PRACTICAL CIRCUITS USING CONSERVATIVE REVERSIBLE LOGIC

by

Andrew Lewis Ressler

Submitted in Partial Fulfillment
of the Requirements for the
Degree of Bachelor of Science
at the
Massachusetts Institute of Technology
May, 1979

Signature of Author.....Andrew L. Ressler..... May 24, 1979.....
Department of Electrical Engineering, Date

Certified by.....Edward Fredkin.....
Thesis Supervisor

Accepted by.....
Chairman, Departmental Committee on Theses

PRACTICAL CIRCUITS USING CONSERVATIVE REVERSIBLE LOGIC

by

Andrew Lewis Ressler

Thesis Supervisor

Edward Fredkin

Professor of Computer Science and Electrical Engineering

Designing logic circuits with Conservative Logic has generally followed the same patterns as with normal logic. The nature of Conservative Logic can allow one to adapt a different way of representing a circuit which aids in the design of the circuit. Since Conservative Logic only has one kind of logic gate, the new symbology takes advantage of this by representing the gate as a vertical line connecting the inputs. This allows greater compacting of a circuit while increasing the clarity of the internal functions. The first part of this thesis explains the new symbology.

The second part of the thesis is the design of a reversible accumulator and counter, using the new symbology.

TABLE OF CONTENTS

REVERSIBLE COMPUTATIONS

Designing computers out of reversible logic is a fairly recent idea. For years, it was considered to be impossible because computation was thought of as an irreversible process.

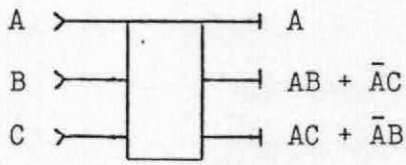
A computation is reversible if, and only if, given the outputs of the computation and the semantics of the computation, the inputs can be determined. This implies a bijective mapping of the inputs to the outputs.

Any computation with a given m input bits and n output bits can be forced to be a reversible computation by adding at most n additional bits to the inputs and m additional bits to the outputs. A trivial method of doing this would be to make the m additional output bits copies of the m inputs. The additional n inputs are necessary for the number of inputs to equal the number of outputs. Now there will be a bijective mapping.

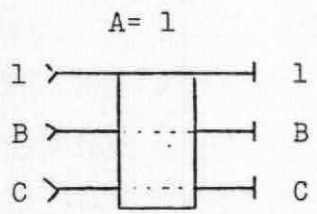
Computations seem to have varying degrees of irreversibilities. An extreme of irreversibility would be a function where all the inputs mapped to the same output state. This would require the maximum number of additional inputs and outputs to make it reversible. A nearly reversible computation would be a divide by 2, using only integers. If truncation were used, any even number would map to the same output as the next odd number. To determine the input from the output, it would only need additional information as to whether the input had been an odd or even number. This could be represented by only one additional output bit regardless of the number of input bits. Many of the useful computations encountered range between these two degrees of irreversibility. Since unnecessary bits are useless to have, any function should be designed to have the minimum additional bits necessary.

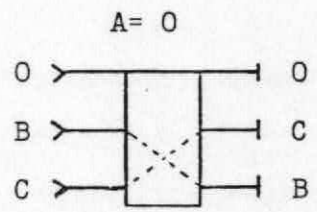
CONSERVATIVE LOGIC

Conservative Logic is a model for expressing reversible computations. It consists of two basic elements; the selector gate and the wire. The selector gate is a 3 input, 3 output gate symbolized as shown:



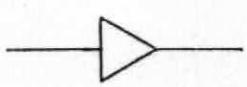
It is shown with the inputs on the left and the outputs on the right as Boolean expressions of the inputs. A better visual conception of its function is as shown for two values of A.





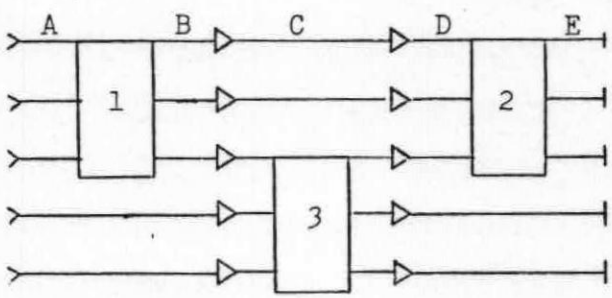
Any inputs are denoted as a little arrow and the outputs as a straight line.

The wire is used to connect gates together. All the inputs or outputs of a gate that are not inputs or outputs of the whole circuit must be connected to a wire. All wires must be connected to a gate or to another wire. The representation of a wire is:



The functions of the wire are to communicate information and also represent time delays in the circuit. This can be visualized by thinking of the

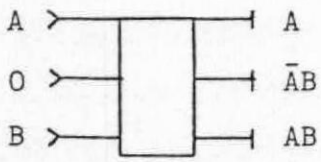
triangle as a D flip flop. All the wires of a system are connected to a central clock and are triggered synchronously. The D flip flop acts as the memory. The physical location of the wire determines the communication channel. The time delay of information communicated through one wire will be the inverse of the frequency of the clock. Also, the gates are to be considered as no time delay. An example of the above concepts are in the following figure.

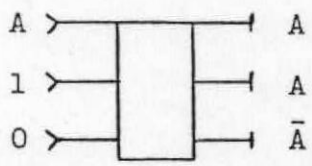


At a given time step, consider that the signal at A is the same as the signal at B, and likewise with D and E, but B, C, and D are all different signals. In the next time step D and E will assume the value of C, and C will assume the value of A and B.

The wire with two triangles is actually two wires that are joined. Two are necessary to keep the signals synchronized with the information in the third line. Also, the bottom two lines have an input connected to a wire. This is to synchronize the inputs to gate 3 because one of the inputs is delayed by the wire from 1 to 3.

An interesting logic model would have to be universal. This is generally considered true if the model has an "and gate" and a "negate" function. Conservative Logic has these, as can be seen from the following examples.





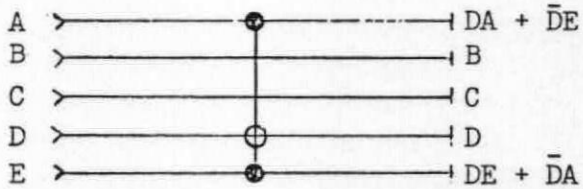
The model gets the name "Conservative" because it also has the property of conserving the ratio of different binary states from the inputs to the outputs. In other words, the outputs will have the same number of "1's" as the inputs.

This model also deals with the method of fanout. Since only one wire may be attached to a given output of a gate, it would seem to prevent fanout. The example of the "negate" also makes a copy of the input, so this can be used for fanout. This gives a precise way of notating fanout. It is also necessary because the traditional method of fanout, with just a branching of wires, would run into problems running in reverse, if two different signals ran into the same branch.

THE NEW SYMBOLOGY FOR CONSERVATIVE LOGIC

The new symbology has several advantages over the old symbology. The most obvious advantage is in the geometrical layout of the circuit in two dimensions. The advantages will become clear after a discussion of the symbology.

Since Conservative Logic has only one kind of gate, it is not necessary to use a two dimensional figure to distinguish between gates, as is necessary in conventional logic. The gate now becomes just a straight line intersecting the three lines it needs for inputs. Since the inputs of the gate can be thought of as a controller and two lines to be controlled, the inputs fall into two classes. To represent the different classes will be an open circle at the intersection of the controller and a closed circle at the intersection of other inputs. Since the two inputs are distinguished by circles at the intersections, it naturally leads to the ability to cross a wire but assume there is no connection if there is no circle at the intersection. The new symbol is as shown:



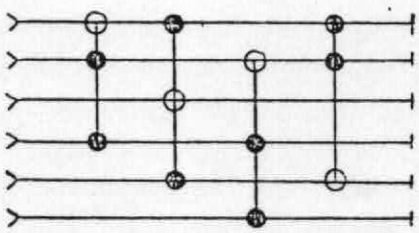
The B and C were not affected and the D was passed through since it was the controller. The A and the E were affected just as would be expected.

Since a gate can be extended over a wire, it is never necessary to move a wire to a gate, just move the gate to the wire. This makes it possible to have all the wires stay as straight lines all the way through the circuit.

The procedure for using the new symbology is as follows:

1. Given n inputs, draw n wires from the beginning of the circuit to the end.
2. Overlay the necessary gates on the wires.

Example:



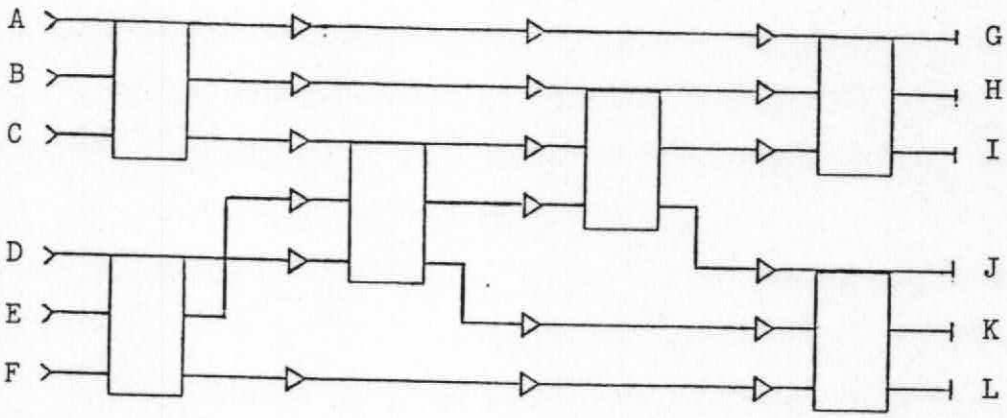
Notice the delays have been left out. Without delays, it is easy to see the dependencies of the gates on their inputs. In other words, the inputs to a gate must be calculated before the gate needs them.

