P. Elias

Computation in the Presence of Noise*

Abstract: The behavior of a system consisting of a preliminary coder, an unreliable computer, and a decoder is investigated. Coding input blocks of k binary digits into output blocks of $n>k$ binary digits, it is shown that a simple combinational computer which can take the *and* or *or* of k or more input blocks can only be made arbitrarily reliable by making n/k arbitrarily large, so that the capacity for computation, in an information theory coding sense, is zero. Incomplete results for a single *and* or *or* circuit give the same result if the output gives no information about the inputs except for the information about their *and* or *or*; if this is not demanded, then for $n>2k$, reliable computation through noisy computing circuits is possible, but the computing is done in the decoder.

Introduction

This paper investigates the problem of doing reliable computation with computing elements which in themselves are unreliable. People are interested in this problem for three reasons. First, the human nervous system seems to be capable of performing reliable computations, while there is considerable evidence that the way in which a particular neuron responds to a particular set of input signals is statistical in character. It is therefore interesting to construct formal models which have the same characteristics. Second, it seems likely that we are at the beginning of a transition away from the current concept of the large-scale high-speed general-purpose electronic digital computer, more concisely described as the one-bottleneck computer. In a one-bottleneck computer all of the actual computation takes place in a single, central bottleneck known as the arithmetic organ. It is therefore possible to demand that the components used in this unit be highly reliable, and it is economic to spend considerable sums on each of them to make it so. However, as we start to build *parallel,* or *diffuse* computers, in which computation is carried out all over the machine and the number of computing elements required increases rapidly, it may be that the techniques which turn out to be useful for making active components in the very large quantities required are not amenable to making the individual elements highly reliable. Third, the problem has considerable formal fascination.

The problem is difficult to formulate. It is necessary to make assumptions as to the repertory of computing elements which is available, and what the character of their unreliability is. It also seems necessary to select some elements which are not unreliable. J. Von Neumann[1] has analyzed computers whose unreliable elements are majority organs—crude models of a neuron. Shannon and Moore[2] have analyzed combinational circuits whose components are unreliable relays. Both papers assume that the wiring diagram is correctly drawn and correctly followed in construction, but that computation proper is performed only by unreliable elements. The unreliability in each case consists in the elementary device considered having errors, the errors in different elementary devices being statistically independent, and each device having no memory. (A recent analysis by Warren McCulloch,[3] based on neuron models, treats the case of common variation in threshold for a group of nearby elements, and is the only treatment of this problem of which I am aware which does not make use of an assumption of statistical independence.)

Although the Von Neumann and Shannon-Moore papers differ greatly in the details of their assumptions, analyses and conclusions, they agree in one major respect. Both show that it is possible to make arbitrarily reliable computers out of individually unreliable computing elements. But in both cases the methods by which the construction is carried out require the use of a large number of unreliable elements—a number which increases with the level of reliability which is required for the computer operation.

The results represent a triumph of analysis. In practical terms, given the very high reliability of current computer components, they say that at the cost of multiplying the number of computing components by not too large a

*This work was performed at the University of California, Berkeley, and was supported by the Office of Naval Research under Contract Nonr-222(53).

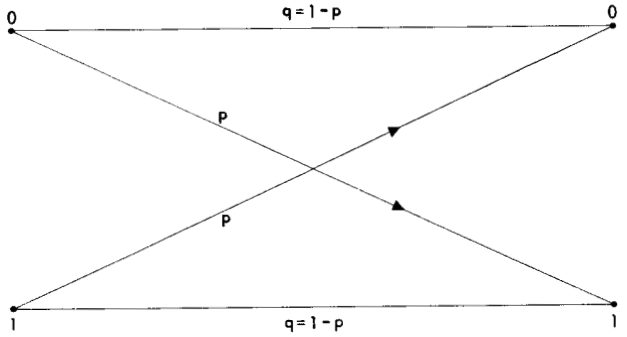


Figure 1 **A noisy binary channel.**

factor, it is possible to achieve truly fantastic reliabilities. But the results are unsatisfying, or at least disappointing, from a theoretical point of view in the context set by information theory. In fact, these results have the character of the pre-information theory results on the reliable transmission of information over unreliable (noisy) channels.

