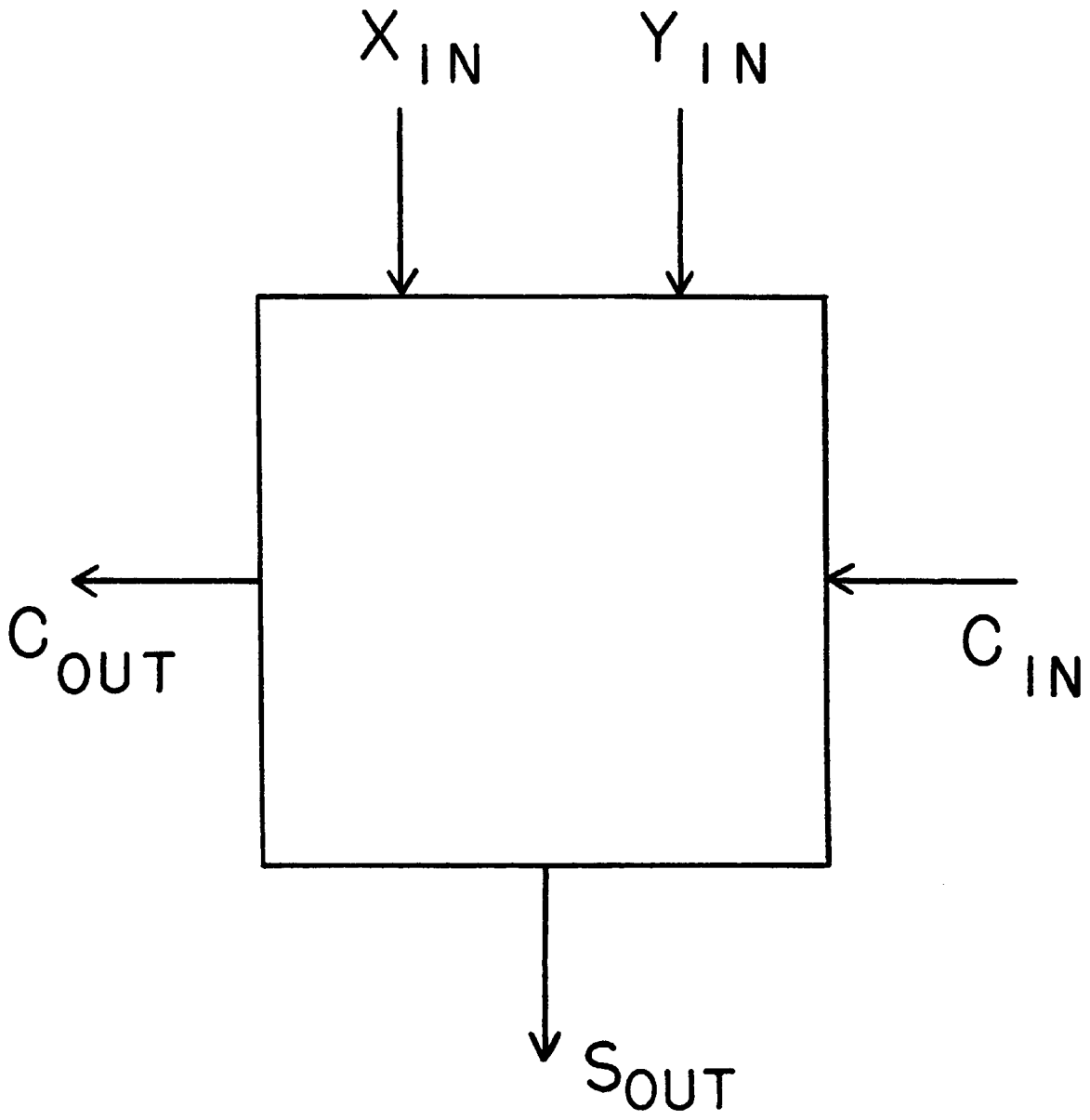


Fig. 1. Standard Binary Adder Stage

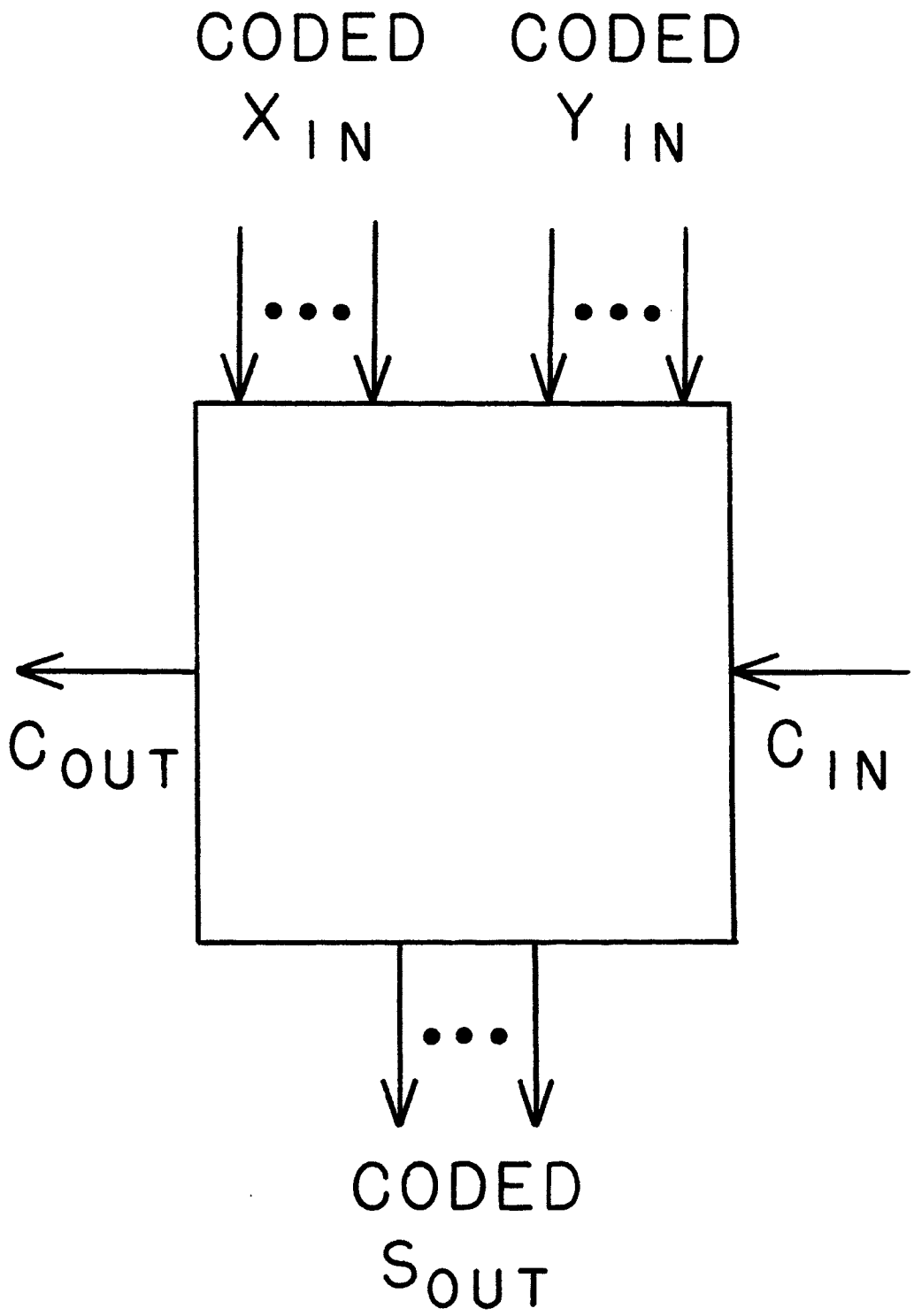
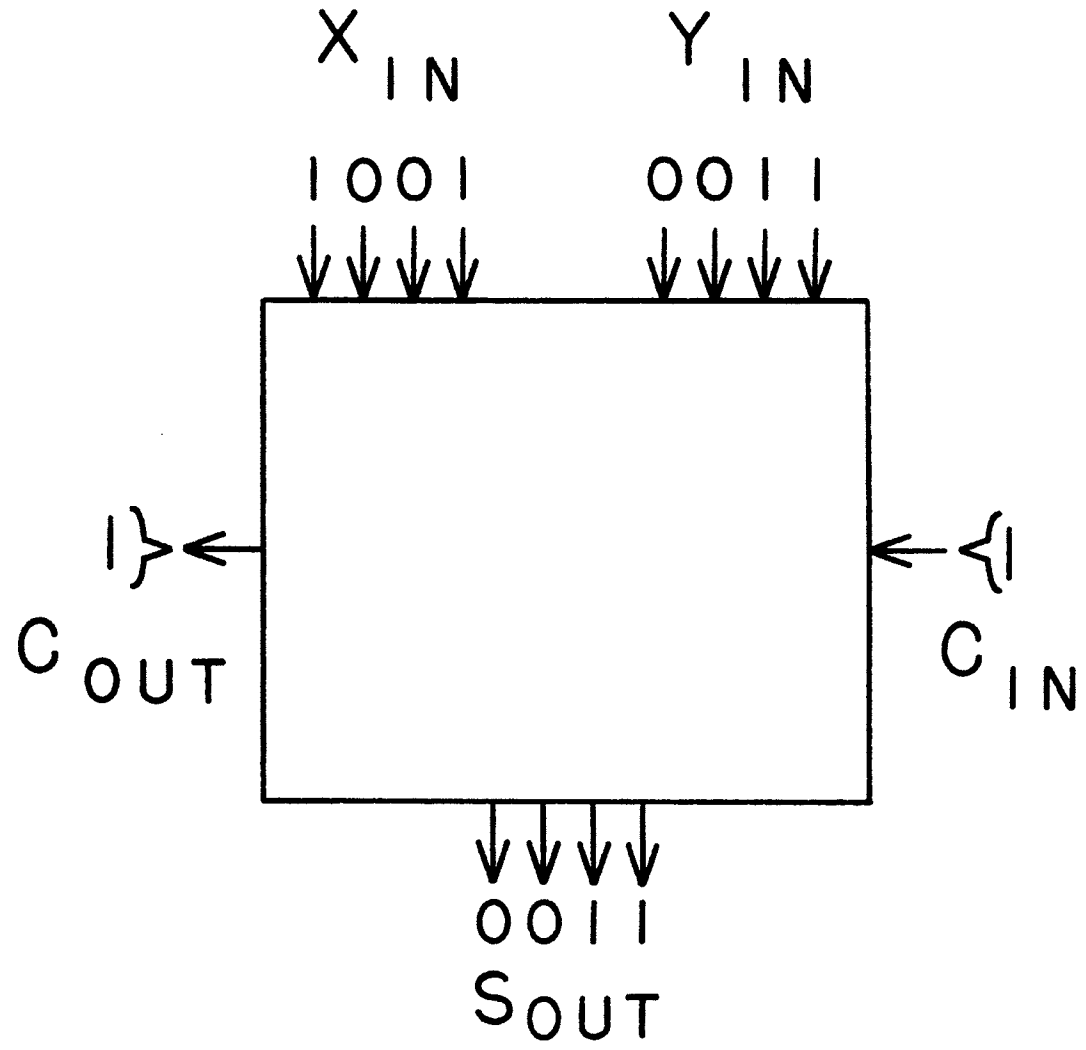


Fig. 2. Binary-Coded Base N Adder Stage

carries. A little thought verifies that, no matter what the base, as long as only two numbers are being added, the input and output carries of any single adder stage can only take on the values 0 or 1. Thus, one binary-valued line can suffice in representing the input and output carries of any adder stage.

As might be expected, there are some problems involved in trying to convert a straight binary adder into a binary-coded adder which will operate successfully on nonbinary numbers. The most important of these problems is that if the base being used is not a power of 2, correction circuitry must be added to each adder stage to correct the sum digit output whenever a decimal output carry occurs [1], [2]. (This correction circuitry is not necessary if one uses a direct logical implementation designed specifically for binary-coded base n addition instead of the traditional approach of using a modified standard binary adder [1].) Consider the example in Fig. 3 using a binary-coded decimal, or BCD, adder stage. The coded x input is decimal 9, or binary 1001, the coded y input is decimal 3, or binary 0011, and the input carry is decimal 1, or binary 1. Since the correct sum is decimal 13, the BCD adder stage should produce a carry output of 1 and a sum digit of decimal 3, or 0011 in binary-coded decimal. However, note that to get the correct sum digit output, the adder stage must perform a correction to the initial result which consists of




```
  1001   X IN
  0011   Y IN
+    1   C IN
------
  1101   INITIAL S OUT
+ 0110   CORRECTION FACTOR
------
  0011   CORRECTED S OUT
```

Fig. 3. Correction Procedure in Traditional BCD Addition

subtracting decimal 10. That is, binary 1010 must be subtracted from the initial result whenever the correct sum of the input data is greater than decimal 9. This is usually done by the equivalent process of adding binary 0110 to the initial sum digit result [1], where binary 0110 is the 2's complement of binary 1010 if the normal sign bits are deleted. Any output carry that would result from the correction process is not needed and therefore ignored.

C. Post Base N Adder

Wojcik and Metze, having found that multi-valued logic could be advantageous in the control circuitry of asynchronous systems, decided to investigate the feasibility of multi-valued logic in adders and storage registers. Post adders and storage registers were therefore researched by Wojcik and Metze in detail [2], [3]. Unlike the standard binary and binary-coded systems already discussed, the Post base n system is not purely 2-valued logic. On the other hand, after detailed study, one finds that the Post base n system is not purely n-valued logic either. Actually, it would be most accurate to call this system a hybrid of 2-valued and n-valued logic. This fact will become clearer as the discussion progresses.

Perhaps the most pressing problem with n-valued

logic systems is deriving a set of basic functions with inexpensive circuit realizations, combinations of which can be used to form any complex logical function [2], [3], [4]. The systems proposed by Wojcik and Metze use three basic functions: (1) the "x^i" functions ($i = 0,1,....,n-1$), (2) the "MIN" function, and (3) the "MAX" function. These functions are described in Fig. 4. (Note that throughout this paper, whenever n-valued functions or n-valued logic are being discussed, a general variable name without a superscript, such as "x," will be used to indicate a single, fully coded, multi-valued line. For a general base n, such a variable name would usually indicate an n-valued line, although in the case of carries, a "c" without a superscript indicates a fully coded 2-valued (0 or 1 volt) carry line. However, a superscripted variable name, such as "x^i," indicates one of a group of 2-valued (0 or n-1 volts) lines. The entire group of "x^i" lines compositely represents, in general, a "1-out-of-n" multi-line coded form of the general n-valued variable "x." In the special case of carries "c^0" and "c^1" are the two lines which together form a "1-out-of-2" coded form of a carry.)