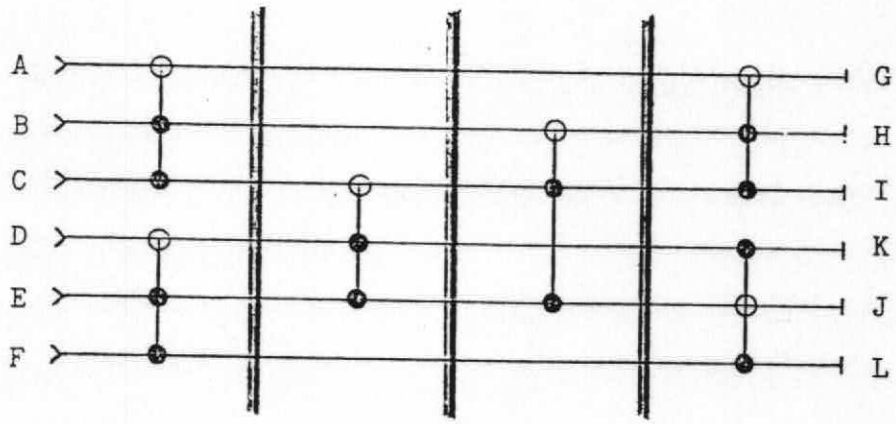
An easy way to put in the time dependencies is as follows. Put as many of the gates to the left as possible with this one restriction; no more than one gate can intersect a line in this group. Now draw a vertical line through the whole circuit directly to the right of the group. Wherever this line intersects the signal lines, that will be interpreted as a time delay. This is the equivalent of the triangle on the wire. Now repeat the process with the remaining gates. This is shown in Figure 1.

This method will assure that all the signals arrive at the gates in a synchronous fashion and that the outputs are synchronous. This also assures that the proper number of wires will be inserted between all the gates and this will also compute the results in the shortest time for the given configuration of gates.

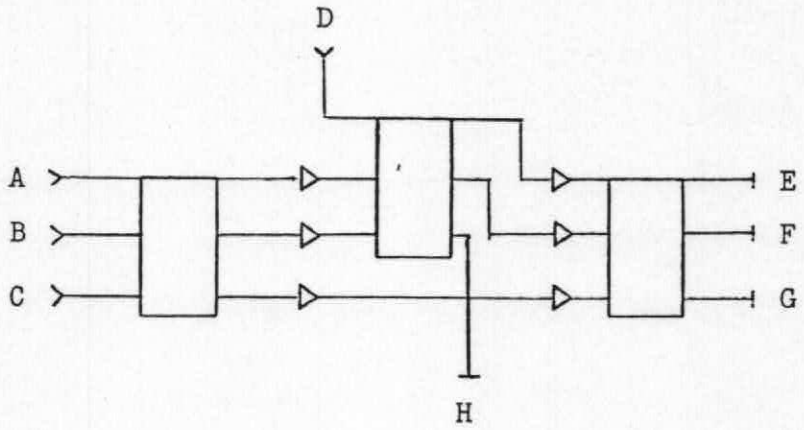
So far, this model will only work for inputs that arrive at the same time. Any inputs that come at a later time can come in straight to a gate.

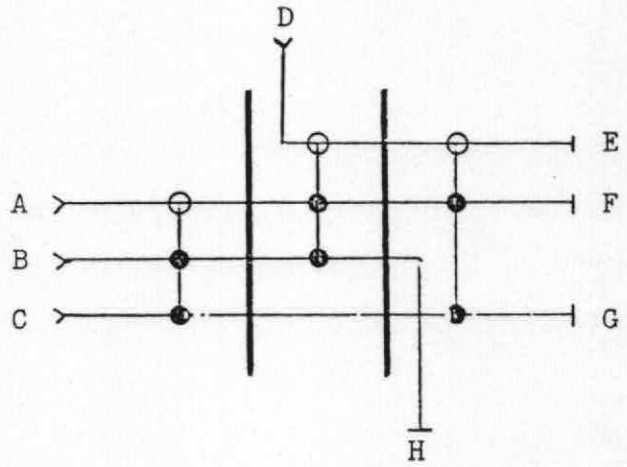
Figure 1. Time Delays





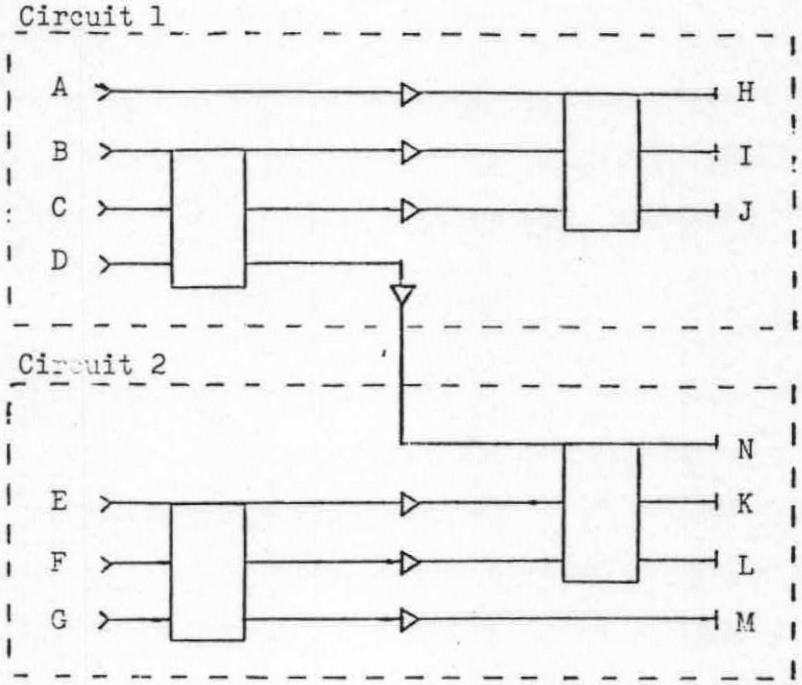
Any output that leaves before the end of the circuit will pass through a time delay before it leaves. For uniformity, show all inputs coming in from the top and outputs coming out from the bottom.

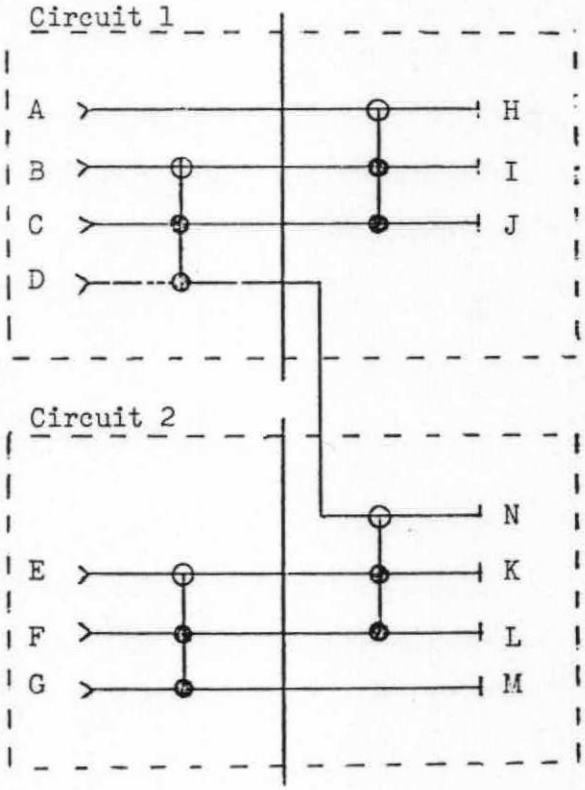




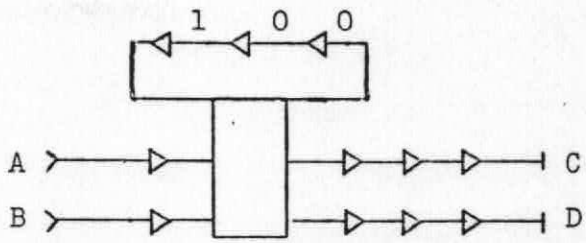
This method makes it fairly easy to stack circuits as in Figure 2 on the following page.

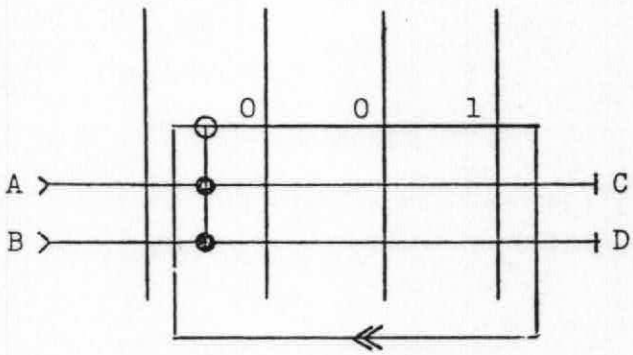
Figure 2. Interconnections





The one necessary exception to the inputs entering from the top will be a feedback loop. Consider the following example of a feedback loop:





The time delays of the loop are all shown to the right of the gate and the path going backwards has no time delays.