Consider a noisy binary channel as shown in Fig. 1. A transmission system sends zeros or ones, and with probability p the channel causes the transmitted symbol to be received incorrectly; with probability $q=1-p$, the transmitted symbol is received without error. Given this channel, with successive errors in transmission assumed statistically independent, how can we arrange to transmit binary digits through the channel and receive them with a probability of error less than p? The pre-information theory solution was iteration. Each digit is sent 3, or 5, or in general $2n+1$ times in a row. This reduces the rate at which new information can be fed into the channel to $1/3,\ 1/5, \ldots, 1/(2n+1)$ of its former value. The receiver then takes a majority rule at the output: if $n+1$ or more digits are received as ones, the receiver decides that $2n+1$ ones were transmitted, and decodes the received sequence of $2n+1$ adjacent digits into a one. This procedure gives an arbitrarily low error probability, but at the expense of an arbitrarily low rate of transmission. For this particular system of coding and decoding, the error probability decreases more or less exponentially with amount of iteration. In terms of rate R of transmission (input digits per channel digit) we have roughly an error probability P_e given by

$$P_e \sim 2^{-k/R}. \tag{1}$$

This error probability is a continuous function of rate, which looks plausible; it is shown as a plot in Fig. 2, and it says that more reliability can be attained only at the expense of less rate.

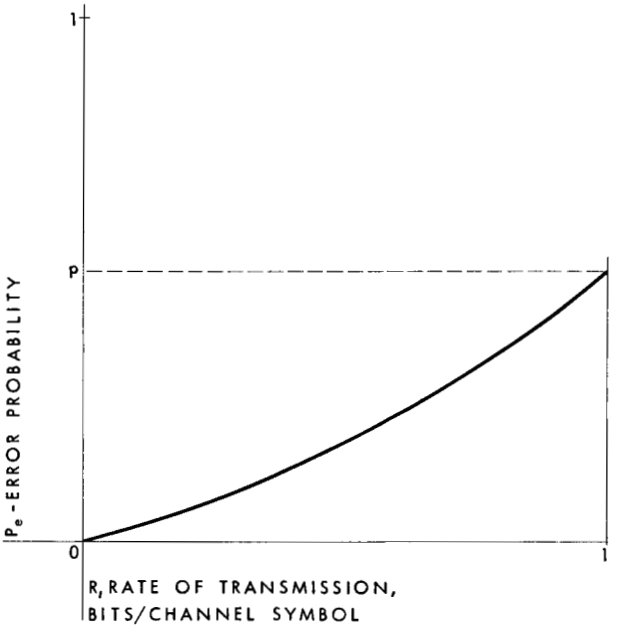
However, the noisy-channel coding theorem of information theory[4] gives a different answer to the same question. It makes this relationship discontinuous. Defining a number C as the capacity for the noisy channel, it says that

$$P_e = \begin{cases} 0, & R<C \\ 1, & R>C. \end{cases} \tag{2}$$

This result is achieved by coding, like the result of Eq. (1), but with a slight modification in the procedure, which is now called *block coding*. The input digits again go into a coder before transmission, and the received digits again go into a decoder after reception, and again the number of digits which go into the coder is smaller than the number which come out of it. But the input digits no longer go in one at a time. The input digits are segmented into blocks, of k digits each. Each block of k digits is converted by the coder into a block of n digits, $n>k$. The receiver receives a noisy block of n digits, and decodes them, usually into the correct block of k digits which was the coder input. Mistakes are still made in decoding, but if the rate of transmission, $R=k/n$, in bits per channel symbol, is less than the channel capacity, then keeping R constant and letting k and n increase reduces the error probability in a way which is exponential in the block size n for given R and C. Thus in the limit of arbitrarily complex encoding and decoding, the error probability becomes arbitrarily small.

A discontinuous relationship like (2) is always surprising, and this one was initially very surprising indeed. But just what is continuous—and just what is surprising—depends on how the results are presented, and I would like to present this result so that it looks ordinary, so that when we get to computation and a different kind of behavior, that can then surprise us. Suppose we plot C, the maximum rate at which it is possible to transmit with arbitrarily high reliability over the channel, as a function of the noise parameter p which measures how poor the channel is. In the case of block coding, we get the result of Fig. 3a: if there is no noise (i.e., $p=0$) then the capacity of the channel is one bit per channel symbol. As the noise increases, the channel capacity $C(p)$ decreases

