A Single Chip Radix-2 FFT Butterfly Architecture Using Parallel Data Distributed Arithmetic

I. ROSS MACTAGGART AND MERVYN A. JACK

Abstract—**This paper describes how distributed arithmetic techniques can be applied in parallel-data arithmetic computations to achieve highly regular and efficient VLSI structures on silicon. Two individual arithmetic processor chips are described as examples of the technique.**

The chips described, which are intended primarily for computation of the FFT Butterfly, each contain the functional equivalence of two parallel pipelined multipliers.

The first chip is an 8-bit prototype device which has been designed and fabricated on a standard 5 μm silicon gate n-channel MOS process. The second chip is a 16-bit CMOS-SOS design which uses a modified architecture to achieve higher clocking rates and improved versatility in systems use.

I. Introduction

REAL-TIME digital signal processing favors the use of very high-speed parallel data arithmetic operations. Distributed arithmetic techniques [1], [2] offer a means of mapping parallel data systems onto silicon with a high degree of regularity and efficiency. The specific structures considered here are two versions of a Radix-2 Butterfly processor for computation of the fast Fourier transform (FFT) algorithm [3], using distributed arithmetic.

The FFT algorithm is introduced and discussed briefly to highlight the Butterfly processing requirements and to indicate how distributed arithmetic approaches can be used in this processing task.

The paper includes a detailed discussion of the distributed arithmetic reformulation of the FFT Butterfly to show in detail how the silicon floorplan for the Butterfly processor can be derived.

Using an available, in-house NMOS process, a prototype 8-bit processor has been realized to validate the distributed arithmetic architecture. Details of this design are presented and test results together with performance data for this chip are discussed. A similar, but much more powerful 16-bit CMOS-SOS design with modified architecture using a commercially available process will also be described.

Manuscript received October 1, 1983; revised December 21, 1983. This work was supported by a Science and Engineering Research Council Grant.

The authors are with the Department of Electrical Engineering, University of Edinburgh, Edinburgh EH9 3JL, Scotland.

II. The FFT Algorithm

Of the several important FFT algorithms which have been developed for efficient computation of the discrete Fourier transform (DFT) [3], the most widely used is the Radix-2 decimation-in-time FFT [3], [4], where the transform length (N) may be any positive integer power of 2. A symbolic representation of this algorithm is shown in Fig. 1 for $N=8$. Here, the time-domain sequence (x_n) is converted to the frequency domain sequence (X_n) by means of 12 identical processing nodes, each of which is known as a Butterfly. Each Butterfly processing node consists of a two-point DFT (vector add and subtract), symbolized by the circle in Fig. 1, with a vector rotation requirement (multiplication by a unit vector) on one of the inputs, symbolized by the arrow in Fig. 1. It is this vector rotation requirement which dominates any silicon implementation of the Butterfly since this entails a complex multiplication for each Butterfly operation.

III. Distributed Arithmetic Concepts

Complex multiplication involves four real multiplications, plus an addition and subtraction as shown in Fig. 2(a), to implement the equations

$$\mathrm{Re}\{Z\}=\mathrm{Re}\{B\}\cdot\mathrm{Re}\{W\}-\mathrm{Im}\{B\}\cdot\mathrm{Im}\{W\} \quad (1)$$

$$\mathrm{Im}\{Z\}=\mathrm{Re}\{B\}\cdot\mathrm{Im}\{W\}+\mathrm{Re}\{W\}\cdot\mathrm{Im}\{B\}. \quad (2)$$

It is clear, from Fig. 2(a), that the two multiplier structure used to form $\mathrm{Re}\{Z\}$ is essentially the same as the two multiplier structure used to form $\mathrm{Im}\{Z\}$, differing only in an add and subtract. This two multiplier structure might therefore be considered to be a suitable candidate for a VLSI implementation of the complex multiply requirement of the Butterfly. In the case of parallel arithmetic, however, this general structure does not map onto silicon very efficiently due to problems arising from bus interconnections and irregular multiplier structures when special multiply algorithms, such as Booth's [5], are used. For this reason, as well as for yield considerations, current parallel-data Butterfly devices use a single, multiplexed, parallel multiplier.