ADVANTAGES OF THE NEW SYMBOLOGY

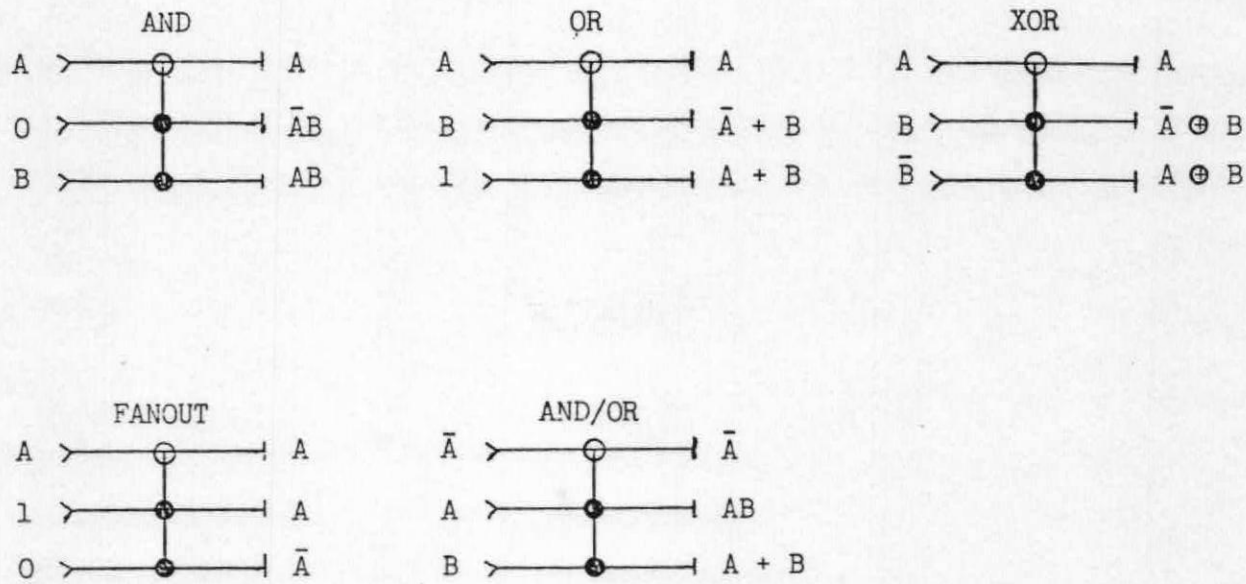
The advantage of the layout should be fairly ovbious now. Since it is now easier to layout the circuit, this also allows more of the time in design to be spent on the actual logic functions. This layout also helps to see two types of dependencies of the gates. A gate is always dependent on any gates that must compute a value that will affect one of the inputs to the original gate. Also, if two gates need the same output of some gate, there is now a time dependency. If both gates need it for an input other than a controller, a copy must be made before either can use it. If one of the gates is using it for a controller and it is important for this gate to be calculated sooner at the expense of the other gate, then it should use the value and then pass it to the next gate. Since this helps to see these dependencies, then it helps when changing the circuit because immediately, all things affected by the change can be predicted.

The regular structure of the circuit also hints at a possible implementation as an integrated circuit almost directly from the drawing.

The use of a simple structure for the gate and the regular structure of the entire circuit also hints nicely at a convenient representation on a CRT for uses of debugging. It might be possible to implement it such that whenever a gate was added, the computer could instantly add the affects in Boolean Algebra form on the screen.

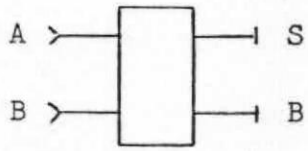
In using selector gates it is often useful to think of the gate in terms of specific sets of inputs, which will result in a close similarity with normal logic as in Figure 3.

Figure 3. Specific Gates



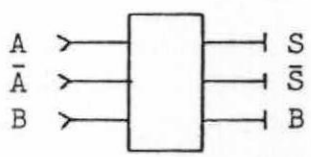
THE ACCUMULATOR

A block diagram for an accumulator is as follows, where S=A+B:



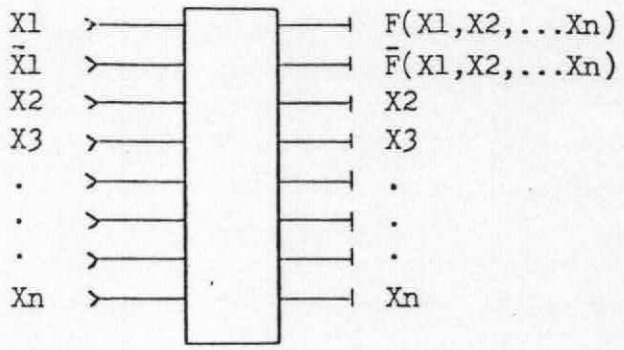
Since the input B is a function of the outputs where B=S-A, all the inputs can be determined from the outputs, therefore, this figure represents a reversible circuit. To be designed out of the selector gate, it must also conserve the number of "1" inputs. As it stands, this figure does not conserve bits. Consider the case where A=1111 and B=0001. There will be five "1's" input to the circuit. The sum of A and B will be S=0000 mod 32. The outputs will only include one "1".

To correct this problem, include as an input the negation of A and as an output the negation of S. Using the above example, there will now be five "1's" in the inputs and five "1's" in the outputs. The resulting block diagram is:

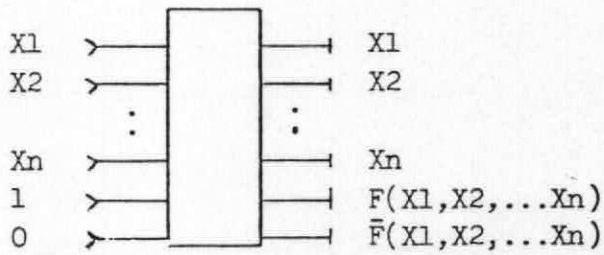


It is not obvious how to design this circuit with a selector logic gate. One method is based on the proof of the following theorem:

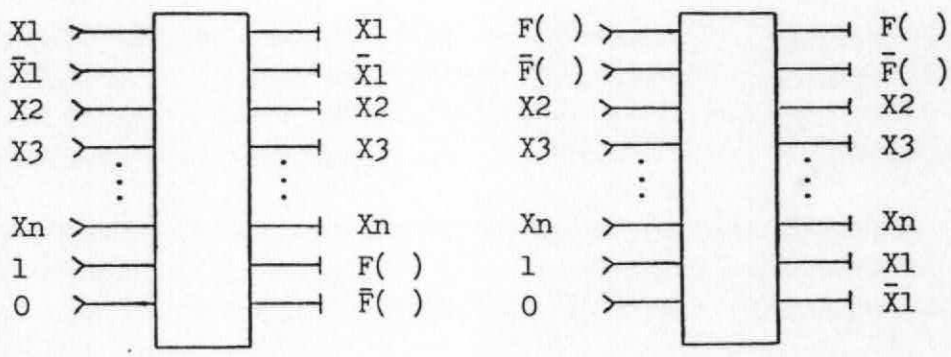
Theorem 1: For any combinatorial function F(X1,X2,...Xn), where it is true that X1 can be expressed as a function of F(X1,X2,...Xn) and the variables X2,X3,...Xn, the following circuit can be constructed.



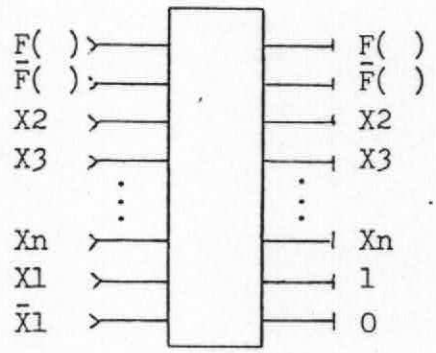
Proof: The first step of the proof will require Theorem #3 from Bill Silver's paper "Conservative Logic" which states, "The following circuit can be constructed for any Boolean function F".