The use of these functions becomes clearer as one studies Fig. 5 which illustrates the basic form of a Post base 10 adder stage. Note that, in this adder stage, the input lines and lines c^1, s^1, s^2,....., s^9 are all

$$X^{\dot{i}} = \begin{cases} N-1, & X = \dot{i} \\ 0, & X \neq \dot{i} \end{cases}$$

$$MIN(X,Y) = \begin{cases} X, & X \leq Y \\ Y, & Y \leq X \end{cases}$$

$$MAX(X,Y) = \begin{cases} X, & X \geq Y \\ Y, & Y \geq X \end{cases}$$

$$\text{WHERE } \dot{i}, X, Y \text{ ALL} = 0, 1, \ldots, N-1$$

Fig. 4. Three Useful Functions in Post Base N Logic

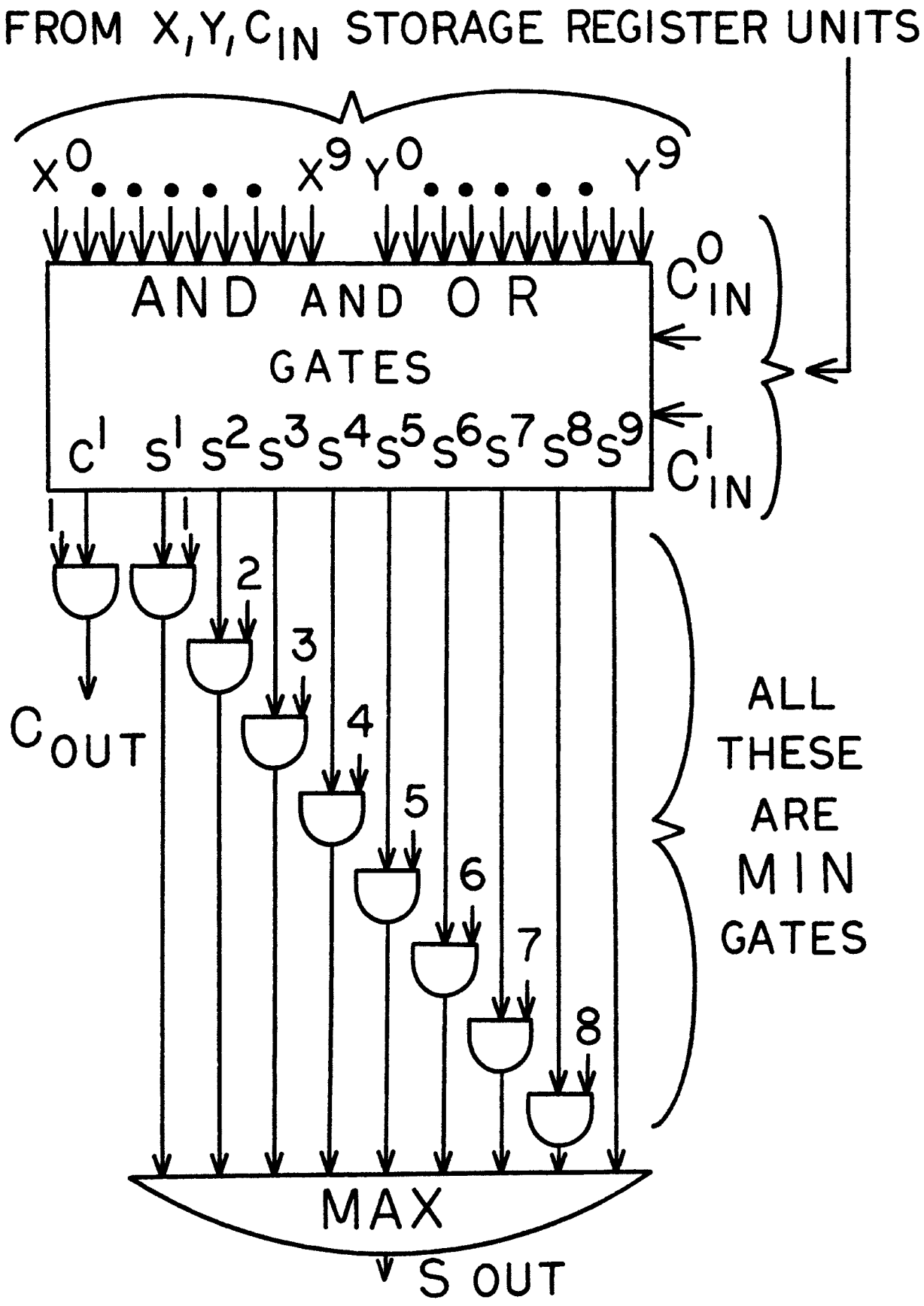


Fig. 5. Post Base 10 Adder Stage

binary valued lines, taking on only the values 0 or 9. It must be pointed out that in a group of lines such as x^0, x^1,....., x^9 or s^1, s^2,....., s^9, only one line in the group, say x^2 or s^5, can exhibit the high voltage (9 volts in base 10) at any given time. At this same instant, all other lines in the group must exhibit 0 volts. That is, line s^5, for example, assumes a voltage level of 9 volts whenever the sum digit output should be 5 and assumes a voltage level of 0 volts otherwise. The final sum output line, s, on the other hand, is a multi-valued line which can have any integer value from 0 through 9. This multi-valued output line, s, is formed essentially by "recombining" the binary-valued lines s^1, s^2, ..., s^9 via the MIN and MAX gates shown. Wojcik and Metze make the traditional assumption that such a multi-valued line can change from one value to another nonadjacent value, say from 0 to 2, in such a way that one need not worry about the effect of intermediate values, such as 1, on the circuitry that the line drives. Obviously, one cannot justify such an assumption by claiming that the line changes instantaneously so that intermediate values do not really even appear. Such instantaneous changes simply do not occur in the real world. Indeed, if the circuitry being driven by a multi-valued line were asynchronous sequential circuitry, extreme design care would have to be taken to insure that the appearance of undesired intermediate

values produced no ill effects such as race conditions. On the contrary, to realistically justify this assumption, Wojcik and Metze limit the circuitry being driven by this multi-valued line to clocked sequential circuitry so that the appearance of these intermediate values poses no problem. Thus, it is important to realize that intermediate values in n-valued logic could create problems. Finally, in Fig. 5, the carry output line as shown is identical in its functioning to the carry output line of a standard binary or binary-coded base n adder stage. It is fully coded so that a level of 1 volt indicates the output carry is a "1" and a level of 0 volts indicates the output carry is a "0." Thus, this carry output line may be separated into the next-stage carry input functions c^0 and c^1. This separation can be accomplished by passage through a Post base 2 storage register unit.

The circuit realizations that Wojcik and Metze assume for the MIN and MAX gates also demand realistic comment. Ordinary diode-resistor AND gates are proposed as MIN gates and diode-resistor OR gates are proposed as MAX gates. Fig. 6 illustrates the most basic forms of these gates. Since, in diode-resistor gates, only the diode with the lowest input voltage will conduct in an AND gate and only the diode with the highest input voltage will conduct in an OR gate, it is true that these gates

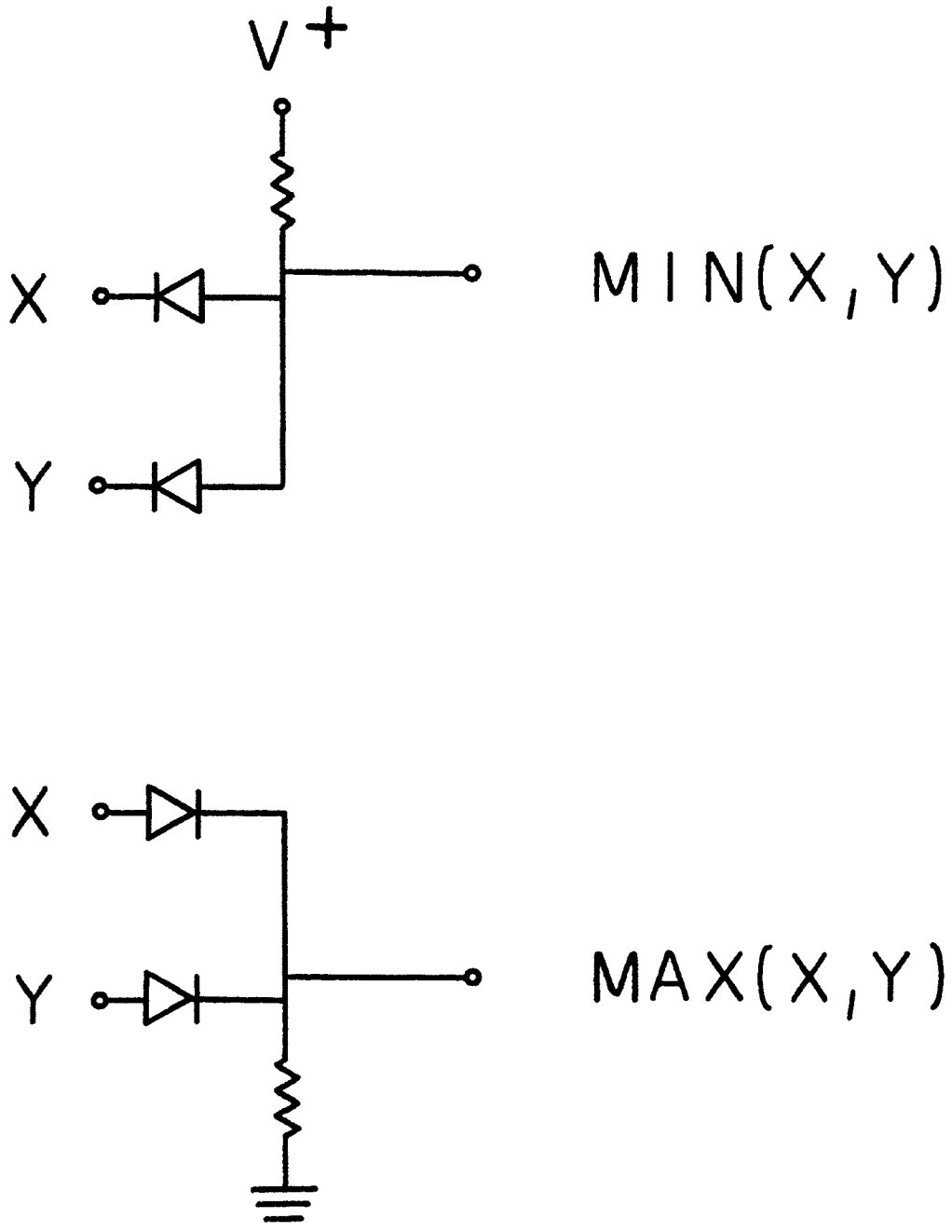


Fig. 6. Possible MIN and MAX Gate Realizations for N-Valued Logic

will function as MIN and MAX gates respectively. However, even though more sophisticated models than the ones of Fig. 6 are readily available, because all diode-resistor gates are basically passive, their ability to transmit a voltage from input to output without degradation is generally poor compared to that of active gates having means for voltage reamplification. As a result, a desired input value of 5 volts, for example, might drop to approximately 4 volts after passing through only a few MAX gates. Such a situation is very dangerous in multi-valued (nonbinary) logic which generally requires accurate voltage levels to insure proper functioning.

D. NBBL Adder

The final type of adder to be discussed is the "n-base binary logic," or NBBL, adder. This adder, of the author's own design, can be viewed as a simplification, or special case, of the Post base n adder. Fig. 7 illustrates a "decimal-base binary logic," or DBBL, adder stage. The realization shown in Fig. 7 is entirely combinational logic. Thus, the "black box" in this figure essentially contains a large decoder possessing 22 inputs and 12 outputs. Detailed analysis of the general realization equations presented by Wojcik and Metze for their Post adders indicates that these equations are very similar to the equations that would be used to

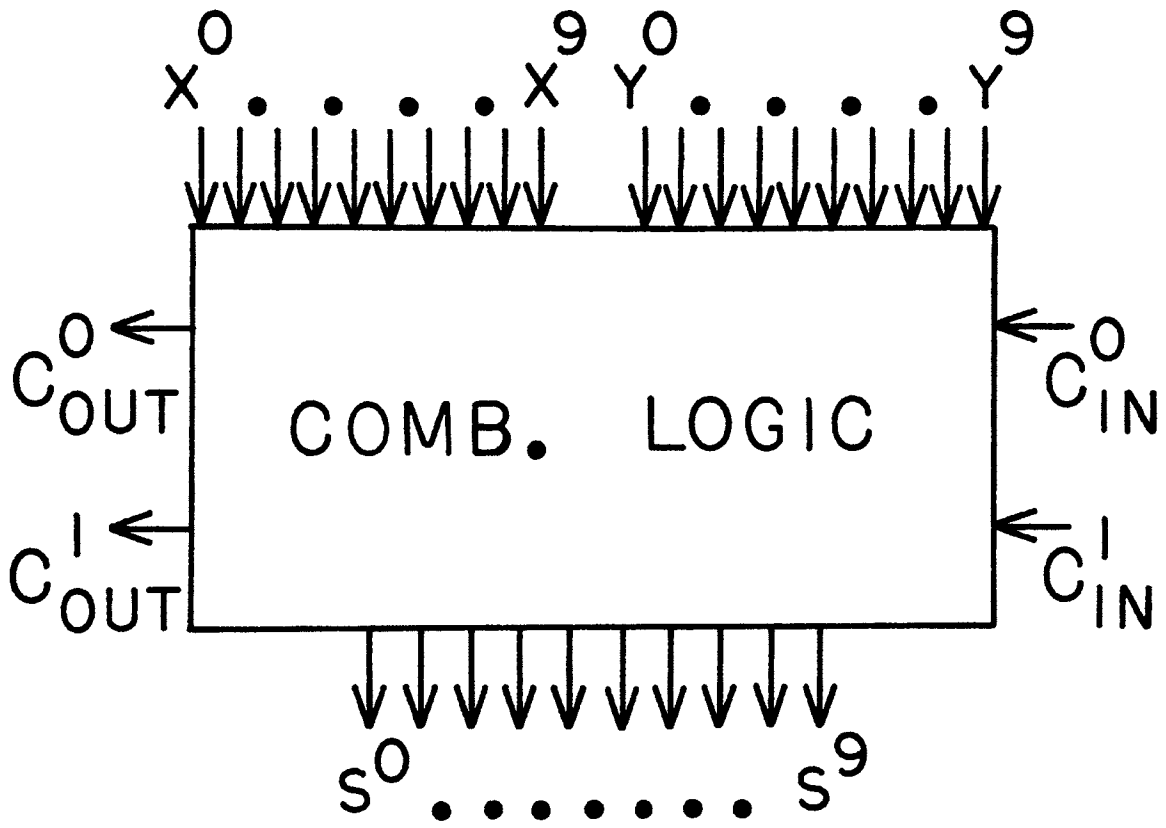


Fig. 7. DBBL Adder Stage (Combinational Logic)

realize NBBL adders. Thus, for example, a Post base 10 adder stage is quite similar to a DBBL adder stage. Note that if one added an s^0 and a c^0 output to the first major block of AND and OR gates in Fig. 5, this block of AND and OR gates would then be the decoder of Fig. 7. That is, the DBBL adder stage is very nearly the first block of AND and OR gates in a Post base 10 adder stage with the final block of MIN and MAX gates removed. Because this block of MIN and MAX gates is not present, the DBBL adder stage has as outputs only the binary-valued lines c^0, c^1, and s^0, s^1,....., s^9. Therefore, unlike the Post base 10 adder, the DBBL adder is entirely 2-valued binary logic.

A combinational logic realization such as the one in Fig. 7 is definitely not the only means of formulating an NBBL adder. Section VI. of this paper proposes several other approaches to constructing a general NBBL adder stage which could offer significant advantages over a combinational realization in important areas such as initial circuitry cost and operational speed.

III. STORAGE REGISTERS

Like the adder circuitry, the storage registers in a digital computer can also be designed to operate on numbers in any base. This section discusses straight binary, binary-coded base n, Post base n, and NBBL storage registers. For the sake of simplicity, all storage register units shown here will be considered unclocked.

A. Standard Binary Register

The internal circuitry of a standard binary storage register unit will purposely not be restricted to a specific type of circuitry at this point in the paper. Therefore, the standard binary storage register unit, or flip-flop, will, for now, be viewed as a "black box" (Fig. 8). The type of flip-flop shown in Fig. 8 is the well known S-R binary flip-flop, having a set input, a reset input, a normal output, x, and an inverted output, $\overline{x}$.

B. Binary-Coded Base N Register

The binary-coded base n storage register unit ($n > 2$) is simply a row of two or more standard binary flip-flops. Like the binary-coded base n adder, the binary-coded base n storage register unit has multiple input lines and multiple output lines, these multiple lines again

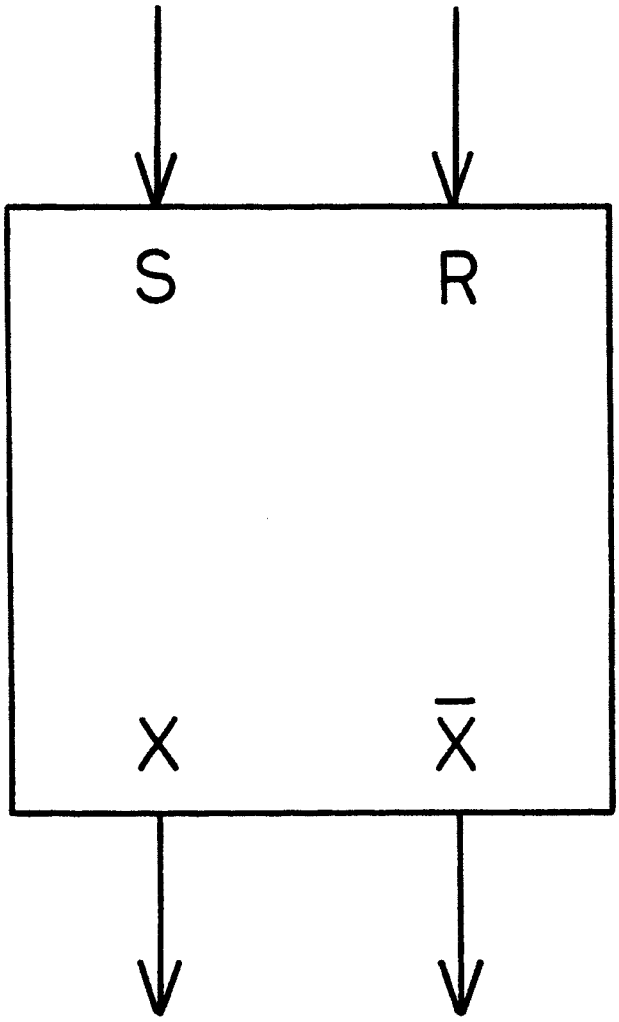


Fig. 8. S-R Binary Flip-Flop

representing a binary-coded form of the input and output data [2]. The multiple set input lines correspond to the single set input of the standard binary flip-flop, while the multiple normal output lines correspond to the single normal output, x, of the binary flip-flop. Fig. 9 illustrates one possible realization of a binary-coded base 10, or BCD, storage register unit.