This paper shows how it is possible to replace the two parallel multiplier structure in Fig. 2(a) with a single dis-

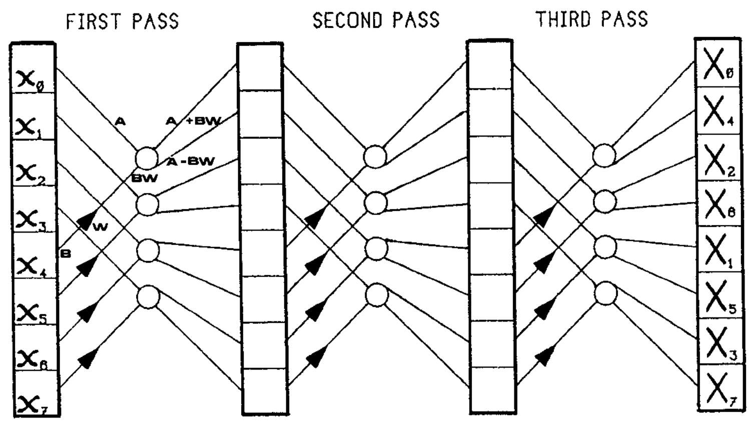


Fig. 1. 8-point FFT.

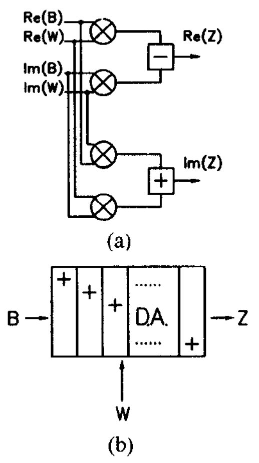


Fig. 2. Distributed arithmetic concept.

tributed arithmetic array which is regular and maps onto silicon efficiently. This is achieved in distributed arithmetic by bringing forward the final add and subtract in the complex multiply structure of Fig. 2(a) to the level of multiplier partial product formation, in order to form new unique arithmetically merged partial products which can be stored in temporary data registers. This allows the formation of real and imaginary complex outputs (Z) by performing a data-select and accumulate operation on these new merged partial products. The importance of this reformulation is that the resulting structure [Fig. 2(b)] involves only a single accumulator and therefore allows a highly regular VLSI structure. The distributed arithmetic approach [1] is generally useful when the products of more than one multiplier are subsequently combined in other arithmetic operations such as add or subtract to form a single output. In the case of the complex multiply, as described in (1) and (2), the real output requires two individual multipliers, each with two possible partial products. The same is true for the imaginary output. The two shift and add multipliers thus present four possible combinations of partial products, corresponding to the four possible combinations of the two real and imaginary coefficient bits being considered. It is the number of combinations of partial products that is important, as this determines the number of arithmetically merged distributed arithmetic partial products that will need to be stored. It will be shown later, however, that only two merged partial products are nontrivial in this case, and actually need to be stored.

Let us assume that N-bit, fixed point, two's complement arithmetic is used so that Re$\{W\}$ and Im$\{W\}$ might be described as

$$\mathrm{Re}\{W\} = -W_{RO} + \sum_{n=1}^{N-1} W_{Rn}\cdot 2^{-n} \tag{3}$$

$$\mathrm{Im}\{W\} = -W_{IO} + \sum_{n=1}^{N-1} W_{In}\cdot 2^{-n}. \tag{4}$$

This allows equation (1) for Re$\{Z\}$ to be expressed as

$$\mathrm{Re}\{Z\} = \left[-W_{RO} + \sum_{n=1}^{N-1} W_{Rn}\cdot 2^{-n}\right]\cdot \mathrm{Re}\{B\} - \left[-W_{IO} + \sum_{n=1}^{N-1} W_{In}\cdot 2^{-n}\right]\cdot \mathrm{Im}\{B\}. \tag{5}$$

