2.2.0

Digital Computer Laboratory
Massachusetts Institute of Technology
Cambridge, Massachusetts

SUBJECT: THE USE OF BOOLEAN ALGEBRA IN LOGICAL DESIGN

To: N. H. Taylor

From: R. C. Jeffrey, I. S. Reed

Date: April 15, 1952 (Revised April 28, 1952)

Abstract: This note is a practical description of Boolean algebra and its application to the analysis and synthesis of digital computers. It is argued that knowledge of the theory and methods described here is equivalent in value to considerable experience and ingenuity in the logical design of computers, and that it provides a way of bringing a novice in the field up to the point where he can make contributions considerably more quickly than this is done at present.

1.0 INTRODUCTION

To a first approximation we can describe a binary computer as a set of 2 state memory devices functionally connected by an information processing network. This first approximation to any particular computer represents its logical design; if it has been well engineered and well constructed, the approximation will be useful: for example, we may then ignore the fact that the voltages at critical points in the machine may assume any one of a continuous range of values.

It is customary to represent the logical structure of a machine by block diagrams. Unfortunately, you cannot calculate with block diagrams: they are merely expository devices. Everyone will agree that it would be helpful to be able to represent machines by sets of equations for which we know simple rules of transformation. Much would then become routine which now requires more or less experience and ingenuity, leading the designer more quickly to the important decisions.

There exists a system of mathematics within which such calculation is possible. Its mechanical rules are simpler than those of ordinary algebra, as will be seen in the next section. With a very little practice at it, a novice in the field of digital computers can solve, with understanding, a large class of non-trivial problems. For example, the following problem is solved later in the text.

Design a three bit "counter" with the following "loops":

FF1	FF2	FF3	
1	1	0	
0	1	1	
1	1	0	(alternates between 110 and 011)
0	0	0	
0	1	0	(passes from 000 into its
1	0	0	cycle, but 000 is not
0	0	1	included in the cycle)
0	1	0	
1	1	1	
1	0	1	
1	0	1	(sticks on 101)

Such devices might be used as operation counters.

We do not by any means suggest that facility at Boolean algebra will supercede experience and ingenuity in the logical design of computers. Rather

(1) the algebra provides a way of efficiently channeling the experience and ingenuity of the novice: a unified theory accelerates and deepens learning.

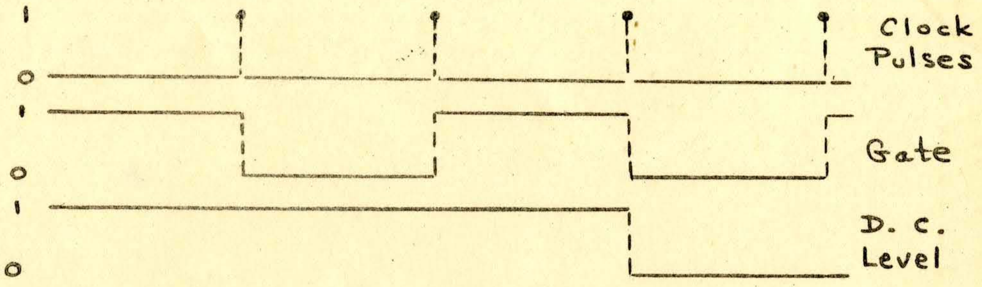
(2) it allows the practicing designer immediate access to the important, non-routine problems: they allow him to use his skill where it counts.

2.0 BOOLEAN ALGEBRA

Boolean algebra is most often developed as an abstract mathematical system, the interpretation being left open. Here, however, we parallel each step in the exposition of the theory with its counterpart in terms of the familiar block diagrams in the hope of promoting a sense of confidence and familiarity with the new technique.

The voltage (or current or whatever physical magnitude represents information) at any logically important point in a machine may be represented to a first approximation as a function of time which, for every value of t, is either 0 or 1. Any change in such a function will then be a jump discontinuity.

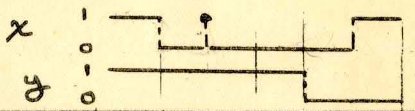
Examples



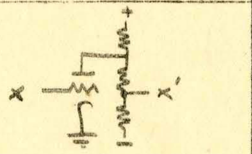
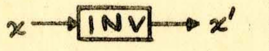
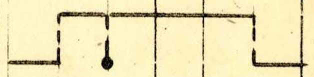
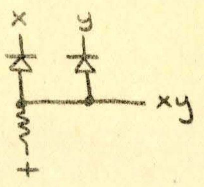
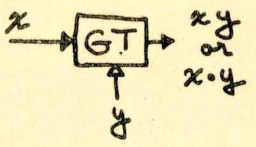
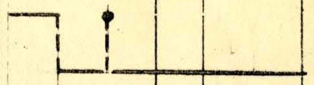
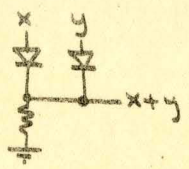
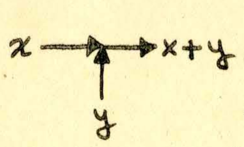
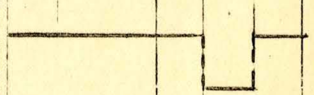
The elements of our algebra are such Boolean functions of time.

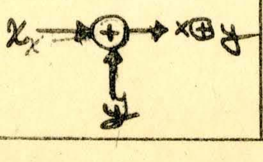
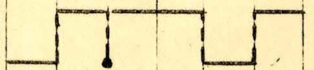
We define four operations on such functions: ways of compounding from x(t) and y(t) new Boolean functions. For conciseness we shall omit the time variable in the following table, in which, for example, "x'" is an abbreviation for "x'(t)", and "x + y" abbreviates "x(t) + y(t)".

Under "Graph" we show the output waveform when the inputs, x(t) and y(t), are:



DIF. FROM BINARY ADDITION

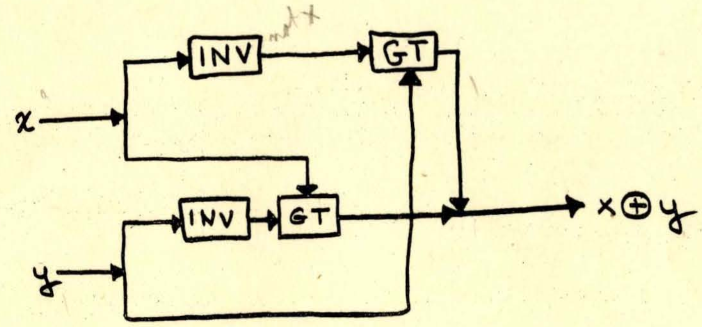
Name	Table	One Physical Realization	Block Diagram	Graph
Not; ~ Complement	x x' 0 1 1 0			
And; Logical Product	x y xy 0 0 0 0 1 0 1 0 0 1 1 1			
Or; Logical Sum	x y x+y 0 0 0 0 1 1 1 0 1 1 1 1			

Partial Sum;
Cyclic Sum;
Symmetric Difference
	x y $x \oplus y$
	0 0 0
	0 1 1
	1 0 1
	1 1 0
		See P. 4	Proposal:
			
				

Since there are a finite number of different ways of assigning 0 and 1 as values to n variables, it will always be possible to completely describe any Boolean function by a table as we have done for the four above.

For n inputs there are 2^{2^n} Boolean functions, any one of which *might* be realized electronically in some simple way. The algebra is neutral on this issue: new physical realizations of functions which previously had to be built up out of others have algebraic representations waiting for them and can be integrated, without changing the algebra, into the data of the design problem.

To proceed with the formalism: there are 2^n ways of assigning one of the two values, 0, 1, to each of n variables. Then it is practical to check any presumed theorem of our algebra by substituting (in tabular form) each possible combination of values for the variables on each side of the equation. Thus we can prove that the cyclic sum, $\oplus$, is represented by this combination of gates, mixers and inverters:



$x \oplus y =$	$(x \circ y')$	$+$	$(x' \circ y)$
0 0 0	001	0	1 0 0
0 1 1	000	1	1 1 1
1 1 0	111	1	0 0 0
1 0 1	100	0	0 0 1

by definition

identical for all x,y

Note that once the 4 pairs of values of x, y are listed, the values of x' and y' are determined, and from these, $x \circ y'$, $x' \circ y$ and $xy' + x'y$.

By the same tabular method each of the following theorems of the algebra can be proved. Again, for conciseness, we have omitted time variables.

