of the control memory does permit the design of a much more versatile machine. The next step in designing such a machine is to provide a method of building in all possible transfers between existing generalized registers.[3] Suppose the set of generalized registers is designated by $\hat{M}$ and suppose that a control word in F contains three addresses, $(Ad_1[F])$, $(Ad_2[F])$, and $(Ad_3[F])$ each of which refers to one of the generalized registers in $\hat{M}$. Then the general transfer is stated as follows:

$$|f_i|: \mu(\hat{M}<Ad_1[F]>)+ \mu'(\hat{M}<Ad_2[F]>)\rightarrow\hat{M}<Ad_3[F]>$$

The function μ is a Boolean function designated in the machine. If this function is true the generalized register determined by the contents of $Ad_1[F]$ are transferred to the generalized register determined by the contents of $Ad_3[F]$; otherwise the contents of the register determined by $Ad_2[F]$ are transferred.

When viewed in this way, all the registers and derived registers are connected together through a large selection switch (or several such switches for simultaneous transfers) which is actuated by the state of control. Changing the nature of the machine is then accomplished by changing the contents of the control memory which essentially sets up a sequence of selection switch positions to perform the desired instructions.

References

1. Symbolic Design of Digital Computers, **I. S. Reed.** *M.I.T., Lincoln Laboratory Technical Memorandum, No. 23,* Lexington, Mass., Jan. 1953 not generally available.

2. Logical Design of CG24, **G. P. Dinneen, J. A. Dumanian, I. L. Lebow, I. S. Reed, P. B. Sebring.** *M.I.T. Lincoln Laboratory Technical Report, No. 139,* Lexington, Mass., Apr. 1956, not generally available.

3. Symbolic Design Techniques Applied to a Generalized Computer, **I. S. Reed.** *M.I.T. Lincoln Laboratory, Technical Report, No. 141,* Lexington, Mass., Jan. 1957, not generally available.

Discussion

D. P. Boone (Astronautic Company): You will notice that the feeder is tied to the radar. How do you accomplish the feeder from the radar?

Dr. Lebow: The way it is set up right now is the following:

There are three registers which take the data from the radar, and associated with these registers is another bit which tells the computer that some data has come in.

Then upon sensing this bit, the data are transferred directly into the computer memory, without going through the arithmetic unit.

W. J. Seiple (Federal Laboratories): In discussing the conversion of the data, how do you get them into the computer so that it is tagged to that correlation?

Dr. Lebow: One of the registers I spoke about is a register that contains the actual real time, the time at which the return was received, and this represents part of the data that are introduced into the machine.

K. L. Deane (Variomatic): Does this include some analog to digital conversion?

Dr. Lebow: Yes, for the data part of the word, not for the time part, of course.

Question: What consideration, or what are the considered optimizations of these controls?

Dr. Lebow: This kind of optimization was considered by the people who actually deal with circuit design. I, perhaps, did not say enough about that at the beginning of the talk when I explained what we meant by logical design. I think our use of the term is a little different than what most people mean by the term.

You can see from the talk that the term logical design does not get down to the detailed logical configuration.

Obviously, a lot of work is necessary in going from the transfer level to the actual circuit details, and this is a function of hardware that the machine will be built of.

This was optimized in some sense by the people who actually built this machine.

Design Criteria for Autosynchronous Circuits

J. C. SIMS, JR. H. J. GRAY

Synopsis: The circuits and organization of present computers are such that possible operating speeds are lower than the capabilities of the components. The speed limitations of such synchronous computers will be described, and design criteria for higher speed operation set forth. Examples will be given for a logic and circuit organization which results in both faster operation and improved performance to cost ratios. In particular, circuits which are free of transient logical malfunctions, sometimes called "spikes," will be developed and a typical autosynchronous system will be shown.

COMPUTING circuitry is generally organized to transfer digital information from a first storage through a logical net into a second storage register. As the information passes through storage, the wave shapes are standardized and the relative and absolute timing of the signals are restored. This situation is generalized in Fig. 1. Here information stored in two registers is read out on arrival of a timing pulse, CP_1, passes through a logical net, and is received by an output register. Upon receipt, it can be read out again to the same or to a further network in response to a timing pulse, CP_2. In synchronous machines, the signals CP_1 and CP_2 are clock pulses, and in the common single-phase systems are the same signal.