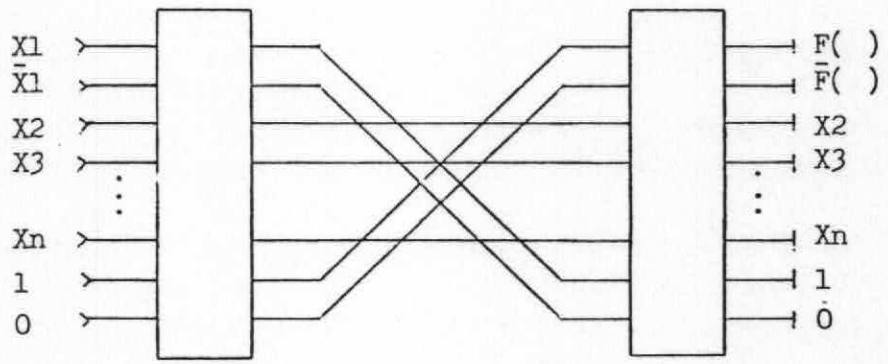
Using this theorem, the following two circuits may be constructed:



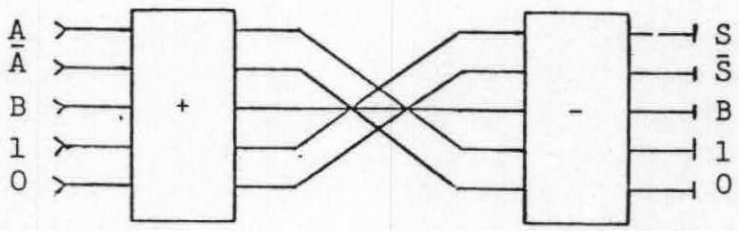
Since the selector gate is reversible, and only combinatorial circuits are being considered, the mirror inverse of the second circuit will result in the following:



Using the above circuits it is now possible to construct the final configuration as follows:



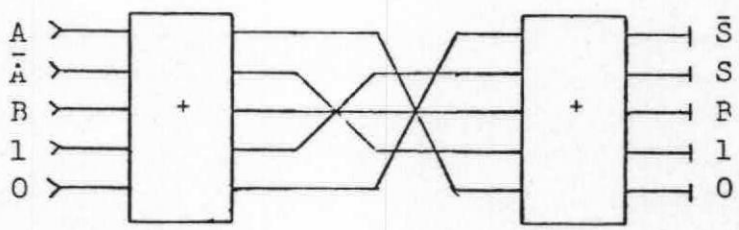
Applying this construction to the accumulator results in the following configuration:



The first block will be an adder. The second block will be a subtractor, where A=S-B. This subtractor can be implemented with an adder. Using two's complement notation, the subtraction S-B can be performed by negating the sum of B and the negation of S.

$$S-B= \overline{\bar{S} + B}$$

Using two back to back adders the circuit will be as follows:



The design is now simplified to the design of an adder. The adder can be divided into sub-modules for the sum of each bit.

A_i = i-th bit of A.
B_i = i-th bit of B.
C_i = carry into sum of i-th bits.
C_{i+1} = carry out of i-th bits.
S_i = sum of A_i, B_i, C_i.

The Boolean algebra functions for C_{i+1} and S_i can be calculated from their truth table.

A_i	B_i	C_i	S_i	C_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

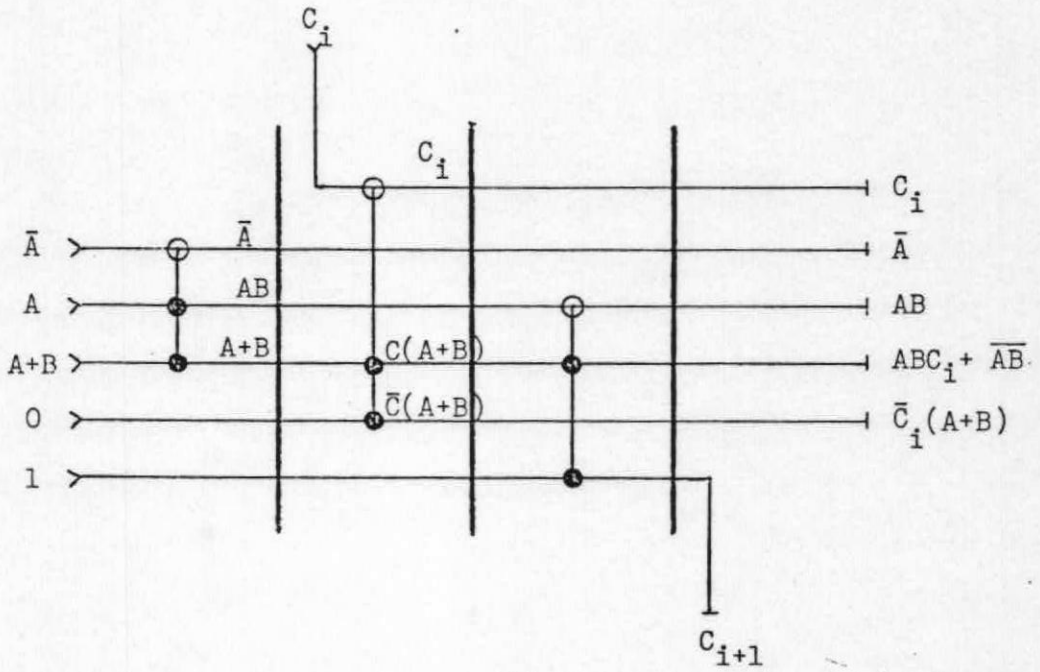
$$S_i = A_i \oplus B_i \oplus C_i$$

$$C_{i+1} = A_iB_i + A_iC_i + B_iC_i$$

The final design of the adder has been optimized for speed. The constraint on an adder for time is the delay due to propagation of the carrys. Therefore, the time necessary for the carry to be propagated through one bit must be a minimal value. The optimal delay would be one time delay to propagate one bit.

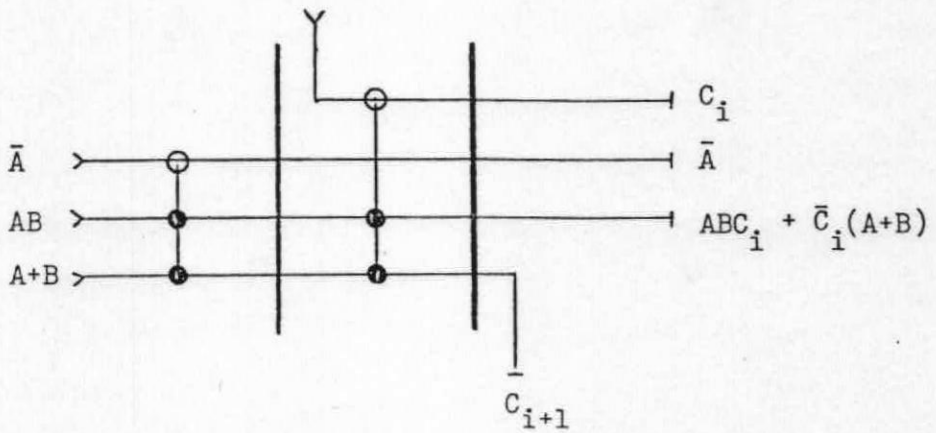
Several iterations of the design of the adder resulted in acheiving a much more optimal circuit.

On the first attempt, it became obvious that designing straight from the Boolean Algebra doesn't always lead to an optimal circuit. To calculate C_{i+1} requires the following circuit if the straight forward implementation is used.

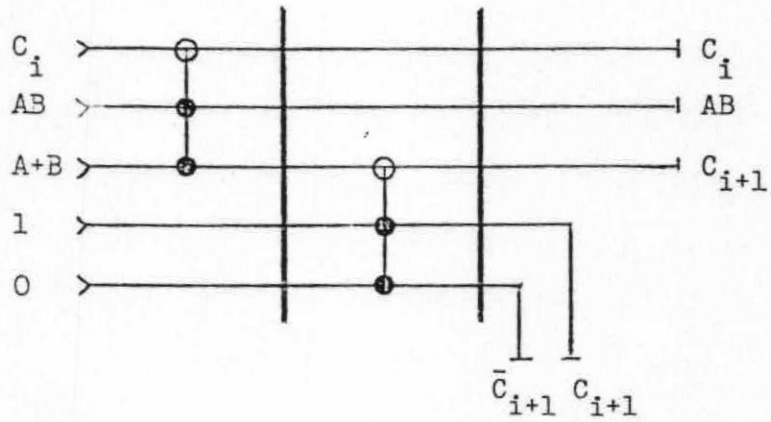


If the expression is rearranged to an equivalent algebra expression, it becomes obvious how a quicker implementation can be found. By quick, it only requires the one delay hoped for.

$$C_{i+1} = A_iB_i\bar{C}_i + (A_i + B_i)C_i$$

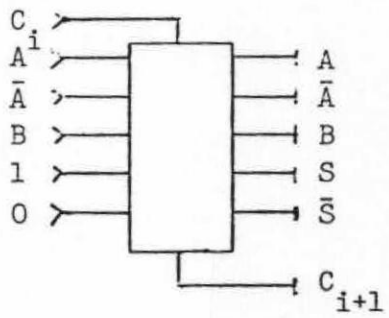


The first approach assumed a need for the carry to propagate with its complement. This was done with a "Fanout" gate after the carry had been calculated as shown.

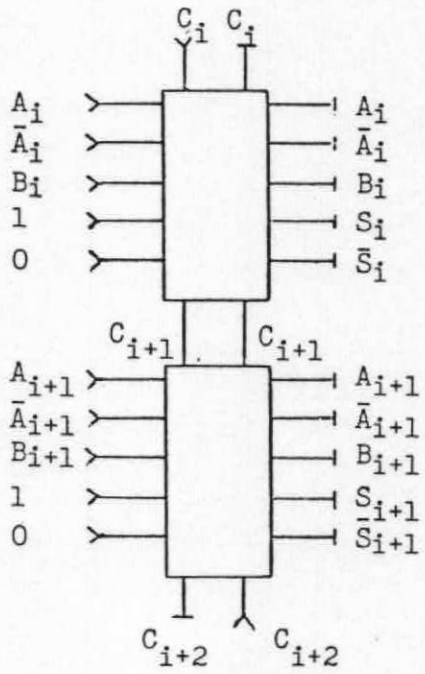


Unfortunately this results in two delays per bit for the carry to propagate. Two methods are discussed for solving this problem.

The first method was to only propagate the carry. Since the complement is not there the inputs and the outputs must be checked to be bit conserving.