Remember that in the binary-coded base n adder, when the base being used was not a power of 2, the problem of normally having to use extra correctional circuitry appeared. Similarly, in the binary-coded base n storage register unit, when the base is not a power of 2, a problem is created. This time, however, the problem is that of easily producing a complement form of the number held in the storage register unit. On the straight binary flip-flop, the inverted output, $\bar{x}$, is actually the 1's complement form of the normal output, x. Thus, to obtain the 1's complement of a multi-digit binary number stored in an entire binary storage register (merely a string of binary flip-flops), one simply uses all the inverted outputs instead of the normal outputs. Consider two binary storage registers containing two separate binary numbers. In order to subtract the second register from the first, one has only to gate the normal outputs of the first register into the adder, gate the inverted outputs of the second register into the adder,

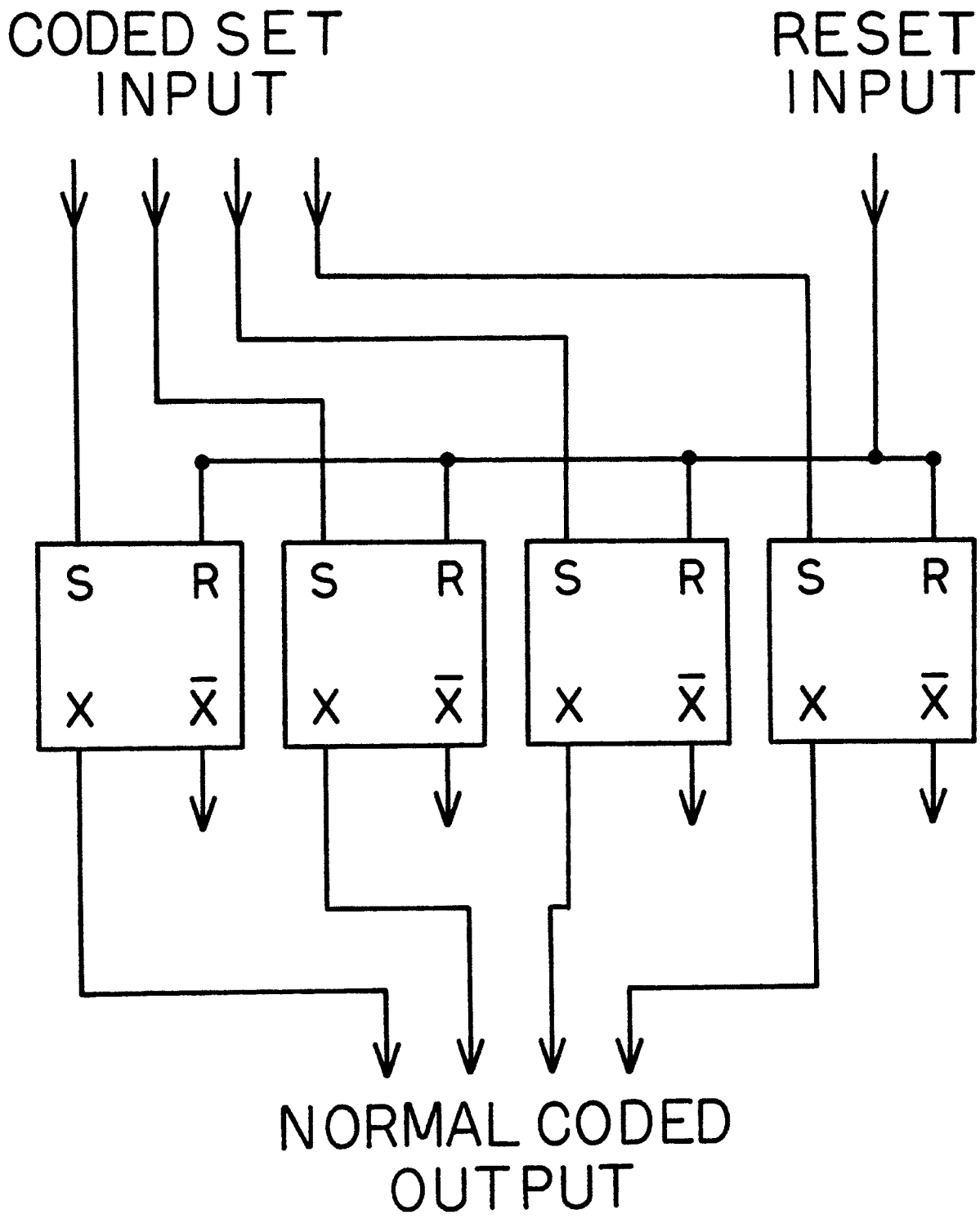


Fig. 9. BCD Storage Register Unit

and also gate in a "pre-carry." This procedure is equivalent to subtracting the second register from the first by adding the 2's complement of the second register to the first register. It should now be obvious that it is quite desirable for a general base n storage register to be capable of easily producing the n-1's complement of its normal contents. In a binary-coded base n storage register, if the base being used is a power of 2, the n-1's complement of the register contents is again formed by merely using all the inverted outputs in place of the normal outputs. On the other hand, if the base being used is not a power of 2, merely using the inverted outputs will not produce the correct n-1's complement (unless special, sophisticated codes are employed). In such cases, additional combinational circuitry must be used to produce this complement. Fig. 10 illustrates the logic operations that must be performed on the 4 normal outputs of a single BCD storage register unit in order to produce the 9's complement of the decimal digit represented by these 4 outputs [1]. (In base 10, the "excess-3" code, for example, can produce the 9's complement merely by direct complementation of the 4 normal outputs. However, note that the "excess-3" code adder still requires correction after the addition. In fact, unlike the traditional BCD adder, it requires correction in all cases, whether or not a decimal carry was produced [1].)

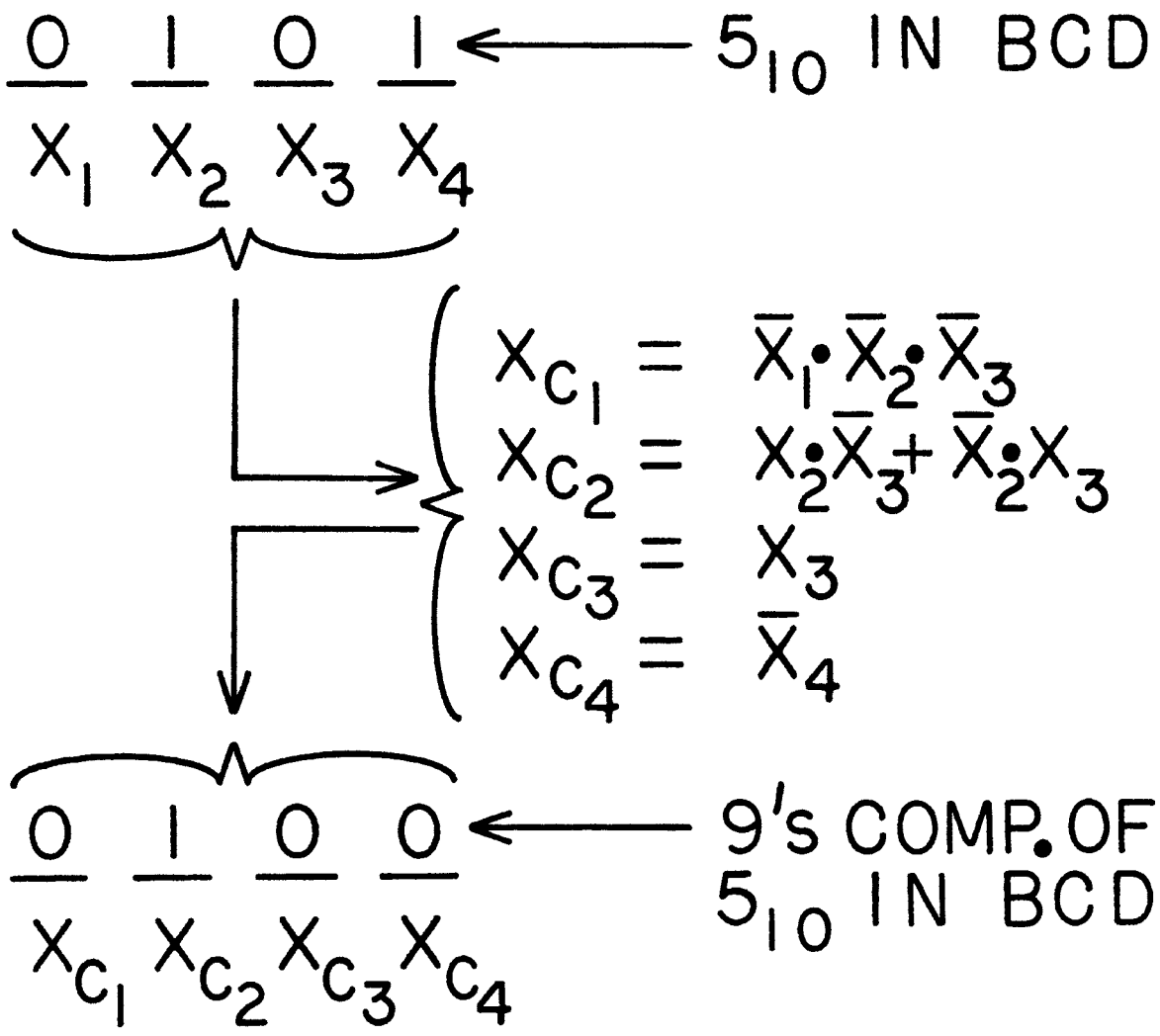


Fig. 10. 9's Complement of a BCD Digit

C. Post Base N Register

The Post base n storage register unit [2], [3], like the Post base n adder stage, is essentially a hybrid of n-valued and 2-valued logic. The reader will recall that the Post base n adder stage basically took several binary, or 2-valued, input lines and transformed them into an n-valued sum output line. On the other hand, the Post base n storage register unit takes as input one n-valued line, x, and transforms it into the 2-valued output functions x^0, x^1,....., x^{n-1}. Fig. 11 depicts a Post base 10 storage register unit.

A Post base n storage register unit merely stores the value of voltage on its input by impressing a voltage of n-1 on the appropriate output line. For example, if a voltage level of 6 volts were supplied as input to the base 10 unit of Fig. 11, the x^6 output line would assume a value of 9 volts while all other lines assume 0 volts.

D. NBBL Register

The NBBL storage register, again of the author's own design, is the last type of storage register that will be discussed. Such a storage register could be used to receive and hold data such as teletype input or core memory output, to feed input into a DBBL adder, etc. Just as the NBBL adder was essentially a modification of the Post base n adder, the NBBL storage register is

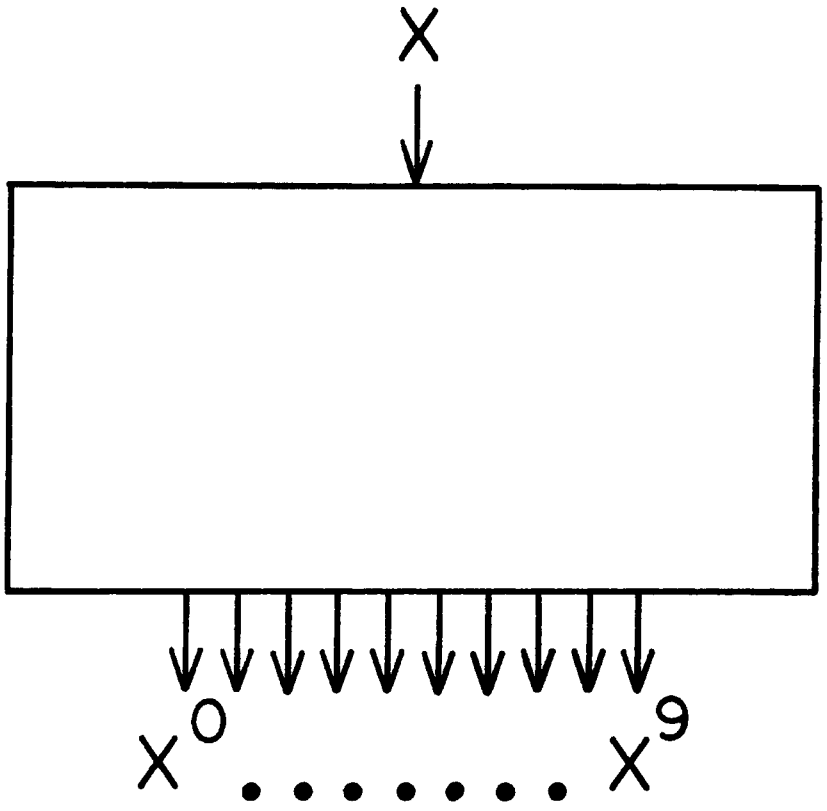


Fig. 11. Post Base 10 Storage Register Unit

actually a simplification, or special case, of the Post base n storage register. Unlike the Post base n storage register unit with its single n-valued input line and n binary output lines, the NBBL unit has n binary input lines and n binary output lines. Actually, in this NBBL unit, the n input lines are labeled identically to the n output lines. In essence, these input lines receive the functions x^0, x^1,....., x^{n-1} which the output lines then assume and hold. Remember that only one input and only one output can possess the high voltage level, or logic 1 level, at any particular instant. Thus, when one puts a logic 1 on input line x^i, output line x^i then assumes this logic 1. Fig. 12 illustrates a DBBL storage register unit.

Obviously, since the NBBL storage register unit need not transform a multi-valued line into binary lines as the Post base n unit does, the internal circuitry of the former can be much simpler. Fig. 13 shows one possible realization for a base 3 NBBL unit using three binary S-R flip-flops. Section VII. of this paper proposes many more possible realizations for an NBBL storage register unit. A sizeable portion of these realizations are not mere interconnections of ordinary binary flip-flops like the realization of Fig. 13, but instead, are original creations designed by the author solely for use in NBBL logic systems. These creations can offer advantages

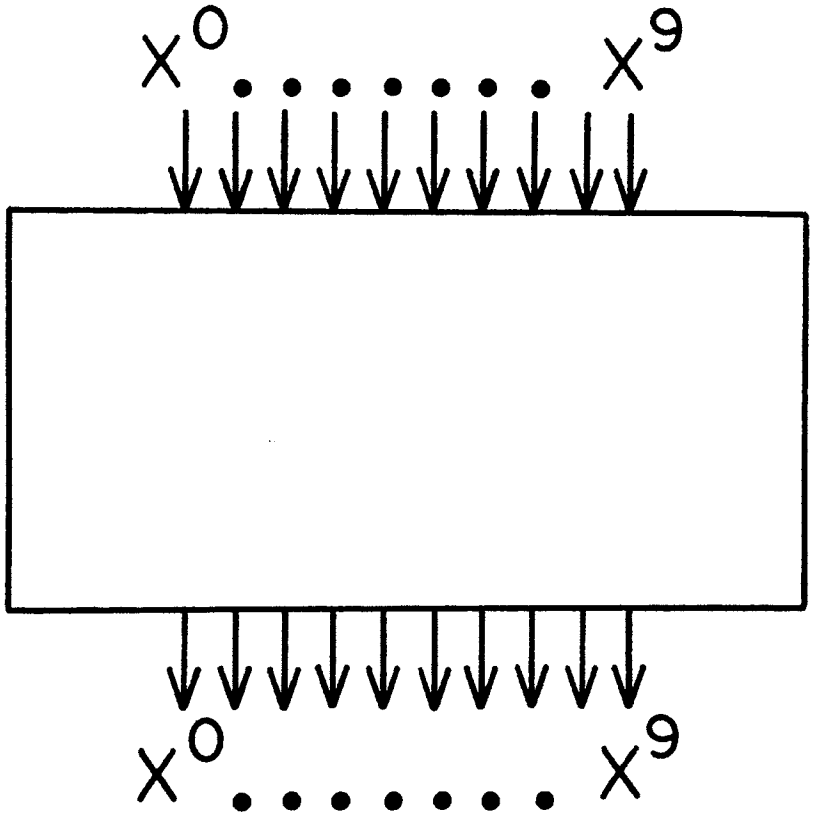


Fig. 12. DBBL Storage Register Unit

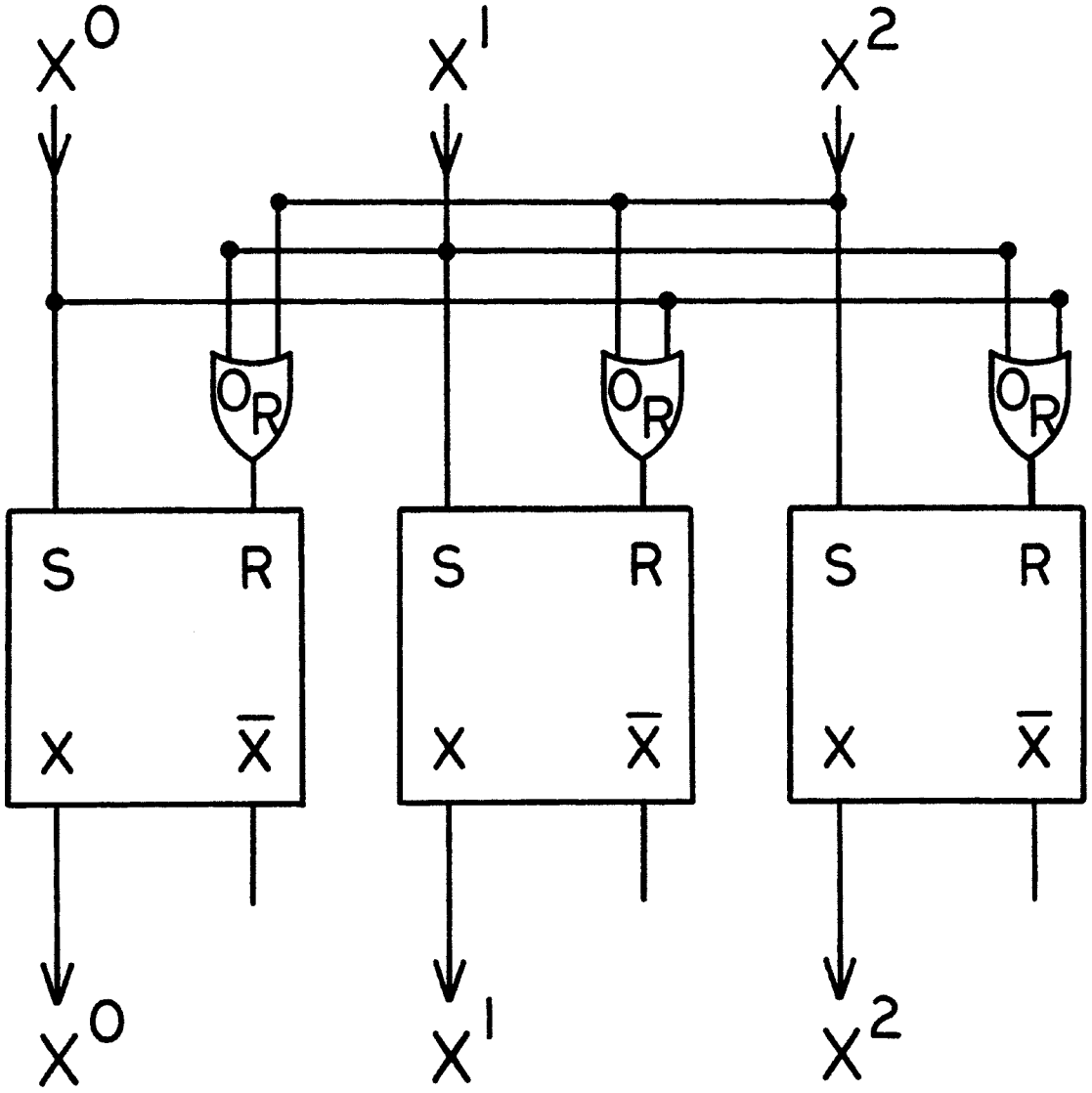


Fig. 13. Possible Base 3 NBBL Storage Register Unit

over the more ordinary realizations in areas such as power consumption and total number of circuit components.

