TABLE III
Flow Table of Example Machine

| | | next state | | output | |
|---|---|---|---|---|---|
| | input | 0 | 1 | 0 | 1 |
| state | (0,3,6) | (2,4) | (1,5) | b | a |
| | (1,5) | (0,3,6) | (2,4) | a | a |
| | (2,4) | (1,5) | (0,3,6) | b | a |

1 always goes into blocks containing states 3 and 4, and so forth. The flow table is then constructed following the order of the integers and the states in the partitions associated with them.

As mentioned above, obtaining a realization of an unknown machine is a reasonably straightforward process. No more storage is necessary for the algorithm than that needed to store the matrix $A(i,j)$. Note that all input sequences need not be applied in general. Each tape as in Table I may be processed sequentially in step 4), the algorithm halting when any row of the $A$ matrix contains all the states in $n$ blocks.

With $r$ being the number of input symbols, at most $(n)(r^{2n-1})$ states may be considered before halting at step 5) of the algorithm. If we view the results of steps 1)–3) as implicit in the structure for a given $(n,r)$ the time for the algorithm is directly related to the number of states to be considered.

The algorithm for multiple experiments considered here, with prior knowledge of a bound on states and input symbols as well as the complete reachability assumption, can be considered a worst case situation. In most practical situations, prior knowledge of the properties of the unknown machine may exceed these facts. As has been shown, the length of multiple preset experiments in this case increases as a factor $2n$ with multiplicity $r^{2n-1}$. In these terms, preset experiments are limited not by algorithmic complexity, but rather by running time and storage capacity.

## References


[1] M. A. Aiserman, L. A. Gusev, L. I. Rozonoer, I. A. Smirnova, and A. A. Tal, *Logic Automata and Algorithms.* New York: Academic Press, 1971.

[2] A. Gill, *Introduction to the Theory of Finite-State Machines.* New York: McGraw-Hill, 1962.

[3] S. Ginsburg, *An Introduction to Mathematical Machine Theory.* Reading, MA: Addison-Wesley, 1962.

[4] J. Hartmanis and R. E. Stearns, *Algebraic Structure Theory of Sequential Machines.* Englewood Cliffs, NJ: Prentice-Hall, 1966.

[5] E. F. Moore, "Gedanken experiments on sequential machines," in *Automata Studies,* C. E. Shannon and J. McCarthy, Ed. Princeton, NJ: Princeton Univ. Press, 1956, pp. 129–153.

[6] C. L. Sheng and S. R. Das, "Identification of synchronous sequential machines by merging of states," in *Southwestern IEEE Conf. Rec.,* 69C16-SWIEEECO, 1969, pp. 4F1–4F8.

[7] B. A.Trakhtenbrot and Ya. M. Barzdin, *Finite Automata: Behavior and Synthesis.* Amsterdam, The Netherlands: North Holland, 1973.


## Some Comments Concerning Design of Pipeline Arithmetic Arrays

J. C. MAJITHIA

***Abstract*—Cellular arrays for arithmetic operations usually consist of identical cells connected in an iterative or near iterative pattern. By introducing latch circuits between the rows of the array, the entire unit can be pipelined. The effect of this modification is to increase the throughput on a continuous processing basis. In most of such designs, however, the amount of hardware required for a maximally or fully pipelined array is prohibitively large. Pipeline arrays with reduced amount of intermediate latch circuits imply partially pipelined designs which of course also have a lower throughput. However, several such pipeline arrays can be operated in parallel to achieve some specified total throughput. In this correspondence this aspect is analyzed and illustrated by the design of 48-bit parallel adders.**

***Index Terms*—Cellular arrays, figure of merit, parallel processes, pipelining.**

Pipeline arrays for arithmetic operations have attracted considerable attention in recent years. Notably the effort has been in designing generalized arrays for operations such as multiplication, division, etc. [1]–[5]. These arrays usually consist of identical arithmetic cells or units connected in a simple iterative array pattern. Latch circuits providing temporary storage are introduced between the rows thereby achieving a pipeline design.

Analysis of such pipeline designs has primarily concerned itself with the throughput [1] and a figure of merit [1], [4]. The latter attempts to give an insight into the effectiveness of the use of large amount of hardware in terms of raising the throughput. More recently Deverell [4] has analyzed the design of various maximally pipelined arrays for multiplication. In this investigation again the figure of merit is considered for various multipliers while attempting to preserve the iterative nature of the array. This latter aspect is significant as it is envisaged that such arrays would be implemented as LSI units. Maximally pipelined units where the basis is the iterative array invariably require very large amount of hardware for which even an LSI implementation may be uneconomical. If, however, we consider reduced pipelining, i.e., where only some rows of the array are latched then the hardware is reduced but at the expense of lower throughput. In this case it is feasible to use several units in parallel to achieve the total throughput as specified. We now consider this aspect in greater detail.