Figure 2 **Error probability vs rate under iteration.**



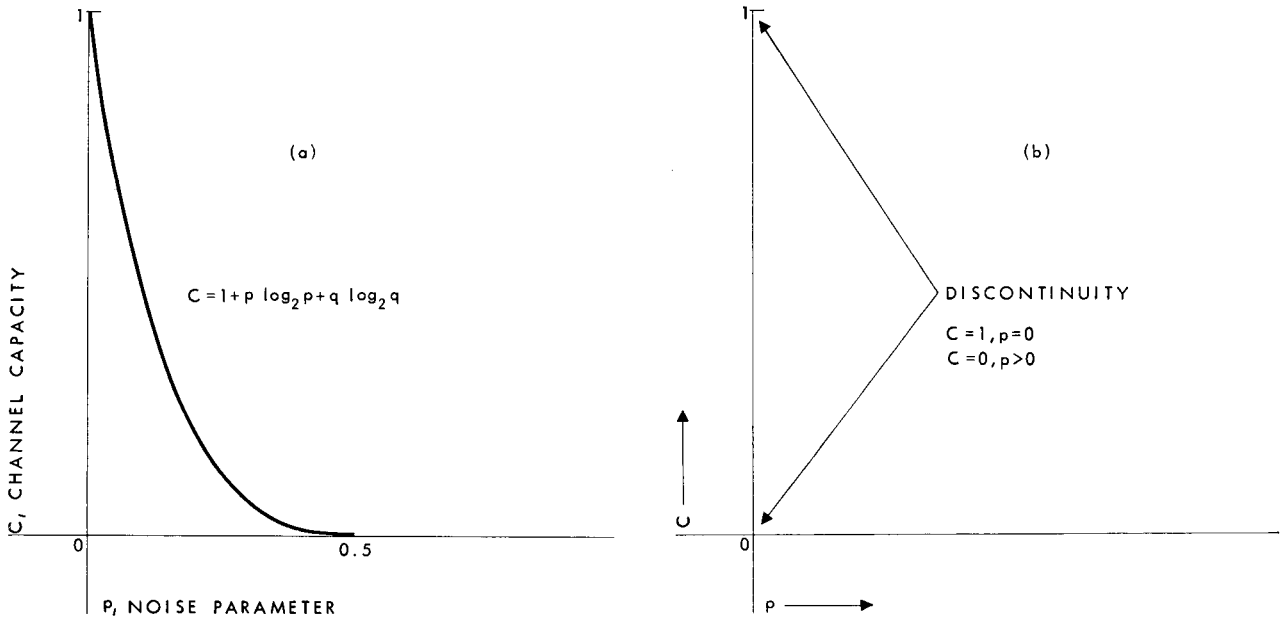


Figure 3 **Channel capacity vs noise under block coding and iteration.**

in a reasonable, continous fashion, reaching zero only when $p=½$ and the channel output is statistically independent of its input. On the other hand, in the case of simple iteration, we get the unreasonable, discontinuous behavior illustrated in Fig. 3b: if $p=0$, then the channel capacity is one bit per channel symbol as before, but as soon as $p>0$, no matter how small, C for iteration drops to zero. We can transmit with high reliability at a high rate when p is small, but to make the reliability greater it is necessary to iterate more and reduce rate, and to get arbitrary reliability it is necessary to iterate arbitrarily often, and to transmit at an arbitrarily small rate. Thus in the block coding result, a small difference in reliability makes only a small difference in C; in the iteration case, a small difference in reliability makes a large difference in C.

The Von Neumann and the Shannon-Moore procedures are both like the iterative solution to the noisy-channel coding problem. Time and number of components are interchangeable, in the sense that if we use twice as many components in order to make a computer operate at the same rate more reliably, we could instead use the components to build two computers and compute twice as fast at the old level of reliability. In this sense, as we increase the number of components to get greater reliability at the same rate, we are reducing the rate of computation per component. And we must reduce it arbitrarily far in order to get arbitrarily reliable computation, as soon as the components go from perfect to something slightly less than perfect. This kind of discontinuity no longer seems natural after the noisy-channel coding theorem, and the logical question is whether it can be eliminated—i.e., whether introduction of a small probability of error for components necessarily causes a discontinuous drop in the attainable rate of arbitrarily reliable computation, and if so, whether the drop is necessarily to rate zero.

This paper explores that question in a preliminary way, in the block coding context. The results are incomplete, but such as they are, they are negative. That is, they show an abrupt drop of computational rate, in some cases to zero, caused by a small component error probability. This is disappointing, and it is probably due to the formulation used for the problem and not to the problem itself. As in the Von Neumann and Shannon-Moore papers we assume that errors in different elements, or in the successive operations of one element, are statistically independent of one another. Instead of adding additional unreliable components to increase reliability, however, we add additional input data to provide error detection and correction. A recent paper by Peterson[5] is very close to the spirit of the present investigation, and a review paper by Löfgren[6] discusses related problems.

Block computation

We will consider a simple computer, or data-processor—simple in the sense that it does not change its wiring diagram as the result of prior computations. In the noiseless case, one or more streams of input binary digits enter the computer and interact through two-input single-output no-memory combining circuits. The only memory is in the form of pure delays in multiples of one pulse-period. Any output digit of such a computer is some combinational function of a set of present and past input digits.