Combining the separate summations into one summation and decoding all possible combinations of the real and imaginary w bits to select these new merged partial products gives

$$\begin{aligned}\mathrm{Re}\{Z\} &= \overline{W}_{RO}\cdot \overline{W}_{IO}(0)\\ &\quad + \overline{W}_{RO}\cdot W_{IO}(\mathrm{Im}\{B\})\\ &\quad + W_{RO}\cdot \overline{W}_{IO}(-\mathrm{Re}\{B\})\\ &\quad + W_{RO}\cdot W_{IO}(-\mathrm{Re}\{B\}+\mathrm{Im}\{B\})\\ &\quad + \sum_{n=1}^{N-1} [\overline{W}_{Rn}\cdot \overline{W}_{In}(0)\\ &\quad + \overline{W}_{Rn}\cdot W_{In}(-\mathrm{Im}\{B\})\\ &\quad + W_{Rn}\cdot \overline{W}_{In}(\mathrm{Re}\{B\})\\ &\quad + W_{Rn}\cdot W_{In}(\mathrm{Re}\{B\}-\mathrm{Im}\{B\})]\cdot 2^{-n}.\end{aligned} \tag{6}$$

Equation (6) shows how Re$\{Z\}$ can be formed in a single accumulator by selecting one of four merged partial products. This was not the chosen solution, however, as only two are actually required if we define

$$K = (\mathrm{Re}\{B\}+\mathrm{Im}\{B\})/2 \quad \text{and}$$
$$K' = (\mathrm{Re}\{B\}-\mathrm{Im}\{B\})/2. \tag{7}$$

Replacing the Re$\{B\}$ and Im$\{B\}$ terms in (6) by the K and K' terms shown in (7) (and Table I), yields (8), which

TABLE I
PERFORMANCE

PARAMETER	NMOS	CMOS-SOS *
CLOCK CYCLE	250 nS	25nS
DATA RATE	8 M Wds/s	40 M Wds/s
WORD LENGTH	8 Bits	16 Bits
POWER	0.5 Watts	0.25 Watts
PACKAGE	40 Pin Dil	64 Pin Dil

* Available for testing December '83

can be further simplified to give (9):

$$\begin{aligned}\mathrm{Re}\{Z\} = {} & \overline{W}_{RO}\cdot\overline{W}_{IO}(-K'+K') \\ & +\overline{W}_{RO}W_{IO}(-K'+K) \\ & +W_{RO}\overline{W}_{IO}(-K'-K) \\ & +W_{RO}W_{IO}(-K'-K') \\ & +\sum_{n=1}^{N-1}\Big[\overline{W}_{Rn}\cdot\overline{W}_{In}(K'-K') \\ & \quad \overline{W}_{Rn}\cdot W_{In}(K'-K) \\ & \quad W_{Rn}\cdot\overline{W}_{In}(K'+K) \\ & \quad W_{Rn}\cdot W_{In}(K'+K')\Big]\cdot 2^{-n} \end{aligned} \tag{8}$$

$$\begin{aligned}\mathrm{Re}\{Z\} = {} & -K'\cdot 2^{-(N-1)}+\overline{W}_{RO}\cdot\overline{W}_{IO}(+K') \\ & +\overline{W}_{RO}\cdot W_{IO}(+K) \\ & +W_{RO}\cdot\overline{W}_{IO}(-K) \\ & +W_{RO}\cdot W_{IO}(-K') \\ & +\sum_{n=1}^{N-1}\Big[+\overline{W}_{Rn}\cdot\overline{W}_{In}(-K') \\ & \quad +\overline{W}_{Rn}\cdot W_{In}(-K) \\ & \quad +W_{Rn}\cdot\overline{W}_{In}(+K) \\ & \quad +W_{Rn}\cdot W_{In}(+K')\Big]\cdot 2^{-n}. \end{aligned} \tag{9}$$