The concept of pipelining an arithmetic process is a well-known one and revolves around the possibility of processing independent subtasks which make up the complete arithmetic tasks. A general model of a pipelined processor is shown in Fig. 1. Thus, each subprocessor having completed the subtask of the $K$th task, would store the results in temporary storage (latch circuits) and then commence the processing of the subtask for the $(k+1)$th task. As such the delay on a continuous basis would simply be the delay to process a subtask. We define the subtask as that operation which cannot be divided or reduced any further either due to limitation of the hardware or of the subtask itself. Thus, for example, in the design of a pipelined adder an appropriate subtask would be 4-bit addition implemented by a CLA adder available commercially (e.g., SN7483). A fully pipelined adder design requires latch circuits after each 4-bit addition. Referring to Fig. 1 the subprocessor is a 4-bit adder. The storage after this must store the remaining data and the result of addition. In general, if each stage consists of a subtask followed by an appropriate amount of storage, we have a fully or maximally pipelined unit. In order to reduce the amount of hardware one may combine several subtasks and then introduce latch circuits after the larger subtask. In this case the design is referred to as a reduced pipelined design. As an example, a 48-bit adder can consist of a subtask of 4-bit addition and requires twelve stages each stage followed by latch circuits. A possible reduced pipeline design would be 12-bit addition per stage and requires three stages.

In order to examine the effect of pipelining, two factors are commonly used. One is $\tau$ the maximum throughput that can be achieved. The other is a figure of merit which in our case is de-

Manuscript received February 18, 1975; revised December 1, 1975 and February 3, 1976. This work has been supported by a grant from the National Research Council of Canada.

The author is with the Department of Electrical Engineering, University of Waterloo, Waterloo, Ont., Canada.

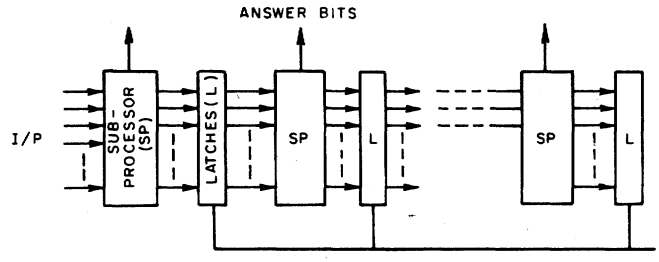


Fig. 1. A general model of a pipelined unit.

fined as follows:

$$f = \frac{N \cdot \tau}{I}$$

where

$N$ = word size or width of pipeline,
$\tau$ = throughput, and
$I$ = total number of IC modules used in the design.

The maximum throughput $\tau$ computed on the basis that data arrive at the pipeline on a continuous basis, is given by the reciprocal of the total delay per stage (including latch delay). Thus, it is obvious that a maximally pipelined array would yield a maximum throughput. However, if $I$ is large then the figure of merit can be unacceptably low. This is the case for most unpipelined cellular arrays proposed earlier, and can be true even in pipelined arrays if the number of latch circuits required is very large. It will now be shown that under certain conditions it is possible to use two or more pipelined arrays (each with reduced pipelining) in parallel to achieve the same or higher throughput as a maximally pipelined array but have a much higher overall figure of merit. Thus, consider the following analysis.

Suppose a fully pipelined array achieves a maximum throughput of $\tau$. Assume that it uses $A$ modules for arithmetic functions and $l$ for latch circuits. The figure of merit in this case is given by

$$f_1 = \frac{N \cdot \tau}{A + l}$$

If $M$ partially pipelined arrays are used in parallel then for the purpose of our evaluation, each of these units should have a throughput of at least $\tau_i = \tau/M$ so that the overall throughput is the same as for a fully pipelined unit. Each of the partially pipelined arrays will still require the same number of IC's for arithmetic, viz., $A$ units. However, the number of latch units will be reduced to, say, $\alpha l$, where $0 < \alpha < 1$ for each array. The individual figure of merit assuming that $\tau_i = \tau/M$ is given by

$$f_2 = \frac{N \cdot \tau/M}{(A + \alpha \cdot l)} = \frac{N \cdot \tau}{M \cdot (A + \alpha \cdot l)}.$$

The overall figure of merit for an $M$ array system is of course the same as $f_2$. Note that both $f_2$ and $f_1$ imply the same overall throughput. The ratio $Y = f_2/f_1$ defines a measure of the improvement. Thus,

$$Y = \frac{A + l}{M(A + \alpha l)} = \frac{1 + \theta}{M(1 + \alpha \cdot \theta)}, \quad \text{where } \theta = \frac{l}{A}.$$