The combining circuits are *and, or, if and only if* and so forth: combinational circuits which produce an output digit which is a function of two input digits. A noiseless combining circuit of this type is specified by a 2 by 2 binary matrix. Let x and y be the two input digits, and

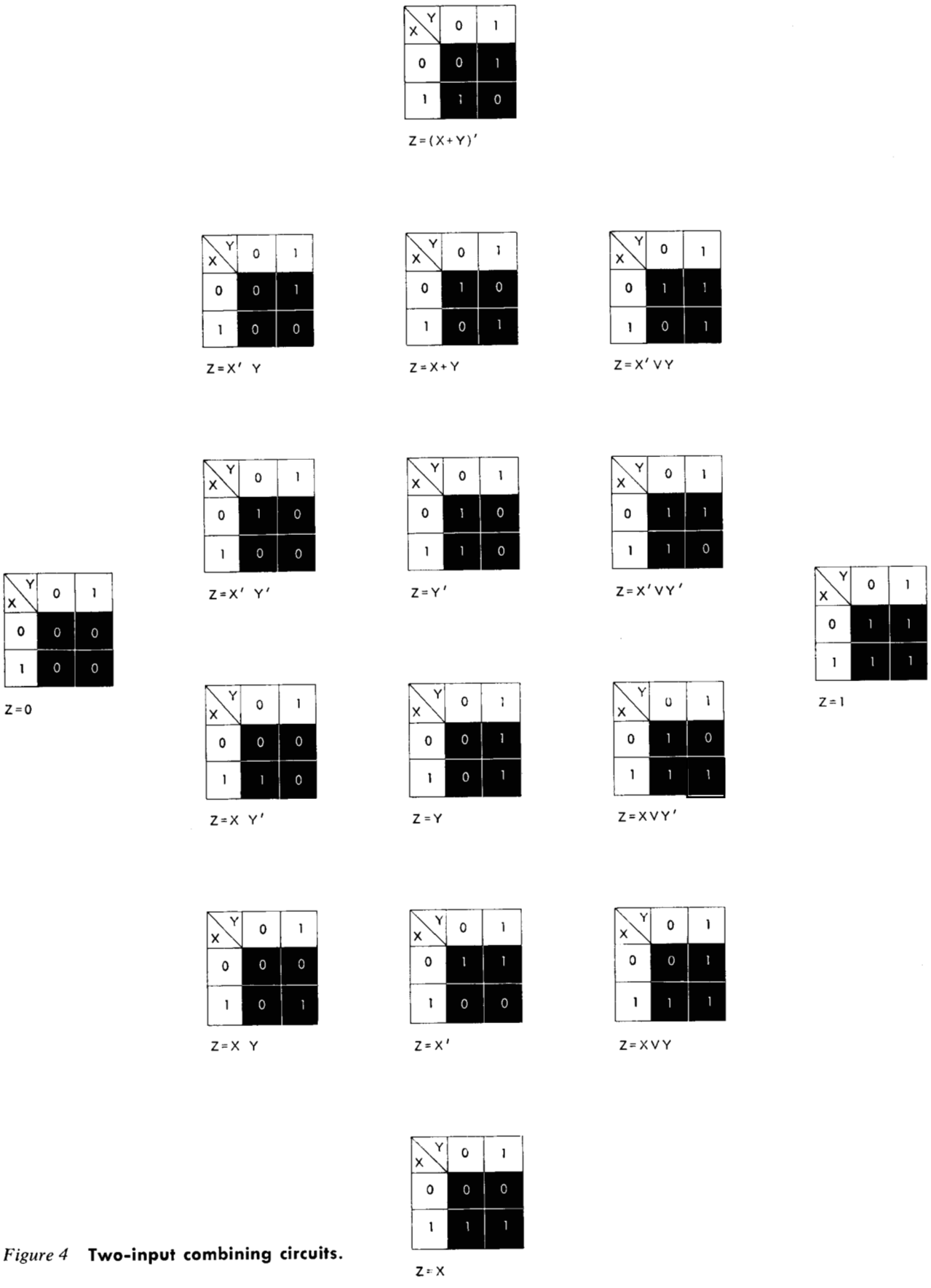


Figure 4 **Two-input combining circuits.**

let z be the output digit. Matrices giving $z=f(x, y)$ are shown for all such functions in Fig. 4. A value of x (0 or 1) selects a row, a value of y (0 or 1) selects a column, and the entry at that row and column is the resulting value of the function z. The sixteen possible functions are generated by filling in the four positions in the diagram with 0's and 1's in all possible ways.