Equation (9) shows how $\mathrm{Re}\{Z\}$ can be formed in a single accumulator by adding in or subtracting a selected K or K' as a function of the real and imaginary w bits. The selection of K or K' can be based on the Exclusive-OR of the real and imaginary w bits and the add/subtract logic can be derived from the appropriate w bit directly, as shown in (10):

$$\begin{aligned}\mathrm{Re}\{Z\} = {} & -K'\cdot 2^{-(N-1)}+\overline{W}_{RO}(W_{RO}\overline{\oplus}W_{IO})(+K') \\ & +\overline{W}_{RO}(W_{RO}\oplus W_{IO})(+K) \\ & +W_{RO}(W_{RO}\oplus W_{IO})(-K) \\ & +W_{RO}(W_{RO}\overline{\oplus}W_{IO})(-K') \\ & +\sum_{n=1}^{N-1}\Big[+\overline{W}_{Rn}(W_{Rn}\overline{\oplus}W_{In})(-K') \\ & \quad +\overline{W}_{Rn}(W_{Rn}\oplus W_{In})(-K) \\ & \quad +W_{Rn}(W_{Rn}\oplus W_{In})(+K) \\ & \quad +W_{Rn}(W_{Rn}\overline{\oplus}W_{In})(+K')\Big]\cdot 2^{-n}. \end{aligned} \tag{10}$$

The expression for $\mathrm{Im}\{Z\}$ can be obtained similarly, giving

$$\begin{aligned}\mathrm{Im}\{Z\} = {} & -K\cdot 2^{-(N-1)}+\overline{W}_{IO}(W_{IO}\oplus W_{RO})(+K') \\ & +W_{IO}(W_{IO}\overline{\oplus}W_{RO})(-K) \\ & +\overline{W}_{IO}(W_{IO}\overline{\oplus}W_{RO})(+K) \\ & +W_{IO}(W_{IO}\oplus W_{RO})(-K') \\ & +\sum_{n=1}^{N-1}\Big[+\overline{W}_{In}(W_{In}\oplus W_{Rn})(-K') \\ & \quad +W_{In}(W_{In}\overline{\oplus}W_{Rn})(+K) \\ & \quad +\overline{W}_{In}(W_{In}\overline{\oplus}W_{Rn})(-K) \\ & \quad +W_{In}(W_{In}\oplus W_{Rn})(+K')\Big]\cdot 2^{-N}. \end{aligned} \tag{11}$$

Table I depicts this algorithm for nonsign bits only, alongside the conventional arithmetic approach using shift and add multipliers. This table serves to illustrate how the individual merged partial products in the distributed arithmetic approach are related to the individual partial products in the conventional shift-and-add multiplier scheme. For example, in the $\mathrm{Re}\{Z\}$ formation columns (1–4) of Table I, row 3 shows how $\mathrm{Re}\{B\}$ can be expressed as $K'+K$, with a W independent K' term. In the same columns, row 4 shows how $\mathrm{Re}\{B\}-\mathrm{Im}\{B\}$ can be expressed as $K'+K'$.

In the shift and add multiplier, the final product is formed by the successive accumulation of partial products which are formed by the logical "AND" of the data word (B) with successive coefficient bits (W) at various levels of significance which are all powers of 2. The partial products for the four multipliers in the conventional arithmetic case are shown in columns 1,2; 5,6 in the table as a function of the individual bits of W. Table I shows how the data word is added in, only if the coefficient bit (W) is a "1." However, in the conventional approach, the final subtract (for $\mathrm{Re}\{Z\}$) and add (for $\mathrm{Im}\{Z\}$) is not considered until final product formation in the individual multipliers. Table I shows how the final add and subtract operation can be brought forward to the level of partial product formation so as to form four new merged partial products. Thus, individual multiplier partial products in columns 1 and 2 are now considered to be combined arithmetically to form a single column containing merged partial products for $\mathrm{Re}\{Z\}$. Similarly, columns 5 and 6 are now considered to be merged to form a single column from which $\mathrm{Im}\{Z\}$ can be formed directly. Table I goes on to show how these merged multiplier partial products can be replaced with the expressions involving K and K' (7) in columns 3,4,7,8. The K' term in column 3 for $\mathrm{Re}\{Z\}$ and the K term in column 7 for $\mathrm{Im}\{Z\}$ are both independent of the W coefficient bits. This means that these columns do not need to be included in the main accumulation process used to form real and imaginary Z. Instead, they can be accounted for during array initialization. The table shows how, by the W-controlled selection of $+K$, $-K$, $+K'$, $-K'$ [+/−(K