The above analysis neglects the overheads, if any, caused by the multiplexing and demultiplexing of data for $M$ pipelined arrays operating in parallel. The above analysis therefore is an approximate one only, but nevertheless indicative of any potential improvement. Thus, whenever $Y > 1$, a more efficient system would be achieved by using $M$ partially pipelined units as compared to one fully pipelined unit. Thus, for any improvement we require that

$$\frac{1 + \theta}{M(1 + \alpha \cdot \theta)} > 1$$

or

$$\frac{1 + \theta}{1 + \alpha\theta} > M.$$

For any realistic design comparison $M \geq 2$, i.e., $\alpha$ must be less than 0.5. Furthermore, if $\theta < 1$, i.e., if the ratio of latch units to arithmetic units in the array is less than one then there will be no improvement possible by using $M$ partially pipelined arrays. For the purpose of illustrating the above aspect of pipelined arrays, we now consider a simple example, viz., the design of a 48-bit adder.

*Example:* The 48-bit adder uses 4-bit CLA adders. Hence, the subtask is a 4-bit addition. The packing density is 4-bit adder/IC and four latches/IC. The worst case delay specifications used are as follows.

*Adders:* $D_{\text{sum}} = 50$ ns $D_{\text{carry}} = 20$ ns (per IC).
*Latches:* $D_{\text{latch}} = 40$ ns.

In this design a fully pipelined adder would require latch circuits after each 4-bit addition, i.e., there will be twelve stages in the pipeline. Such an adder requires twelve full adder IC's. The total storage required is 852 bits which can be achieved by 213 latch IC's. This gives $\theta = 17.78$ with a throughput of $\tau = 11.1 \times 10^6$ tasks/s. Various partially pipelined adders were investigated and their two significant parameters have been plotted in Fig. 2. If we now consider two 4-stage pipelines, then each of these has a throughput of nearly $7.8 \times 10^6$ tasks/s, implying that the two units operating in parallel would be able to handle $15.6 \times 10^6$ tasks/s a figure substantially higher than the fully pipelined adder. Furthermore, it can be shown that in this case $\alpha < 0.5$ there is also an improvement in figure of merit. This can be also seen from Fig. 2, where the 4-stage pipeline has a figure of merit of $4.8 \times 10^6$. If the total throughput is reduced to match that of the fully pipelined unit, even then the figure of merit is higher, viz., approximately $3.52 \times 10^6$. In many applications, however, the availability of the higher throughput from the two units operating in parallel may be considered a significant advantage. It should however be noted that the additional requirements such as multiplexing and demultiplexing hardware for operating two or more pipelines in parallel have not been included in the above analysis. However, this is not expected to be significant where operation of only two or three units in parallel is considered.

We conclude from the above discussion that while fully pipelined arithmetic arrays achieve maximum throughput, their figure of merit which reflects the amount of hardware and its utilization can be unacceptably low. If the ratio of latch units to arithmetic units exceeds unity than it may be possible to improve this figure of merit by using several partially pipelined units in parallel. A second advantage can also arise from the latter configuration. This is concerned with the possibility of data being

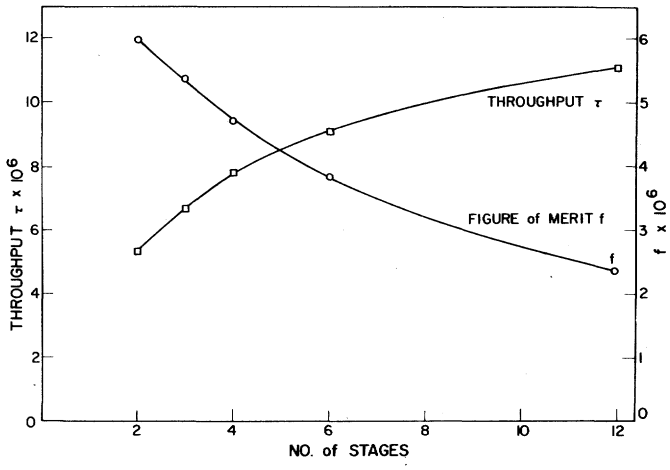


Fig. 2. Throughput and figure of merit for a 48-bit adder.

rejected from service. Since each pipeline will most likely be operative at a throughput below the maximum possible, the probability of data rejection becomes extremely small [5].

## References

[1] T. G. Hallin and M. J. Flynn, "Pipelining of arithmetic functions," in *IEEE—TCCA Symp. on Computer Arithmetic*, University of Maryland, vol. 1, May 1972.
[2] J. Deverell, "Sequential generalized array," *Electron. Lett.*, vol. 8, pp. 9–10, Jan. 1972.
[3] J. C. Majithia, "A pipelined array for square root extraction," *Electron. Lett.*, vol. 9, pp. 4–5, Jan. 1973.
[4] J. Deverell, "Pipeline iterative arithmetic arrays," *IEEE Trans. Comput.*, vol. C-24, pp. 317–321, Mar. 1975.
[5] J. C. Majithia and M. Gouda, "Buffer size requirements for pipelined arithmetic processors," in *1974 CEC Conf.*, Calgary, Canada, June 1974.