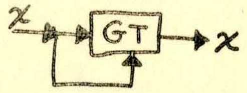
Law of Double Negation: $(x')' = x$. $$\begin{tikzcd} {} \arrow[r, "x"] & \boxed{INV} \arrow[r, "x'"] & \boxed{INV} \arrow[r] & x \end{tikzcd}$$

Dual Theorems (The result of interchanging '0' and '1', '+' and '.' in an expression is the complement of that expression. The result of that interchange in a theorem is another theorem.)

For Products

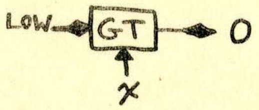
<u>No Powers</u>

$xx = x$

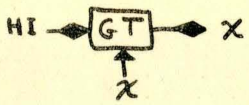


<u>Multiply by a constant as in arithmetic:</u>

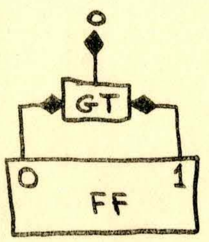
$0.x = 0$



$1.x = x$

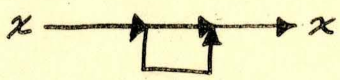


$x.x' = 0$



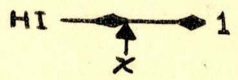
For Sums

<u>No Numerical Coefficients:</u> $x + x = x$

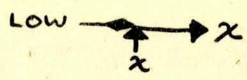


<u>Addition of a constant:</u>

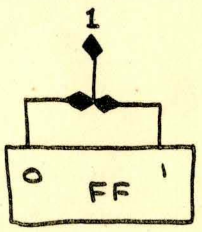
$1 + x = 1$



$0 + x = x$



$x + x' = 1$



<u>Associative and Commutative Laws: Ignore grouping and order</u>

<u>in pure products:</u>

$x(yz) = (xy)z = xyz$

$xy = yx$

<u>in pure sums:</u>

$x + (y + z) = (x + y) + z = x + y + z$

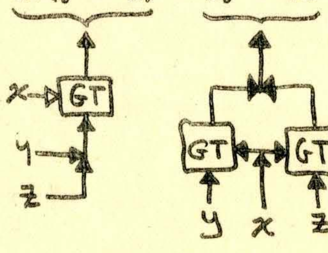
$x + y = y + x$

<u>De Morgan's Theorem:</u> $(x + y)' = x'y'$ $(xy)' = x' + y'$

<u>Distributive Laws</u>

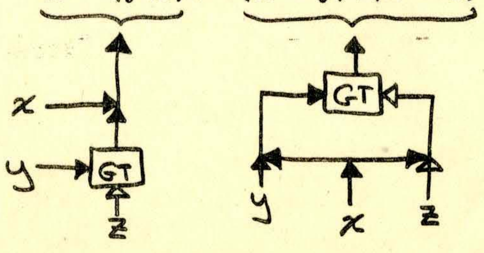
<u>"Multiply through" and factor as in arithmetic:</u>

$x.(y + z) = xy + xz$

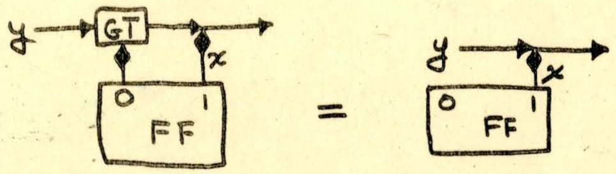


<u>Unlike arithmetic: you may also "add through" a product:</u>

$x + (y.z) = (x + y).(x + z)$



A very handy simplification: $x + x'y = x + y$



Theorems of more purely theoretical interest:

Expansion of 'I'

I is the sum of all 2^n possible products of $\underline{n}$ variables and their primes.

$$\begin{aligned} I &= x + x' \\ &= xy + x'y + xy' + x'y' \\ &= xyz + x'yz + xy'z + x'y'z + xyz' + x'yz' + xy'z' + x'y'z' \\ &= xyzw + \ldots\ldots (14 \text{ terms}) \ldots\ldots + x'y'z'w' \\ &= \end{aligned}$$

Each function of $\underline{n}$ variables can be represented by dropping some of the terms of the above sum:

$$f(x) = f(1)x + f(0)x'$$

$$f(x,y) = f(1,1)xy + f(0,1)x'y + f(1,0)xy' + f(0,0)x'y'$$

$$f(x,y,z) = f(1,1,1)xyz + \ldots\ldots + f(0,1,0)x'yz' + \ldots\ldots + f(0,0,0)x'y'z'$$

etc.

Note the relation between zeros in the argument places and primes on the corresponding variables.

This last theorem is of special importance since it allows us to write an algebraic expression for a function directly from its table:

x	y	f(x,y)
1	1	1
0	1	0
1	0	1
0	0	1

The table is an abbreviation of 4 statements:

$f(1,1) = 1$
$f(0,1) = 0$
$f(1,0) = 1$
$f(0,0) = 1$

$$\begin{aligned} \therefore f(x,y) &= f(1,1)xy + f(0,1)x'y + f(1,0)xy' + f(0,0)x'y' \\ &= 1.xy + 0.x'y + 1.xy' + 1.x'y' \\ &= xy + 0 + xy' + x'y' \\ &= xy + xy' + x'y' \end{aligned}$$

A further example:

x	y	z	g(x,y,z)
1	1	1	0
0	1	1	0
1	0	1	1
0	0	1	0
1	1	0	1
0	1	0	0
1	0	0	0
0	0	0	1

$$g(x,y,z) = xy'z + xyz' + x'y'z'$$

As an excercise, note that in the first example, $f(x,y)$ can be further simplified to $x + y'$. (Factor, and use the theorems: $a + a' = 1$; $1.a = a$; $a + a'b = a + b$)

3.0 APPLICATION TO PASSIVE NETWORKS

We may now illustrate the technique of reducing networks which do not contain memory elements. It is assumed that all pulses occur at the same time, so the time variable will be dropped.

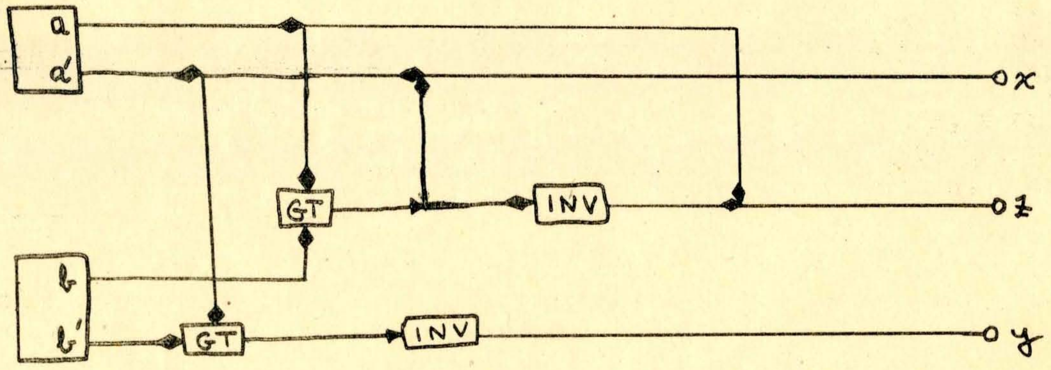
Example of translation of equations into block diagram

$x = ab + a' = a' + b$

$y = (a'b')' = (a + b)$

$z = a + (ab)' = 1$

Here regard a and b as inputs, x, y and z as outputs and assume that a and b are obtained from FF's so that both a, b and a' and b' are available.