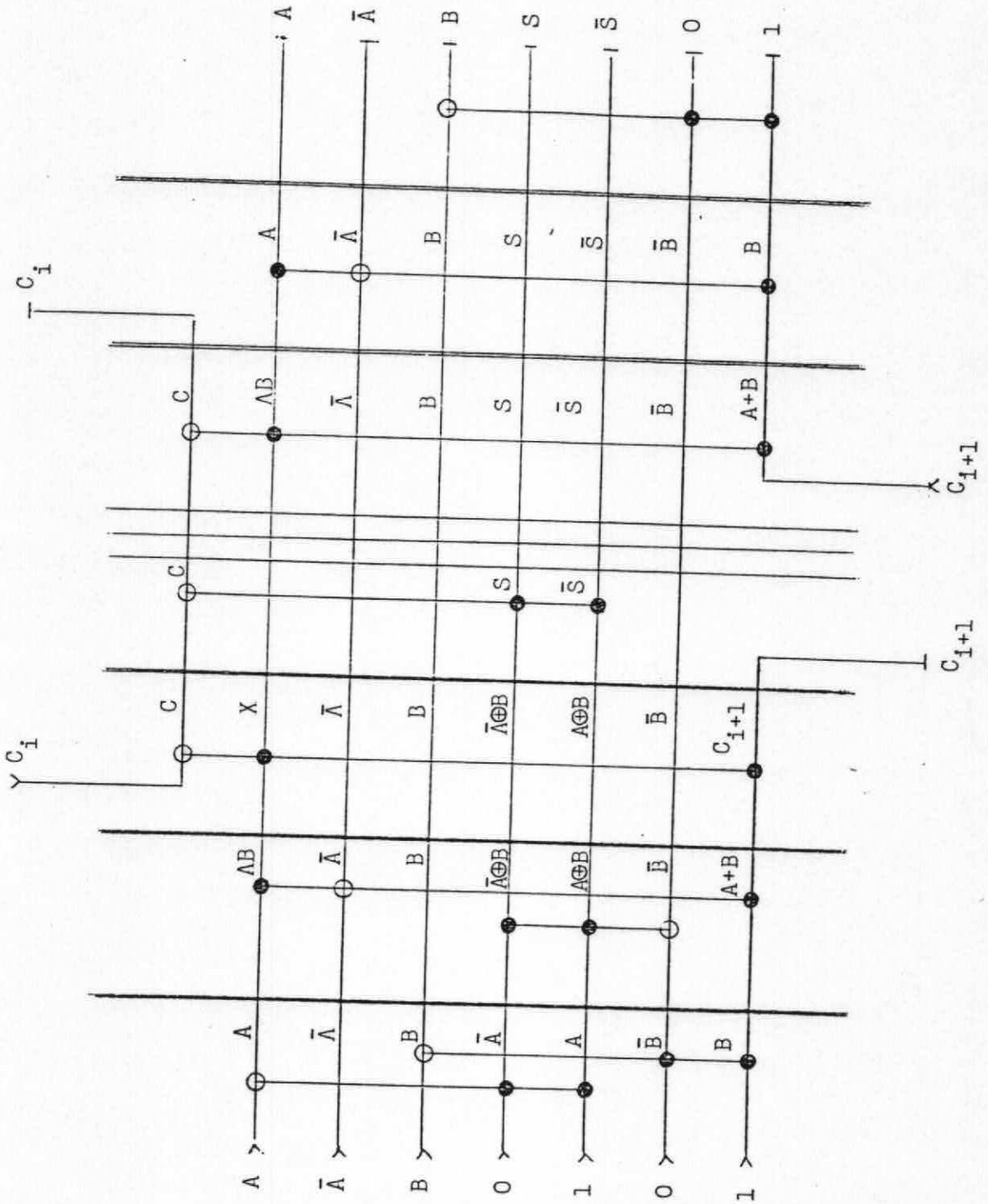
Except for the carry in and the carry out all the other inputs balance with the outputs for the total number of ones. The carry in doesn't balance with the carry out. An easy way to conserve the bits would be to output the carry in and to also input the carry out. This is accomplished by outputting the carry in back to the previous bit adder after it is no longer needed. This will also accomplish the necessary input of the carry out as shown.



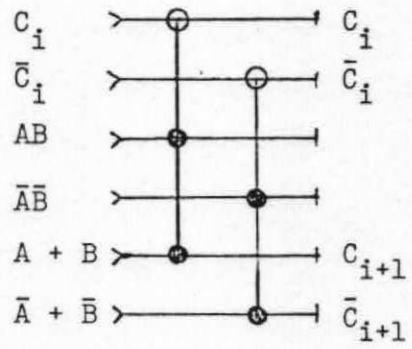
When the carry out comes back to the circuit it feeds into a mirror image of the gates used to calculate it. This will eliminate it. The carry in is used to calculate the carry out and then to eliminate it. For the propagation of the carry back up the circuit to be only one delay the carry in must be output as soon as the carry out is eliminated. In between these uses the carry in will be used to calculate the sum bit. This is easy since it only requires the carry in to control one gate. All these points are shown in Fig. 4

The fatal drawback of this design is the total time delay required for the circuit to finish. The propagating of the carry back up must wait until the carry propagates all the way down. Thus there is still a delay of two per bit. An interesting point about the delays is that the sum is finished being calculated after the carry has propagated down. So there exists a possibility for the sum to be used after only one delay per bit. The additional delays are only necessary to clean up the garbage.

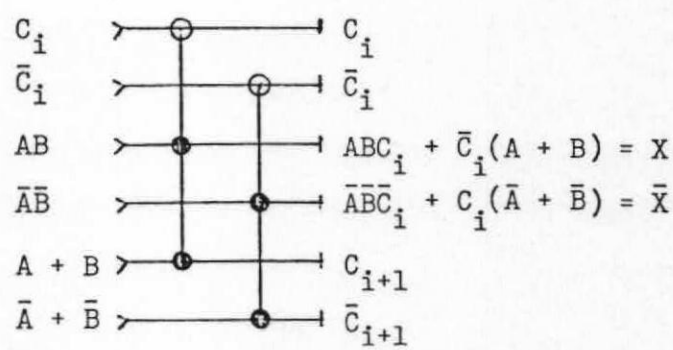
Figure 4. Trial Adder



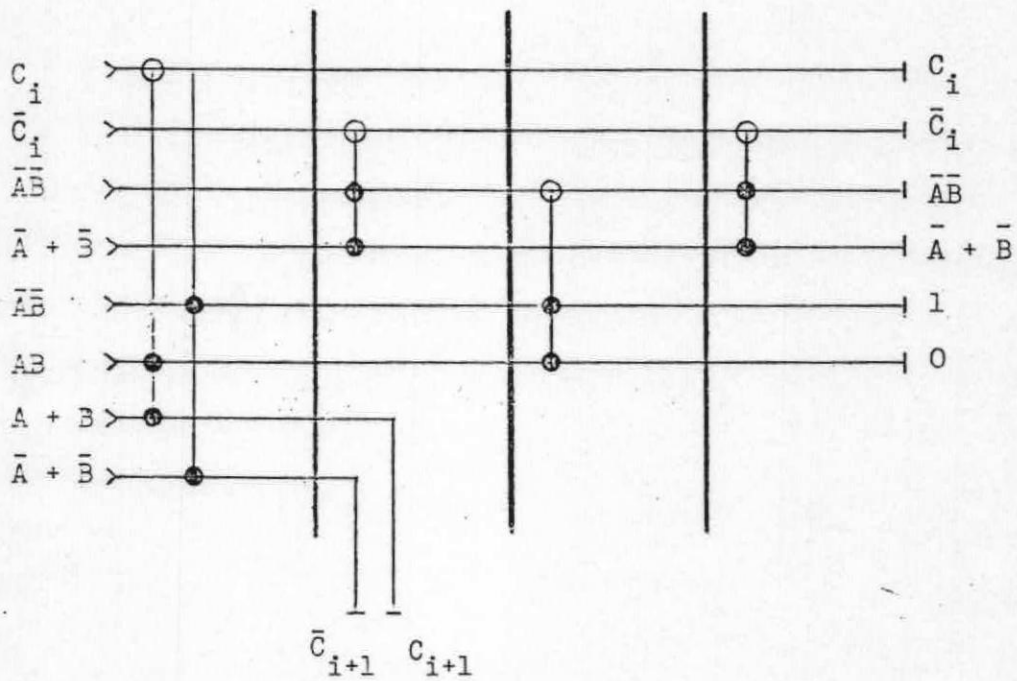
The second method is to calculate the complement of the carry out independent of the carry in using the complement of the carry in. This way the carry out and its complement will be calculated in parallel. The complement of the carry out must also be calculated in one time delay. This can be done as shown:



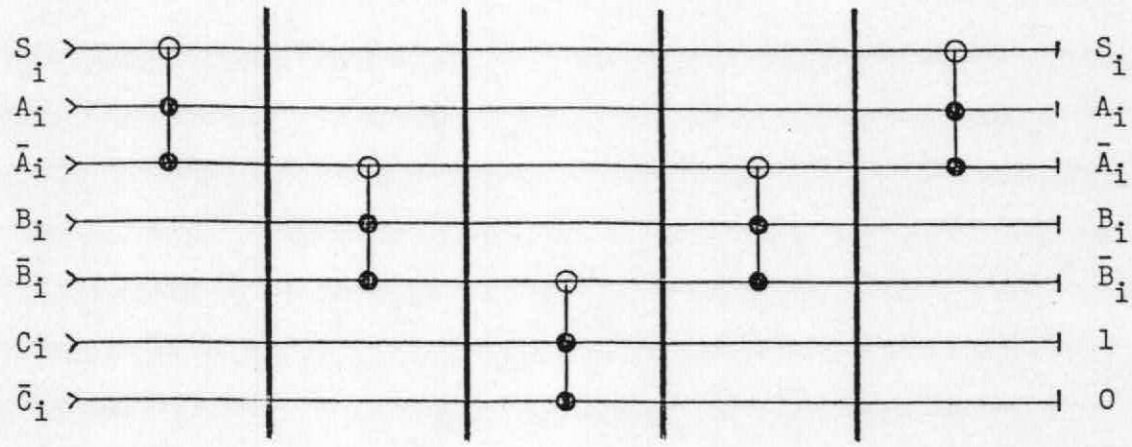
Since the carry's are accompanied by their complements the bits will be conserved. Therefore there will be no need to input or output anything else. Now the problem lies in eliminating the garbage and the carry in. Since the carry out doesn't come back it is not possible to use a simple mirror image as before. The same trick that is used to eliminate the A's in the whole circuit can be used to get rid of the garbage. The first garbage to destroy is the garbage generated by the gates calculating the carry outs. The middle outputs of these gates are the garbage and one is the inverse of the other as shown.



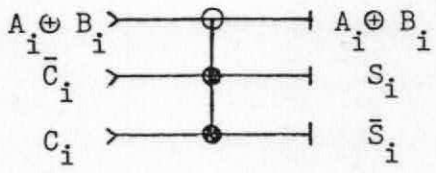
The trick is to generate the garbage again by calculating the carry out again. Now the three garbage outputs can be fed into an inverted fanout. This eliminates two of the garbage bits as shown. Now the last bit of garbage can be fed into a mirror of how it was created which will eliminate it.



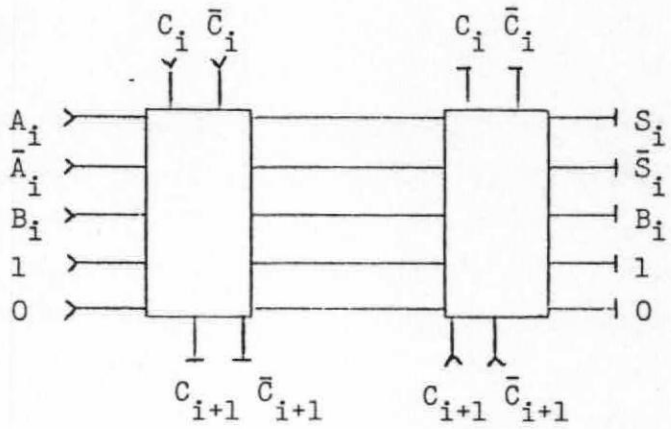
Now the carry in must be eliminated. Before it can be eliminated it must be used to calculate the sum bit. This can be done the same as before. Now the carry in could be eliminated by solving for it with the sum bit and A and B. Then use an inverted fanout as before. This would result in the following:



The unusual function required for the sum allows a savings on the method just described. Since the concern is to eliminate the carry in and its complement and calculate the sum bit, if the XOR of A and B is used to control the carry in and its complement, that would calculate the sum and get rid of the carry in at the same time.



The final configuration of the one-bit adder is Figure 5. To complete the circuit, Figure 6 shows the inverse of the adder necessary to eliminate the input A. The inverse adder's inputs have been permutated so that it can be connected directly to the first adder as follows, where 5 means from Figure 5 and 6 means from Figure 6.



Between the two adders must be a fixed time delay corresponding to the delay of the carry propagating down the number and back up to that point. To clarify why these delays are needed, Figure 7 shows the connections for a 4-bit accumulator. In Figure 7 the time lines are not meant to show any delay between the stages. They are to show the delays necessary between the adders at any particular bit.

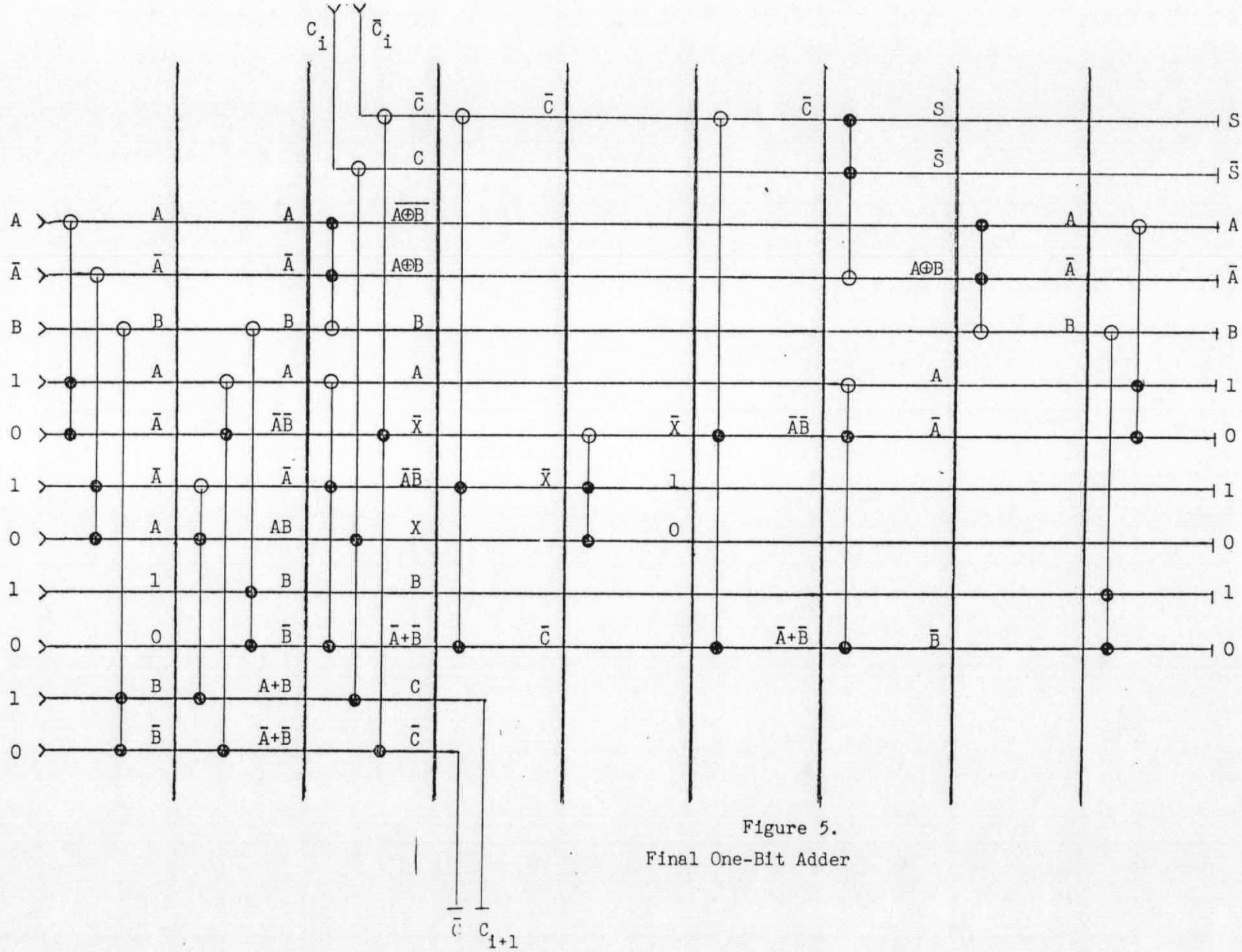


Figure 5.

