TABLE III

LOWER BOUNDS OF SINGLE ERROR-CORRECTING *BN* MODULO *A* CODES

m	a	j	n
7	19	5	2
8	19	5	3
10	23	5	5
12	29	5	7
13	29	5	8
15	37	6	9
16	37	6	10
17	37	6	11
19	47	6	13

TABLE IV

LOWER BOUNDS OF DOUBLE ERROR-CORRECTING *BN* MODULO *A* CODES

a	j	n	m	Optimal Codes*
171	8	1	9	
205	8	2	10	*
557	10	2	12	
565	10	4	14	
941	10	5	15	*
1069	11	5	16	
1417	11	7	18	
1939	11	8	19	*

In the previous section it was shown that a double error-correcting *BN* modulo *A* code is a subclass of double error-correcting *AN* codes if $b=-2^j$, where $j=[1+\log_2 a]$ is the bit length for the check symbol. The computational algorithm for generating such codes is similar to that for generating double error-correcting *AN* codes [5]. We describe this algorithm as follows.

1) Pick up all a's such that $\|a\| \geq 5$ by sorting the table of norms of the natural numbers.

2) Compute the least positive integer N_a such that $\|a \cdot N_a\| \leq 4$. Then set the following.

3) $m=[\log_2(1+a \cdot N_a)]$, $j=[1+\log_2 a]$ and $n=[\log_2(1+a \cdot N_a)]-[1+\log_2 a]$.

Some computed data and the optimal codes are shown in Table IV.

ACKNOWLEDGMENT

The authors would like to thank one of the referees for pointing out errors in the manuscript and his thorough criticism.

REFERENCES

[1] D. T. Brown, "Error detecting and correcting binary codes for arithmetic operations," *IRE Trans. Electron. Comput.*, vol. EC-9, pp. 333–337, Sept. 1960.
[2] E. I. Goldberg and W. E. Webb with Sections II and III by N. Zierler and I. S. Reed, "Application of *BN* modulo *A* codes to aerospace vehicle control systems," Rep. ASD-TDR-62-169, July 1962.
[3] W. W. Peterson, "On checking an adder," *IBM J. Res. Develop.*, vol. 2, pp. 166–168, 1958.
[4] ——, *Error-Correcting Codes*. Cambridge, Mass.: M.I.T. Press, 1961, ch. 13.
[5] A. C. L. Chiang and I. S. Reed, "Arithmetic norms and bounds of the arithmetic *AN* codes," *IEEE Trans. Inform. Theory*, vol. IT-16, pp. 470–476, July 1970.
[6] T. R. N. Rao and O. N. Garcia, "Cyclic and multiresidue codes for arithmetic operations," *IEEE Trans. Inform. Theory*, vol. IT-17, pp. 85–91, Jan. 1971.
[7] Y. G. Dadayev, "Arithmetic divisible codes with correction for independent errors," *Eng. Cybern.*, vol. 6, 1965.
[8] I. Niven and H. S. Zuckerman, *An Introduction to the Theory of Numbers*. New York: Wiley, 1966.
[9] A. Cunningham, H. J. Woodall, and T. G. Creak, "On least primitive roots," *Proc. London Math. Soc.*, vol. 21, pp. 343–358, Mar. 1952.

New Results for Rado's Sigma Function for Binary Turing Machines

DONALD S. LYNN, STUDENT MEMBER, IEEE

***Abstract*—A computer program was written and executed to search for better lower bounds to Rado's noncomputable sigma and shift functions for binary Turing machines. Former results in this search (called by Rado the Busy Beaver logical game) are reviewed and new bounds found by this program are presented.**

***Index Terms*—Binary Turing machine, Busy Beaver logical game, halting problem, noncomputable functions, Rado's sigma function.**

Rado defined and proved to be noncomputable two functions related to what he called the Busy Beaver problem in [1]. He defined a binary Turing machine and showed that exactly $[4(n+1)]^{2n}$ such machines exist with a given number of states n. Among these machines will be found some that halt when started on an empty (all zero) tape and some that continue to run forever. Let E_n denote the set of all such machines of n states that halt when started on an empty tape. Rado's first noncomputable function $\Sigma(n)$ is defined as the maximum number of ones written on a Turing machine tape (when the machine halts) by any member of the set E_n. Rado showed [1] this function to be noncomputable by showing that it must grow with increasing n faster than any computable function.

The noncomputability of $\Sigma(n)$ is closely related to the Turing machine halting problem. One is not able to tell (by computable or algorithmic methods) for all n which of the $[4(n+1)]^{2n}$ possible machines halt and which do not. Hence the membership of E_n cannot in general be determined, and the machine in the set E_n that produces the most ones cannot be found. This does not preclude the possibility of certain values of this function, $\Sigma(1)$ for instance, being discovered or calculated.

The second noncomputable function $S(n)$ is the maximum number of moves (shifts) made by a machine in the set E_n. If the function $S(n)$ were computable, then one could compute $\Sigma(n)$ by simply running all possible n-state machines until they halted or reached $S(n)$ moves.

$\Sigma(1)$ and $S(1)$ can both be found by inspection to be one (since Rado allowed printing a one on the move that halts). $\Sigma(2)$ was known to be four when Rado proposed

Manuscript received October 8, 1971; revised February 22, 1972.
The author is in the Computer Science Program, University of Southern California, Los Angeles, Calif. 90007.

the Busy Beaver problem; that problem is to find halting binary Turing machines that produce more ones or make more moves than previously known machines, and thus raise the known lower bound on $\Sigma(n)$ and $S(n)$.

In [2], Lin and Rado gave the results of a study that showed $\Sigma(3)$ to be 6 and $S(3)$ to be 21. All different three-state machines were examined by computer or by hand to determine which halted and which displayed a looping behavior that definitely would not halt.

Brady [3] concluded after a nearly exhaustive examination of four-state machines that $\Sigma(4)$ is probably 13 and $S(4)$ is probably 107 (counting the move that halts). As with the two- and three-state cases, the machines were published in support of these claims. Even though the number of possible machines has become an astronomical figure at four states, Brady was able to nearly exhaust the four-state machines by examining only what he calls tree-normal machines. By establishing certain rules by which to generate these tree-normal machines, Brady drastically reduced the number of machines to be examined and yet established that all non-tree-normal machines are either equivalent to or no better than some tree-normal machine. But a small percentage of four-state tree-normal machines evaded Brady's attempts to determine whether they halted or not. Hence his numbers remain lower bounds on $\Sigma(4)$ and $S(4)$ rather than established values.

Green [4] defined a four- and five-state machine along with rules for adding groups of two more states to each. He proved that these classes of machines halted when started on empty tapes and derived the number of ones $BB(n)$ produced in a few cases.

$$BB(4) = 7$$
$$BB(5) = 13$$
$$BB(6) = 35$$
$$BB(7) = 22961$$
$$BB(8) = 3(7 \cdot 3^{92} - 1)/2.$$

Green states that better machines are known than his for up to five states, but that his machines are the best known for six or more states. Green mentions that a 17 one five-state machine is known, but does not produce it.

This note presents a program that searches tree-normal machines for better lower bounds on the Σ and S functions than those mentioned above. The only criteria in this program for giving up on a machine (and assuming that it must run forever) are that it uses more tape or that it makes more moves than certain specified limits. This decision method was simple enough to make a fast running program, but could result in missing some valid halting machines.

The program reproduced the best known Turing machines for two, three, and four states. Using an IBM 360 model 65 for this program was enough faster than previous work (Brady used an SDS 920 for his program) that some progress could now be made toward exhausting the five-state problem. There were about 150 times as many four-state machines examined as three-state machines, but it took about 400 times as long for the program to run because four-state machines tended to take longer to simulate. It is reasonable to assume that the five-state case would take at least 400 times longer than the four-state case, and thus would run at least 200 h. Hence the program was run for five states over only a few portions of all possible tree-normal machines. These portions included several initial moves found to be good for the four-state machines.

Present State	Character read 0	1
1	2,1,R	3,1,R
2	3,1,L	4,1,L
3	1,0,R	2,1,L
4	5,1,R	2,0,L
5	halt,1,R	4,1,R

$1011111111111\underline{1}111111111$

Present State	Character read 0	1
1	2,1,R	3,0,R
2	3,1,L	2,0,L
3	4,1,R	2,1,L
4	5,1,R	1,0,R
5	2,0,R	halt,1,R

$11110011111111\underline{1}1000011111111$

Fig. 1. Turing machines making 292 moves (left) and 217 moves and producing 22 ones.

Present State	Character read 0	1
1	2,1,R	1,1,R
2	3,1,L	4,0,L
3	1,0,R	2,1,L
4	halt,1,R	5,0,L
5	3,1,R	2,1,R

$10110110110110\underline{1}101111$

Fig. 2. Turing machine making 435 moves.

Many machines were found to produce more than the previously known high of 17 ones. The best were the two machines given in Fig. 1 which each produce 22 ones. The tables in the figure give next state, character written on Turing machine tape, and left or right (L or R) move, in that order. State number one is the start state in all figures. The final tape pattern for each machine is given beneath its table, with an underscore indicating the starting position. The greatest number of moves found was 435 for the machine in Fig. 2. It produced only 15 ones however.

All five-state runs were set so that 500 moves would be made before abandoning each machine as nonhalting, but the machine of Fig. 2 and one halting on the 317th move were the only two that halted after 300 moves or more.

Present State	Character read 0	1
1	2,1,R	2,0,L
2	3,1,L	halt,1,R
3	4,0,L	3,0,L
4	5,1,L	1,0,R
5	6,0,L	5,0,L
6	6,1,R	4,1,R

111111111111111111111111111111111011111<u>1</u>111

Fig. 3. Turing machine producing 42 ones in 522 moves.

The program, as run on the model 65, would probably take months or years of computing time to examine the six-state case. Hence another approach was tried. During the run for the four-state case, all machines of seven ones or more were printed. These were examined for machines showing the type of loops that Green's class of machines display. Nine were found, including the one Green proposed. Upon expanding to six states, two of these machines surpassed the known maximum of 35 ones. The better of these machines is given in Fig. 3.

No promising machines similar to Green's were found in the few hundred five-state machines printed out. Also the two six-state machines mentioned above did not produce all their ones adjacent to each other, and so were not amenable to expansion to eight or more states by Green's method. Therefore, this investigation ended at six states. The new lower bounds found for Rado's noncomputable functions are

$$\Sigma(5) = 22 \qquad S(5) = 435$$

$$\Sigma(6) = 42 \qquad S(6) = 522.$$

References

[1] T. Rado, "On non-computable functions," *Bell Syst. Tech. J.*, vol. 41, pp. 877–884, May 1962.

[2] S. Lin and T. Rado, "Computer studies of Turing machine problems," *J. Ass. Comput. Mach.*, vol. 12, pp. 196–212, Apr. 1965.

[3] A. H. Brady, "The conjectured highest scoring machines for Rado's $\Sigma(k)$ for the value $k=4$," *IEEE Trans. Electron. Comput.* (Short Notes), vol. EC-15, pp. 802–803, Oct. 1966.

[4] M. W. Green, "A lower bound on Rado's sigma function for binary Turing machines," in *Proc. IEEE 5th Annu. Symp. Switching Circuit Theory and Logic Design*, Oct. 1964, pp. 91–94.

A Flexible Rate Multiplier Circuit with Uniform Pulse Distribution Outputs

R. M. M. OBERMAN

***Abstract*—In digital integrated circuits (IC's) a synchronous binary-rate multiplier is commercially available, generating a programmable number of output pulses during each internal counting cycle having a length equal to a power of two. These output pulses are not equally divided over that fixed internal counting cycle and the generated binary rate has a fixed denominator.**

In this note a new type of binary-rate multiplier is described. This circuit allows the programming of both numerator a and denominator b of the rate of the numbers of output and input pulses. It operates in such a way that during a cycle of b pulses the circuit output generates a number of pulses divided over the cycle of b pulses as equally divided as is possible in digital systems.

***Index Terms*—Applications in arithmetic and conversion operations (as A/D and D/A conversion), division rate of integer numbers, fixed-rate or variable-rate frequency division, programmable logical output circuit, synchronous binary counter.**

Manuscript received December 6, 1971; revised February 23, 1972.

The author is with the Department of Electrical Engineering, Delft Technical University, Delft, The Netherlands.

I. The Problem

In Fig. 1 the symbolic diagram of a four-bit synchronous binary-rate multiplier has been shown. This is the trade name used in the integrated circuit (IC) manufacturer's catalog [1]. In literature its name is discrete multiplier [2]. Fig. 1 consists mainly of a four-bit synchronous binary counter with sections A, B, C, and D with weights 1, 2, 4, and 8, respectively, and a logical circuit with AND gates A_1–A_5 inclusive and output OR gate O_1. The time chart of the operation of this counter is shown in the upper five lines of Fig. 2.

Gate A_1 of the circuit is controlled by clock or counting pulse α and a strobe signal determining the beginning of the series of output pulses. In each of the gates A_2–A_5 inclusive, a series of pulses can be formed, determined by the following group of equations in which the corresponding input rate signal is represented by r_i.

$$\text{gate } A_2: \quad \alpha r_8 a' \tag{1}$$

$$\text{gate } A_3: \quad \alpha r_4 a b' \tag{2}$$

$$\text{gate } A_4: \quad \alpha r_2 a b c' \tag{3}$$

$$\text{gate } A_5: \quad \alpha r_1 a b c d'. \tag{4}$$

During each counting cycle of 16 pulses, gate A_2 will generate a sequence of 8 pulses if $r_8=1$, gate A_3 a sequence of 4 pulses if $r_4=1$, gate A_4 a sequence of 2 pulses if $r_2=1$, and finally, gate A_5 a 1 pulse if $r_1=1$. It follows from the time chart of Fig. 2 that these pulses are equally distributed in the main cycle of 16 pulses. They are shown in this figure on lines a_8, b_4, c_2, and d_1.

The outputs of gates A_2–A_5 inclusive are connected with the inputs of output gate O_1 so that if all rate inputs are characterized with 1 ($1111=15$) the output will show a series of 15 pulses for each input series of 16 pulses. A binary rate of 15/16 is then established between the series of output pulses and the series of input pulses. The number in the numerator of this binary rate can be programmed on the rate inputs. Any binary rate from 1/16 to 15/16 is possible.

It is of course possible to extend the number of sections of this type of binary-rate multiplier. But the denominator of the binary rate will always be a power of two.

The output pulse series for a binary rate of 5/16 is, e.g., shown on line o of Fig. 2. It needs no further explanation that this series of output pulses is irregularly