To describe a noisy computing element, we first reinterpret the tables in Fig. 4. For $x=i$, $y=j$, the entry in the matrix at i,j is no longer the value z will take but the probability that z will take the value 1. If this probability is one, then the output digit will be a 1. If the probability is zero, then the output digit will be a zero. If it is p_{ij}, with $0<p_{ij}<1$, then the output digit will be a 1 with probability p_{ij}, and a 0 with probability $q_{ij}=1-p_{ij}$. It is assumed that the operation of the noisy combiner on successive input digit pairs is statistically independent: whenever x takes the value i and y takes the value j, z takes the value 1 with probability p_{ij} and no knowledge of past history or of what is going on elsewhere in the computer modifies this probability.

In the presence of noise we must expect at least some reduction in rate of reliable computation. If this is not to appear as a reduction in the reliability of the output, it is necessary to reduce the rate of input to the computer—i.e., redundancy must be put into the inputs to the combining circuits. We do this by putting a block coder before the computer and a decoder after it, as illustrated in Fig. 5.

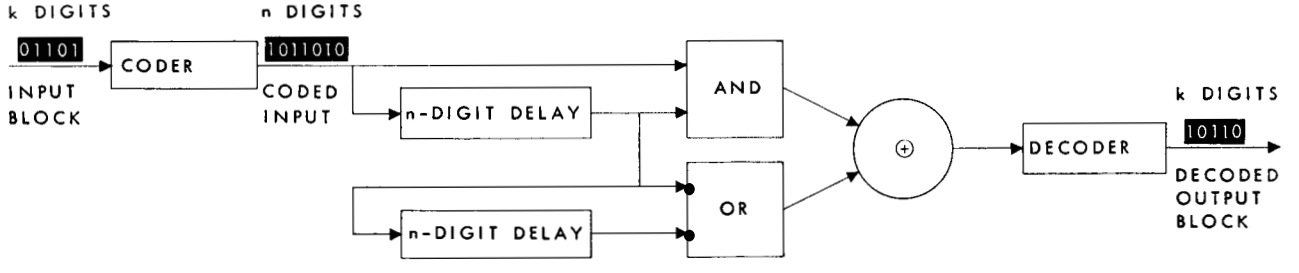
The input to the coder is segmented into blocks of length k input digits. Each such block is fed into the coder, which produces an output block of n digits, $n>k$. We assume that the mapping from input sequences to output sequences is a fixed one-to-one transformation, independent of past inputs and of computer operation. This assumption is necessary because we will also assume that the coder and the decoder themselves are reliable devices. If the coder output could then depend on past input blocks as well as the present block, it could be doing the computing itself. We demand that the only way in which two different coded blocks of n digits can be brought together to produce an output block of n digits which is a function of them both is through one of the noisy combining circuits.

The coder output blocks go into delay circuits, which have outputs going to noisy combining circuits. All delays are multiples of n digits, so that any two blocks brought out of different delay circuits will be in step in going through a combining circuit. The action of a combiner is then to accept two input blocks of n digits each and to produce one output block of n digits. The output block produced is only statistically determined by the two inputs: given a sequence of values $x_1, x_2, \ldots, x_n$, and $y_1, y_2, \ldots, y_n$ for the two inputs, a probability distribution is set up on the 2^n possible output sequences $z_1, z_2, \ldots, z_n$, which is of product form because of the statistical independence with which successive pairs of inputs are treated by the device. Thus, e.g., the probability of the output sequence 1, 1, . . ., 1 is the product $p_{x_1y_1}.p_{x_2y_2} \cdots p_{x_ny_n}$.

A result of the computation will first appear as a sequence of n-digit blocks coming out of some combining circuit, but this is still in coded form. The proper answer to the question which the computer was asked is not an n-digit sequence but a k-digit sequence—the k-digit sequence which is the logical function that the computer is connected to compute of some set of current and past input k-digit blocks. This is obtained from the decoder, which maps sequences of length n into sequences of length k. This transformation must be many-one, in general, since different errors which may have occurred in computation must lead to the same correct decoded answer if the coding and decoding procedure is in fact to be successful. However, we again assume that the mapping is fixed: to each input sequence of length n corresponds exactly one fixed output sequence of length k, independent of past inputs and of computer action. Again we assume that the decoder is a noiseless device.