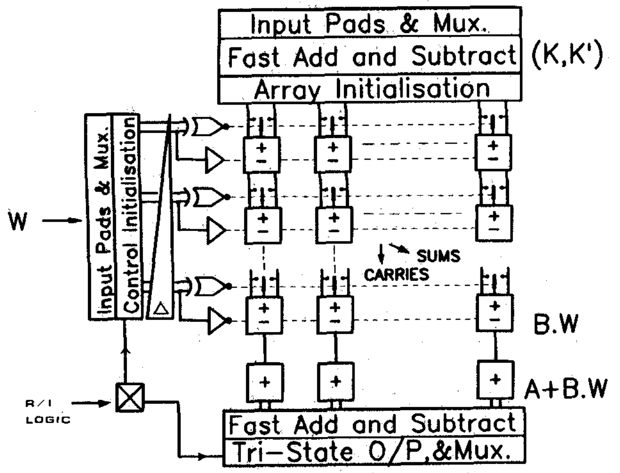


Fig. 3. Floorplan of NMOS chip.

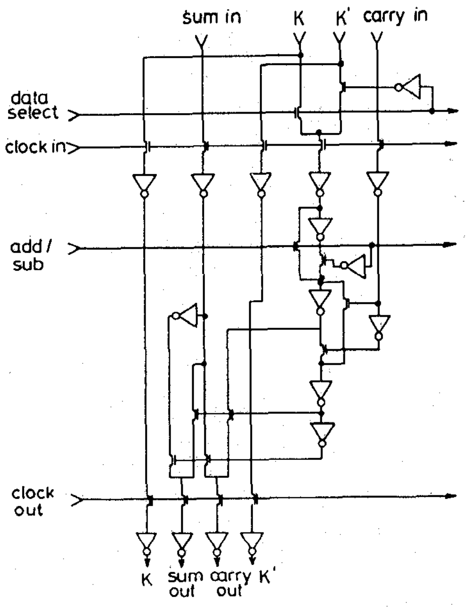


Fig. 4. Basic cell (NMOS).

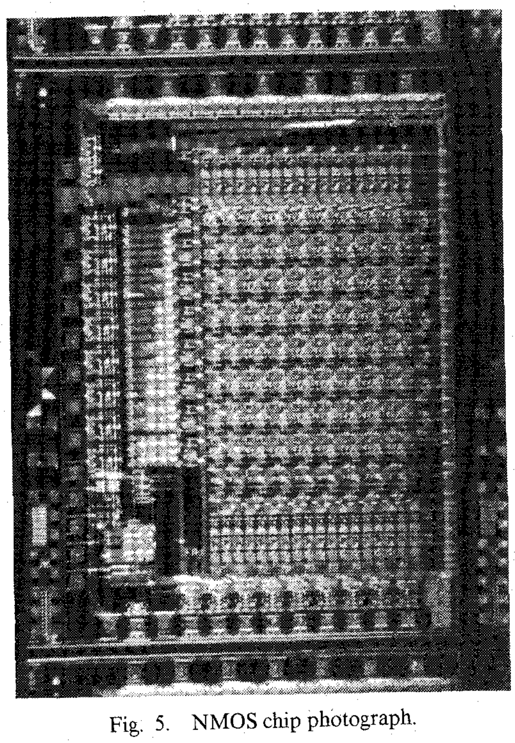


Fig. 5. NMOS chip photograph.

or K')] (as shown in columns 4 for Re$\{Z\}$ and column 8 for Im$\{Z\}$), the complex product Re$\{Z\}$ and similarly Im$\{Z\}$ can be formed in a single data-select and accumulate structure. It can be seen how an Exclusive-OR/NOR type relation of the W bits can be used to select either K or K' and how the real W bit (imaginary W bit) can determine whether this selected K or K' is added or subtracted for Re$\{Z\}$ (Im$\{Z\}$).