<u>Simplification</u>: This design contains redundancies in the sense that fewer gates and inverters may be used to get the <u>same outputs</u> for each input:

Since $x + x'y = x + y$

$$\underline{x = a' + b}$$

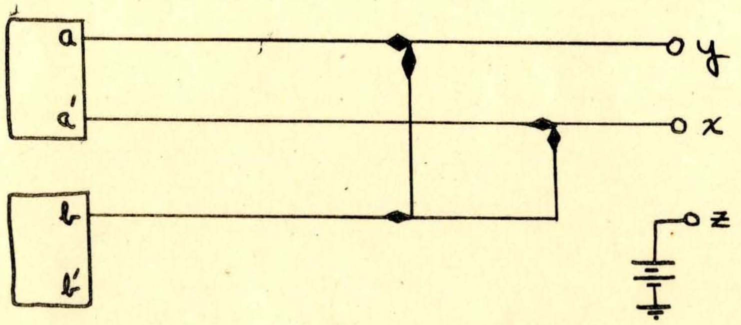
By De Morgan's theorem

$$\underline{y = a + b}$$

$$z = a + a' + b' = 1 + b' = 1$$

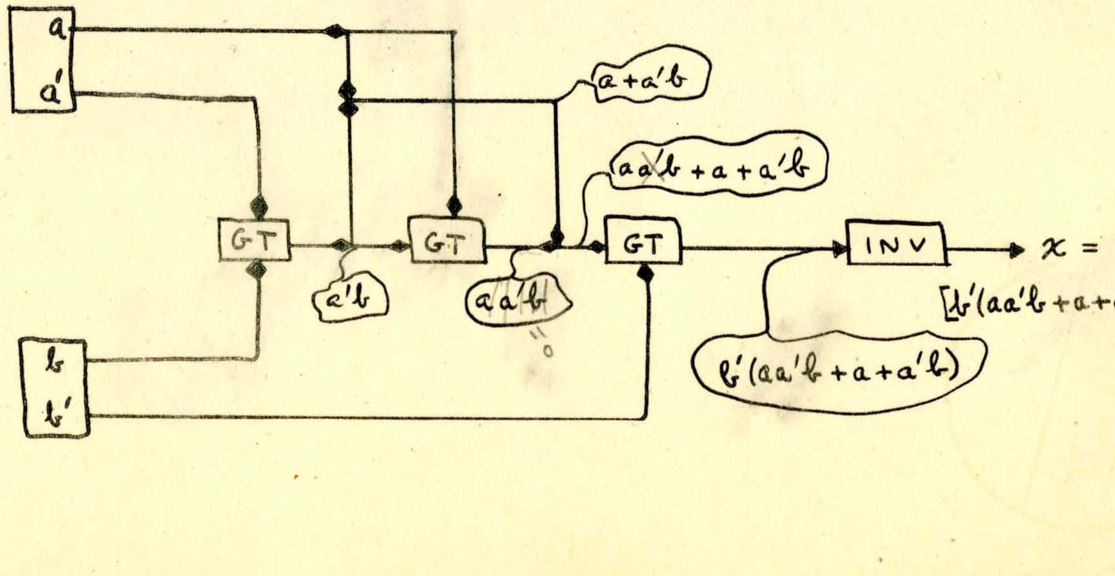
Thus z is simply a point which is permanently at, say, high voltage.

This gives as a simpler equivalent block design



<u>Example of translation of block diagram into symbols</u>:

We may analyze the circuit following and simplify it by first translating it into equations:



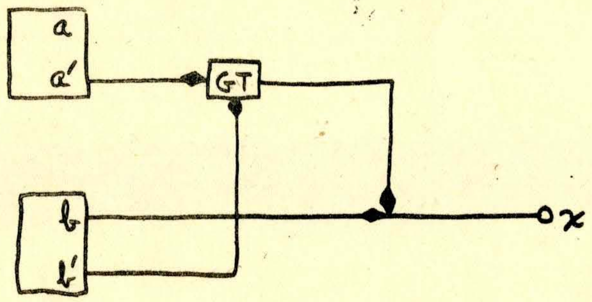
Now we simplify: $aa' = 0$ and $0.b = 0$. Then the $aa'b$ term vanishes.

$a + a'b = a + b$. Hence $x = \left[b'(a + b)\right]'$. By De Morgan's theorem,

$x = b + (a + b)'$ and by another application

$x = b + a'b' = b + a'$

<u>Simplified block diagram:</u>



A further simplification, which will eliminate the gate, is left as an exercise.

<u>Translation of tables into equations</u>

It is desired to construct a circuit with the properties given by the table:

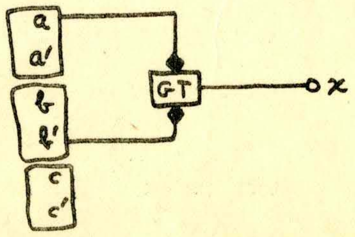
a	b	c	x
1	1	1	0
0	1	1	0
1	0	1	1
0	0	1	0
1	1	0	0
0	1	0	0
1	0	0	1
0	0	0	0

which tells, for each possible triad of input values, the desired output.

We find the desired $x = f(a,b,c)$ as a sum of the products associated with the 1's in the table.

$$x = ab'c + ab'c' = ab'(c + c') = ab'(1) = ab'$$

Then c is superfluous:



The devices indicated above have their analogies in the methods of Aiken and Shannon. However, the treatment of flip-flops in the next section is new, and represents a radical advance over other methods.

4.0 THE FLIP-FLOP EQUATIONS

At this point it becomes necessary to explicitly indicate the dependence on time of the variables in our discussion. Expressions like '$A(t)$' represent voltage (or current or whatever physical quantity is used as the realization of our 0 and 1) at particular points in the machine at time t. If '$A(t)$' is such an expression, '$A(t+T)$' will be the voltage (or whatever) at the same point in the machine T seconds later.

For definiteness, let us assume we are dealing with a clocked machine, i.e., a machine in which any changes in state of the FF's must occur at discrete times, the times at which the clock pulses occur. Call the period of the clock T.

We shall analyze, under the name 'flip-flop' an Eccles-Jordan multivibrator with 2 inputs, a clear and a set, with a cross-over circuit such that when both inputs are 'on' simultaneously, triggering action occurs and the FF is complemented. This is somewhat different (superficially) from the WWI sort of FF which is provided with 3 inputs, no 2 of which may be 'on' at once. However, the transformation to the WW variety is simple, once the equation for the present type is established.

We know that the state of the FF after the inputs have been pulsed depends only on (1) which input has been pulsed (has value I) and (2) the state of the FF at the time the input was pulsed: $A(t+T)=f\left[{}_0a(t), a(t), A(t)\right]$

For example, if neither input is pulsed, i.e., if ${}_0a(t) = a(t) = 0$, then the FF state doesn't change: $A(t+T) = A(t)$. If only the clear side is pulsed $\left[{}_0a(t) = 1,\ a(t) = 0\right]$ the state of the FF goes to 0 regardless of what it was before: $A(t+T) = 0$. And if the FF is complemented by pulsing both inputs $({}_0a(t) = a(t) = I)$ then $A(t+T) = A'(t)$. These characteristics of the FF may be summarized by the following table.

$A(t+T)$	${}_0a(t)$	$a(t)$	$A(t)$	Explanation
0	1	1	1	Complement
1	0	1	1	Set
0	1	0	1	Clear
1	0	0	1	No change
1	1	1	0	Complement
1	0	1	0	Set
0	1	0	0	Clear
0	0	0	0	No change

This table determines $A(t + T)$ as a function of the inputs and the state at time t. To get an equation for the FF we represent the table in the form

$$A(t + T) = f\left[{}_{o}a(t), a(t), A(t)\right] =$$

$$f(1,1,1)\,{}_{o}a(t)a(t)A(t) + \ldots\ldots + f(0,0,0)\,{}_{o}a'(t)a'(t)A'(t),$$

that is, as a sum of those products which correspond to the '1's under "$A(t + T)$".

$$A(t + T) = {}_{o}a'(t)a(t)A(t) + {}_{o}a'(t)a'(t)A(t) + {}_{o}a(t)a(t)A'(t) + {}_{o}a'(t)a(t)A'(t)$$

Factoring "${}_{o}a'(t)A(t)$" from the 1st 2 terms and "$a(t)A'(t)$" from the 2nd 2 terms:

$$A(t + T) = {}_{o}a'(t)A(t)\left[a(t) + a'(t)\right] + a(t)A'(t)\left[{}_{o}a(t) + {}_{o}a'(t)\right]$$

and since $x + x' = I$ and $x.I = x$

$$\boxed{A(t + T) = {}_{o}a'(t)A(t) + a(t)A'(t)}$$

This is the flip-flop equation, which describes the action of a FF the way $x(t) \oplus y(t) = x(t)y'(t) + x'(t)y(t)$ describes the action of a partial sum circuit. Here, however, we deal essentially with a difference in time (it is this fact which made the analysis of the FF come later than that of "instantaneous" networks in Boolean algebra).