We now consider the particular computer shown in Fig. 6 which takes the *and* of k successive n-digit blocks of coded output, using k-1 delay circuits and k-1 noisy *and* circuits. Let us denote the input sequence consisting of k1's by 1^k, the sequence consisting of k-j1's followed by j0's as $1^{k-j}0^j$, and the sequence consisting of k0's as 0^k. Let us denote the n-digit block which is the coded version of a k-digit input sequence s by $T(s)$, so that $T(1^k)$ is the output sequence from the coder when a sequence of k1's is put in. Then consider the following set of n-digit blocks which are possible results of this computation as they

Figure 5 **A computer.**



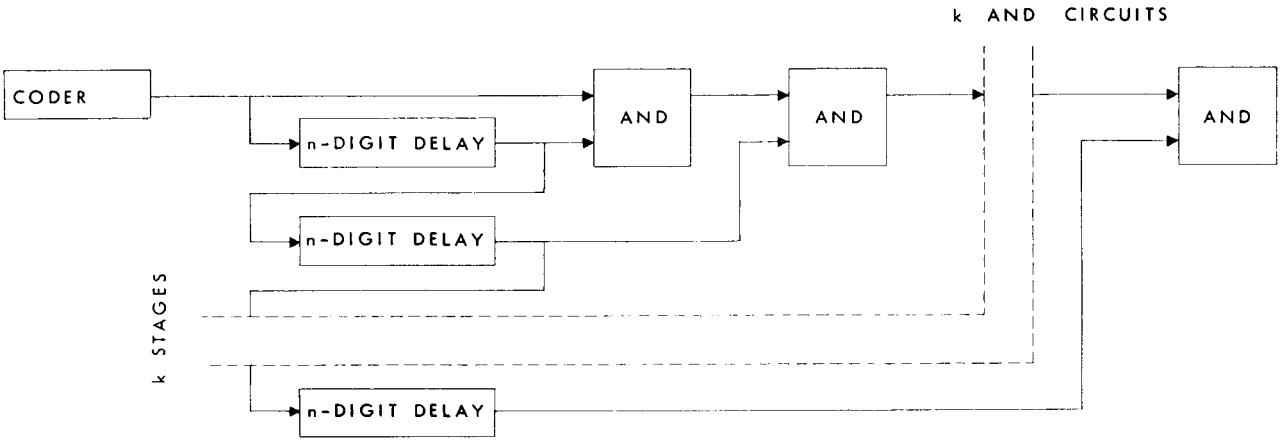


Figure 6 **Multiple *and* computer.**

come out of the last *and* circuit before decoding:

$$T(1^k)\cdot T(1^k)\cdots T(1^k)\cdot T(1^k)$$
$$T(1^k)\cdot T(1^k)\cdots T(1^k)\cdot T(1^{k-1}0^1)$$
$$T(1^k)\cdot T(1^k)\cdots T(1^{k-1}0^1)\cdot T(1^{k-2}0^2)$$
$$\cdot$$
$$\cdot$$
$$\cdot$$
$$T(1^k)\cdot T(1^{k-1}0^1)\cdots T(1^10^{k-1})\cdot T(0^k). \qquad (3)$$

Each line in (3) can be obtained from the preceding line by *anding* with one new factor, since the *and* of $T(1^k)$ with itself any number of times is still just $T(1^k)$. But adding a new factor by *anding* can only strike out some of the 1's which are present in a sequence. All of the n-digit sequences in (3) must be decoded differently, since they all represent computations having different answers; in fact the proper answer—the proper decoder output—for any sequence in (3) is the k-digit sequence which is the argument of its last factor. Thus, in going from one line of (3) to the next we strike out at least one 1 and perhaps more. Let d_i be the number of 1's which are stricken out in going from line i to line $i+1$. Then since there are at most n 1's in the first line and at least no 1's in the last line, we have

$$\sum_{i=1}^{k} d_i \leqq n,\ d_i \geqq 1. \qquad (4)$$

It follows from (4) that if $n<2k$, some of the d_i will be just 1, and a single error in the output of the final adder can cause two of the sequences corresponding to adjacent lines in (3) to become confused, so that the decoder will print out the wrong answer when such an error is made if it prints out the right answer when no errors are made. Thus it is not possible to get a reliability for the block greater than that present in the individual devices until the rate of transmission drops from unity ($k=n$) to rate one-half ($2k=n$). And even then, there are single errors which the decoder can detect but can only attempt to correct, with probability at least one-half of guessing wrong. The rate must drop to one-third before all single errors in the computations listed in (3) can be corrected.

It will be noted that we could do as well, at least in terms of the minimum Hamming distances[7] between output codewords listed in (3), by not using block coding at all, but merely by iterating the i'th digit of the input block d_i times. Thus at least in terms of minimum distance, no block code for the computer of Fig. 6 does better than simple iteration of the input digits, and as we have seen before this leads to a computational capacity—i.e., a maximum attainable rate of computation of arbitrary reliability—which is zero.