It has been mentioned that the availability of the n-1's complement from a register of general base n is a desirable feature which permits an adder to easily perform subtraction. Fortunately, the production of the n-1's complement from the n binary output lines of the Post base n or NBBL storage register unit is not at all difficult. In fact, an n-1's complement "gate" for this purpose can be constructed from n pieces of wire, a structure so simple it would not ordinarily be thought of as a gate. Consider a "black box" having n input pins and n output pins, where each of these sets of n pins represent the functions x^0, x^1,....., x^{n-1}. An n-1's complement gate is formed simply by connecting each x^i input to the $x^{(n-1)-i}$ output, where i = 0,1,....,n-1. Fig. 14 depicts a 3's complement gate for use in base 4 NBBL logic.

Decimal machines will very probably become increasingly attractive in the future. Sections IV. and V. of this paper will demonstrate sufficient justification for such an increase in the popularity of decimal machines. Furthermore, given this future demand for decimal machines, Sections IV., V., VI., and VII. will further contrast NBBL logic with its competitors and will specifically show why DBBL logic could be quite advantageous over traditional

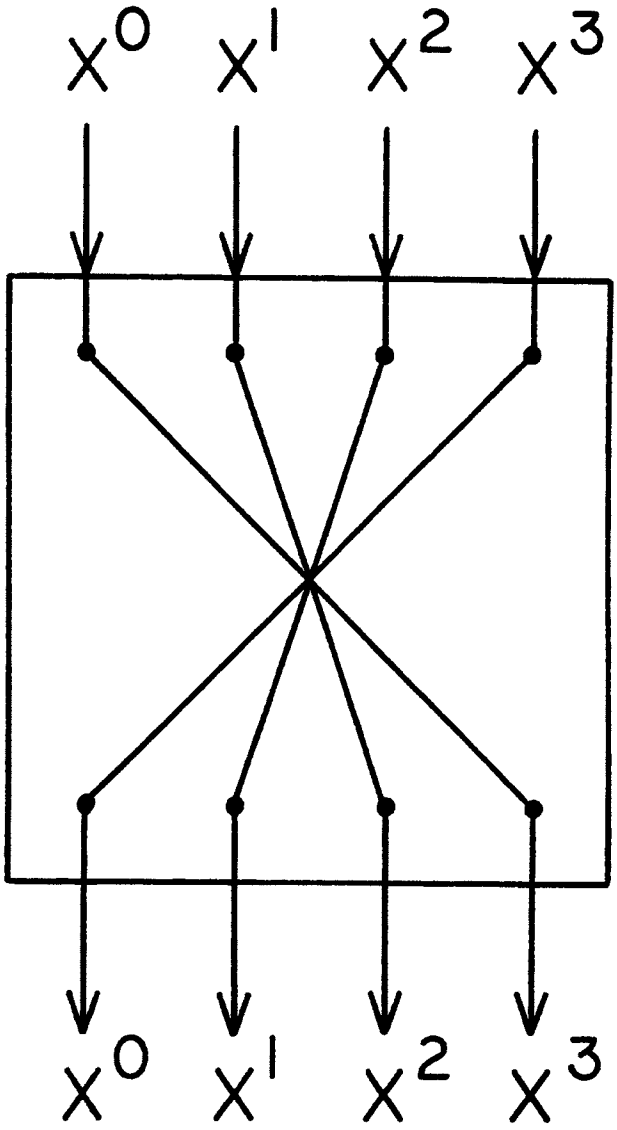


Fig. 14. 3's Complement Gate for Base 4 NBBL Logic

BCD and Post base 10 logic in the construction of these machines.

IV. COST ANALYSIS

This section presents a cost analysis of DBBL adders and storage registers relative to straight binary adders and storage registers. This analysis is performed on a twofold approach, considering both initial purchasing cost (or rental cost) and daily operational efficiency of the two types of circuitry. Note that it is really quite logical to consider pruchasing (or rental) cost and operational efficiency as both being related to the overall cost of using some particular type of machine. Obviously, machine purchasing (or rental) cost represents a deficit that the user must suffer. On the other hand, machine operational speed and efficiency are directly related to the daily assets that the user can realize. Thus, for example, a user of a digital machine might be willing to pay more for a more efficient machine initially with the expectation that its greater operational efficiency and productiveness would eventually offset his greater initial expense.

In the analysis of relative initial purchasing cost, or price, of the circuitry, two entirely different methods are employed. The first method is the traditional, but virtually outdated, technique of comparing the number of gates and the number of gate inputs needed for different realizations. The second technique, far more relevant to today's technology, assumes the circuitry in question to

be integrated onto an MSI or LSI chip so that merely counting the relative number of pins on different chips gives a fair measure of their relative cost.

A. Purchasing Cost - Traditional Approach

Wojcik and Metze have derived a detailed traditional cost analysis of Post base n adders and registers relative to their straight binary equivalents [2], [3]. Since their approach was the outdated "number of gates and gate inputs" technique, it would be futile to run through the same procedure in detail in this paper. Instead, by making a few simple approximations, one can directly transfer the results of Wojcik and Metze to the author's DBBL adders and registers. Such transferral of the traditional Post base 10 results to the author's DBBL circuitry is indeed only approximate. However, this transferral does yield a rough indication of the DBBL/binary cost ratio using traditional techniques. These traditional techniques, in view of their general irrelevancy to modern integrated circuit technology, simply do not merit extreme accuracy.

The reader will recall from Section II. that a Post base 10 adder stage (Fig. 5) could be converted to a DBBL adder stage (Fig. 7) by adding the relatively small number of extra AND and OR gates needed to produce the c^0 and s^0 outputs from the first block of AND and OR gates and by

removing the relatively small number of final MIN and MAX gates. In other words, the Post base 10 and the combinational logic DBBL adder stage contain approximately the same number of gates and gate inputs, so that from a traditional point of view, these two types of adders should differ little in price. Thus, by extrapolating the results of Wojcik and Metze out to base 10, one finds that a DBBL adder would cost about 5 times as much as a straight binary adder with enough stages to handle equivalent size numbers.

Concerning registers, Wojcik and Metze assume basically that the ratio of the number of components in a single Post base n stage to the number of components in a single standard binary stage is always about n/2. Thus, the traditional cost viewpoint implies that the relative cost of a Post base n stage compared to a binary stage is also about n/2. Certainly this assumed Post base n register stage versus binary register stage cost ratio of n/2 is only approximate and Wojcik and Metze give no real evidence of its validity. However, this n/2 factor should be just as applicable to an NBBL register stage versus binary register stage cost ratio. For example, if one examines the number of SCR's required for the SCR-type NBBL register stage realizations of Section VII., one finds that a 2BBL stage requires 2 SCR's (just as a standard binary S-R flip-flop can be made with essentially 2 cross-coupled

transistors), a 3BBL stage requires 3 SCR's, ..., a DBBL stage requires 10 SCR's, etc. Thus, at least when comparing the types of NBBL register stage and standard binary register stage realizations just mentioned, one finds that the ratio of NBBL stage SCR's to binary stage transistors is exactly n/2 for any general base n ($n \geq 2$). Therefore, this crude n/2 register stage cost factor assumed by Wojcik and Metze can be just as relevant in comparing an NBBL register stage to a binary stage as it is in comparing a Post base n stage to a binary stage. Furthermore, now consider the difference between a Post base 10 and a DBBL storage register stage. The reader will again recall that whereas the Post base 10 storage register unit (Fig. 11) had only 1 input, the DBBL storage register unit (Fig. 12) had 10 inputs. On the other hand, the Post unit required special components immediately following the input in order to convert this 10-valued input into 2-valued, binary outputs. These special components were not required in the DBBL unit. Thus, since the n/2 cost factor of Wojcik and Metze has some validity for the author's NBBL units and since the differences between a Post base 10 and a DBBL unit tend to approximately "cancel each other," one is finally led to the conclusion that DBBL units can be at least as cheap as Post base 10 units with similar output structures. That is, if the DBBL unit has a row of S-R flip-flops forming its

outputs (see Fig. 13), the Post base 10 unit should also have a row of S-R flip-flops forming its outputs. Thus, again extending the results of Wojcik and Metze to base 10, one finds that a DBBL register would cost about 1.5 times as much as a straight binary register with enough stages to hold equivalent size numbers.

It should now be obvious that, by traditional cost analysis, DBBL adders and registers would definitely have a higher initial cost than their standard binary equivalents.

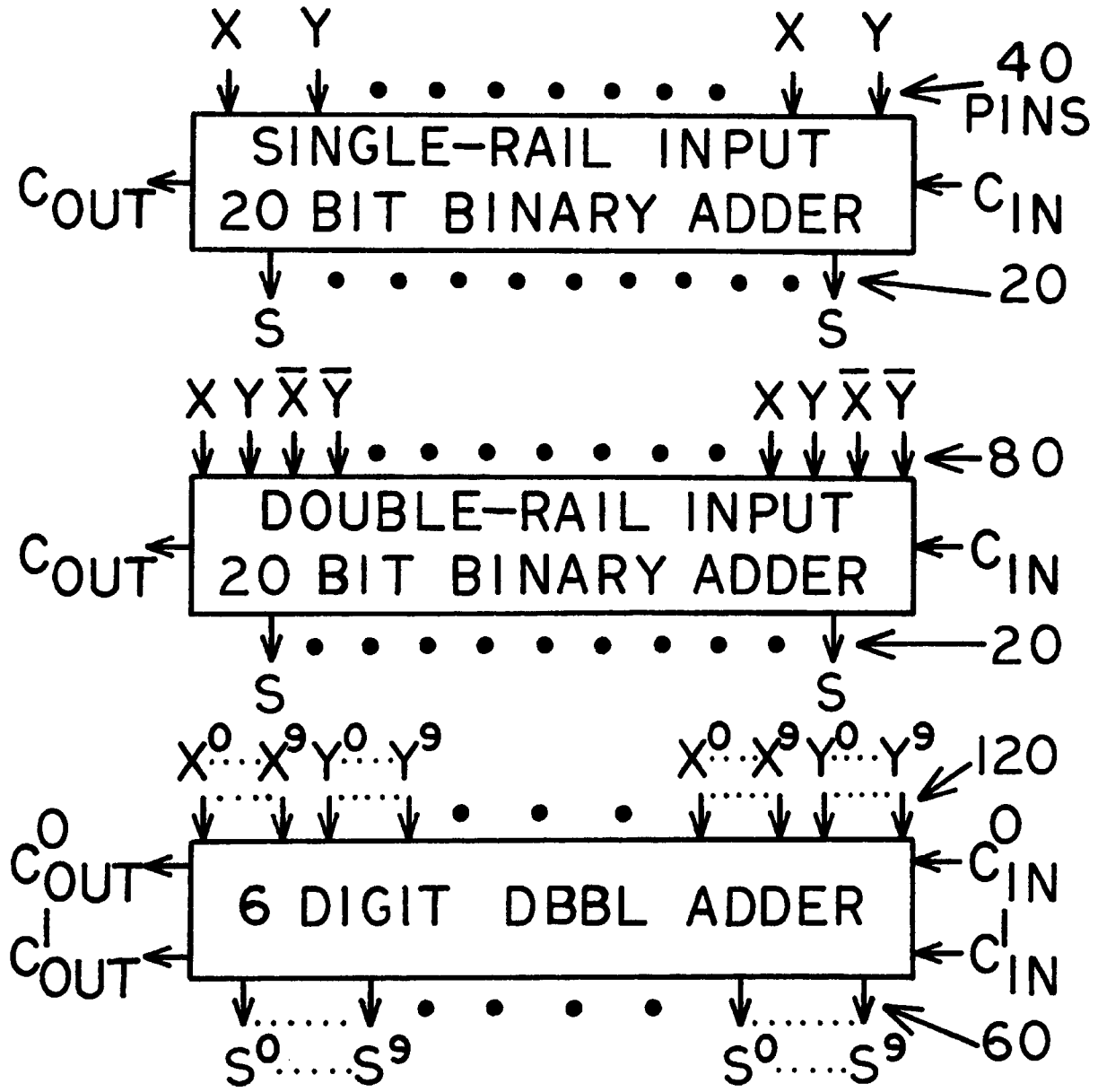
B. Purchasing Cost - Modern Approach

The modern cost analysis approach which follows not only is more applicable to present computer technology but also is easier to perform. The assumption in this approach is that the circuitry in question is all integrated onto one chip and that the chips are all produced in large volume quantities by modern automated techniques so that by far the most important factor in the cost of any single chip is the number of connecting pins which must be fastened on the chip. With this assumption in mind, one can then obtain a very reasonable measure of the relative cost of different chips merely by comparing the number of pins on the various chips [5].

First of all, in order to make a fair comparison between the cost of a base 10 and a base 2 adder or

storage register, one needs to know the number of stages required in each base to handle equivalent size numbers. That is, one needs to know the number of digits necessary to represent equivalent numbers in each base. Some thought will verify that if one represents a number as 10^d, then d, if rounded up to the next greatest integer value, is the number of decimal digits needed to represent that number in base 10. In fact, d, even without rounding, is a very good indication of how many decimal digits are required to represent a given number. Similarly, if one represents a number as 2^b, then b is a very good indication of how many binary digits are required to represent that number in base 2. Therefore, the equation $10^d = 2^b$ essentially means that some particular decimal number having d digits is exactly equal to a binary number having b digits. Solving this equation for d/b yields the approximate ratio of decimal to binary digits necessary to represent equivalent numbers in base 10 and base 2 respectively. Taking $\log_{10}$ of both sides gives $d/b = 1/3.33$. In other words, if a base 10 and a base 2 adder or register are intended to handle equivalent size data, then the base 2 device must have about 3.33 stages for every one stage of the base 10 device.