Now the problem of designing a circuit using flip-flops is simply that of connecting the proper network onto the two inputs. That is, if we know ${}_{o}a(t)$ and $a(t)$ as functions of the ultimate inputs to the circuit we can draw block diagrams for the inputs to the flip-flops and hence have the circuit.

<u>Illustrations of Circuit Design via Flip-Flop Equations</u>

<u>2 Stage Binary Counter</u>

(This example is chosen because the result is familiar. In the next example we illustrate the use of this method in analyzing a more difficult problem.) We wish the counter to be cyclic, i.e., the successive states of the flip-flop are as shown. The flip-flop will progress from each state to the next after a <u>count</u> command (P(t)).

A_1	A_2
FF1	FF2
0	0
0	1
1	0
1	1
0	0

Apparently we need to clear FF1 in only one case: when A_1 and A_2 are both 1 and there is a count pulse.

Thus $_0a_1(t) = A_1(t)A_2(t)P(t)$

In other cases where $A_1 = 0$ the previous state was also 0, so no pulse is required to maintain the state. Note that we are designing in terms of the changes in state of the flip-flops, and not in terms of the states themselves.

We must set A, when $A_1 = 0$, $A_2 = 1$ and there is a count pulse:

$$a_1(t) = A_1'(t)A_2(t)P(t)$$

Similarly for A_2:

Clear: $_0a_2(t) = A_1'(t)A_2(t)P(t) + A_1(t)A_2(t)P(t)$

i.e., we wish to clear A_2 when either of these two conditions exist:

$A_1 = 0$, $A_2 = 1$, pulse

$A_1 = 1$, $A_2 = 1$, pulse

Now we can factor: $A_1'(t)A_2(t)P(t) + A_1(t)A_2(t)P(t)$

$$= \left[A_1'(t) + A_1(t)\right] A_2(t)P(t) = [1]\, A_2(t)P(t)$$

$$\therefore \boxed{_0a_2(t) = A_2(t)P(t)}$$

Thus we wish to clear A_2 on the next pulse whenever it holds a 1, a fact which might have been read directly from the table (regardless of what is in the left hand column, the successor of any '1' under 'A_2' is a 0).

Finally, to set A_2:

$$a_2(t) = \left[A_1'(t)A_2'(t) + A_1(t)A_2'(t)\right] P(t)$$
$$= \left[A_1'(t) + A_1(t)\right] A_2'(t)P(t)$$
$$= A_2'(t)P(t)$$

which means that A_2 is to be cleared on the next pulse whenever it holds a '0' regardless of what A_1 holds. This also might have been seen from the table.

We now have, for the equations for the changes to the grids of the flip-flop, the following (abbreviated by dropping the 't'):

$$ {}_{o}a_1 = A_1A_2P $$

$$ a_1 = A_1'A_2P $$

$$ {}_{o}a_2 = A_2P $$

$$ a_2 = A_2'P $$

These may be reduced by replacing 'A_2P' in the first equation by '${}_{o}a_2$' (from the third) and ditto in the second:

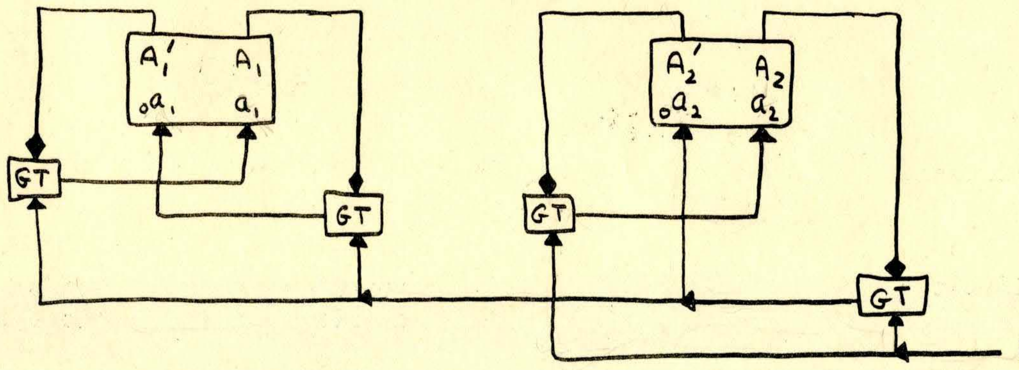
$$ {}_{o}a_1 = A_1 \; {}_{o}a_2 $$

$$ a_1 = A_1' \; {}_{o}a_2 $$

$$ {}_{o}a_2 = A_2P $$

$$ a_2 = A_2'P $$

Thus we have eliminated several gates. The block diagram is unfamiliar because of the use of two-input flip-flops.



Using only the trigger inputs of flip-flops results in a simpler circuit:

Here ${}_{o}a = a = {}_{c}a$. The flip-flop equation becomes

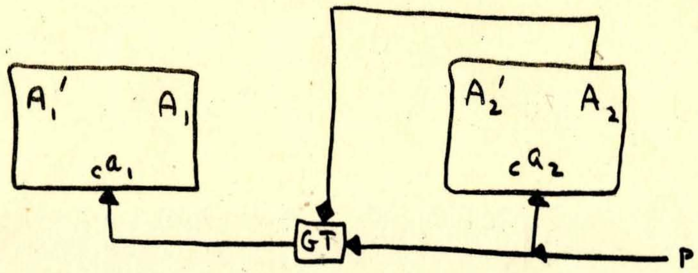
$$ \boxed{A(t+T) = {}_{c}a(t)A'(t) + {}_{c}a'(t)A(t) = {}_{c}a(t) \oplus A(t)} $$

Returning to the table, A_1 should be complemented whenever $A_2 = 1$:

$${}_c a_1 = A_2 P$$

and for A_2: ${}_c a_2 = P$ (complemented on <u>every</u> pulse)

<u>The Block Diagram</u>



Note that the delay necessary so that A_2 will change <u>after</u> P has tried to pass the gate is assumed to exist in the FF. This was implicit in our original FF equation.

We now indicate, without much comment, the solution of the less familiar problem proposed in the introduction. We shall use only the complement input to the flip-flops (except for reading in numbers, a simple process which we wont include in the problem). The "counter" changes state when it receives a command pulse, P(t).

A_1	A_2	A_3	
1	1	0	
0	1	1	
1	1	0	
0	0	0	
0	1	0	
1	0	0	
0	0	1	
0	1	0	
1	1	1	
1	0	1	
1	0	1	(stick)

$$A(T) = {}_c a \oplus A$$

(abbreviated form of the full equation:

$$A(t + T) = {}_c a(t) \oplus A(t))$$

We are concerned only with the <u>changes</u> in the states of the flip-flops.

For A_1: $${}_c a_1 = (\underline{A_1A_2A_3'} + A_1'A_2A_3 + A_1'A_2A_3' + \underline{A_1A_2'A_3'})P$$

$$= \left[A_1A_3'(A_2 + A_2') + A_1'A_2(A_3 + A_3')\right]$$

$$\boxed{{}_c a_1 = (A_1A_3' + A_1'A_2)P}$$

For A_2: $${}_c a_2 = (\underline{A_1'A_2'A_3'} + A_1'A_2A_3' + \underline{A_1'A_2'A_3} + A_1A_2A_3)P$$