The concept of channel capacity

There are two well-defined concepts of channel capacity for communications purposes. One of these is the maximum rate at which it is possible to obtain mutual information from the output of a (possibly noisy) channel about its input. The other is the maximum rate at which it is possible to receive information over the channel with arbitrarily high reliability. In the communications situation the noisy-channel coding theorem proves that these two quantities are equal. The demonstration just given shows that for the computation case we have discussed, the two quantities are different. The maximum rate for computing with arbitrarily high reliability is zero in this case. However, the rate at which the output of the computer gives information about the function which is being computed is high when the probability of error is low; the receiver has only a small amount of equivocation, and it would take only a small additional amount of information to correct the occasional errors which are present in the computer output. But it is not possible to provide this extra information by coding the different blocks of k digits independently of one another before the computation starts.

Our negative result applied specifically to iterated *and* operations only. It is natural to ask, first, whether there are other computational operations which are easier to noiseproof, and second, whether there are enough of them to generate all combinational functions of two variables by cascading. The answer to the first question is "yes," to the second, "no." Referring to Fig. 4, we can

divide the 16 possible combining circuits into five classes, classifying by how many 1's they have as entries in their 2 by 2 matrices. The iterated *or* has the same character as the iterated *and*, and a dual of the demonstration in the last paragraph can be constructed to show that a computer which takes the *or* of a number of input blocks has no capacity for computation of arbitrary reliability. This is also true for all of the other functions in Fig. 4 which have an odd number of 1's in their matrices. If they have a single 1, they are the *and* of x or x' and y or y'. If they have three 1's, they are the *or* of x or x' and y or y'. In either case the result is negative.

On the other hand, a computer constructed entirely of combining elements which have an even number of 1's in their matrices can have its input blocks independently coded and its output blocks decoded to give reliable results despite errors in computation. The functions $z=0$ and $z=1$ do not really represent combiners at all; the output is independent of the input, and these functions can be reliably computed if reliable sources of 0's and 1's are available to the receiver. The other functions with an even number of 1's either reproduce one input (e.g., $z=x$, $z=y$) or its complement (e.g., $z=x'$, $z=y'$), or take the modulo two sum of x and y, or its negation. The Hamming codes,[7] or any other group codes[8] which include the sequence of n 1's, may be used at the input to such a computer. Each stage of the computation then maps the set of 2^k n-digit input blocks into itself, preserving minimum distance properties—and average distance properties too. Using such codes in such a computer, in the absence of noise, the decoding operation is one-to-one, rather than many-to-one; it is only noise which can cause the same answer to be represented as two different n-digit blocks. If the effect of the noise is symmetric, i.e., if a noisy output sequence is as likely to differ from the noiseless version in having 0's replaced by 1's as in having 1's replaced by 0's—then the capacity for noisy computation is just the capacity for transmission over a binary channel with the same symmetric error probability. It is known[9] that group codes can be used to obtain transmission of arbitrary reliability over such a channel at any rate less than the channel capacity. If the noise is not symmetric, but is small, then it is not obvious whether the capacity for reliable computation is the same as the capacity for reliable transmission, but in any case the computational capacity is decreased only a small amount by the introduction of small error probabilities in the computing elements.

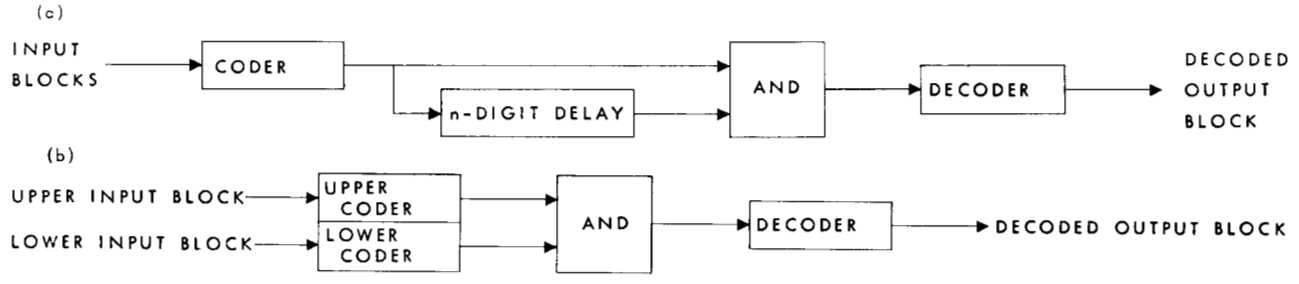
Unfortunately, however, it is not possible to generate all combinational functions of two variables by using only the functions which have an even number of 1's in their matrices. It is impossible to generate *and* or *or* from such functions, for example.