Thus, for example, a 20 bit binary adder could handle the same data as a 6 digit DBBL adder. Fig. 15 illustrates three chips, the first containing a 20 bit binary adder



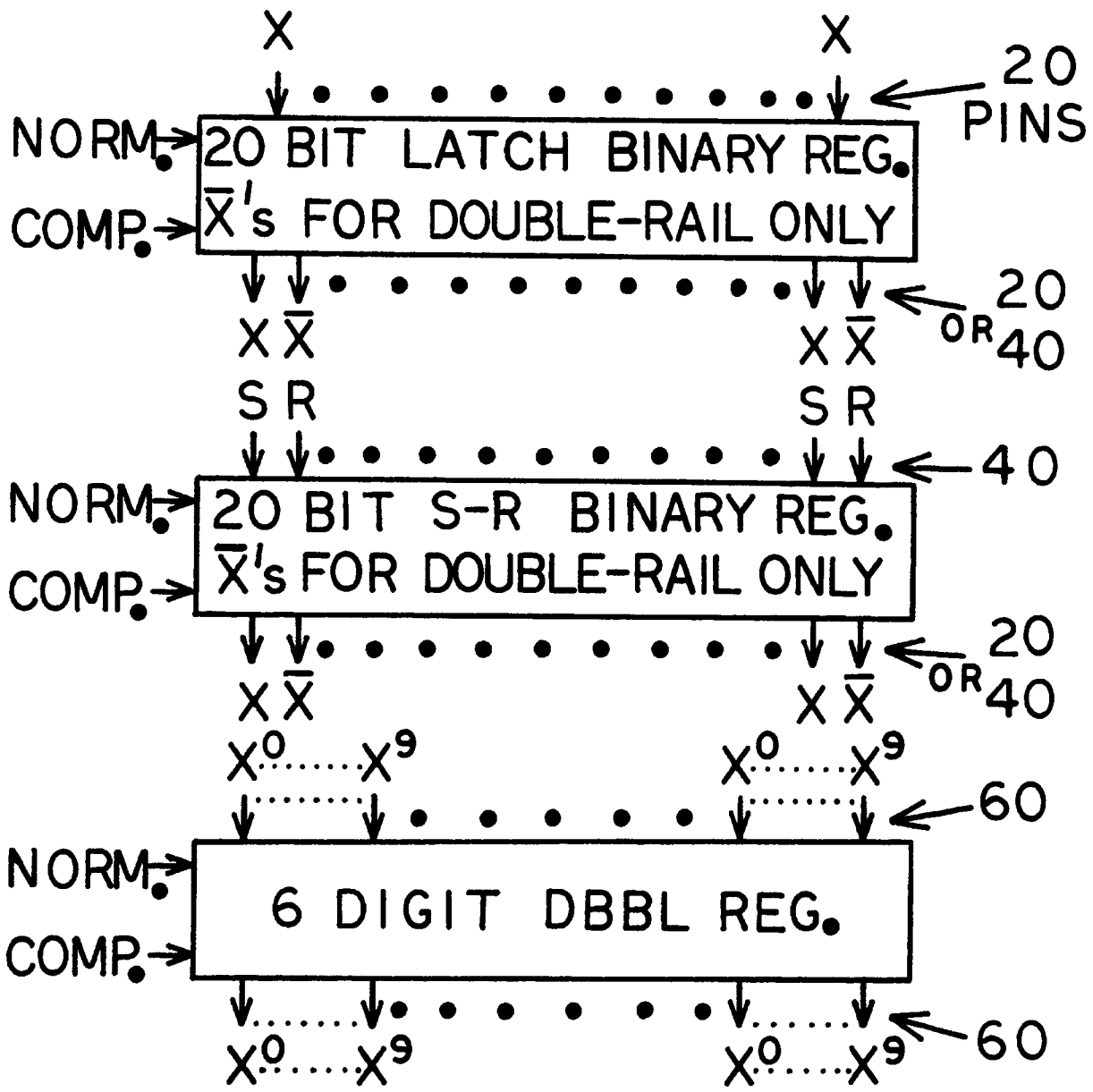
$$\frac{\text{DBBL COST}}{\text{S.RL. BINARY COST}} = \frac{184}{62} \approx 3$$

$$\frac{\text{DBBL COST}}{\text{D.RL. BINARY COST}} = \frac{184}{102} \approx 1.8$$

Fig. 15. Cost of DBBL Adder/Cost of Binary Equivalents

with "single-rail" inputs (only normal, or "true," inputs), the second containing a 20 bit binary adder with "double-rail" inputs (both normal and complemented inputs), and the third containing a 6 digit DBBL adder. It must be pointed out that a binary adder with "double-rail" inputs, even though it obviously costs more than its "single-rail" binary counterpart when using a pin-oriented cost approach, is quite desirable in cases in which one wants the actual adder circuitry on the chip itself to be a fast, basically two level type of realization such as AND-OR or NAND-NAND. In other words, since one might want to use either the "double-rail" or the "single-rail" binary adder, both types are included in the DBBL adder versus binary adder cost comparison. Note that the ratio of DBBL adder pins to binary adder pins runs from approximately 1.8 to 3.0, indicating that, in general a DBBL adder would cost about 1.8 to 3.0 times as much as its straight binary equivalent.

Fig. 16 compares a 6 digit DBBL register to both a 20 bit "D flip-flop" [6], or latch, type binary register and a 20 bit S-R flip-flop type binary register. Fig. 16 again allows both types of binary registers to have either "single-rail" or "double-rail" outputs. Note that in all the registers shown, either the normal or the complement outputs can be selected by means of the two selection lines. Of course, one binary-valued line entering each chip could, with a little extra logic on the chip itself,



$$\frac{\text{DBBL COST}}{\text{LATCH BINARY COST}} = \frac{122}{42 \text{ OR } 62} \simeq 3 \text{ OR } 2$$

$$\frac{\text{DBBL COST}}{\text{S-R BINARY COST}} = \frac{122}{62 \text{ OR } 82} \simeq 2 \text{ OR } 1.5$$

Fig. 16. Cost of DBBL Register/Cost of Binary Equivalents

serve the same purpose as these two selection lines. However, the difference between one or two output selection pins is insignificant when compared to the total number of pins on each chip. Also note that no clock lines have been shown in any of these registers even though some units, such as the latch type binary register, are always clocked. Since the number of clock inputs would, in all cases, again be insignificant compared to the number of other pins, ignoring these clock lines changes the accuracy of the calculations very little. Thus, a DBBL register would, in general, cost from 1.5 to 3.0 times as much as its binary equivalents.

If the DBBL adder and register chips shown thus far were enlarged to include maximal encoding on the inputs and outputs, the number of pins on a DBBL chip could be greatly reduced. For example, in a DBBL adder, 4 binary lines would suffice to encode each normal group of 10 DBBL inputs and outputs and 1 line would suffice to encode each of the normal pairs of DBBL carry inputs and outputs. Obviously, such encoding would significantly reduce the cost of these DBBL chips relative to their binary equivalents. However, such encoding will not be considered in this cost analysis because it would unfortunately induce additional propogational delays in the DBBL units, cutting down substantially on their speed. The inherent possibility of high speed operation in DBBL

logic will soon be shown to be one of its best "selling points." Thus, seriously hampering this speed by the introduction of encoding merely to cut down on initial cost is not practical.

It has now been shown that the traditional and the modern methods of initial cost analysis both indicate that DBBL adders and registers would be more costly than equivalent standard binary units. In fact, given the present state of the art in computer circuitry, by no reasonable stretch of the imagination or plausible series of assumptions could one ever propose that the DBBL system would be cheaper to construct. Thus, since the DBBL system would be more costly to build than would an equivalent binary system, it is only natural to assume that the DBBL machine user would suffer a greater purchasing cost if he actually buys his machine or that he would suffer larger periodic rental payments if he rents his machine.

C. Daily Operational Efficiency

Unlike purchasing, or construction, cost, when one considers the daily operational efficiency and speed of the DBBL system versus that of its binary equivalent, the picture is much brighter. First of all, consider the simple fact that, in the DBBL machine, conversion and reconversion of all input and output data from decimal

to binary and back again is not necessary. Not only does this mean that the binary system must spend time doing this conversion and reconversion, but it must also possess the hardware or software to do these operations. Obviously, this extra time and hardward or software mean money spent by the binary machine user that the DBBL machine user need not spend. For instance, in the IBM System/360 Model 50, the two machine instructions "Pack" and "CVB" (Convert To Binary) are necessary to convert the initial coded numerical data read in from the card reader into binary data. Similarly, the two instructions "CVD" (Convert To Decimal) and "Unpack" are required to convert binary machine data back into the coded data to be printed by the printer. The "CVD" instruction, as a specific example, can take anywhere from 13.00 to about 44.75 microseconds of machine time depending on the size of the data being reconverted [7]. It is clear that in a commercial operation having relatively large amounts of input and output data, such as a bank's use of a computer, the amount of time spent doing conversion and reconversion could represent a substantial portion of the machine's daily running time [1].

Now consider the potential speed of a DBBL adder compared to the speed of its standard binary equivalent. The reader should by now have grasped the idea that DBBL logic has a highly parallel form of structure. This

inherent parallelism is a definite advantage over the traditional BCD adder system with its basically serial approach of "first add -- then correct." Consider, for the sake of argument, that one wants to create AND and OR gate realizations of both a DBBL adder stage and a binary adder stage. These stages will be assumed to not have carry inputs or carry outputs, but only sum outputs, in order to keep the comparison simple. Furthermore, since the binary adder stage will require its inputs in both normal and inverted form ("double-rail" form), the inverted form of the inputs will be assumed to come either from storage register units feeding the adder stage or from inverters preceeding the adder stage so that no inverter gates need be included in the realization. Thus, the equation for the sum output line of the binary adder stage is $s = x\bar{y} + \bar{x}y$. This equation implies a two-level logic network, 2 parallel AND gates feeding into 1 common OR gate. Each sum output line of the DBBL adder stage can be formed from exactly the same type of two-level realization. For instance, the equation for the s^0 output is $s^0 = x^0y^0 + x^1y^9 + x^2y^8 + ... + x^8y^2 + x^9y^1$, again implying a two level AND and OR realization. Thus, an entire DBBL adder stage is nothing more than 10 of these two-level realizations in parallel. In other words, as long as a DBBL and a binary adder stage are realized via similar structures, the propogational speed of both stages should

be the same. Obviously, if one insisted on building his entire adder by merely connecting individual stages together to form a row of connected, but discrete, stages, the determining speed factor would be the time required for the carry to serially propogate down through all stages. Therefore, given such an adder layout, since a DBBL adder requires only 1/3.33 times as many stages as its binary equivalent, it should be capable of adding a number at least 3.33 times as fast as the equivalent binary adder.

However, such a binary adder layout employing a mere connected row of discrete single adder stages is a virtually outdated type of layout. For example, it is quite common to see 4 individual binary adder stages grouped together as one block having a fast "look-ahead" carry available from the fourth individual stage on the block of 4 stages. Such blocks using carry "look-ahead" help to increase the speed of the entire adder. Obviously, to further increase the speed of a binary adder, one could in fact form the binary adder by connecting blocks together, each of which contains a purely two-level realization of a 3 or 4 bit binary adder. These separate 3 or 4 bit binary adders thus could each complete their 3 or 4 bit addition in the same time required by a single DBBL adder stage to complete its addition. Therefore, since 3.33 binary digits are about equivalent in numerical size to 1 decimal digit, a binary adder constructed according to this scheme could add numbers approximately as fast as its DBBL equivalent

providing the DBBL adder is a mere connection of discrete, single-digit DBBL stages.

Finally, in situations requiring the ultimate in speed, one could also extend the idea of a two-level realization to an entire adder. That is, one can form two-level combinational logic realizations for both an entire 20 bit binary adder and its 6 digit DBBL equivalent. Of course, such two-level realizations of entire multi-stage adders require huge numbers of logic gates and gate inputs, far more than are required in the slower, more serial realizations previously mentioned. However, the modern integrated circuit cost approach already presented indicates that the complexity, or size, of the internal circuitry of a chip containing such a two-level adder has very little effect on the cost of the adder. Therefore, it is reasonable to assume that, in the interest of overall adder speed, one might want to construct such two-level realizations. Given then, that both the 20 bit binary adder and its 6 digit DBBL equivalent are both entirely two-level realizations, both adders would be capable of adding their equivalent size numbers with identical speeds after these numbers <u>reach</u> the adders. The only difference is that the binary adder demanded previous "uncoding" of each of its binary inputs into "double-rail" inputs including both the normal and the inverted forms of each input variable. Whether this uncoding was done

by passing the inputs through a binary register or whether it was done by separate inverter gates, this necessary uncoding potentially implies, in general, one extra level of propogational delay which is not present in the DBBL system. The alert reader will note that this particular advantage of DBBL results directly from the fact that the DBBL system, as proposed in this paper, is a highly "uncoded" form of logic. That is, the DBBL system does not transmit its signals from circuit to circuit via maximally encoded lines as today's binary system usually does. At best, one could refer to a group of 10 DBBL data transmission lines as using a "1-out-of-10" code to indicate each possible value of the decimal digit represented by such a group of lines. (Of course, one could create a binary machine which has all variables that are transmitted throughout the machine available in both true and complemented form. However, such a machine would be, in reality, a base 2 NBBL, or 2BBL, machine.)

Thus, at one extreme, if one organizes equivalent size DBBL and standard binary adders using single-digit adder stages as the basic building block (allowing the carry to "trickle down" serially through each stage), the resultant DBBL adder would be much faster than its binary equivalent. At the other extreme, if one uses a purely two-level realization of both adders in their entirety (such as AND-OR or NAND-NAND), then both adders would

themselves be equal in speed. However, even at this purely two-level extreme, the DBBL system due to its highly "uncoded" form of data transmission, still holds the general advantage over the standard binary system of never needing additional uncoding delays preceeding the adder itself. Therefore, the DBBL system exhibits the definite potential of faster overall arithmetic operation.

Whether or not the overall DBBL system offers any advantage over its overall binary equivalent in average power consumption is questionable. Consider the combinational logic realizations of each adder discussed thus far. The traditional initial cost analysis demonstrated that the DBBL adder would cost more to purchase because it contained many more gates and gate inputs than its binary equivalent. It is therefore reasonable to deduce that this DBBL adder realization consumes more average power than its binary counterpart. On the other hand, now consider a 6 stage DBBL storage register and its 20 stage binary equivalent. If the DBBL register stages are the SCR types shown in Section VII.,then only one SCR is on at any given time in each register stage. Similarly, if the flip-flops in the binary register are the familiar type consisting of two cross-coupled transistors, then one transistor is also on in each binary stage. Assuming then that the SCR's and the transistors have

identical voltage supplies and load resistors, the average power consumption of the DBBL register would be less than that of its binary counterpart. Note that this would not be true if the DBBL stages were built from a row of binary flip-flops like those in the binary register (see Fig. 13). Therefore, although the DBBL and the binary system each have some advantage in the area of power consumption, it is not fair to say that either system as a whole is definitely superior.

By far the most significant advantage of the DBBL system over the equivalent size binary system lies in the area of daily operational efficiency and speed. The DBBL system need not waste time performing conversion and reconversion, and thus also need not have any hardware or software to perform this conversion and reconversion. Furthermore, when compared to a standard binary system, the DBBL system has the definite potential of faster overall arithmetic operation within the machine. Such considerations indicate that the DBBL system is generally capable of doing more work per unit of time.

V. NBBL SYSTEMS

This section discusses some of the advantages and disadvantages of NBBL systems compared to straight binary, binary-coded base n, and Post base n systems. The major asset of a base 10 NBBL, or DBBL, system, namely greater daily operational efficiency than that of a standard binary system, has already been discussed in detail in Section IV. Therefore, this particular advantage will not be mentioned again here. Instead, this section concentrates on basic characteristics of NBBL systems which might easily be overlooked.

A. General Advantages

One important advantage of an NBBL system that must be stressed is its good, binary noise immunity. In a multi-valued Post base n system, voltage levels would, in general, be much more critical than in purely binary systems such as the straight binary, the binary-coded base n, and the NBBL systems.

It cannot be said that the NBBL system offers any advantage in the area of computer architecture, or "lay-out." However, it can be said that NBBL should not force architecture to be more complex than it is in today's standard binary systems. Actually, one can picture an NBBL machine as merely having n lines routed wherever the binary machine has one. Thus, the architecture of an

NBBL machine is not more complex, but simply on a larger scale than that of the equivalent binary machine.

Similarly, the NBBL system creates no new problems in the area of fault diagnosis. For example, the very commonly assumed "stuck-at-1" and "stuck-at-0" faults which are used in ordinary binary systems are also applicable to the NBBL system. Obviously, a more complete list of possible faults would be necessary in a multi-valued Post base n system.

Another important advantage of an NBBL system is the possibility of simpler peripheral devices. In a straight binary or binary-coded device, all characters, <u>including the numerical digits 0, 1,...., 9</u>, which enter the computer from a device such as a card reader or teletype must be put in some form of binary code. Obviously, this means that these peripheral devices must contain coding networks to perform this coding. Even in a Post base 10 system, the numerical digits would have to be transformed into 10 accurate discrete voltage levels. However, in the DBBL system, the "coding" of the numerical digits would consist only of sending each of the digits 0, 1,....., 9 into the computer on the corresponding correct line of the group of 10 lines x^0, x^1,....., x^9. Of course, this lack of actual numerical coding in DBBL peripheral devices should also make the DBBL system capable of getting numerical data into and out of the machine faster than is possible

with the other systems.

Finally, it must be mentioned that since NBBL logic is entirely binary in nature, it is, in general, just as capable of successful asynchronous operation as are binary systems. Thus, unlike a Post base n system [2], [3], an NBBL system could take advantage of the speed increase that asynchronous operation offers over synchronous operation.

B. General Disadvantages

Naturally, just as NBBL has several inherent advantages, it also has some basic inherent disadvantages. The most obvious of these general NBBL disadvantages is the increased inter-module wiring and the increased number of individual circuit components required by most bases in comparison to an equivalent binary system. Using the equivalent 20 bit binary and 6 digit DBBL adders of Section IV. as an example, these wiring and component increases are evidenced respectively by the greater modern and traditional purchasing costs of the DBBL adder. Among other things, more inter-module wiring and more components mean, for the general NBBL system, increased physical size, and greater probability of at least one component fault occurring during any given time period.