$$= \left[A_1'A_2'(A_3' + A_3) + A_2(A_1'A_3' + A_1A_3)\right]P$$

$$\boxed{{}_c a_2 = \left[A_1'A_2' + A_2(A_1'A_3' + A_1A_3)\right]P}$$

For A_3: $${}_c a_3 = (A_1A_2A_3' + A_1'A_2A_3 + A_1A_2'A_3' + A_1'A_2'A_3)P$$

$$= \left[A_1A_3'(A_2 + A_2') + A_1'A_3(A_2 + A_2')\right]P$$

$$\boxed{{}_c a_3 = \left[A_1A_3' + A_1'A_3\right]P}$$

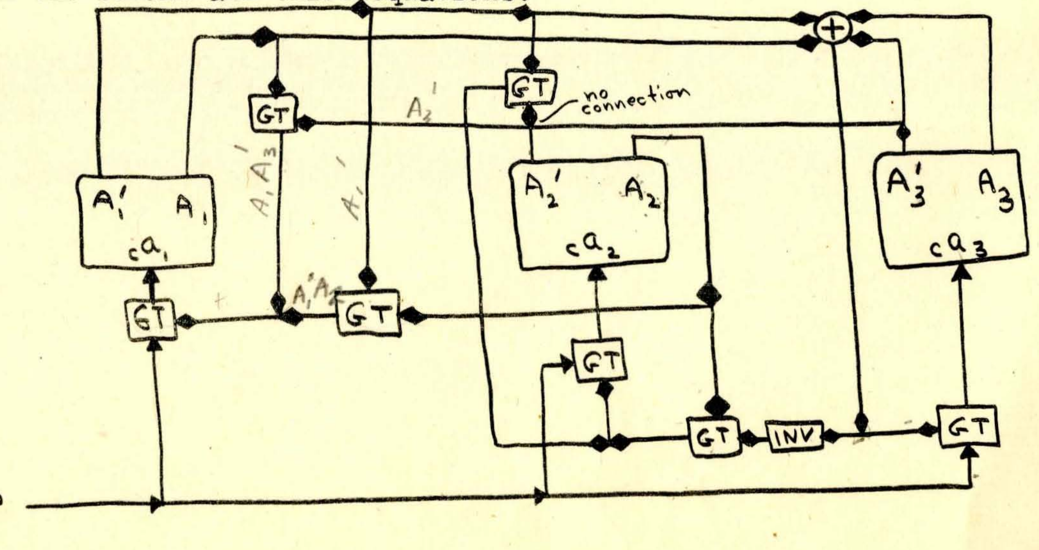
In order to simplify the block diagram for this counter, let us assume that we have available a "package" realization of $x \oplus y$.

$${}_c a_1 = (A_1A_3' + A_1'A_2)P \text{ as before}$$

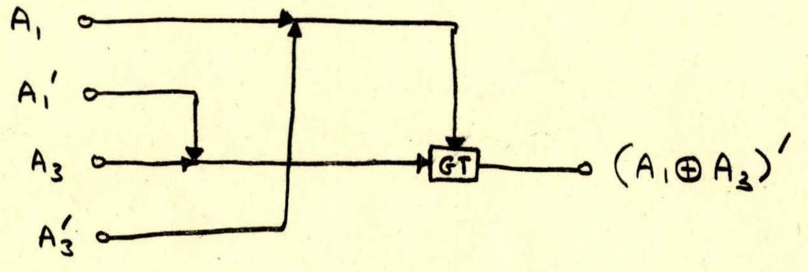
$${}_c a_2 = \left[A_1'A_2' + A_2(A_1 \oplus A_3)'\right]P$$

$${}_c a_3 = (A_1 \oplus A_3)P$$

We may now (assuming that inverters are cheap) use the same circuit for $A_1 \oplus A_3$ in the second and third equations.



In case inverters are more expensive than gates, we might synthesize $(A_1 \oplus A_3)'$ directly



[Note that this is the dual of the circuit for $\oplus$: we interchanged gates(.) and mixers (+)].

As a final example of the use of the algebra in synthesizing circuits we shall design an adder.

Let A_i and B_i be the digits to be added in stage #i, and let the carry into this stage be C_i. Then the carry out will be C_{i+1} and the well known table governs the action of the stage:

$A_i(t)$	$B_i(t)$	$C_i(t)$	$C_{i+1}(t)$	$A_i(t + T)$
1	1	1	1	1
0	1	1	1	0
1	0	1	1	0
0	0	1	0	1
1	1	0	1	0
0	1	0	0	1
1	0	0	0	1
0	0	0	0	0

Note that we require the carry to be instantaneous: there is to be no cumulative delay from stage to stage. Also note that the sum is to be stored in A_i. This time our table does not represent the successive states of a counter! We are interested in successive states of A_i. These are indicated row by row by looking first at column one and then at the parallel entry in the last column. Use complementable FF: (we wish to complement in cases 3, 4, 5, 6). Note that these are just the cases in which $B_i(t) = C_i'(t)$, i.e., $B_i(t) \oplus C_i(t) = 1$

$$\boxed{{}_{c}a_i(t) = \left[B_i(t) \oplus C_i(t)\right] P(t)}$$ [We have introduced the add command: P(t).]

Now we need to know how to get $C_{i+1}(t)$ as a function of the three parameters on the left. Apparently

$$C_{i+1} = (A_iB_iC_i + A_i'B_iC_i + A_iB_i'C_i + A_iB_iC_i')P$$

which can be factored:

$$\boxed{C_{i+1} = \left[B_iC_i + (B_i \oplus C_i)A_i\right] P}$$

or in unabbreviated form:

$$C_{i+1}(t) = \left[B_i(t)C_i(t) + (B_i(t) \oplus C_i(t))A_i(t)\right] P(t)$$

Note that because of the delay presumed inherent in the flip-flop C_{i+1} will depend on the original A(t), before complementing.

Further reduction: Since $B_i(t) \oplus C_i(t)$ appears in both equations:

$${}_{c}a_i(t) = \left[B_i(t) \oplus C_i(t)\right] P(t)$$

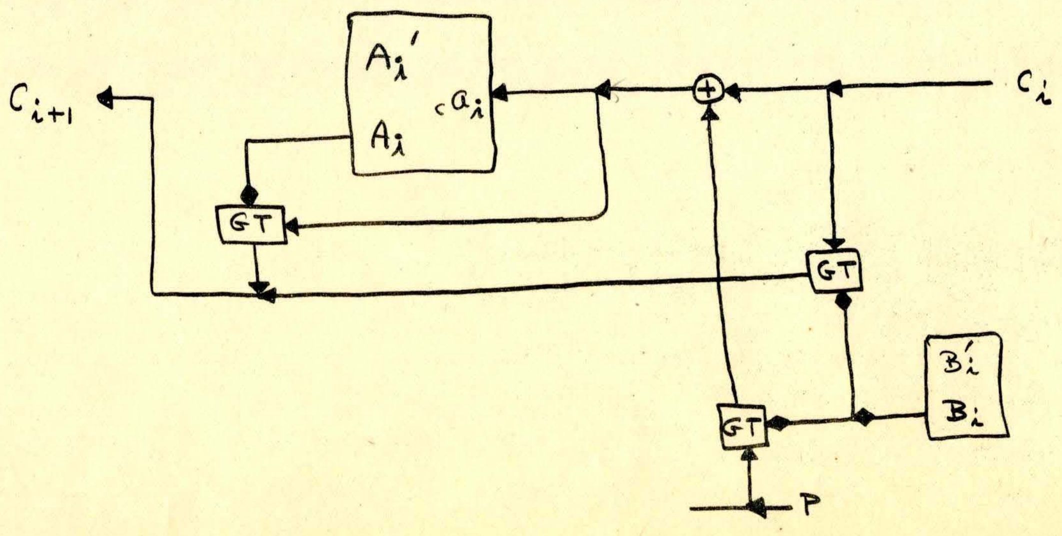
$$C_{i+1}(t) = B_i(t)C_i(t)P(t) + {}_{c}a_i(t)A_i(t)$$

Now $C_i(t)$ is a result of gating various inputs from previous stages with the command pulse, P(t). Therefore, we need not multiply it again by P(t):

$${}_{c}a_i(t) = B_i(t)P(t) \oplus C_i(t)$$

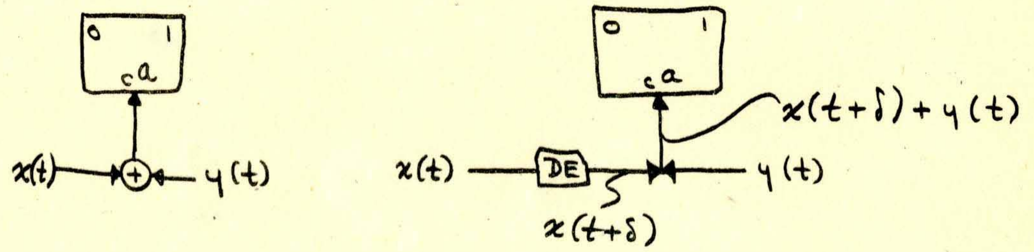
$$C_{i+1}(t) = B_i(t)C_i(t) + {}_{c}a_i(t)A_i(t)$$

Block Diagram



5.0 DELAY ELEMENTS

To illustrate our method of treating delay elements, consider the following device for realizing $x \oplus y$ on the trigger input to a flip-flop.



It has already been noted that the realization of $\oplus$ with ordinary electronic components requires two gates, two inverters (unless the complements of the inputs are also available) and a mixer; it is advantageous to trade for all this a delay element and a single mixer. The action of the second circuit is simple: if x and y both occur, the flip-flop is complemented twice, resulting in no change, whereas if one but not the other occurs it is complemented only once.