In order to establish a point of departure for the present discussion, a brief analysis will be given of synchronous systems of this form. The storage registers are usually flip-flop or shift registers, depending on whether the system is parallel or serial. The logical net is made up of combinations of "and," "or," and "not" elements, each element usually consisting of an amplifier and a group of logical diodes or resistors. The computing device is thus constructed of four basic devices, a store or flip-flop, an "and" circuit, and "or" circuit, and a "not" circuit. In some systems, a stroke element is used to function as an "and," "or," and, "not" device. Stroke elements are typified by the "Larc 1 c" circuit and the "nor" circuit.[1,2]

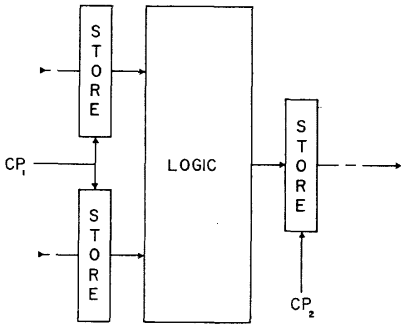
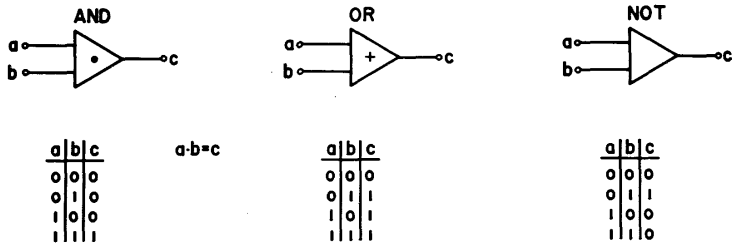


Fig. 1. General block diagram

J. C. Sims, Jr. is with Sylvania Electric Products Inc., Waltham, Mass.

H. J. Gray is with the Moore School of Electrical Engineering, University of Pennsylvania, Philadelphia, Pa.



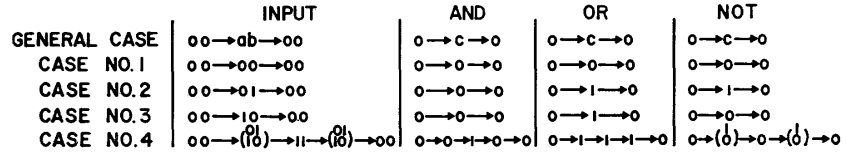


Fig. 2. Truth tables for "and," "or," "not" logic

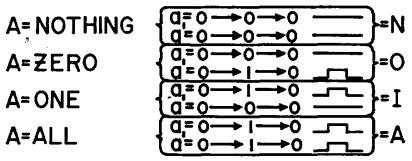


Fig. 3. Two-line notation

Conventional Logical Elements

Regardless of how the logical "and," "or," "not" elements may be constructed, they must follow the truth tables of Fig. 2. In analyzing the behavior of any logical element, it is necessary to consider the way in which the input signals can arrive and correspondingly the way in which the output signals, according to the truth table, will be developed. Consider these conditions for a return-to-zero signal notation. A signal is represented by unity, "or" by +, and "and" by "times."

The arrival and departure of a pulse at input *b* is indicated by the sequence of input states 00→01→00, etc. In the general case, two inputs (*a*) and (*b*) can change independently from zero to information and back to zero. There are four cases for binary signals, taken two at a time, and it will be noted that the first three do not introduce any ambiguity. In the fourth case, however, either input (*a*) or input (*b*) can arrive first and can leave first. The element thus passes through some intermediate input state during the set-up and drop-out periods.

The "and" and "or" functions are symmetrical, and accordingly, the fourth case ambiguity only serves to widen or narrow the output signal. Observe that the "and" function output occurs only during input signal overlap, and that misalignment narrows the output. Conversely, the "or" function widens the output.

The fourth case of the "not" function, however, exhibits a hazard or spike condition. Here coincidence of (*a*) and (*b*) in the truth table calls for zero output, but the set-up and drop-out can pass through intermediate input states calling for a "one" output. These spurious outputs, or "spikes," can operate as information on succeeding stages to cause errors.[3,4]

The most common method for eliminating spikes is through clock pulses. The outputs of the "not" gate can be sampled with a narrow probe occurring safely between the set-up and drop-out spikes. To allow this sampling to be done, the signals must be synchronized with the sampling pulses. A system so constructed uses a central clock and is called a synchronous machine.

One pays serious penalties in speed for synchronous operation. In a typical synchronous system, upwards of half the time will be expended in retiming and in time tolerances. For example, consider a recent computing system operating at 2 megacycles (mc), or one pulse every 500 microseconds (μsec). The system uses eight levels of logic between storage, and each level has a maximum signal delay of 40 μsec. The delay of an average element may, of course, be only 20 μsec, and, indeed, may be as little as 12 μsec. But in designing the circuits and assigning the repetition rate, one must assume the worst end-of-life case. Thus, allowing 40 μsec per level for eight levels, the result is a delay of 320 μsec.