Final One-Bit Adder

Figure 6. Final Inverse One-Bit Adder

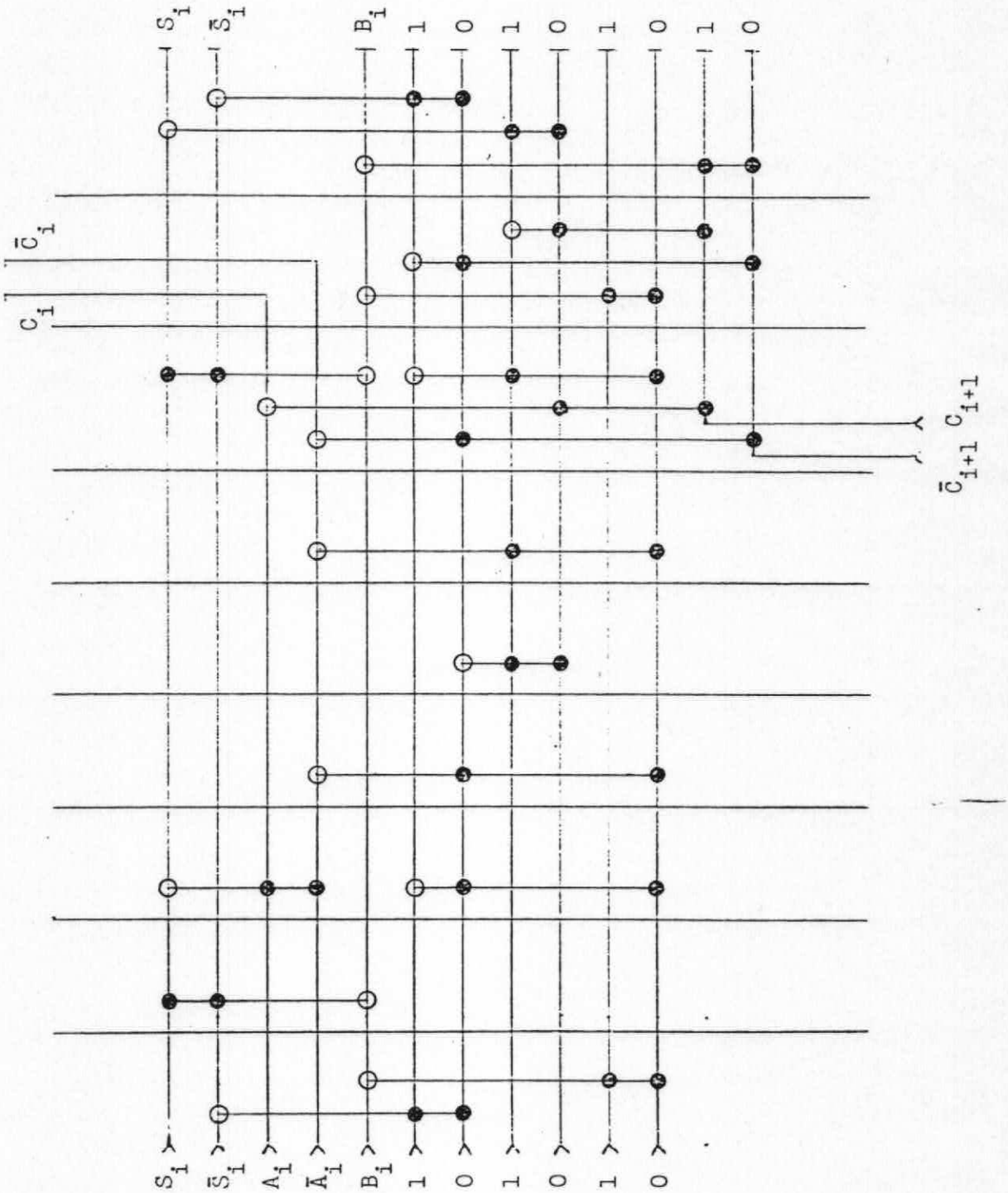
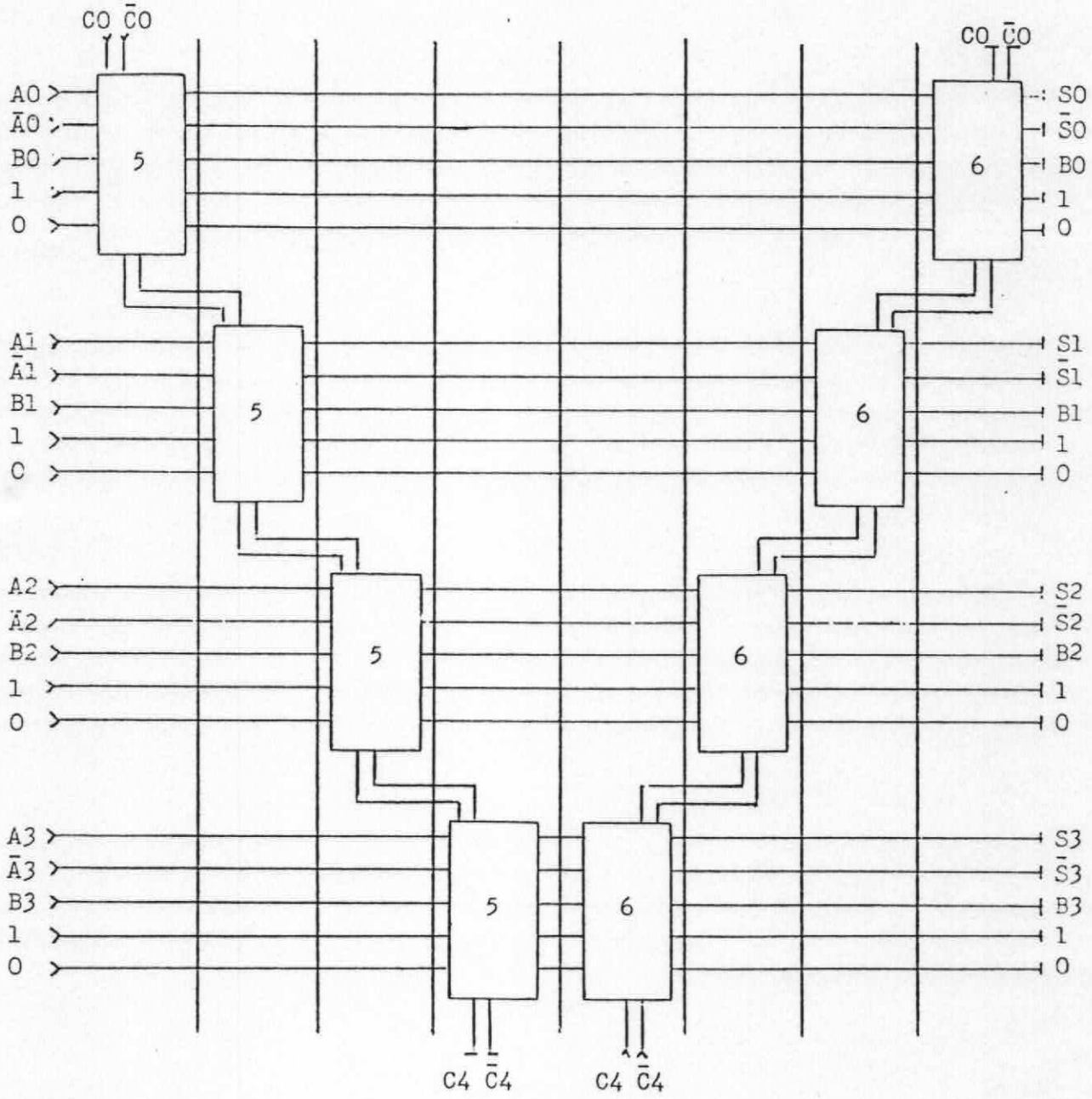
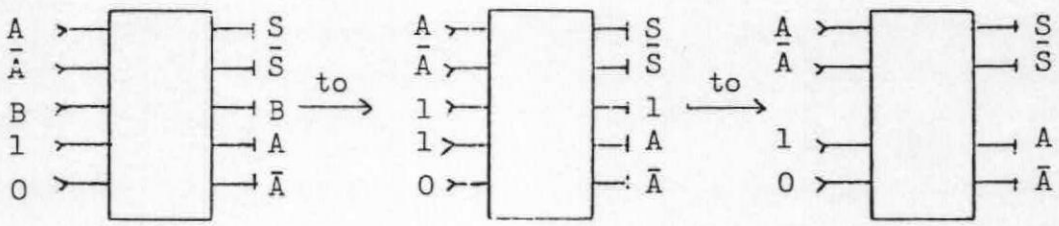


Figure 7. 4-bit Accumulator



THE COUNTER

A counter can be thought of as a simplified accumulator where the number being added is always "1". To use the accumulator as a counter would only require replacing B by "1". Since this is a constant the circuit can be simplified. Using the same construction methods as for the accumulator it is only necessary to modify the adder circuit.



Since B in binary is "00000001", the first bit B_0 will be replaced by a "1", and all other bits will be replaced by a "0". Figures 8 and 9 show the results of the simplifications for the adder and the inverse adder. The circuit for the first bit is Figure 10. A 4-bit counter would have the same structure as the 4-bit accumulator so its diagram would look like Figure 7 without the B inputs and outputs.

Figure 8. 1-bit Counter (First Half)