# Suggestion for a Fast Binary Sine/Cosine Generator

P. W. BAKER

***Abstract*—A combinational two-dimensional array for the rapid generation of sines and cosines is described. The array is a spatial realization of a continued product algorithm which uses multiplication by $(1 + jE_k 2^{-k})$. Using 10-ns logic, it is estimated that 16-bit sines and cosines may be generated in less than 2 $\mu$s.**

***Index Terms*—Cascaded carry-save adders, continued products, cosines, digital arithmetic, sines.**

## Introduction

The rapid improvement in computer solid-state technology makes possible complicated special purpose hardware structures that were unthinkable several years ago. As a result, much effort has gone into designing structures, suitable for LSI or hybrid technology, which allow the combinational computation of the elementary arithmetic procedures such as multiplication, division, the extraction of square roots and the interconversion of binary and BCD numbers. These structures are known as iterative cellular arrays. As the LSI and thick-film technologies become cheaper it is likely that combinational structures for the generation of more complex arithmetic procedures will be sought after. This correspondence develops a semiiterative array structure for the simultaneous generation of sines and cosines which should be suitable at least for construction in thick-film hybrid technology.

The array is a spatial realization of a continued product algorithm which uses multiplication by $(1 + jE_k 2^{-k})$. The next section discusses the radix-2 continued product generation of sines and cosines and develops a fast algorithm for the case where 16-bit accuracy is required. The last section describes the logical realization of a 16-bit sine/cosine generator. While the structure presented is for the generation of 16-bit results, the method is applicable to results of any desired accuracy.

Manuscript received June 25, 1974; revised September 1, 1975. This work was supported by the Australian Research Grants Committee.

The author is with the Department of Computer Science, School of Electrical Engineering, University of New South Wales, Kensington, N.S.W., Australia.

## Continued Product Sine/Cosine Algorithm

Iterative algorithms for the generation of $\sin X$ and $\cos X$ were introduced by Volder [1] and later by Specker [2] in a succinct mathematical notation. Specker's algorithm relies on the identity:

$$\exp(jX) = \beta \cdot \prod_{k=0}^{n} (1 + jE_k 2^{-k}) \exp \cdot \left\{ j \left[ X - \sum_{k=0}^{n} \tan^{-1}(E_k 2^{-k}) \right] \right\},$$

where

$$\beta = \prod_{k=0}^{n} (1 + E_k^2 2^{-2k})^{-1/2}, \tag{1}$$

$n$ is the length in bits of $X$, and $0 \le X < 1$. The $E_k$'s are chosen such that

$$X - \sum_{k=0}^{n} \tan^{-1}(E_k 2^{-k}) \simeq 0.$$

Specker confines the $E_k$ to values of $\pm 1$, in which case $\beta$ is a constant, prestored, and used as an initial argument in the algorithm:

$$X_0 = X,\ U_0 = \beta + j0, \qquad \beta = 0.60\cdots$$
$$X_{k+1} = X_k - E_k T_k,\ X_{k+1} \to 0, \qquad 0 \le k \le n \tag{2}$$

where $T_k = \tan^{-1}(2^{-k})$,

$$U_{k+1} = U_k(1 + jE_k 2^{-k}), \qquad E_k \in \{-1, 1\},$$
$$\operatorname{Im}\{U_n\} \simeq \sin X,\ \operatorname{Re}\{U_n\} \simeq \cos X \tag{3}$$

(here, $\operatorname{Im}\{\cdot\}$ denotes imaginary part of $\{\cdot\}$ and $\operatorname{Re}\{\cdot\}$ denotes real part of $\{\cdot\}$).

As pointed out by de Lugish [3], not all the $E_k$'s need to be confined to $\pm 1$. If the value of $\beta$ in (1) is replaced with

$$\beta = \prod_{k=0}^{n/2} (1 + 2^{-k})^{-1/2}$$

the deviation from the true value is less than $2^{-n}$. Hence, for $k > n/2$, $E_k$ may be taken from the nonredundant set $\{0,1\}$. However, this algorithm may be speeded up further because the $E_k$s can be predicted (cf., [4]), a circumstance which allows most of the additions required in the converging of $X_k$ to be of the fast carry-save type. Prediction of the $E_k$s is possible since

$$\tan^{-1}(2^{-k}) = 2^{-k} - \frac{2^{-3k}}{3} + \cdots.$$

Hence, given an $X_l$ with $l - 1$ leading zeros: