Let analogously (14)–(16)

$$\hat{f}_0^{((\beta_0,\cdots,\beta_{m-1}),(\alpha_0,\cdots,\alpha_{m-1}))}(\omega_0,\cdots,\omega_{m-1}) = S_f^{((\beta_0,\cdots,\beta_{m-1}),(\alpha_0,\cdots,\alpha_{m-1}))}(\omega_0,\cdots,\omega_{m-1}) \quad (20)$$

$$\hat{f}_1^{((\beta_1,\cdots,\beta_{m-1}),(\alpha_1,\cdots,\alpha_{m-1}))}(x_0,\omega_1,\cdots,\omega_{m-1}) = \sum_{R_{\omega 0}} \sum_{\alpha_0,\beta_0} \hat{f}_0^{((\beta_0,\cdots,\beta_{m-1}),(\alpha_0,\cdots,\alpha_{m-1}))}(\omega_0,\cdots,\omega_{m-1})\, R_{\omega 0}^{(\alpha_0,\beta_0)}(x_0) \quad (21)$$

$$\cdots$$

$$\hat{f}_l^{((\beta_l,\cdots,m-1),(\alpha_l,\cdots,\alpha_{m-1}))}(x_0,\cdots,x_{l-1},\omega_l,\cdots,\omega_{m-1}) = \sum_{R_{\omega l-1}} \sum_{\alpha_{l-1},\beta_{l-1}} \hat{f}_{l-1}^{((\beta_{l-1},\cdots,\beta_{m-1}),(\alpha_{l-1},\cdots,\alpha_{m-1}))} \cdot (x_0,\cdots,x_{l-2},\omega_{l-1},\cdots,\omega_{m-1})\, R_{\omega l-1}^{(\alpha_{l-1},\beta_{l-1})}(x_{l-1}), \quad (l=1,\cdots,m). \quad (22)$$

Then by (19)–(22)

$$f(x_0,\cdots,x_{m-1}) = \hat{f}_m(x_0,\cdots,x_{m-1}). \quad (23)$$

Formulas (20)–(23) defined the fast inverse Fourier transform (FIFT) on G.

The number of memory cells for storage $\hat{f}_l$ is g, and calculation of f_l by (22) involves $g_{l-1} \cdot g$ multiplications. The total number of multiplications for the FIFT is $g \cdot \Sigma_{j=0}^{m-1} g_j$; i.e., the same as for FFT.

V. Conclusions

We have considered the properties of Fourier transforms on finite non-Abelian groups, FFT, and FIFT algorithms for such groups and estimations for the complexity of these algorithms have been presented.

Acknowledgment

The author wishes to thank Dr. B. Moroz for useful discussions.

References

[1] M. G. Karpovsky and E. S. Moskalev, "Realization of a system of logical functions by means of an expansion in orthogonal series," *Automat. Remote Contr.*, vol. 28, pp. 1921–1932, 1967, (Translated from: *Automat. Telemekh.*, no. 12, pp. 119–129, 1967).

[2] B. L. N. Kennet, "A note on the finite Walsh transform," *IEEE Trans. Inform. Theory*, vol. IT-16, pp. 489–491, July 1970.

[3] M. G. Karpovsky and E. S. Moskalev, "Utilization of autocorrelation characteristics for realization of systems of logical functions," *Automat. Remote Contr.*, vol. 31, pp. 243–250, 1970 (Translated from: *Automat. Telemekh.*, no. 2, pp. 83–90, 1970).

[4] K. C. Andrews and K. L. Caspari, "A generalized technique for spectral analysis," *IEEE Trans. Comput.*, vol. C-19, pp. 16–25, Jan. 1970.

[5] G. Apple and P. Wintz, "Calculation of Fourier transforms on finite Abelian groups," *IEEE Trans. Inform. Theory*, vol. IT-16, pp. 233–236, Mar. 1970.

[6] R. J. Lechner, "Harmonic analysis of switching functions," in *Recent Development in Switching Theory*, A. Makhopadhyay, Ed. New York: Academic, 1971.

[7] M. G. Karpovsky and E. S. Moskalev, *Spectral Methods for Analysis and Synthesis of Digital Devices* (Monograph), "Energia" (in Russian), 1973.

[8] M. G. Karpovsky, *Finite Orthogonal Series in the Design of Digital Devices* (Monograph). New York: Wiley, and IUP Press, Jerusalem, 1976.

[9] J. Pearl, "Optimal dyadic models of time invariant systems," *IEEE Trans. Comput.*, vol. C-24, June 1975.

[10] V. E. Benes, "Optimal rearrangeable multistage connecting networks," *Bell Syst. Tech. J.*, vol. 43, pt. 2, pp. 1641–1656, July 1964.

[11] D. C. Opferman and N. T. Tsao-Wu, "On class of rearrangeable switching networks," *Bell Syst. Tech. J.*, vol. 50, pp. 1579–1618, May 1971.

[12] K. Harada, "Sequential permutation networks," *IEEE Trans. Comput.*, vol. C-21, pp. 472–479, May 1972.

[13] M. G. Karpovsky and E. A. Trakhtenberg, "Some optimization problems for generalized convolution systems over finite groups," *Informat. Contr.*, to be published.

[14] E. Hewitt and K. Ross, *Abstracts Harmonic Analysis* (Monograph), vol. II. Berlin: Springer, 1963.

The Skip-and-Set Fast-Division Algorithm

PANOS A. LIGOMENIDES

***Abstract*—A fast-division algorithm, effectively implementable with available IC technology, has been developed and implemented. Especially suited for asynchronous division units, the skip-and-set algorithm provides simple control provisions for the division of numbers in any format and to any fractional precision.**

***Index Terms*—Digital computer arithmetic functional unit, IC division circuit, zero-skipping fast-division algorithm.**

An improved zero-skipping fast-division algorithm has been developed and implemented. The basic characteristic of the skip-and-set division algorithm is that the quotient is formed by selectively setting only the 1 bits needed to form the quotient in very simple and fast itteration cycles. Thus, the number of itteration cycles for each division is limited to the number of 1 bits in the quotient.

This asynchronous division algorithm is implemented effectively with the available IC technology, because the operations included are easily implementable with commercially available IC chips. With simple control provisions it can perform division of two fixed or floating-point numbers to any desired degree of fractional precision, always providing the correct quotient and remainder. A 16-bit bench demonstration model was constructed and tested to show feasibility, to illustrate some special circuit features, and to provide measures of speed and cost. It is estimated that with ECL technology, where available, the average division time of two 30-bit normalized floating-point fractions is about 1.75 μs. For fixed-point operands of the same word length, the division time for two 30-bit unsigned numbers is about 1.0 μs.

Let $\langle\ \rangle$ designate "order of;" i.e., the position of the highest 1 bit in a binary word. Then, given the unsigned numbers N(dividend) and D(divisor), where $j_1 = \langle N\rangle - \langle D\rangle$ and $\langle 2^{j_1}D\rangle = \langle N\rangle$, we have $N{:}2^{j_1}D = 2^{-b_1} + (N - 2^{j_1-b_1}D){:}2^{j_1}D$, where $b_1 = 0$, if $N > 2^{j_1}D$, and $b_1 = 1$, if $N < 2^{j_1}D$. Then $N{:}D = 2^{J_1} + (N - 2^{J_1}D){:}D$, where $J_1 = j_1 - b_1$ and the first term of the quotient is 2^{J_1}; i.e. $2^{J_1} \rightarrow Q$. The remainder, $N - 2^{J_1}D$, becomes the new dividend and $2^{J_1}D$ is the new divisor, where $N - 2^{J_1}D < 2^{J_1}D$ and $j_2 = \langle 2^{J_1}D\rangle - \langle N - 2^{J_1}D\rangle = \langle N\rangle - \langle N - 2^{J_1}D\rangle - b_1$. The process is repeated until after the pth itteration we should have

Manuscript received April 29, 1976.

The author is with the Department of Electrical Engineering, University of Maryland, College Park, MD 20742.

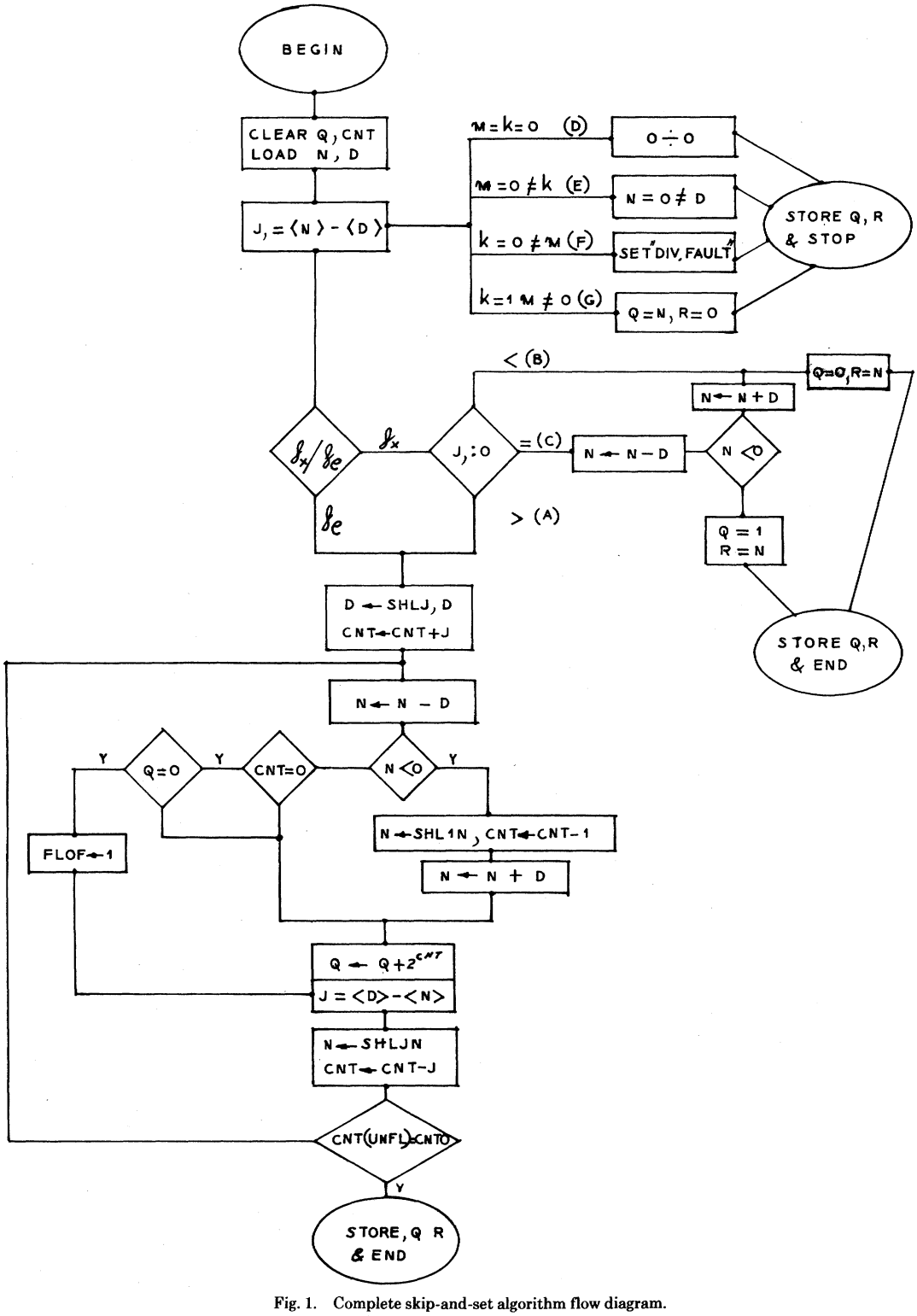

Fig. 1. Complete skip-and-set algorithm flow diagram.

$$N{:}D = 2^{J_1} + 2^{J_1-J_2} + \cdots + 2^{J_1-\sum_{i=2}^{p} J_i} + N - 2^{J_1}D - 2^{J_1-J_2}D - \cdots - 2^{J_1-\sum_{i=2}^{p} J_i}D/D$$

where the quotient is $Q = 2^{J_1} + 2^{J_1-J_2} + \cdots + 2^{J_1-\sum_{i=2}^{p} J_i}$, and the remainder left in the N register is $R = N - QD = N - 2^{J_1}D - 2^{J_1-J_2}D - \cdots - 2^{J_1-\sum_{i=2}^{p} J_i}D$.

The division algorithm suggested by the just given short mathematical derivation consists of finding successively the numbers $J_1, J_2, \cdots, J_p$ in p itteration cycles. The quotient and remainder are then formed as previously shown.

The complete skip-and-set division algorithm diagram is shown in Fig 1. Details of the mathematical derivation, the complete algorithm and flow diagrams, the circuit design and construction, with examples of applications and with actual IC wiring diagrams used for the bench model, are provided with the technical report "The skip and set division algorithm," Dept. Elec. Eng., Univ. of Maryland.

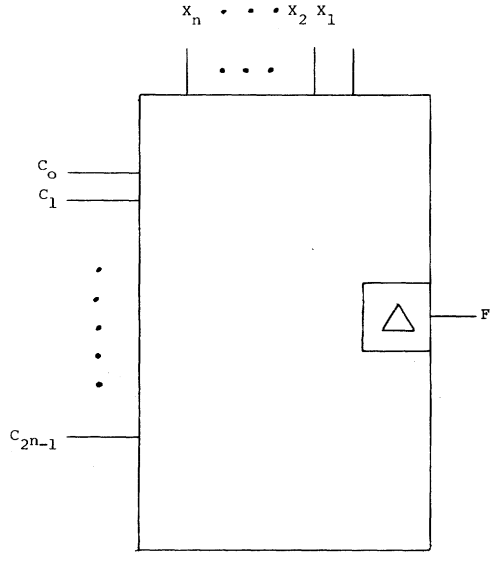

Fig. 1. Delayed ULM-n.

Minimum Universal Logic Module Sequential Circuits with Decoders

RAYMOND P. VOITH

***Abstract*—Universal logic modules of n variables (ULM-n) can be used to synthesize sequential machines. It has been shown that any machine can be realized using delayed ULM's and OR gates. Addition of an n-to-2^n decoder allows minimization of the number of ULM's necessary.**

***Index Terms*—Decoder, decomposition, multiplexer, sequential machine, universal logic module.**

I. Introduction

Recent articles by Newborn [1], Arnold *et al.* [2], Williams [3], and Le Van and Van Houtte [4] discuss realization of sequential machines by interconnection of delayed universal logic modules (ULM's). Newborn discusses uniform decompositions of a machine to allow realization using only ULM's. Williams extends the technique to incompletely specified machines. Arnold *et al.* present a technique that results in a highly regular structure using only ULM's. Le Van and Van Houtte present a technique using only ULM's and OR gates, in which each state is represented by a delayed ULM.

A ULM of n variables (ULM-n) is a module implementing the function

$$F = \sum_{i=0}^{2^n-1} C_i \cdot (i_2 * x) \tag{1}$$

where $x = (x_n \cdots x_1)$ is an n vector; Σ is inclusive OR; $\cdot$ is logic product; $C_i \in \{0,1\}$; i_2 is the binary representation of i; $*$ is an operator defined by the example

$$(10010) * (x_5x_4x_3x_2x_1) = x_5\bar{x}_4\bar{x}_3x_2\bar{x}_1.$$

A ULM-n cascaded with a delay element is used as a sequential circuit building block (see Fig. 1).

Le Van's method [4] can best be illustrated by a simple example. Given the state diagram in Fig. 2, each state is replaced by one ULM-2. Transitions are implemented by interconnecting the module outputs and inputs. Note that OR gates are used when one input symbol enters one state two or more ways. For more details see Le Van's paper [4].

Manuscript received December 1975; revised July 22, 1976.

The author is with the Department of Electrical Engineering, University of Toledo, Toledo, OH 43606.

II. Minimization

If a reduced sequential machine has p states, then there must be k binary state variables such that $2^k \geq p$. Therefore, using one delayed ULM-n for each state variable, rather than one for each state, should yield the circuit using the fewest number (k) of ULM-n's (n is the number of input variables). The interconnections are, of course, affected since there is no longer a one-to-one state-to-module mapping. Furthermore, each state is coded as a combination of the k state variables, and there is no longer a single module output for each state.

The one-to-one design procedure of Le Van [4] needs to be modified, but the modified procedure is straightforward. it is simplified by addition of a k-to-2^k decoder to provide a single line for each state. These lines are then fed back to the inputs of the ULM-n's (through OR gates where necessary) to complete the design. The basic structure is shown in Fig. 3. Note that the structure is regular, except for the OR gates at the inputs to the ULM-n's.

The design procedure is straightforward. Suppose that the states are coded arbitrarily. Then it is necessary to determine the inputs to each ULM-n. Each ULM-n will have 2^n inputs, which in general will come from an OR gate, as shown in Fig. 3. Examine one of the ULM-n's. Call its output y_i. Given some coding, the equation for y_i at time $t+1$ is

$$\begin{aligned} y_i(t+1) &= (S_b + S_c + \cdots + S_g)_{t+1} \\ &= (S_r + \cdots + S_p)_t \cdot I_0 \\ &\quad + (S_q + \cdots + S_s)_t \cdot I_1 \\ &\quad \vdots \\ &\quad + (S_m + \cdots + S_k)_t \cdot I_j \\ &\quad \vdots \\ &\quad + (S_x + \cdots S_\omega)_t \cdot I_{2^n-1}. \end{aligned} \tag{2}$$

where $S_b, S_c, \cdots, S_g$ are states for which $y_i = 1$ and I_j is one of the inputs.

The coding induces the first equality and the next state mapping induces the second equality. For (ULM-n)$_i$, (2) is all the information needed to specify its inputs. The ULM-n has an input C_j corresponding to each I_j (see Fig. 1). The input C_j is found from (2) to be $(S_m + \cdots + S_k)_t$. This is simply the OR of outputs of the decoder.