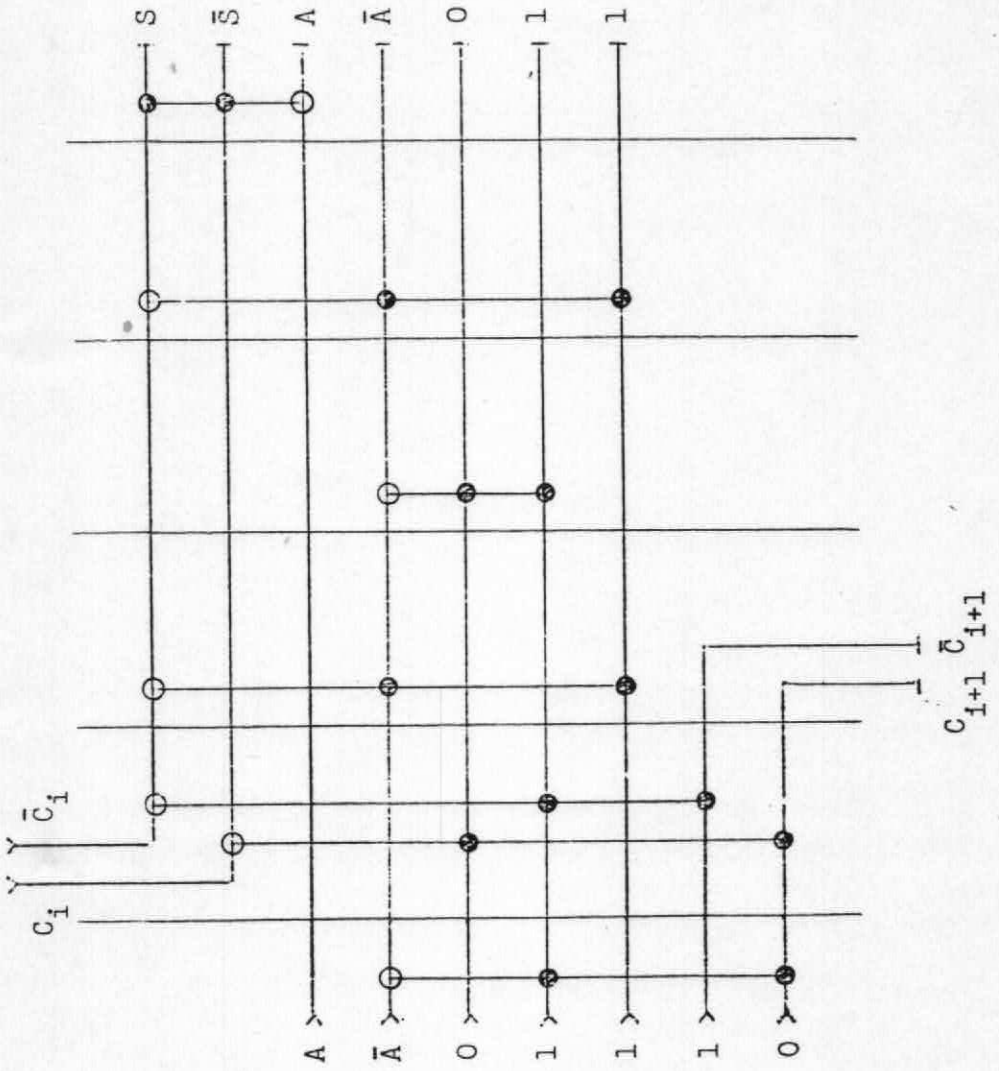


Figure 9. 1-bit Counter (Second Half)

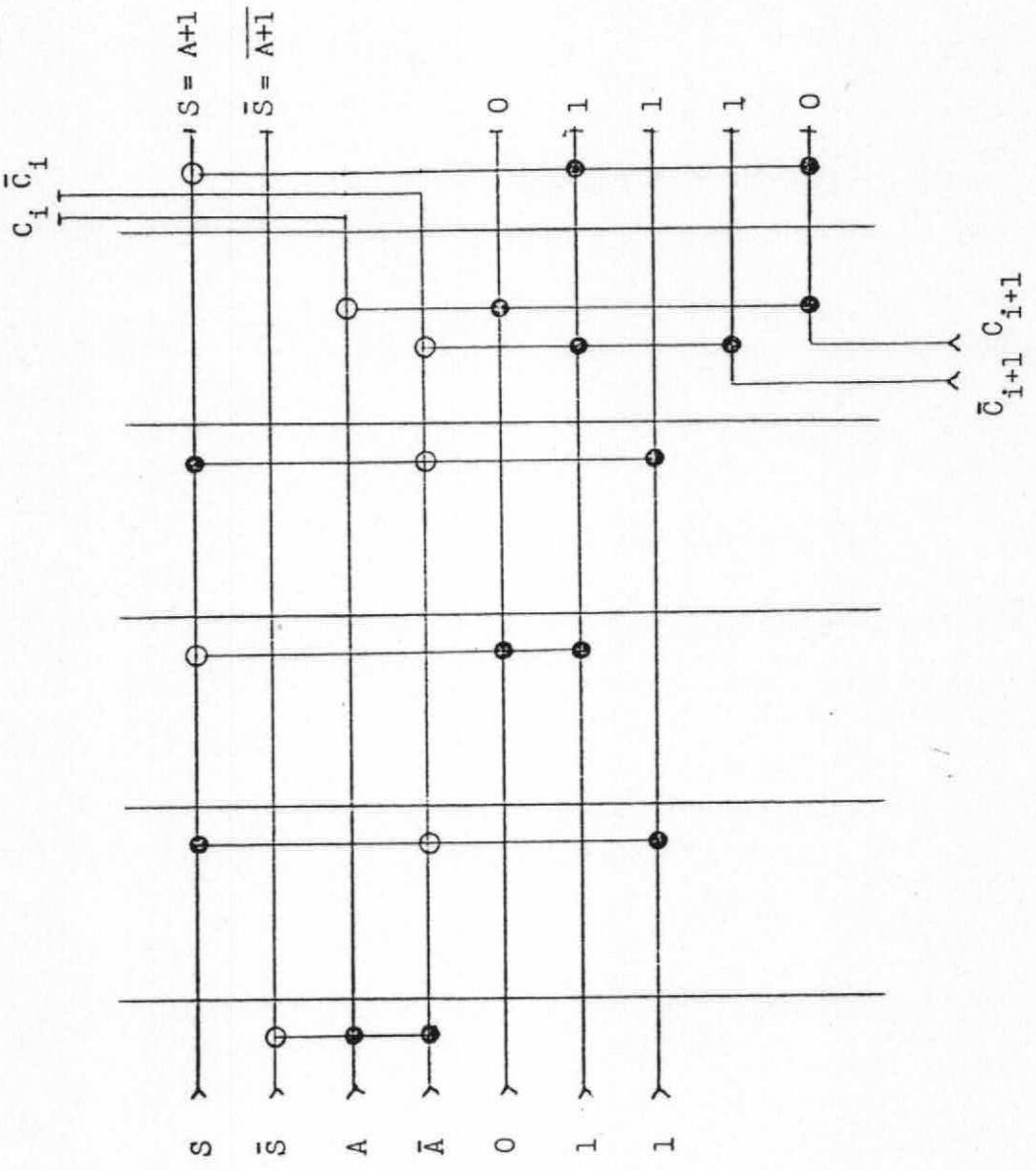
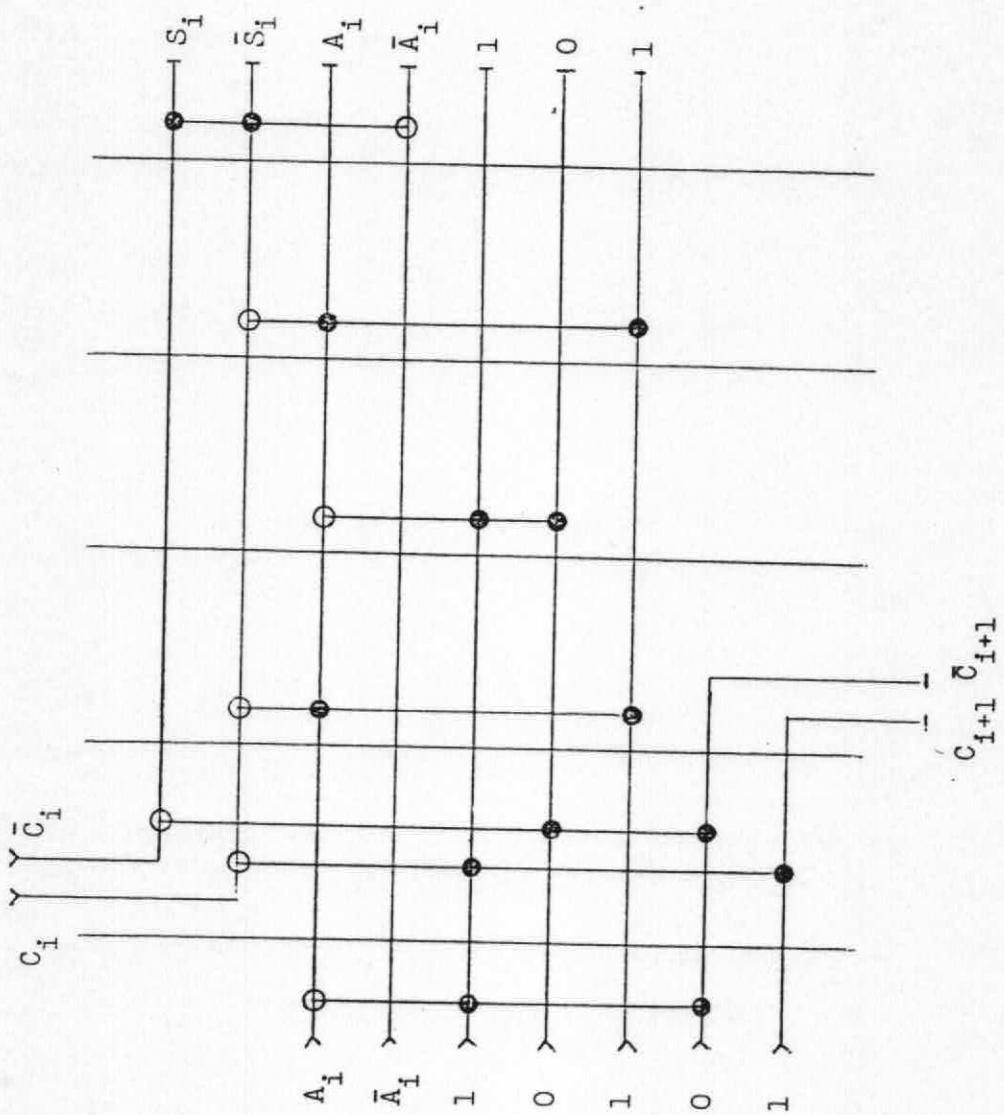


Figure 10. First Bit of Counter



BIBLIOGRAPHY

Conservative Logic, Bill Silver (Informal paper)