IV. Distributed Arithmetic Architecture

Using these distributed arithmetic concepts, the two multiplier structure of Fig. 2(a) can now be replaced with a regular array of bit-level data-select/accumulate cells to form the floorplan of the 8-bit NMOS chip shown in Fig. 3. Data words (A, B) enter the chip and are demultiplexed into real and imaginary components. A fast adder and subtractor is used at this point to convert Re$\{B\}$ and Im$\{B\}$ to K and K', (7) which are then fed down to the first row of cells in the distributed arithmetic array, together with an array initialization word which comprises the very low significance $-K'$ or $-K$ present as the first term in (10) and (11) and a rounding word which is fixed. This rounding word was equal to the mean value of all the sums which had to be truncated in the array. At each cell, K or K' was selected under the control of the Exclusive-OR (Re$\{Z\}$) or Exclusive-NOR (Im$\{Z\}$) gates whose inputs were the real and imaginary W coefficient bits. Each cell was also fed an add/subtract control signal which was derived from the buffered real or imaginary W coefficient bits directly, as outlined in Table I. Only for the sign bits of W, when the bits have a negative significance is the add/subtract logic inverted so that the selected K or K' is added if the appropriate W bit is a "zero" instead of a "one"—as is the case with the nonsign bits of W.

In the NMOS chip, the CARRY data is fed forward along with the SUMS, so that it is necessary to assimilate SUM and CARRY data of equal significance at the output of the array. This was accomplished by means of a fast adder employing a precharged carry-chain. SUM and CARRY data in the array were latched, so it was necessary to skew the coefficient W input data to the control gates as depicted in Fig. 3. There was no need for skewing input or output data because of the use of the fast adder at the output. The nonrotated Butterfly input (A), which is shifted directly through the complex multiplier was finally added to the complex output (BW) to form the Butterfly output ($A+BW$). The other Butterfly output ($A-BW$) was formed as (2A$-$($A+BW$)) as this avoided the need to feed BW forward, through the row of cells used to form ($A+BW$).

Fig. 4 shows the basic cell logic in detail. Invertor controlled data-select type Exclusive-OR gates were used in the carry-save adder, as this offers a good tradeoff in area–speed–power.

TABLE II
DISTRIBUTED ARITHMETIC ALGORITHM (COMPUTERS $Z = B \cdot W$)

W		REAL(Z)	IMAG(Z)
Re	Im	1 2 3 4	5 6 7 8
0	0	0−0 =K'−K'	0+0 =K−K
0	1	0−Im(B)=K'−K	Re(B)+0 =K+K'
1	0	Re(B)−0 =K'+K	0+Im(B)=K−K'
1	1	Re(B)−Im(B)=K'+K'	Re(B)+Im(B)=K+K

(Bit-Level)

Where K=(Re(B)+Im(B))/2 and K'=(Re(B)−Im(B))/2

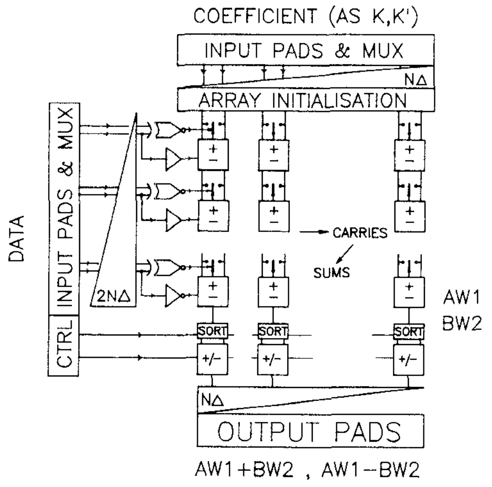


Fig. 6. Floorplan of SOS chip.