Another disadvantage of a general NBBL system would be increased initial cost of core storage compared to

equivalent core storage in a standard binary system. The reader will recall from Section IV. that a 6 digit DBBL storage register was equivalent to a 20 bit straight binary storage register. Similarly, a 6 digit word of DBBL core storage would be the equivalent of a 20 bit word of straight binary core storage. However, note that 6 digits of DBBL core storage actually require 60 individual magnetic cores while 20 bits of straight binary core storage require only 20 magnetic cores. Thus, this comparison yields, for core memories, a DBBL to binary initial cost ratio of about 3 to 1. Thus, one can conclude that, in general, "NBBL" implies costlier core memories than does "binary".

Finally, one last point must be mentioned concerning the use of adders to realize logic functions such as the "Exclusive Or" function. In a standard binary machine, the bit-by-bit Exclusive Or of two binary numbers is easily performed by merely adding the two numbers while inhibiting the carries in the adder. Even though one might never have any need for such an Exclusive Or in a base n machine ($n > 2$), if one attempted to form the Exclusive Or of two numbers (each of which were all 1's or 0's) in an NBBL machine using the above technique, the result would not always be correct. For example, with the carries inhibited in a DBBL adder, the sum of two 1's would yield "2," not "0" as desired in an Exclusive Or

operation. This does not mean that comparably easy techniques for realizing such binary logic functions in NBBL machines cannot be found. It simply means that the normal techniques now in use in standard binary machines would not work.

VI. NBBL ADDER REALIZATIONS

This section proposes some NBBL adder realizations other than the strictly combinational one of Section II. It is possible that thorough research with the realizations shown here might yield advantages over such combinational realizations in areas such as simplicity, initial cost, and speed. For the sake of simplicity only, the adder stage circuits shown in this section do not contain input and output carries. That is, each adder stage shown here produces a sum digit from only the addend and augend digits without allowing an input carry digit and without producing a carry output digit. Eliminating these carry inputs and outputs does not detract from the real world practicality or applicability of these circuits. It does however allow the presentation and explanation of these circuits to be of reasonable length.

A. SCR Steering Array Realization

The first circuit shown in Fig. 17 is an SCR steering array base 3 NBBL adder stage. Note that this circuit, like all others shown in this section and the next section, can easily be extended to a larger base, such as base 10. A simple example can best illustrate how this adder stage functions. Consider feeding two 1's into this circuit to be added. Obviously, the answer should be 2 in base 3. One of these 1's would enter the circuit on input line

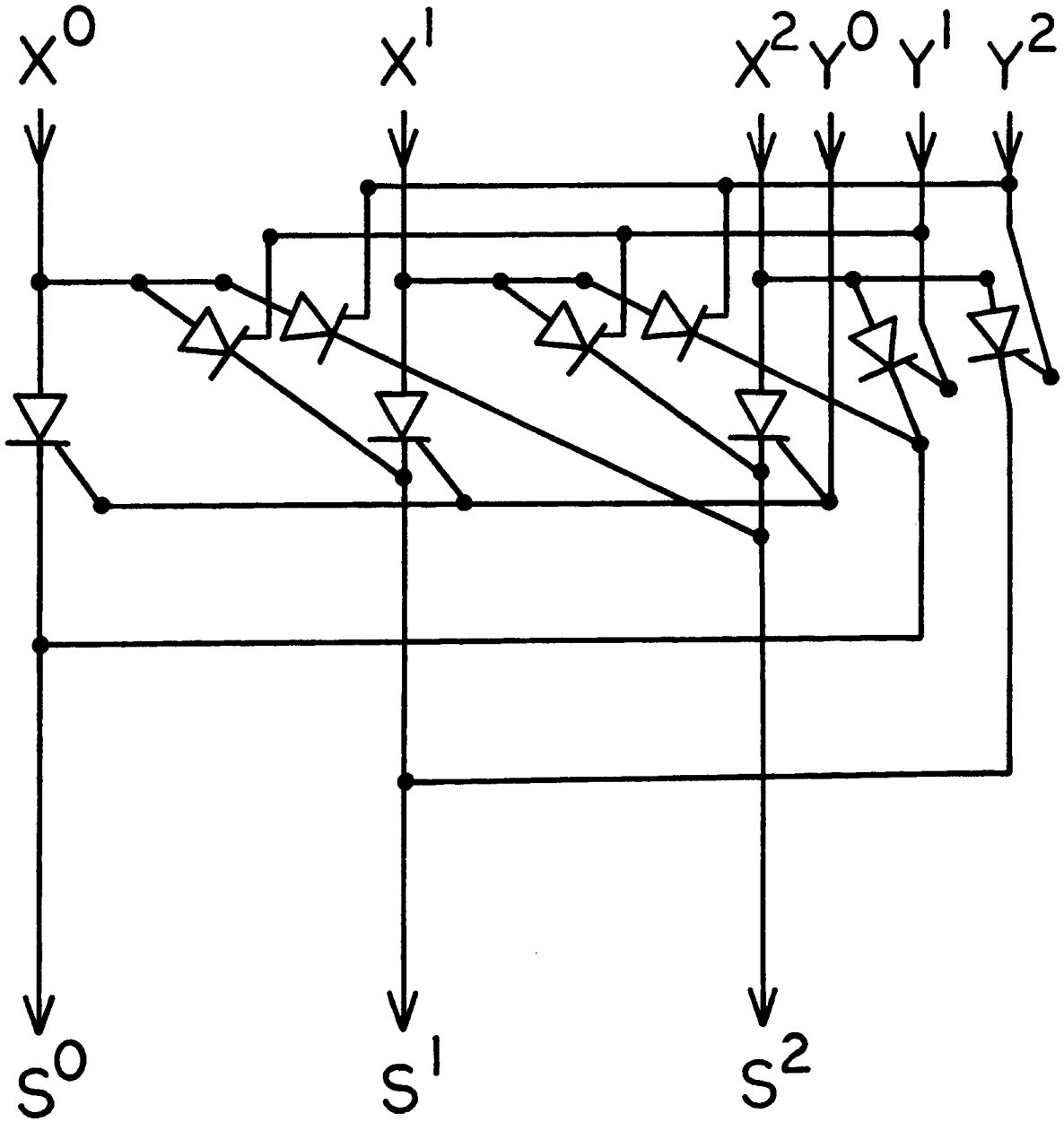


Fig. 17. SCR Array 3BBL Adder Stage

x^1 while the other "1" would enter on line y^1. The function of the latter "1" is to shift the logical one voltage on line x^1 over to the s^2 output line. Thus, only the s^2 output is at the logic one voltage level, indicating a sum output of 2 as desired. The reader might wonder why no load resistors are shown leading from each of the sum output lines to ground. Such resistors are not shown because they would not be necessary in all cases. For example, if this adder stage were feeding one of the SCR-type storage register stages shown in Section VII., the load for each sum output is already contained in the storage register stage. On the other hand, inputs, y^0, y^1, and y^2 would definitely need resistors not shown in order to keep the SCR gate currents in the adder at the proper level.

B. ROM Realizations

The next two types of adder realizations are termed Read Only Memory (ROM) realizations because their structures are very similar to common types of ROM's. These realizations do not actually contain standard ROM's because, for instance, they do not contain the addressing circuitry included at the input of standard ROM packages. However, these realizations do contain diode arrays and rows of transformer cores exactly like those found in various ROM's.

Fig. 18 illustrates a diode ROM [8] realization for a base 3 NBBL adder stage. This realization is really nothing more than a group of diode-resistor AND and OR gates set up so that each sum output line represents a sum-of-products equation for the proper sum digit. For example, the top three horizontal lines and the vertical s^0 line actually represent the equation $s^0 = x^0y^0 + x^1y^2 + x^2y^1$. Note that, just as was the case with the SCR array of Fig. 17, the resistors shown in Fig. 18 between each of the sum output lines and ground may not be needed if the adder stage is feeding an SCR-type register stage such as those shown in Section VII. Obviously, without these resistors, when a particular sum output line is "off," it would essentially be providing the circuitry it feeds an open circuit condition instead of a ground. With most types of logic, such an open circuit would not produce the same results as a ground. However, with many SCR's, an open circuit on a gate lead will function identically to a ground on a gate lead.

Finally, Fig. 19 depicts a braid transformer ROM [8] realization of a base 3 NBBL adder stage. Note that each transformer core contains a list of the input lines that pass through it. Any input lines not listed in a particular core pass around that core. Further note that each particular core is bypassed by a different

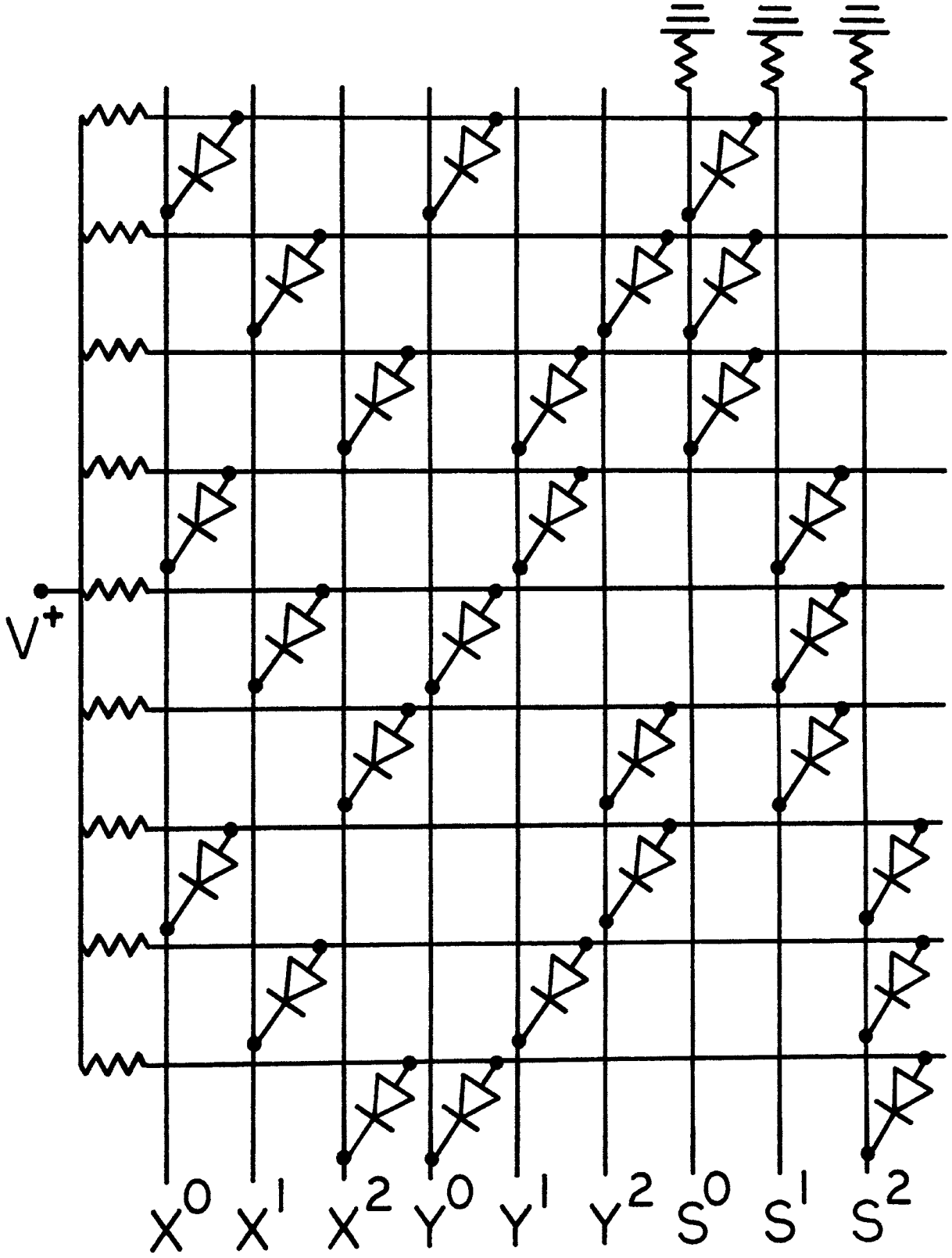


Fig. 18. Diode ROM 3BBL Adder Stage

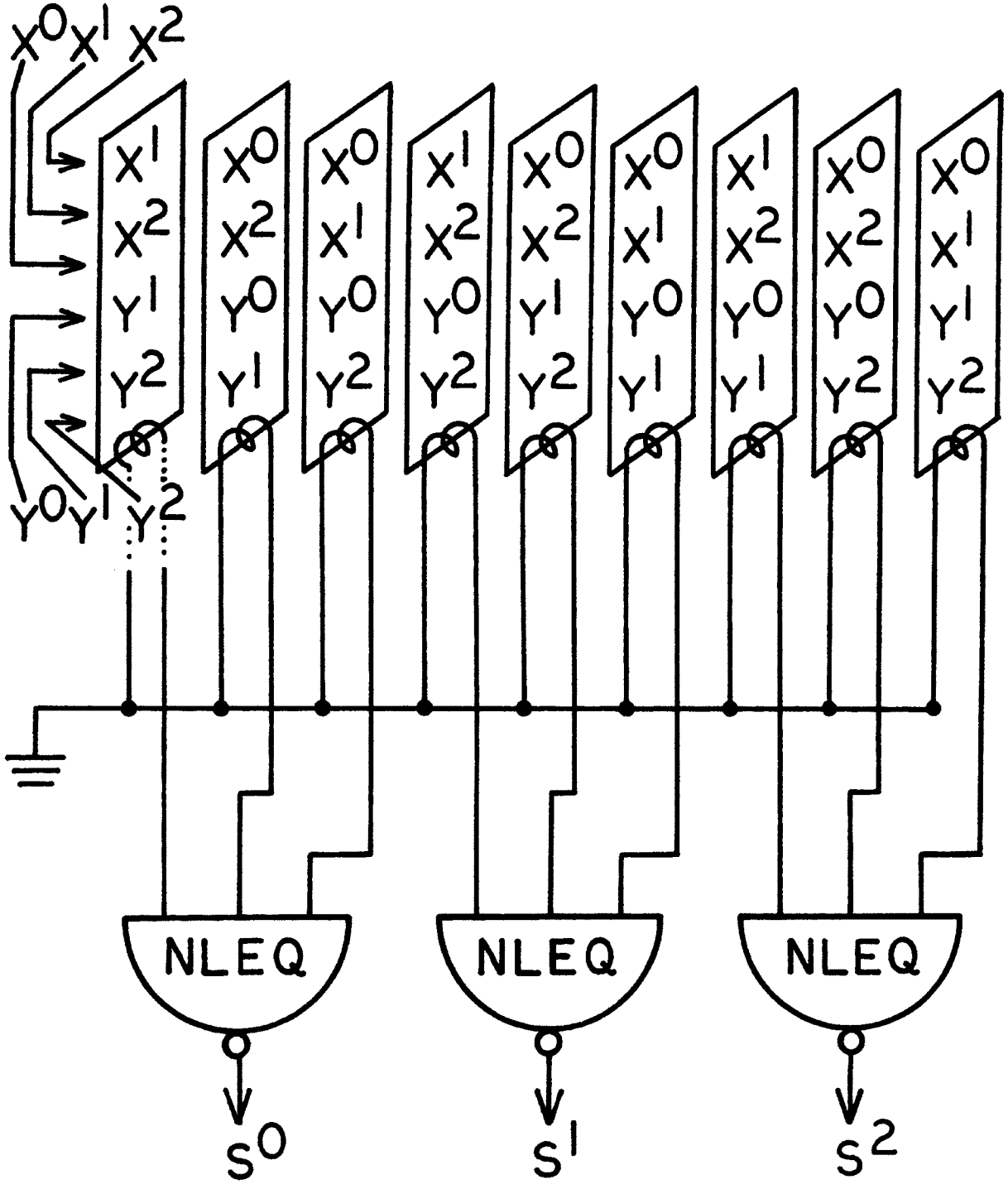


Fig. 19. Braid Transformer ROM 3BBL Adder Stage

pair of input lines. In fact, each core is bypassed by a different one of the nine unique input combinations. For example, the first, second, and third cores on the left are bypassed by x^0y^0, x^1y^2, and x^2y^1 respectively, where each of these combinations should produce a "0" sum digit. Thus, when any of these three input combinations are gated into the adder, the left NLEQ (Not Logical Equivalence) gate will always have one of its three inputs at ground level while the other two NLEQ gates will have all high inputs. The result is that the left NLEQ gate will produce a logical 1 output while the others produce logical 0 outputs, thus forming the correct s^0, s^1, and s^2 outputs. Note that since all transformer outputs will be zero when the adder stage is not being used, all NLEQ outputs will then be zero. Thus, if these adder stage outputs were connected directly to an SCR register stage like those in Section VII., this special case of all register stage input lines being zero would simply not affect the register stage outputs.