The signals are propagated into the logical network from flip-flop storage. The flip flops, on the average, will read out with a delay of 20 μsec, but in the worst case, the delay may be 40 μsec. With a similar tolerance for reading into flip-flop storage, and allowing ±40-μsec clock jitter, the repetition time between clock pulses becomes 480 μsec. If, however, the clock and its jitter are eliminated, and average (rather than maximum) times are used, the delay becomes 200 μsec for a 5-mc repetition rate.

Spike-Free Logical Elements

One cannot realize this speed improvement unless two things are done. First a logical clock must be provided in place of the fixed time clock, and secondly, one must avoid the spike problem. Some work has been done in this direction, notably by Pomerene,[3] Meagher,[4] Mealy,[5] and Huffman.[6] Progress to date has been largely confined to the development of logical clocks. The spike problem has been resolved by careful control of network delays and inhibit pulse widths. It will be noted by referring to the "not" table of Fig. 2 that if the inhibit input signal (*a*) is wider than signal (*b*) and completely overlaps it, the last line of the transition table can be made to read: 0→0→0→0→0. This design method, however, is not easily accom-

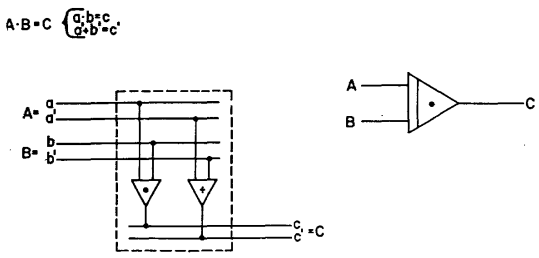


Fig. 4 (left). Two-line buffer

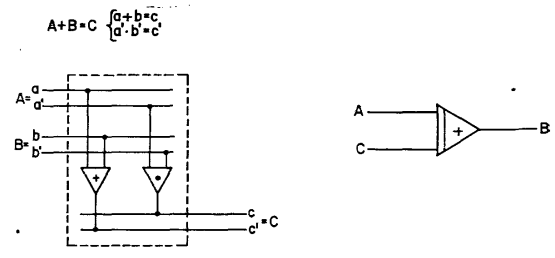


Fig. 5 (right). Two-line gate

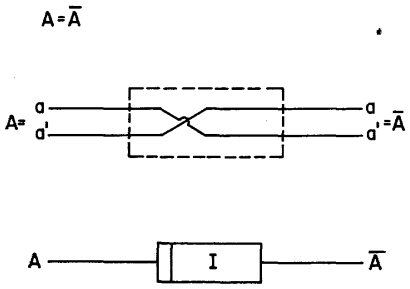


Fig. 6. Two-line negator

plished, particularly in large systems. In systems using this spike correction method, a situation called "races" or "dynamic hazard" exists which can be difficult to resolve and whose solution reduces the permissible circuit speed.

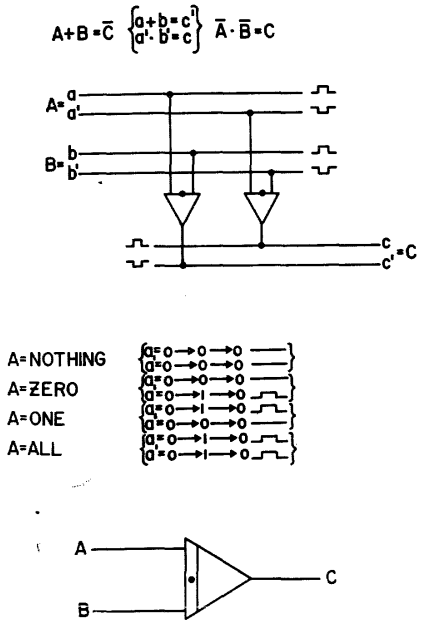
A more satisfactory solution to the spike problem would be to have a logical element which did not produce spikes. In Figs. 4, 5, and 6 are shown the well-known "2-line" notation, which imposes symmetry even in the inhibit case and results in spike-free operation. Two-line notation is shown in Fig. 3, there being four states for binary signals taken in pairs. The zero—zero case is defined as nothing, zero—one as 0, and one—zero as 1 and 1—1 as all or excluded. The signal forms and transitions are shown for these cases.