We can easily derive the equivalence of the two circuits in our formalism, but we have as yet no mechanical way of determining where it would be judicious to introduce delays. In this respect our treatment of delays parallels that of flip-flops: the introduction of both flip-flops and delays is a problem of planning. The design problem takes those elements as data together with their operation cycle, and asks for the most economical connecting network which meets the "boundary conditions". (The analogy between flip-flops and delay elements has as its theoretical basis the fact that any delay element can be represented as a flip-flop gated with a clock of appropriate frequency and phase.)

6.0 OTHER PROBLEMS AND APPLICATIONS

Magnetic Devices

Ceramic and ferromagnetic cores act as memory devices in such a way that there is generally no continuous signal output indicating that the core holds a '0' or a '1'. It has already been proposed that such devices be interpreted as flip-flops with built-in gates.

The main apparent difficulty in applying this algebra to magnetic devices is that from a 2 state core, three outputs are possible: a pulse of "+" polarity, a pulse of "-" polarity and no pulse. We wish, if possible, to avoid going over to a three-valued algebra for the analysis of these devices, first because of the complexity of such a formalism, and second because the cores are themselves devices which have only <u>two</u> states of magnetization in current applications.

It would be easy enough to resort to some such device as lumping together two of the three outputs from a core for the purposes of analysis; but it remains to be seen whether such a device will result in a theory which ignores important logical possibilities in circuits using magnetic cores. For further remarks, see Appendix III.

Probability in the Boolean Machine

When a computer is interpreted as a physical realization of a set of Boolean equations it becomes possible to apply probability theory in such a way that we obtain information about the density of information in critical registers. The application to input-output problems (buffer storage, etc.) is apparent. (See articles by Reed in Bibliography.)

Combinatorial Problems; Planning vs. Design

We have shown a method whereby, given the desired cycle, a logically optimum counter may be designed (relative to existing "packaged" realizations of logical functions). But this theory sheds no direct light on the problem: what is the most desirable cycle for a given application? Rather, we have suggested that the theory, in reducing the actual design to a routine process, leads the designer more quickly to that crucial question. <u>N</u> flip-flops are capable of 2^n different configurations, and there are 2^{2^n} different "counting" cycles which might be obtained. The optimum electronic realizations of these are not all of the same complexity; and often, in a particular application (say where arbitrary meanings are assigned to the various stages of the count) any one of a number of these cycles would be equally useful. The problem is then not: "What, for the given cycle, is the optimal realization?", but rather, "Comparing a number of usable cycles and their optimal realizations, which of these is optimum?" (which of a number of relative minima is least?)

We might call such decisions <u>combinatorial</u> rather than <u>logical</u>. The algebra, as developed here, provides no complete solutions to such problems. However, it appears that the problem may be soluble by further analysis. (One possibility is this: formulate a set of <u>fully mechanical</u> rules for deciding between two circuits on grounds of relative complexity in terms of available components; then program a computer to work out all optimal designs (relative minima) and decide between them as to the absolute minimum.)

In general, the broader questions of computer planning are illuminated but not solved by the present theory. It is entirely possible that further theoretical development, incorporating combinatory, statistical and information-theoretic elements with the present theory may lead to a mathematical treatment of the broader questions concerning the organization of computing machinery.

The present theory gives something like the following general picture of digital computers. Any particular analog computer may be regarded as a physical realization or analyzer of a set of differential equations. Similarly we may regard any digital computer, general purpose or otherwise, as a physical realization or analyzer of a set of Boolean difference equations.

The equations analyzed by a machine may be studied in a perfectly abstract way (this has not been done here) just as the machine may be studied as a physical entity. Then two points of view are possible:

(1) The Boolean difference equations describe the working of the machine.

(2) The machine realizes or analyzes the equations.

It is the validity of the second point of view which motivates the building of any machine.

History of this Theory and Relation to Other Theories

The English mathematician, George Boole, presented, in 1847, the first workable but cumbersome predecessor of the present sort of formalism. He was interested in its interpretation as an algebra of logic and of probability, and it was the application to logic which inspired the investigations of his successors, W. S. Jevons, C. S. Peirce, E. Schroeder and others in increasing the power and simplicity of the algebra.

One logical interpretation of the present algebra is this: let the variables (dropping the time arguments altogether) represent sentences such as "$3>2$", "$3>5$" and "Water boils at 100°C." The two values 0 and I are interpreted respectively as falsity and truth, so that $3>2 = I$ but $3>5 = 0$. "x'" is the sentence which is true when "x" is false, and vice versa: (the contradictory of "x") "it is not the case that x" or briefly "not x". Therefore, $(3>5)' = I$. "$x.y$" is the sentence "x and y", which is true only when both x and y are true: $(3>5).(3>2) = 0$. $x + y$ is x and/or y (briefly: x or y), which is true if x is true or y is true or both are true: $(3>5) + (3>2) = I$. Similar interpretations may be found for the other Boolean functions, and it will be seen that, on this interpretation, the theorems of the system are those laws of logic which apply to propositions and their combinations.

When we drop the assumption that the variables may have only the two values 0 and I there results a formalism which has as one of its interpretations a logic of classes containing the classical theory of syllogisms.

It was, as far as we know, Claude Shannon who, in 1938, first published an interpretation under which the algebra becomes a theory of relay and switching circuits. Shannon's interpretation led the way to an application of the algebra to static information-processing networks of all kinds, including those used in present-day electronic digital computers. However, Shannon's theory took no account of the dependence on time of the states of a computer. Therefore, while it led to an analysis of networks of gates, mixers, inverters and the rest, it did not permit an analysis of flip-flops. That theory was no substantial help in designing counters, adders and so on.

After Shannon, the principal development of algebraic methods in this field came from Burkhart, Kalin and Aiken of the Harvard Computation Laboratory. In the form in which it was published in 1951, their algebra (which shared with Shannon's a lack of adequate means for representing time variables) was an arithmetic of 0 and I.

Boolean Algebra (+ and . have the meanings used in this text)	Aiken's Formulation (+ and . have their ordinary arithmetical meanings)
x'	$1 - x$
xy	xy
$x + y$	$x + y - xy$

Superficially, the Aiken algebra is easier to use than Boolean algebra, since it is merely ordinary arithmetic restricted to 0 and I. However, for every new law which one must learn in order to use Boolean algebra, one must learn an arithmetical trick to use the Aiken algebra. Consider, for example, the transformation which in Boolean algebra is accomplished by De Morgan's theorem:

$$(xy)' = x' + y'$$

In Aiken's algebra it becomes necessary to delicately introduce 1's and parentheses in order to go from $1 - xy$ (our "$(xy)'$") to $(1-x) + (1-y) - (1-x)(1-y)$ (our "$x' + y'$"). Then Aiken's arithmetic is at least as hard to handle as Boolean algebra. Furthermore, it has the disadvantage that while an entire expression such as '$x + y - xy$' (our '$x + y$') is always either 0 or 1, yet such expressions often contain parts which are neither 0 nor 1, and hence meaningless. Thus if $x = y = 1$, $x + y - xy = 1$, but part of it, $x + y$, is 2, which is meaningless in the interpretation. We have examined this matter here in some detail in order to justify our use of a simple but unfamiliar formalism rather than a familiar but unexpectedly complicated one.

The present theory, with its use of time variables throughout, is the work of Irving S. Reed. To our knowledge it is the first analysis of computing machines powerful enough to provide a general method for synthesizing flip-flop circuits such as accumulators by straightforward calculation. Furthermore, by designing not in terms of the sequence of states of the flip-flops, but rather in terms of the changes in their states, we are led directly to a minimal design, without the use of such cumbersome devices as "minimization charts".

It should be stressed that the present report is an account of the most direct and easily used practical outcomes of the theory. For a rigorous mathematical account of the theory the reader is referred to the papers by Reed in the Bibliography. The introduction of time variables makes possible an extension of Boolean algebra into analysis. The theoretical background of the results presented here is an analogue of the calculus and theory of ordinary differential equations, based, not on ordinary arithmetic, but rather on Boolean algebra.

These methods were used, in a restricted form, in the design of the MADDIDA and CADAC computers, beginning in the winter of 1947. One of the results of the theoretical studies has been that the present method is applicable to pulse circuits in general, and not only to computers using a special sort of clock waveform.