Other results

We are left with the negative result that if we code all blocks identically at the point of entry to the computer and demand reliability through cascaded computing stages which include *and* or *or* operations, we have an essentially iterational situation as soon as the cascade length is equal to the input block size, and the rate for computation of arbitrary reliability goes to zero. The obvious assumption to attack is the cascading requirement. Since we are permitting coders and decoders to be reliable, we may be wary of using large numbers of them; however, we can certainly do so to obtain negative results. Taking the most extreme situation, we can permit coding to take place immediately before each noisy combining circuit, and decoding immediately after it. Two such situations are illustrated in Fig. 7. At the top (7a) we have a single input coder, which codes blocks of k input digits into blocks of n output digits. An n-digit delay line provides the second input to an *and* circuit; the two identically coded n-digit blocks are *anded*, and the result decoded. Below (7b) we also have an *and* circuit, but the two input sequences are no longer required to be identically coded; each goes through its own coder, and the two output n-digit blocks are *anded* and decoded.

The results here are fragmentary, and the problem of correct formulation becomes more acute. In Fig. 7b, if $n>2k$, it is possible to do reliable *anding* through a noisy *ander*—but the *ander* does not do the computation. If the upper input coder generates first $n/2$ 1's and then a coded version of its input k-digit sequence, and the lower input coder generates first a coded version of its input sequence and then $n/2$ 1's, the output will be a noisy version of the two coded inputs in sequence. The decoder can then correct errors in each of the two inputs, and take their *and* by means of a table. Here the noisy *and* circuit is being used only as a noisy channel, and does

Figure 7 **Single-layer *anding*.**



no real computation. The same result can be obtained in Fig. 7a by a random-coding argument, although it is not clear whether a reliability which is uniform on input pairs can be obtained in this way.

To avoid this obvious cheating, we can demand that whatever information is present in the output n-digit sequence is information about the *and* of the two input k-digit sequences only. This requires that the decoder be one-one in the absence of noise, for as soon as two pairs of inputs which have the same *and* produce distinct outputs in the absence of noise, the decoder has some information about the inputs which is not information about their *and*. Under this restriction, the negative result follows as before: the only noise-resisting codes are those in which the i'th input digit in a block of k is iterated m_i times, with

$$\sum_{i=1}^{k} m_i \leqq n.$$

This again leads to zero rate of reliable computation. In fact, under the milder restriction that only one n-digit sequence is decoded into the sequence of k 0's in the absence of noise, the behavior is still the same, at least in minimum distance terms, and so is the conclusion.

It seems very likely that, even for computations of depth one, as illustrated in Fig. 7, coding can improve the reliability of a noisy *and* or *or* circuit only through either iteration or cheating. That is, it seems likely that for $n<2k$ there is nothing better to do than to iterate some of the input digits, while for $n>2k$ the only alternative to iteration is to squeeze both inputs through the noisy computer and do the computation in the decoder. However this has not yet been proved.

Acknowledgment

Like so many of the other topics which have been discussed at this meeting, this one originated with Claude Shannon. He raised the question, several years ago in conversation, as to what the channel capacity of a noisy *ander* was. I did not return to the topic until a few months ago, when I picked the title for this paper.

References

1. J. Von Neumann, "Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components," pp. 43-98 in *Automata Studies,* C. E. Shannon and J. McCarthy, editors, *Annals of Math. Studies,* **34,** Princeton Univ. Press (1956).
2. E. F. Moore and C. E. Shannon, "Reliable Circuits Using Less Reliable Relays," *Jour. Franklin Inst.,* **262,** 191-208 and 281-297 (1956).
3. W. S. McCulloch, "Three of Von Neumann's Biological Questions," *Quarterly Progress Report,* 129-138, Research Laboratory of Electronics, M.I.T. (Oct. 15, 1957).
4. C. E. Shannon, "A Mathematical Theory of Communication," *Bell Syst. Tech. J.,* **27,** 379-423 (July 1948).
5. W. W. Peterson, "On Checking an Adder," *IBM Jour. of Research and Devel.,* **2,** 166-168 (1958).
6. L. Löfgren, "Automata of High Complexity and Methods of Increasing their Reliability by Redundancy," *Information and Control* **1,** 127-147 (1958).
7. R. W. Hamming, "Error Detecting and Error Correcting Codes," *Bell Syst. Tech. J.,* **29,** 147-160 (1950).
8. D. Slepian, "A Class of Binary Signalling Alphabets," *Bell Syst. Tech. J.,* **35,** 203-234 (1956).
9. P. Elias, "Coding for Noisy Channels," *Inst. Rad. Eng. Convention Record,* **Pt. 4,** 37-46 (March 1955).

Received May 27, 1958