The NLEQ gates, if driven directly by the transformer output windings as shown in Fig. 19, must be carefully chosen. For example, when the current passing through a transformer core is returning to zero, that transformer would actually produce a negative output voltage. Also, for any given adder input combination, some of the cores will have one current carrying wire passing through them

while, at the same time, other cores will have two such wires. Thus, the NLEQ gate inputs will have to withstand two different positive voltage levels, ground level, and two negative voltage levels. More specifically, these gates will have to be able to respond to both positive voltages as a logical 1 and respond to ground and the two negative voltages as a logical 0. Furthermore, consider what happens to all but one of the NLEQ gates every time data is fed into the adder stage. All gates whose correct sum output is "0" now have their inputs attempting to all rise simultaneously from logical 0's to logical 1's. If one of these inputs were to reach the logical 1 level before the others, transient undesired 1's would appear on sum output lines which should remain fixed at "0." Interface circuitry between the transformer outputs and the NLEQ gates could remove these critical characteristics. However, such circuitry would also add to the cost of the adder stage.

Despite its many cirtical areas and potential problems, a realization such as that of Fig. 19 merits real consideration. Rows of braid transformers are often capable of driving logic gates directly, are extremely cheap, and are capable of high speeds [8].

VII. NBBL STORAGE REGISTER REALIZATIONS

This section proposes some NBBL storage register realizations other than the one of Section III. It is quite possible that the SCR designs shown here can yield advantages over the more conventional designs consisting merely of a row of n standard binary flip-flops and some combinational logic, such as the one shown in Section III. These possible advantages lie mainly in the areas of lower power consumption and fewer internal components, where fewer components can offer smaller physical size, less frequent faults, and depending on the method of manufacture, maybe lower cost.

A. Latch Realization

The first circuit shown in Fig. 20 is another of the more conventional designs. It is a base 4 NBBL storage register stage made from "D flip-flops" [6], or latches. Unlike all other register stages shown in this entire paper, the unit of Fig. 20 is shown clocked. The only reason this unit is shown clocked is that standard binary latches are strictly clocked items. Again note that this circuit, like all others shown in this section, can easily be extended to a larger base, such as base 10.

B. SCR Realizations

Fig. 21 illustrates the first of the SCR model

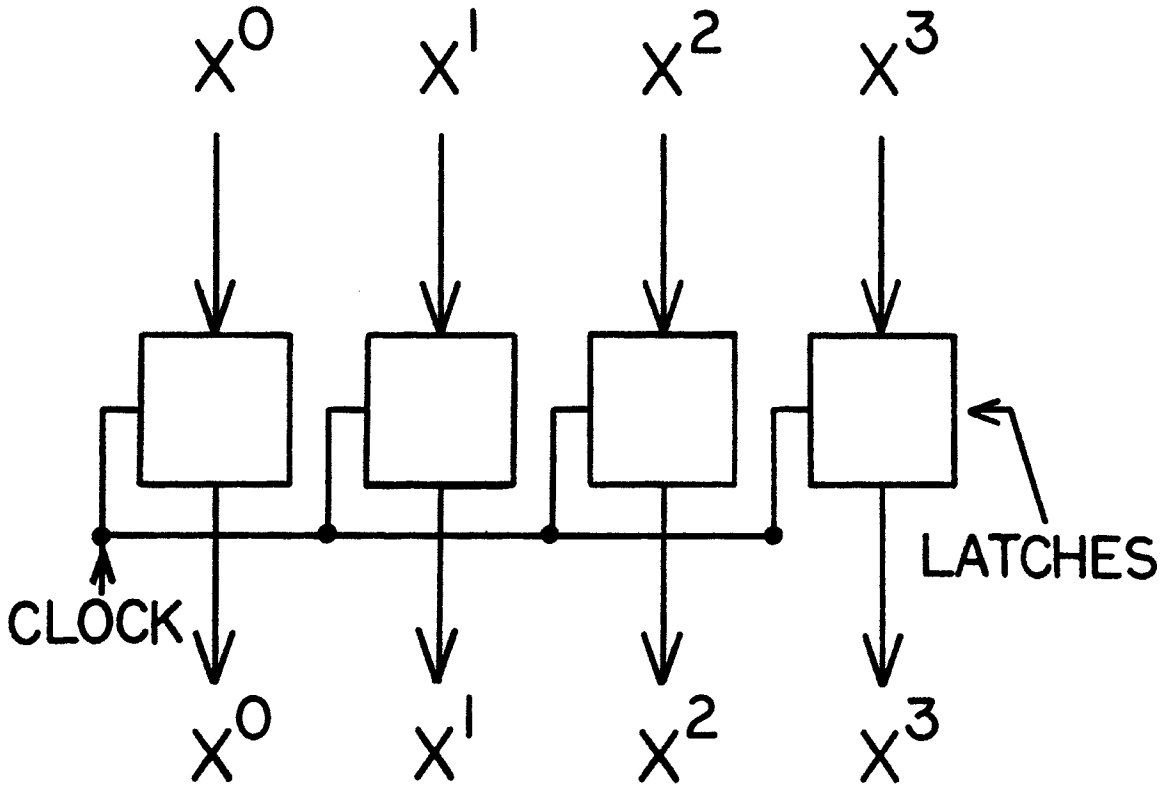


Fig. 20. Latch 4BBL Register Stage

register stages. There is no real reason why a "pnpn" device such as an SCR cannot be integrated onto a chip just like a bipolar transistor. Now consider two different types of DBBL register stages, one made with SCR's like those shown in this section and the other made with binary S-R flip-flops like the one shown in Fig. 13 of Section III. The SCR register stage would have only 1 of its 10 SCR's on and drawing current at all times. However, assuming the S-R flip-flops, for example, to be the type consisting of two cross-coupled transistors, the S-R flip-flop register stage would always have 10 of its 20 transistors on and drawing current. Thus, in DBBL register stages, the SCR realizations can hold a decided advantage over the S-R flip-flop realization in the area of average power consumption. Actually, this power advantage holds true over all types of standard binary flip-flop realizations, not just the S-R flip-flop realization. Now consider the number of "major" components in the two types of DBBL register stages just compared, where "major" here will refer only to SCR's and transistors for the sake of simplicity. A single SCR register stage contains only 10 SCR's (and possibly 2 transistors) while the S-R flip-flop register stage contains 20 transistors, 2 transistors being in each of the 10 flip-flops. (If the S-R flip-flops are more sophisticated types, such as "master-slave" units, far more than 20 transistors would be required for

such a DBBL register stage realization.) Of course, such a crude comparison proves nothing conclusive about the relative costs of the two different realizations. It does, however, hint very strongly that, in general, the SCR realizations, which are designed specifically for NBBL registers, require significantly fewer individual components than the more conventional binary flip-flop realizations. Circuitry containing fewer components implies smaller physical size and less faults, two areas which are critically important in large digital machines.

Fig. 21 depicts a "current sharing" (or current robbing) SCR base 2 NBBL register stage. The basic idea of this design is that the SCR's, the resistors, and the voltage supply are all chosen so that there is ample current available to sustain the ignition of only one SCR. Suppose that a "0" has been previously stored in the unit of Fig. 21 so that the left SCR is on. If one now attempts to alter this storage to a "1" by making input line x^1 high and input line x^0 low, the effect is that the right SCR, in attempting to turn on, robs the left SCR of enough of its original current so that this current falls below the required holding current of the SCR's. Therefore, the left SCR turns off, thus making enough current now available to the right SCR so that it can turn on. The major drawback of this design is that it requires SCR's which exactly follow ideal SCR

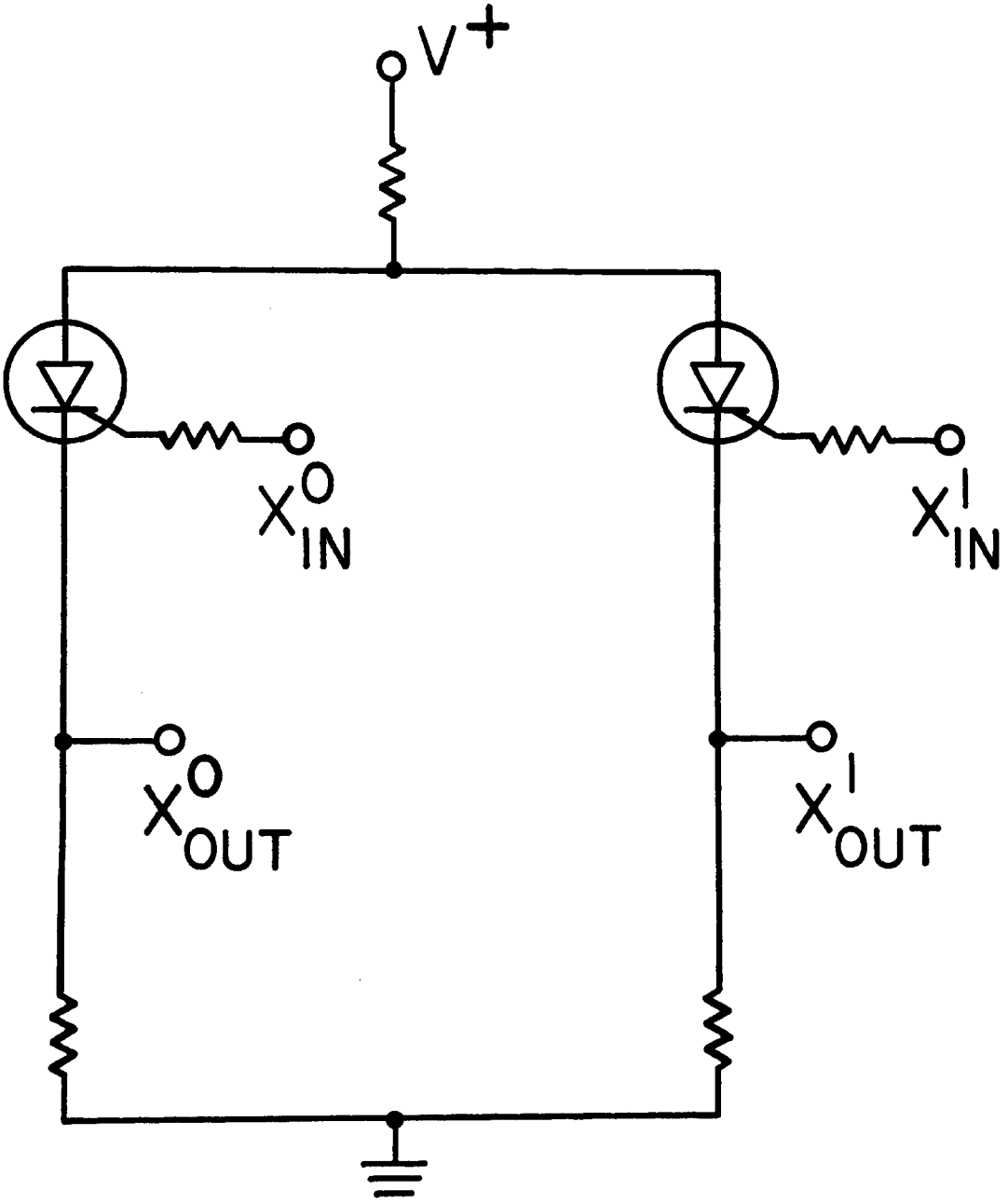


Fig. 21. Current-Sharing SCR 2BBL Register Stage

volt-ampere characteristic curves. Furthermore, all the SCR's used must be almost perfectly matched. Obviously, such SCR's are very hard to find in the real world.

Fig. 22 illustrates a capacitive-coupled SCR base 2 NBBL storage register stage. Fig. 23 shows some actual input and output waveforms for this capacitive-coupled model. It must be pointed out that this model and the final model that soon follows have both actually been built and thoroughly tested.

Imagine, that in this capacitive-coupled model, the left SCR is already on and the right SCR is off, this situation being the result of a previous input. Thus, the coupling capacitor is charged with the polarity indicated in Fig. 22. Now, if one turns on the right gate lead and turns off the left gate lead, the right SCR turns on. When this happens, the charge that was on the coupling capacitor forces the voltage level on the cathode of the left SCR above the voltage level of its anode, thus causing the left SCR to turn off (Fig. 23). The major drawbacks of this model are (1) the appearance of momentary voltages higher than the normal logic 1 level at an SCR's output when that SCR is turning off and (2) the relatively slow turn off times of each SCR caused by the inherent RC time constant of the circuit. The reader should not be alarmed by the 200 ohm load resistors in Fig. 22, which indicate a very large power