The 2-line "or" circuit of Fig. 4 consists of two logical elements, an "and" and an "or" circuit. Inputs A and B each have two lines, $a-a'$ and $b-b'$. Signals on (a) and (b) pass through the "or" circuit to output (c') while inputs (a') and (b') operate through the "and" circuit to output (c). It performs the function $A+B=C$ as $a+b=c'$ and $a'.b'=c'$. The proposed symbol for the 2-line "or" element is shown.

A 2-line "and" gate is shown in Fig. 5, together with the 2-line "and" symbol. The "and" circuit is similar to the "or" circuit, except that the connections are reversed to perform $A.B=C$ as $a.b=c$ and $a'+b'=c'$. Negation is accomplished by inversion simply by crossing over the a and a' lines as shown in Fig. 6.

A 2-line stroke element or buffer-complementer is shown in Fig. 7. It is constructed with two single-line stroke elements, the signals on the lines a and b being turn-on pulses and those on lines a' and b' being turn-off pulses. Also shown in Fig. 7 is a truth table for the 16 states of the element. The permissible states are 1 through 3, 5 through 7, and 9 through 11. The other states are excluded by definition since the 1—1 "all" case is not used, and, indeed, serves as a check, as will be described later.

It is a requirement of 2-line circuits that the signals return to "nothing" (line 1) between information. Input signals arrive on a or a' and on b or b'. Line 6 states the "and" operation, and it will be noted that the setup and drop-out conditions can pass only through lines 2 and 5, both of which give intermediate outputs of nothing. When operating as an "or" element, however, a "nothing" input does not inhibit, as can be noted in lines 3 and 9.



CASE	A a a'	B b b'	C c c'	
1	0 0	0 0	0 0	NOTHING
2	0 1	0 0	0 0	UNDETERMINED
3	1 0	0 0	0 1	OR
4	1 1	0 0	0 1	EXCLUDE
5	0 0	0 1	0 0	UNDETERMINED
6	0 1	0 1	1 0	AND
7	1 0	0 1	0 1	OR
8	1 1	0 1	1 1	EXCLUDE
9	0 0	1 0	0 1	OR
10	0 1	1 0	0 1	OR
11	1 0	1 0	0 1	OR
12	1 1	1 0	0 1	EXCLUDE
13	0 0	1 1	0 1	EXCLUDE
14	0 1	1 1	1 1	EXCLUDE
15	1 0	1 1	0 1	EXCLUDE
16	1 1	1 1	1 1	EXCLUDE

Fig. 7. Two-line stroke element (buffer complimenter)

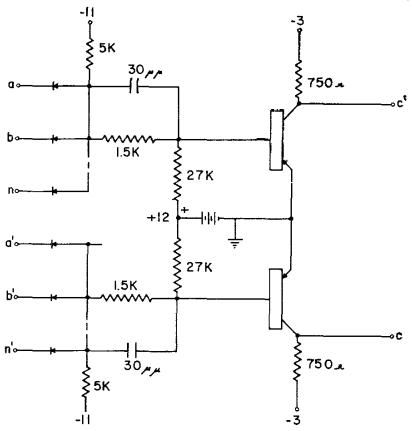


Fig. 8. Two-line stroke circuit

It is evident that the "not" gate is identical with the "and" gate; all 16 input states are shown, and inhibit operation is obtained by gating B against A. If $\bar{A}$ is zero and B is zero, the output is one. But if $\bar{A}$ is one, the output is zero, independent of the B input. In all permissible operations, the element is spike-free.

The circuit of a stroke element is shown in Fig. 8. It consists of two SB-100 transistors operating in the grounded emitter connection. Each amplifier is a complementer with a cluster of "or" diodes on its input. The fact that the system is spike-free allows the circuit time constants to be optimized for the signal bandpass without compromise for spikes. If spikes were present, it would be necessary to choose a value for the speed-up condensers such that the circuit delay for spikes would be the same as the delay for signals. Obviously, if this is not done, spikes may fall in the clock sampling period. But if spikes are not present, it is sufficient to optimize for signals alone, and an improvement in power gain or bandwidth of 25% is realized.

Two more elements are needed to construct a time-independent or autosynchronous system. These elements are: A 2-line flip flop and a 2-line checker. The flip flop is shown in Fig. 9 and consists, in fact, of two single-line flip flops since two bits must be stored. The circuit provides direct and complement outputs and for a clear to "nothing." The 2-line checker of Fig. 10 tests the lines for signals of "nothing," "something" (one or zero), and "error" (the excluded case 1—1).