SIGNED Irving S. Reed
Irving S. Reed

[illegible]
Richard C. Jeffrey

APPROVED [illegible]
Norman H. Taylor

ISR/RCJ/cp
Appendix I, Page 23
Appendix II, Page 27
Appendix III, Page 29

cc: H. R. J. Grosch
R. P. Mayer
I. Mann
H. K. Rising
R. C. Sims
G. R. Briggs
W. N. Papian
D. R. Brown
W. Ogden
K. H. Olsen
D. A. Buck
R. Pfaff
J. H. Hughes
C. J. Schultz
J. Jacobs
J. Ishihara
A. Katz
W. A. Hosier
F. E. Irish
J. A. O'Brien
A. M. Werlin

APPENDIX I

Summary

x	y	x'	$x + y$	xy	$x \oplus y$	$x \downarrow y$	$x \mid y$
1	1	0	1	1	0	0	0
0	1	1	1	0	1	0	1
1	0	0	1	0	1	0	1
0	0	1	0	0	0	1	1

There are 2^{2^n} distinct functions of <u>n</u> variables, some of which are repetitions of functions of $n - 1$, $n - 2$, ..., 1, 0 variables. E.g., x' appears above as a function of two variables (its value is defined for all 4 values of x, y; but since it is actually definable in terms of x alone, it is really a function of 1 variable). The two functions of 0 variables are 0 and I.

There are an infinite number of functions in terms of which all functions can be defined. The two such functions of 2 variables are $\downarrow$ and $\mid$. For example:

$$x \mid x = x'$$

$$(x \mid y) \mid (x \mid y) = xy$$

It follows that all other functions are definable in terms of $\mid$, since all functions are definable in terms of <u>not</u> and <u>and</u>:

$$x + y = (x'y')'$$

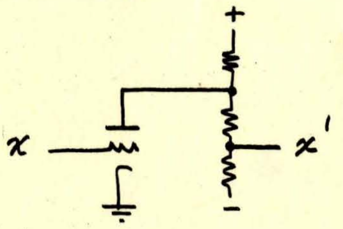
$$x \oplus y = xy' + x'y$$

etc.

<u>Physical Realizations of Functions</u>:

(The list is not complete) (In the magnetic circuits current in the indicated directions represents 1; <u>no current</u> represents 0; diodes might be necessary to prevent back flow and for clamping.)

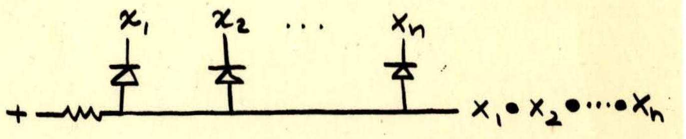
Complement



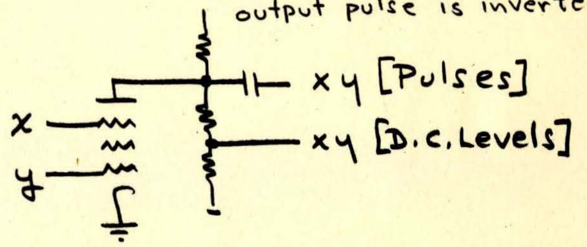
(Bias and voltage dividing network are such that when the grid goes positive, the x' point goes down to zero.)

Product

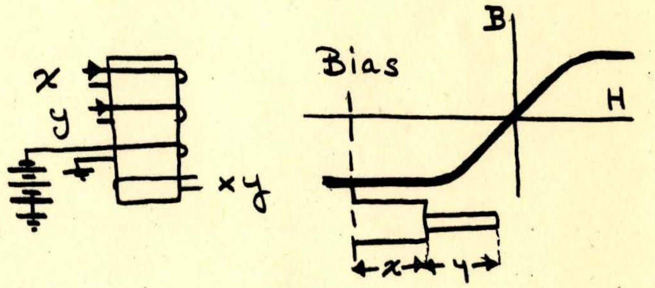
For n inputs:



For 2 inputs; gives amplification; output pulse is inverted.

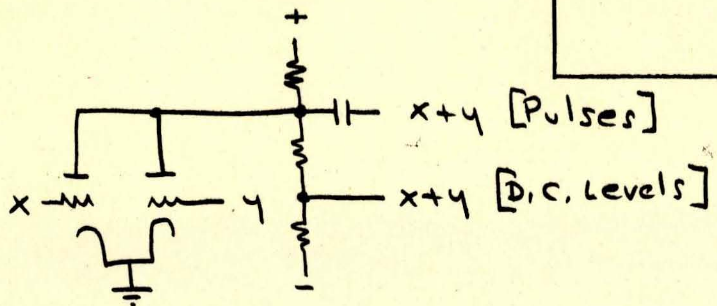


For n inputs (shown with n=2):

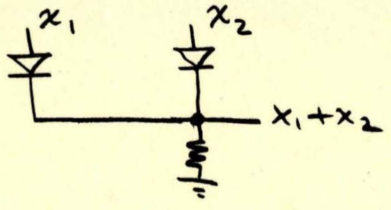


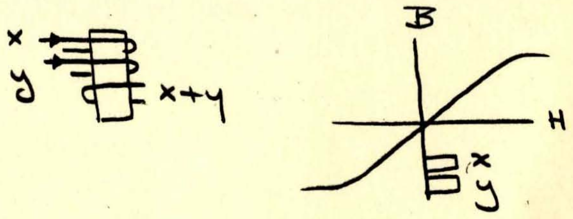
Sum

All circuits may be used with n inputs; shown here with n=2.



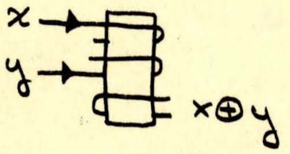
Provides Amplification; output pulse is inverted.

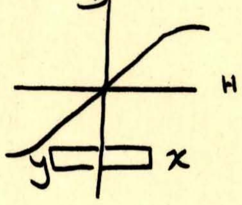


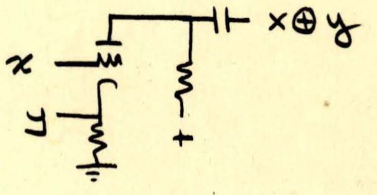


Partial Sum

If you can ignore the polarity of the output *and* depend on the coincidence in time, duration, amplitude and shape of x and y ----------



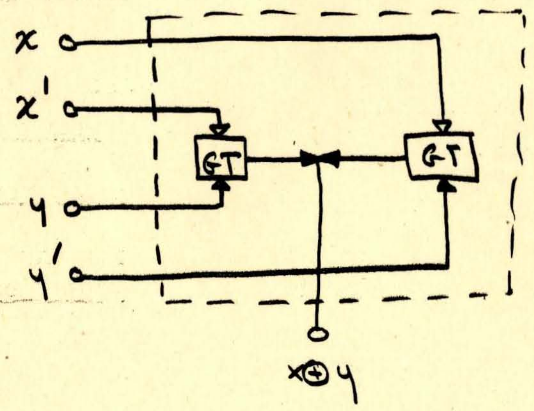




Otherwise

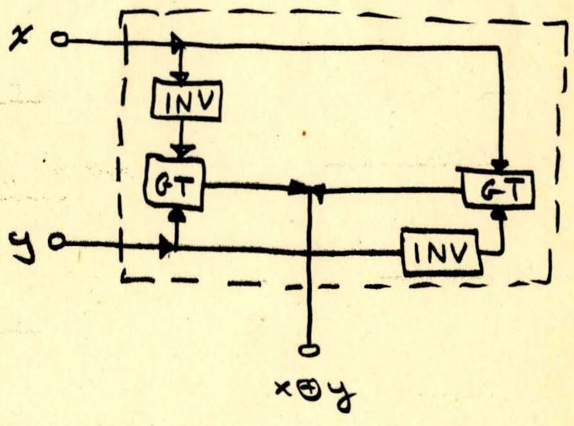
Black Box #1

(If the variables *and also their primes* are available.)



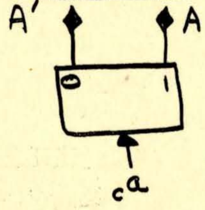
Black Box #2

(If the variables, *but not their primes*, are available.)