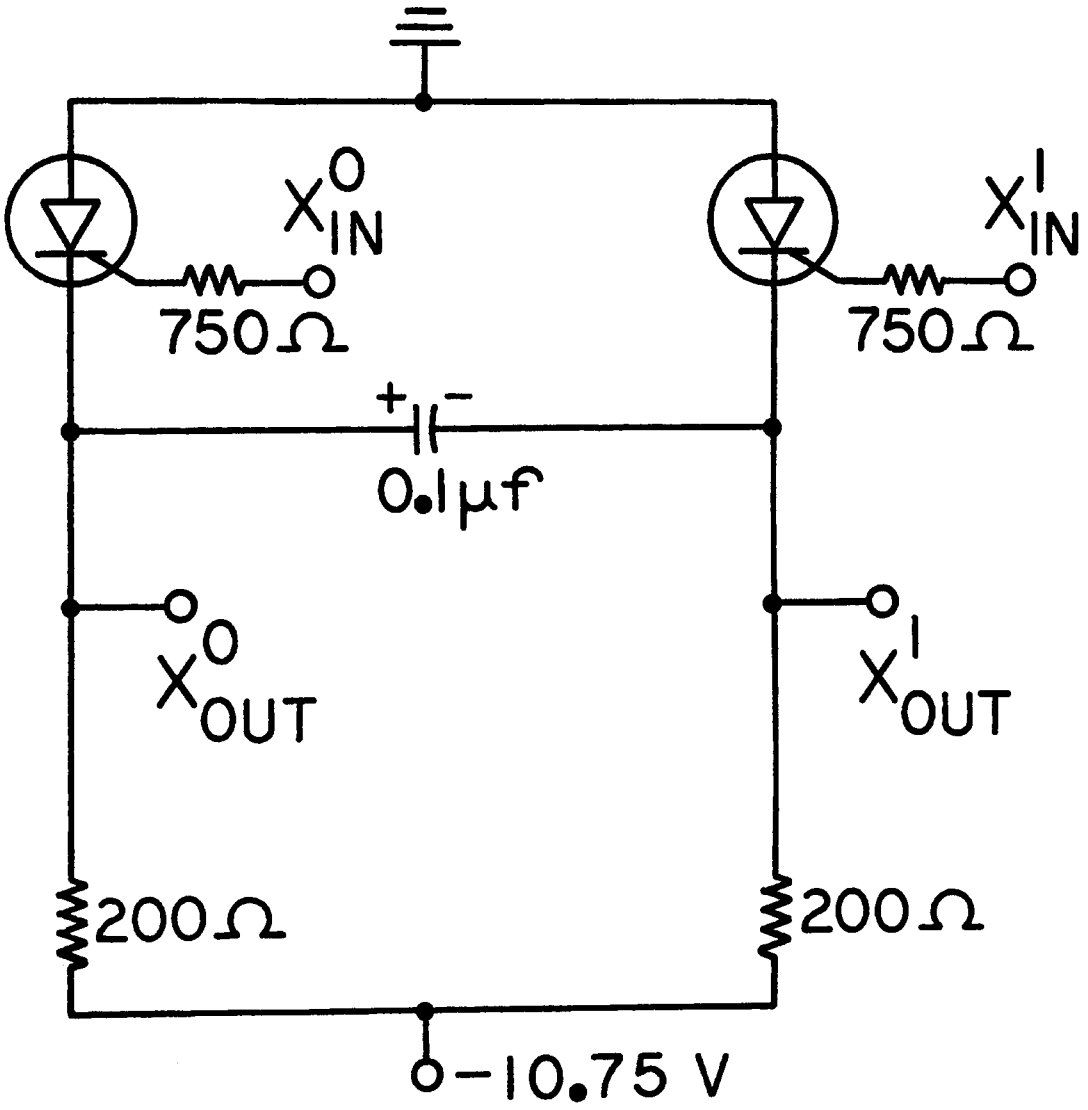


Fig. 22. Capacitive-Coupled SCR 2BBL Register Stage

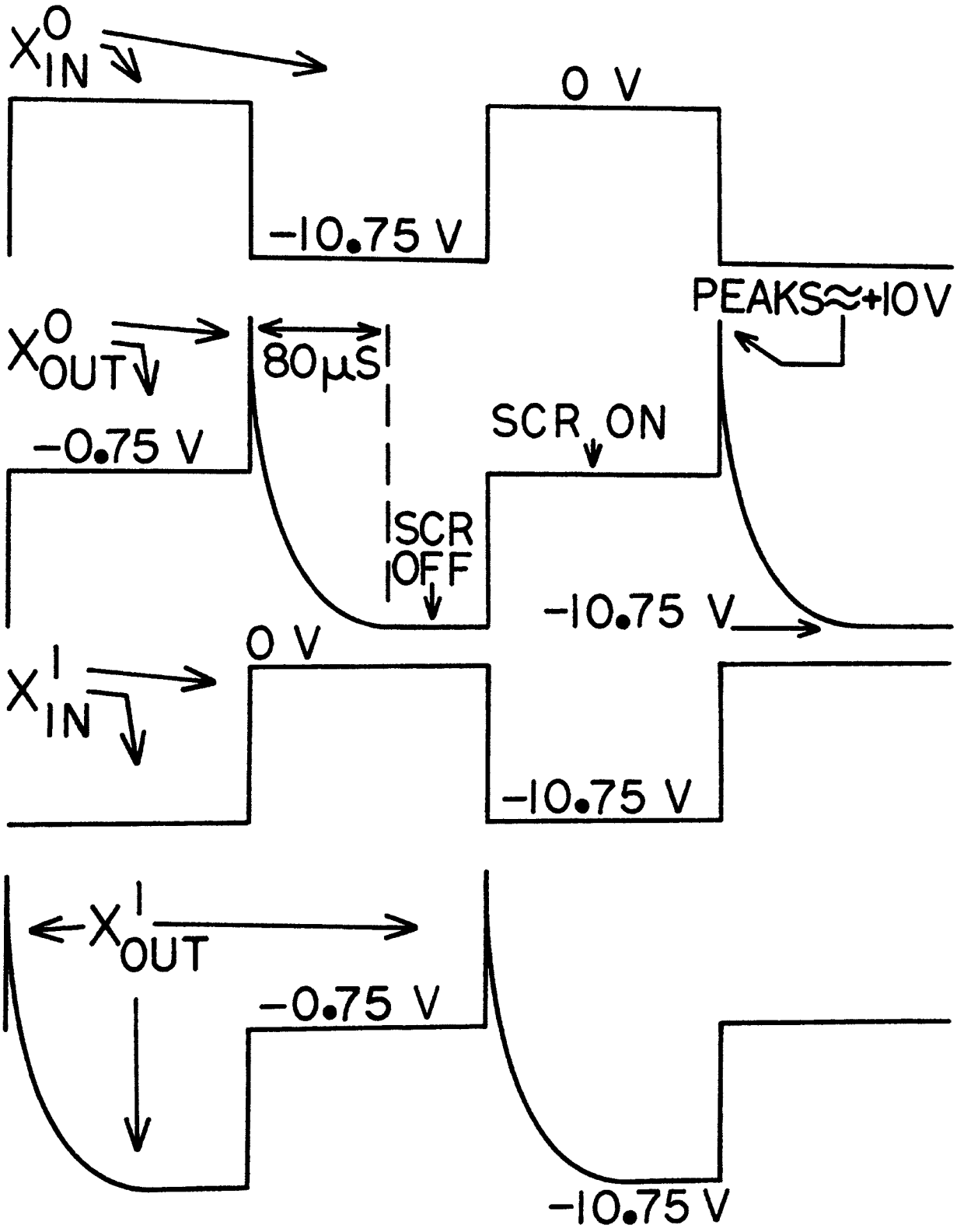


Fig. 23. Waveforms for Fig. 22

consumption. Such resistors were necessary only because the SCR's used were heavy-duty 8 ampere units with fairly large holding currents [9].

The final SCR register stage model to be discussed is termed the transistor pre-clear model. Fig. 24 illustrates the circuit itself and Fig. 25 illustrates some actual input and output waveforms that this circuit produced. The operation of this circuit is really quite simple. Whenever any gate signal rises from the logic 0 to the logic 1 level, this rise is differentiated by the series combination of the 390 picofarad capacitor and the 180 ohm resistor, thus producing a positive voltage spike. This positive spike momentarily turns on the two "pull up" transistors on the left side of the circuit, causing them to momentarily pull the voltage level of all SCR cathodes up to the same level as their anodes. This momentary shunting of all SCR's attempts to turn them off (Fig. 25). However, the SCR whose gate lead is now at the logic 1 level turns on and stays on after this momentary shunting effect has died away. The net effect then is that the circuit stores the logical 1 supplied by the one high input line. This transistor pre-clear model improves greatly on the two major drawbacks listed for the capacitive-coupled model. The transistor pre-clear model does not allow the appearance of any voltages on the outputs higher than the normal logic 1

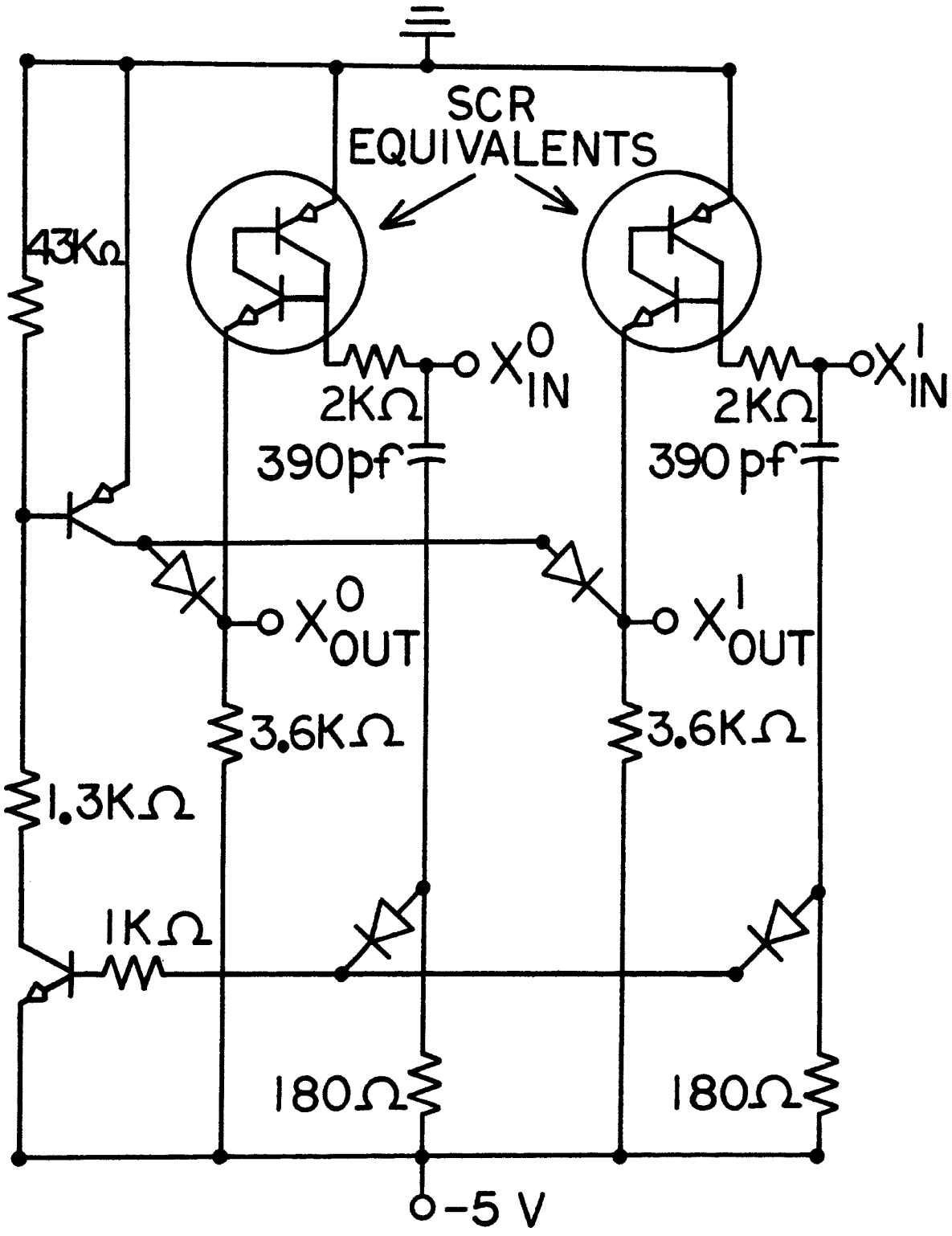


Fig. 24. Transistor Pre-Clear SCR 2BBL Register Stage

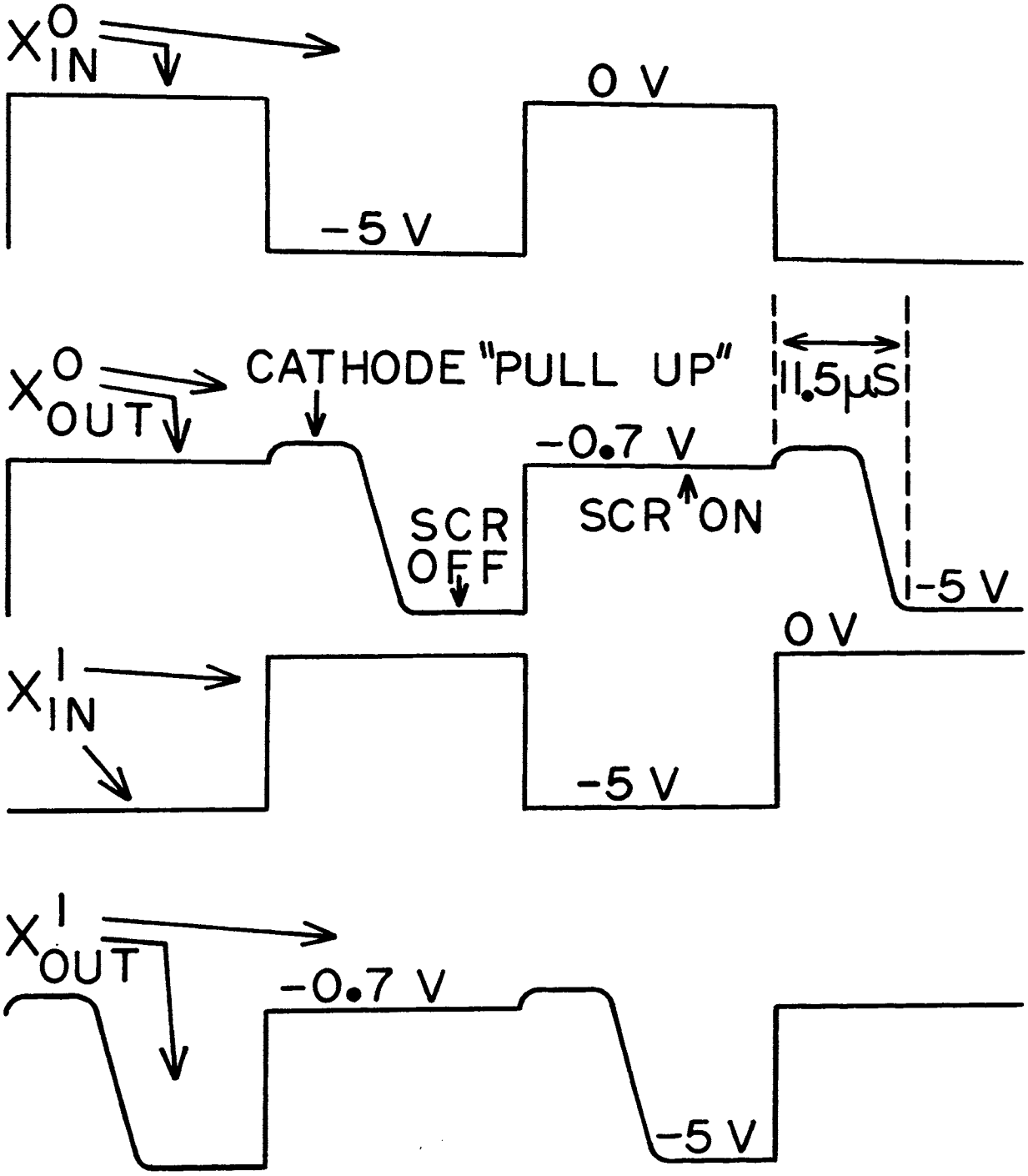


Fig. 25. Waveforms for Fig. 24

level. Furthermore, the lack of a prohibitive RC time constant, such as was found in the capacitive-coupled model, allows, in general, much lower turn off times. Therefore, the transistor pre-clear model is definitely the most promising of the three SCR models discussed in this section. It does not place the nearly impossible ideal requirements on the SCR's as does the current sharing model and it offers "cleaner," higher speed output waveforms than does the capacitive-coupled model.

It must be stressed here that neither the capacitive-coupled model nor the transistor pre-clear model, as they are shown in this section, have been refined as far as possible. None of the electronic components available to the author, especially the SCR's, were extremely high speed switching components. This is one reason why, in the transistor pre-clear model shown in Fig. 24, two-transistor equivalents [9], [10] were used to replace the original heavy-duty SCR's used in the capacitive-coupled model. These equivalents were capable of lower power, higher speed switching. Thus, the point is that, with really good high speed components, these SCR type register stages, especially the transistor pre-clear model, should be capable of speeds which are quite respectable by today's computer standards. For example, one text, already almost 3 years old, cites SCR's capable of turn off times as fast as 100 nanoseconds [6].

VIII. CONCLUSION

This paper has proposed a new type of logic called "n-base binary logic," or NBBL. The basic structure of NBBL adders and storage registers has been compared with the structures of Post base n, traditional binary-coded base n, and straight binary adders and registers. Also, several realizations, both conventional and unconventional, have been shown for NBBL adders and registers. Some of these more unconventional realizations, such as the SCR circuits shown for NBBL storage register stages, are quite promising and deserve further research and extensive developmental work.

This paper has also performed cost analyses of "decimal-base binary logic," or DBBL, adders and registers relative to their straight binary equivalents. These analyses indicated that, without a doubt, the DBBL adders and registers would cost more both to construct and to buy (or rent). On the other hand, a DBBL machine offers greater operational efficiency and speed over its binary counterpart as a result of the following major advantages: (1) no decimal-to-binary and binary-to-decimal conversion, (2) much simpler and faster "coding" of numerical input and output data, and (3) the potential of faster overall arithmetic operation within the machine. These advantages just listed mean that the DBBL machine should be capable of doing more work per unit of time, especially in high

numerical input-output usages. Thus, for example, a DBBL machine owner who allows others to use his machine on a shared-time basis can get more customer jobs done in a day's time, each customer having to pay for less used time than would customers of a binary machine shared-time system. Also, the DBBL system need not contain hardware or software to do conversion and re-conversion, and thus, the DBBL machine user, unlike the binary machine user, need not suffer the cost of these items.

Today the development of computer circuitry is rapidly approaching the point where logic speeds are simply as fast as they can ever be. For example, emitter-coupled logic gates are capable of propogational delays of less than a nanosecond [6], not much more time than it takes for an electrical signal to travel the length of a small piece of plain wire. As a result, the use of DBBL adders and registers, like the use of parallel processors, could be one means of further increasing machine speeds even after logic speeds have reached their limit. Thus, at a time in the computer industry when speed seems to be taking precedence over everything else, even circuitry cost, a DBBL machine indeed has much to offer.

REFERENCES

[1] M. S. Schmookler and A. Weinberger, "High Speed Decimal Addition," IEEE Transactions on Computers, Vol. C-20, No. 8, pp. 862-866, Aug., 1971.

[2] A. S. Wojcik and G. Metze, "On the Cost of Base N Adders," IEEE Transactions on Computers, Vol. C-20, No. 10, pp. 1196-1203, Oct., 1971.

[3] A. S. Wojcik and G. Metze, "Cost Considerations of Base N Adders Using N-Valued Logic Elements," Proceedings of the 7th Annual Allerton Conference, pp. 680-689, 1969.

[4] L. Sintonen, "On the Realization of Functions in N-Valued Logic," IEEE Transactions on Computers, Vol. C-21, No. 2, pp. 610-612, June, 1972.

[5] S. S. Yau and C. K. Tang, "Universal Logic Modules and Their Application," IEEE Transactions on Computers, Vol. C-19, No. 2, pp. 141-149, Feb. 1970.

[6] L. Strauss, Wave Generation and Shaping, Second Edition. United States: McGraw Hill, Inc., 1970, pp. 183, 270-273, 500-502, 569.

[7] IBM System/360 Model 50 Functional Characteristics, Second Edition, IBM Corporation, Systems Development Division, Product Publications, Poughkeepsie, N. Y., Form A22-6898-1, 1967.

[8] J. Marino and J. Sirota, "There's a Read-Only Memory That's Sure to Fill Your Needs," Electronics, Vol. 43, No. 6, pp. 112-116, March 16, 1970.

[9] GE Silicon Controlled Rectifier Manual, Fourth Edition, General Electric Company, Application Engineering Center, Semiconductor Products Department, Syracuse, N. Y., 1967, pp. 7-9, 424.

[10] GE Silicon Controlled Rectifier Manual, Second Edition, General Electric Company, Application Engineering Center, Rectifier Components Department, Auburn, N. Y., 1961, pp. 2-3.

VITA

James Oliver Bondi was born on May 29, 1949 in St. Louis, Missouri where he received both his primary and secondary education. He began his college education in September, 1967 at the University of Missouri-Rolla at Rolla, Missouri, later receiving his Bachelor of Science degree in Electrical Engineering from this institution in May, 1971.

He then enrolled in the Graduate School of the University of Missouri-Rolla in September, 1971 to begin work on his Master of Science degree in Electrical Engineering. He was awarded a three-year NDEA Fellowship which has financed his graduate studies to date and which will continue to support him in his future Ph.D. endeavors.