It should be emphasized that all of these elements operate in "return-to-nothing" notation, each line operating in "return-to-zero." In this mode of op-

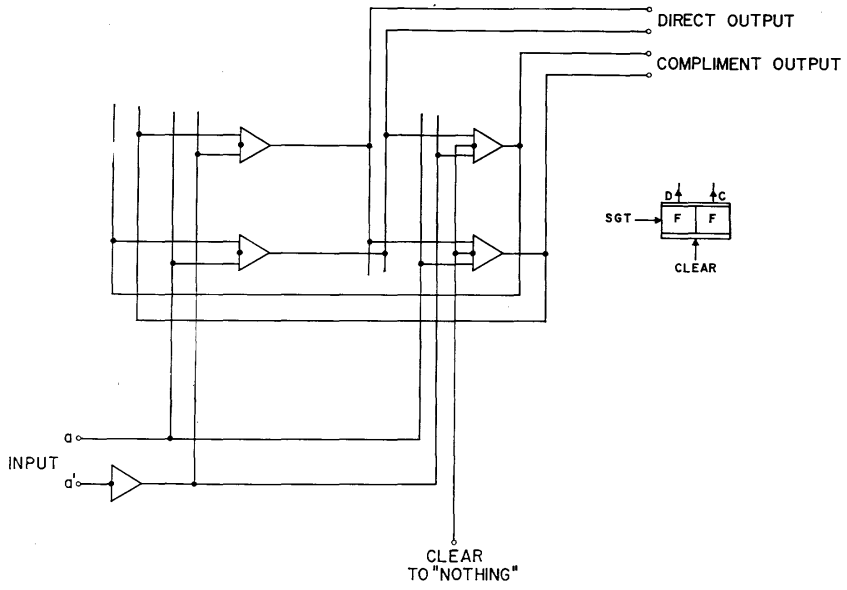


Fig. 9. Two-line flip-flop

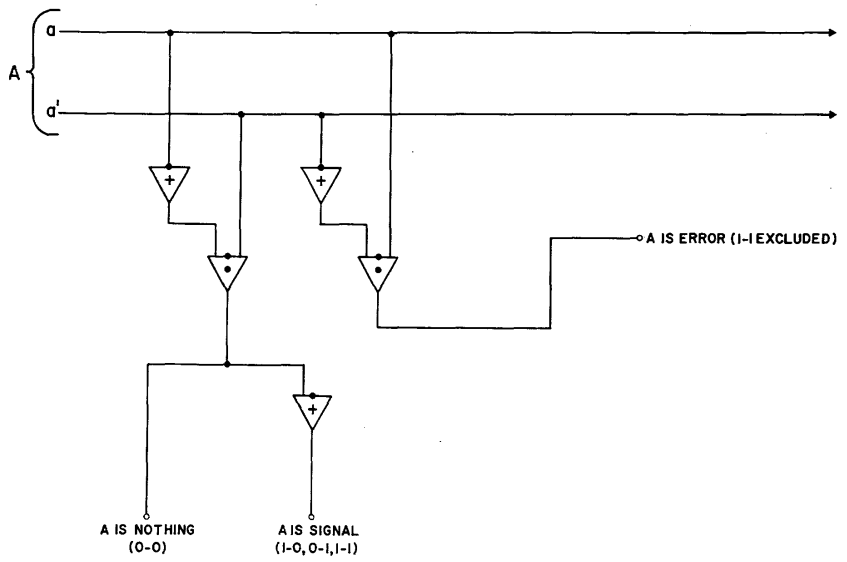


Fig. 10. Two-line checker

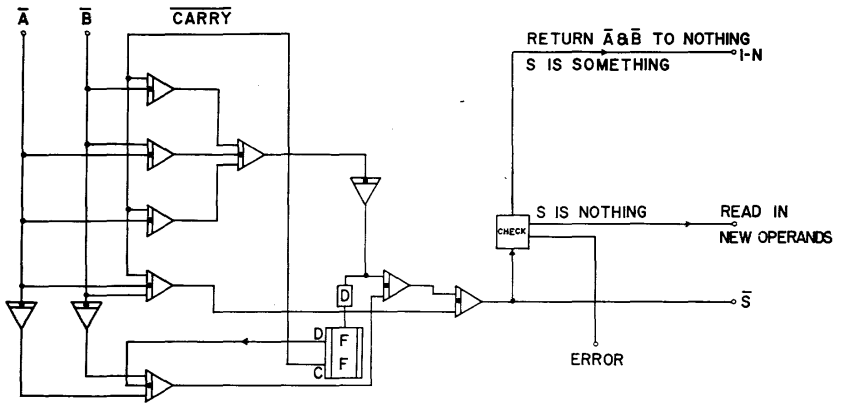


Fig. 11. Two-line adder

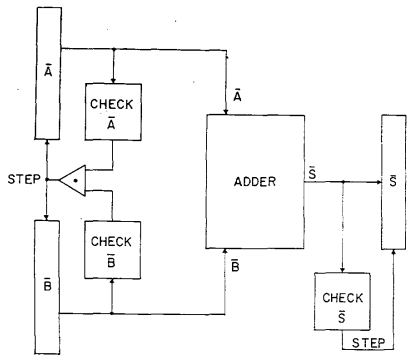


Fig. 12. Autosynchronous equivalent of Fig. 1

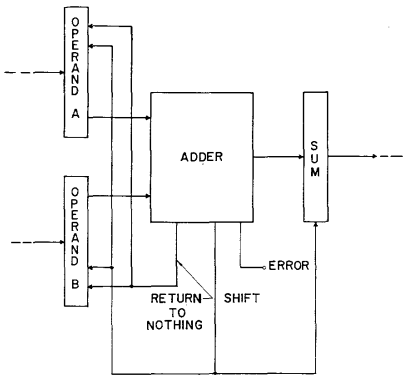


Fig. 13. Autosynchronous adder

eration they are spike-free. There are no known elements that are spike free in "non-return-to-zero."

Spike-Free Logical Networks

Using the elements first described, one can construct a serial adder, and this adder will take the classical 5-level form shown in Fig. 11. It will be noted that essentially no change in logical form is required, and the new 2-line elements can be substituted directly for single-line elements in existing logical structures. Of course, it is necessary to provide a logical clock to replace the conventional oscillator, and the 2-line checker executes this function.

In the serial adder, the first operand bits $\bar{A}$ and $\bar{B}$ are propagated into the network to initiate the process. When a sum output appears, the checker sends a pulse to the operand registers, returning the inputs to "nothing." This 0−0 condition then propagates through the net, and on appearing at the checker, sends a shift pulse to the registers to read in the next operand bits, and shifts the sum register for receipt of the result. This sequence proceeds in a self-synchronized manner until the addition is complete.

The speed is determined only by the circuit delays. This process, illustrated in Fig. 12, is the autosynchronous equivalent of the general system of Fig. 1.

At this point it would be in order to review critically the new organization, comparing it against the equivalent synchronous system. The first thing to note is that the amount of hardware has been doubled. Every logical element is push-pull and every storage holds twice as many bits. Has an improvement in performance which justifies the increase been achieved?

If this adder were a conventional single-line synchronous device, it would operate at 2.8 mc in non-return-to-zero. One would have to allow 200 μsec for the network delay and another 150 μsec for storage delay and clock jitter. Using autosynchronous elements optimized for signals, there is an average delay of around 15 μsec per level. The total network delay around the loop will be 135 μsec for signals and 120 μsec for return to "nothing" for a total cycle time of 255 μsec or 4 mc. The 2-line system is therefore a factor of $1^2/_3$ faster.

The 2-line system has other important advantages. First, it is inherently self-checking, and redundant logic and parity checks can be eliminated. The clock and distribution system is eliminated. One practical feature of considerable importance is that the power supply loading is constant and line transients are minimized. The autosynchronous system, which is nearly twice as fast, requires less than twice as much hardware since, in a typical computer, the checking circuits, redundant adder, and clock involve about one third of the equipment. There is a cost-to-performance ratio for the two systems that is about equal.

It will be noted that the discussion has been limited to serial circuits and logic. This is no accident. The advantages of 2-line autosynchronous logic cannot be fully realized in a parallel system. When a logical network has many parallel related paths, the slowest path provides the determining delay. The larger the number of paths, the greater the probability that one path will have a delay approaching the limiting worst case. It is conceivable that a parallel asynchronous system may be no faster than its synchronous equivalent. Indeed, it can be said that parallelism is never a pay-off from a performance-to-cost ratio standpoint and is only justifiable from the point of view of timeliness.

It is, therefore, ironic that most of the work to date on asynchronous system has been done in connection with large parallel logic computers. The effort, of course, has been to attain the highest possible speed of the equipment. For example, the serial adder described previously performs a full 50-bit addition including carry propagation in $12^1/_2$ μsec and with only 32 transistors. A full parallel asynchronous adder with its carry net and controls may be 10 to 20 times faster, but involves over 100 times as much equipment.

There is a philosophical lesson to be learned here. Prior asynchronous devices have been structurally parallel but logically serial; i.e., the machines have handled data words in a parallel manner, but instructions have been handled serially. It is felt that autosynchronous equipment should be structurally serial but logically parallel.

In the system described in Fig. 12, the checker is on the output line of the logical network. It therefore operates so that only valid results can be generated. Where speed is not essential, this is the preferable procedure. If higher speed is required, the checker can monitor the input lines to propagate signals and a further checker can monitor the output lines both for validity and to step the output storage. This has the disadvantage that information stored in the network may be lost in the event of an error. This system is shown in Fig. 13.

Since the checker has three levels of logic for "information" and two levels for "nothing," and assuming one delay for readout from storage, the readout period will be $6 \times 15 = 90$ μsec per bit or 11 mc. The delay in the adder no longer determines the bit rate, and the logic can have as many levels as desired provided the wave shapes can be preserved. Since the signals are, by definition, "return-to-zero," a-c coupled pulse forming amplifiers can be used and the logical chains can be indefinitely long. In such a network it will be necessary to include adjustable compensating delays so that all paths through the network have the same transit time. It should be pointed out, however, that this is different from the old "race" or "dynamic hazard" problem since changes in circuit time constants will not produce spikes or wrong answers, but will only cause the equipment to stall. It follows that the adder of Fig. 13 will perform the 50-bit addition in $4^1/_2$ μsec, which begins to compare with a parallel synchronous adder.

One final point should be made. The synchronization problems in clocked machines are such that they determine the limiting speeds. As the frequency of operation rises, wiring delays, both in the network and the clock lines, become comparable to the pulse rate. Synchronous operation much above 5 mc becomes difficult, even with very fast low delay amplifiers. The autosynchronous approach not only raises possible operating speeds, but reduces the amount of equipment required.

Conclusions

The autosynchronous logic and circuits that have been described are capable of higher performance than their synchronous counterparts. The new logic is self-checking and inherently more reliable, but the technique can be applied to existing logical networks by substitution of 2-line elements for single-line elements, the clock being replaced by information checking or test elements. The advantages of the autosynchronous approach become greater as new short delay amplifiers become available and as clock tolerances and wiring delays in conventional networks become the principal considerations. It is hoped that the autosynchronous concept can contribute to improved reliability and speed in future computing devices.

References

1. The Transistor NOR Circuit, **W. D. Rowe.** *AIEE Conference Paper 57-195.*

2. A New Method of Designating Low Level, High Speed Semi-Conductor Logic Circuits, **W. B. Cagle, W. H. Chen.** *Westcon Convention Record* (Circuit Theory), Institute of Radio Engineers, New York, N. Y., Aug. 1957.

3. On the Length of The Longest Carry in Binary Addition, **Herman H. Goldstine, James H. Pomerene.** *Ordnance Computer Research Report*, vol. II, no. 4, Institute of Advanced Studies, Ballistic Research Laboratories, Aberdeen Proving Ground, Md., Oct. 1955, pp. 23–26.

4. Method for Synthesizing Sequential Circuits, **George H. Mealy.** *Bell System Telephone Journal*, New York, N. Y., vol. 34, Sept. 1957, pp. 1045–79.

5. The Design of Hazard-Free Switching Networks, **D. A. Huffman.** Journal, Association for Computing Machinery, Baltimore, Md., vol. 4, no. 1, Jan. 1957.

Discussion

Question (Bell Telephone Laboratories): As far as I know, you gentlemen have invented the word "autosynchronous." I just want to be sure that I know what you mean by it.

Do you mean by autosynchronous, any system in which clock pulses are generated by checks of some sort on the order of computing circuits, such that the clock does not run at a steady rate, or do you mean specifically what you have described in the paper—the not returning to zero features?

Mr. Sims: I think your first definition is

closest to what we have had in mind. That is, that an autosynchronous device is one in which there are essentially regenerative impulses which, by sensing information flowing in the network, can use this sensing to propagate new information.

Question: There was something I did not understand. I did not quite understand about your difference between parallel operation and the serial operation as far as—well, let us say you spoke about using, possibly adding, one bit at a time.

This could speed up multiplication, but what about addition? Would it not increase the delay or the length of time for addition?

Mr. Sims: The question was what I had intended in discussing the use of sequential logic and parallel operations in present machines.

I said this was the way in which most computers had been built up to this time, and I said that for autosynchronous or asynchronous equipment, it looked as if one should use a lot of independent serial arithmetic units, rather than a single parallel one, because you would get more work done for your money this way.