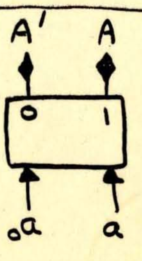


It follows from the statements on the previous page that, given realizations of *not* and *and*, black box realizations of all other functions can be constructed. And given realizations of $\downarrow$ *or* $|$, all needed black boxes can be constructed.

Flip-Flops



$A(t + \epsilon) = {}_{c}a(t) \oplus A(t)$



Triggers when both inputs are pulsed at once.

$A(t + \epsilon) = a(t)A'(t) + {}_{o}a'(t)A(t)$

(Reduces to the first case when $a = {}_{o}a$)

<u>Magnetic Memory Core</u>

${}_0a(t)$	$a(t)$	$A(t)$	$A(t+\in)$	$d_{+(A)}$	$d_{-(A)}$	$R(t)$
1	1	1	1	0	0	0
0	1	1	1	0	0	0
1	0	1	0	0	1	1
0	0	1	1	0	0	0
1	1	0	0	0	0	0
0	1	0	1	1	0	0
1	0	0	0	0	0	0
0	0	0	0	0	0	0

currents as shown represent 1. <u>No</u> current represents 0.

Value of A: $\uparrow = 1, \downarrow = 0$ $R = d_{-(A)}$

$$A(t+\in) = \left[{}_0a'(t) + a(t)\right] A(t) + \left[{}_0a'(t) \circ a(t)\right] A'(t)$$

$$R(t) = {}_0a(t)a'(t)A(t)$$

This can be set and cleared; read out by clearing (if the core held a 1 there will be a pulse out). However: it won't trigger as shown, and the readout is a single pulse, rather than a D.C. level.

APPENDIX II

Some Theorems of Boolean Algebra

Ignore order and grouping in pure sums and pure products.

"Multiply through" and factor as in ordinary algebra, and : "add through" a product.

$$a(b + c) = ab + ac$$

$$a + (b \circ c) = (a + b) \circ (a + c)$$

$$0 + x = x \qquad 0 \circ x = 0 \qquad x + x = x \qquad x + x' = 1$$

$$1 + x = 1 \qquad 1 \circ x = X \qquad x \circ x = X \qquad x \circ x' = 0$$

Expansion of a Function

$$f(x,y,z) = f(0,0,0)x'y'z' + f(1,0,0)xy'z' + \ldots\ldots + f(1,1,1)xyz$$

$$= \left[f(0,0,0) + x+y+z\right] \left[f(1,0,0) + x'+y+z\right] \bullet \ldots \bullet \left[f(1,1,1) + x'+y'+z'\right]$$

In particular, for the constant function $f(x,y,z) = I$ for all x,y,z, we get

$$I = x'y'z' + xy'z' + \ldots\ldots + xyz \quad \text{(all 8 terms are present)}$$

and for the constant $f(x,y,z) = 0$ we get

$$0 = (x+y+z) \bullet (x'+y+z) \bullet \ldots \bullet (x'+y'+z') \quad \text{(all 8 factors are present)}$$

De Morgan's Law: $(xy)' = x' + y'$; $(x+y)' = x'y'$

Elimination of a factor: $x + x'y = x + y$

Theorems relating to $\oplus$:

$$x \oplus (y \oplus z) = (x \oplus y) \oplus z = x \oplus y \oplus z$$

$$x \oplus y = y \oplus x$$

$$x \oplus y = xy' + x'y$$

$$x + y = x \oplus y \oplus xy$$

$(x \oplus y)' = x' \oplus y = x \oplus y'$

[$(x \oplus y)'$ is an interesting function: it is 1 exactly when x and y have the same value.]

$x(y \oplus z) = xy \oplus xz$

If $x \oplus y = z$, $x = y \oplus z$ (Permits solution of equations)

$x \oplus I = x'$

$x \oplus 0 = x$

$x \oplus x = 0$

For a more complete list of theorems, see works of Couturat and Whitehead listed in Bibliography.

Any expression can be put in the form:

$$Ax + Bx'$$

e.g., the equation for the FF with 2 inputs is in that form.

To complement such an expression it is sufficient to complement the "coefficients":

$(Ax + Bx')' = A'x + B'x'$

APPENDIX III

Core Analysis with 3 Valued Logic

${}_0a(t)$	$a(t)$	$A(t)$	$A(t+\epsilon)$	$R(t)$
-1	-1	-1	-1	0
0	-1	-1	-1	0
1	-1	-1	-1	0
-1	0	-1	1	1
0	0	-1	-1	0
1	0	-1	-1	0
-1	1	-1	1	1
0	1	-1	1	1
1	1	-1	-1	0
-1	-1	0	0	0
0	-1	0	-1	-1
1	-1	0	-1	-1
-1	0	0	1	1
0	0	0	0	0
1	0	0	-1	-1
-1	1	0	1	1
0	1	0	1	1
1	1	0	0	0
-1	-1	1	1	0
0	-1	1	-1	-1
1	-1	1	-1	-1
-1	0	1	1	0
0	0	1	1	0
1	0	1	-1	-1
-1	1	1	1	0
0	1	1	1	0
1	1	1	1	0

$a(t) = \pm 1$ has same effect as ${}_0a(t) = \mp 1$

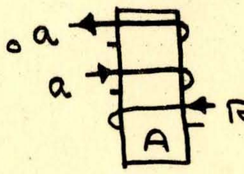
R is wound so that when the state of the core is moving toward 1, R = 1, i.e.,

$$\boxed{R(t) = 1} \Leftrightarrow \boxed{A(t) < A(t+\epsilon)}$$

$$\boxed{R(t) = 0} \Leftrightarrow \boxed{A(t) = A(t+\epsilon)}$$

$$\boxed{R(t) = -1} \Leftrightarrow \boxed{A(t) > A(t+\epsilon)}$$

"$\Leftrightarrow$" means <u>if and only if</u>



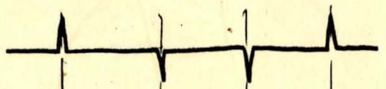
State of the core:
-1 = ↓ ; +1 = ↑ ; 0 = no magnetization.
Currents in the windings are positive as shown by arrows. Opposite current: -1. <u>No</u> I: 0.

The rules of 3 valued algebra are more cumbersome than those of the present theory.

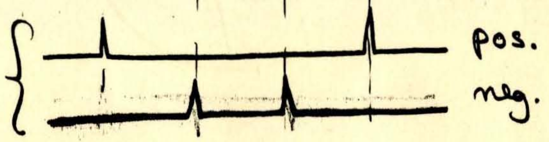
Note that cores require a 3 valued analysis only to the same extent that vacuum tube circuits do. In vacuum tube circuits, too, there are <u>three</u> possible inputs and outputs: positive, negative and zero pulses. In trying to account for such circuits in terms of 0 and 1 alone, we are somewhat farther from reality than when we use -1, 0 and 1. But the three valued analysis is itself a high-order abstraction from the real situation with its continuum of infinitely many values.

The point is that we pay for the additional simplicity of each higher order of abstraction in faithfulness of the resulting black-and-white picture of the real situation.

We believe that a 3 valued analysis would be of use, but the use would be a better evaluation of the limitations of the two valued approach. There may exist combinations of elements whose utility depends on the polarities of the pulses involved. Such designs could be "cranked out" of a 3 valued analysis. But it may be that, having recognized them, they can be introduced into the two valued analysis by special devices. One such device is translating a single-pulse-train:



into two:



BIBLIOGRAPHY

Whitehead, A. N., A Treatise on Universal Algebra, Cambridge, 1898, (Esp. Vol. I, Book III)

Couturat, Louis, The Algebra of Logic, (Eng. Trans.), Chicago, 1914

Lewis, C. I. and Langford, C. H., Symbolic Logic, New York, 1932

Shannon, Claude E., A Symbolic Analysis of Relay and Switching Circuits, Trans. Am. Inst. of Electrical Engineers, Sept. 1938

Aiken et al., Synthesis of Electronic Computing and Control Circuits, Cambridge, 1951

Keister, Ritchie and Washburn, The Design of Switching Circuits, New York, 1951

Reed, Irving S., Some Mathematical Remarks on the Boolean Machine, Project Lincoln Reports, Parts I, II. See also notes contained in Project Lincoln Progress Reports, Division II.
Div. 2 of Pro. Lincoln

Quine, W. V., Methods of Logic, New York, 1951

Tarski, Alfred, Introduction to Logic, New York, 1946