Question: Wouldn't this increase the amount of equipment rather than decrease the amount of equipment for one parallel?

Mr. Sims: No. I think this follows from the fact that a parallel adder has perhaps one hundred times as much equipment as a serial adder, and yet is only perhaps ten times as fast.

Therefore, when you put in, say, ten serial adders which can make ten serial additions and keep up with my synchronous (ten times as fast) parallel adder, I can do this for one-tenth of the price.

Question: Have you considered the problem of locating faults by diagnostic checks?

Mr. Sims: I think as far as locating troubles are concerned, it is to some extent simplified since the propagation of information depends on the propagation of prior information, and since each bit is checked by a little checking arrangement on the output.

Then, a fault of any kind causes the equipment to stall right at the commission of the error, and I would think that suitable indicator lights would tell you where the fault occurred.

Analysis of TRL Circuit Propagation Delay

W. J. DUNNET E. P. AUGER A. C. SCOTT

Synopsis: A program to design transistor-resistor logic (TRL) circuits and compile TRL propagation delay tables on a digital computer is presently underway at the Sylvania Data Processing Laboratory. This paper points to the need for such a program and describes transistor and TRL circuit studies that have resulted in the basic relationships being programmed.

The criterion of TRL circuit performance is taken to be signal propagation delay. This delay can be predicted by applying the transistor large-signal transient expressions of Ebers and Moll.

The accuracy and usefulness of these expressions are increased by measuring pertinent transistor parameters under large signal transient conditions and by considering the practical case where propagated signals have finite, rather than step, rise, and fall times.

COMPLETE knowledge and control of TRL building block performance is necessary to predict the performance of a large data processing system constructed using TRL logic. (The TRL building block is frequently referred to as the NOR circuit.) Control of the circuit performance permits confident application of the circuit thousands of times over, in both controlled and uncontrolled environments.

One of the most important limiting characteristics of a TRL circuit is its propagation delay. Propagation delay is defined as the time required, after application or removal of an input signal, for the TRL transistor output level to begin to change. The magnitude of propagation delay depends upon:

1. The transient performance of the transistor.
2. The way the TRL stage is wired into the system and the number of inputs being energized (circuit environment).
3. The state of the transistor (conducting or nonconducting) prior to the application or removal of an input signal.

Under the section headed Propagation Delay, transistor input capacity transistor rise, decay and storage times and their relationship to propagation delay are discussed. A procedure is outlined for calculating the propagation delay for a given transistor and circuit configuration.

Later, TRL circuit environment is considered and general expressions relating turn-on and turn-off circuit betas to circuit parameters (supply voltages, input and collector resistors) and transistor steady state voltages, V_{BE} and V_{CE} (saturated), are developed.

Finally, relationships developed in the paper that permit prediction of TRL propagation delay are summarized.

Introduction to Transistor-Resistor Logic

The TRL circuit consists of a resistor "or" gate followed by an inverting transistor amplifier. Fig. 1 shows two TRL circuits arranged in cascade. In this figure, the transistor parameters and circuit values are chosen so that the collector voltage of the off transistor Tr−1, is of sufficient magnitude to keep Tr−2 in saturation. If the collector voltage of an "off" transistor is defined as a "1" and the collector voltage of an "on" transistor is defined as a "0," then the logical performance of the TRL circuit can be described as follows. Referring again to Fig. 1, if neither inputs 1, nor 2, nor 3, are present, the collector voltage of Tr−1 will be at "1" level. If any one or more inputs are present, the collector voltage will be at "0" level. Any binary-digital logical function can be performed by combinations of the basic TRL circuit. For example, Fig. 2 shows a TRL flip-flop and Fig. 3 a TRL adder logic.

Compared to other popular types of logic, for example current mode[1] and direct-coupled transistor logic (DCTL)[2], TRL is more economical and reliable due primarily to its lower transistor count and simplicity. The one disadvantage of TRL is that it does not fully exploit the speed capabilities of the transistor.

As previously mentioned, the transistor parameters and circuit values of a TRL circuit are chosen so that the collector voltage of an "off" transistor is sufficient to keep a driven transistor in saturation. However, it is not meant to imply that a TRL circuit is designed on the basis of steady state conditions alone. Generally speaking, excessive propagation delay occurs well before a TRL circuit fails to meet steady state requirements. As a result, TRL circuit design is governed primarily by propagation delay considerations.

W. J. DUNNET, E. P. AUGER, and A. C. SCOTT are with Sylvania Electronic Systems, Division of Sylvania Electric Products Inc., Needham, Mass.