V. DETAILS AND PERFORMANCE OF NMOS PROCESSOR

The NMOS prototype chip (Fig. 5) contains around 8000 transistors and measures about 5.3×6.7 mm. The process used was a standard 5 μm feature size, single polysilicon, single metal n-channel MOS process using depletion mode load devices. Table II shows the measured performance of the 8-bit NMOS processor. The device clocked at 4 MHz corresponding to a data rate of 8 megabytes/s, which was slightly slower than expected owing to the use of a clock input pad which was limiting internal clock risetimes.

VI. CMOS-SOS 16-BIT PROCESSOR

In general, 8-bit word lengths are not adequate to cover most FFT application areas, such as radar signal processing, where 12–16-bit accuracy is typically required and for these reasons a 16-bit processor design which was similarly based on the distributed arithmetic complex multiply algorithm was undertaken. The floorplan for this device is shown in Fig. 6. For larger word lengths it is desirable to pipeline the distributed arithmetic in two dimensions so as to eliminate the fast add requirement. Extra latches (delays) must then be inserted into the basic cell.

CMOS Processor Pipeline

In the CMOS distributed arithmetic processor, (coefficient) data entering from the top is skewed, with the nth input bit receiving a delay of n, going from least to most significant bits. This allows the control and carries in each 17 cell row to be latched (extra cell per row for two's complement operation). The effect of this pipelining scheme is to produce a computation front which moves down through the array at an angle of 45°. This skewed computation front now means that data entering the vertical data port associated with the Exclusive-OR control gates, needs to be skewed by $2n$ delays for the nth input bit, moving from least significant to most significant bits, in order that control signals will match up correctly with data in the array. Fig. 7 shows the basic cell used in the 16-bit CMOS-SOS processor chip which results in a completely systolic architecture [6]. This cell feeds the CARRY right and the SUM is fed down and left to scale down the result of each cell by 2. The control passes from left to right at the same rate as the carries. The distributed arithmetic coefficients, K and K' needed to be delayed by two clock cycles in each 2-D pipelined cell because of the 45° skew on the computation front. As the same K and K' needs to be made available for both real and imaginary computations, this delay was implemented in a single shift register, clocked at half the main clock rate. This was done to save chip area, with the only condition that outputs would have to alternate between real and imaginary. The maximum clock rate of the chip is determined primarily by the time to produce a carry-out from the basic cell. It was stated that the vertical delay through this cell is equal to two clock cycles. This gives the array a latency of the order of $2n$ where n is the word length; however, the time-wedge used at the input to the array and the output of the array to skew and deskew data increases the latency of the chip by another n resulting in a total latency of around $3n$.

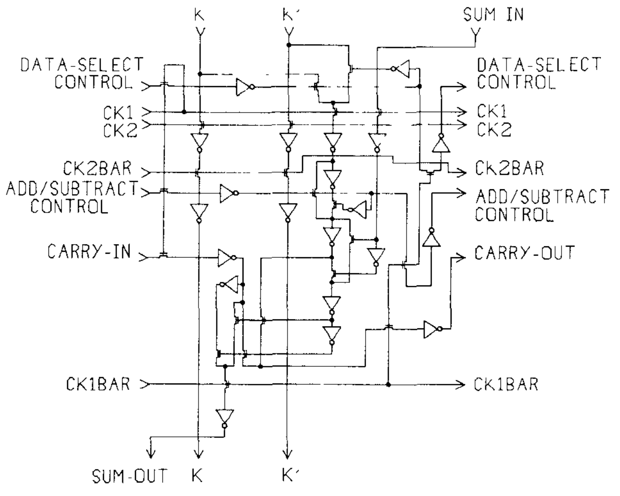


Fig. 7. Basic cell (SOS).

CMOS Architectural Modifications

The CMOS-SOS design contains some other significant architectural modifications. In the NMOS chip, data enters at the top of the chip, and was converted to the form of K and K' as defined in (7). In the CMOS-SOS design, the coefficient enters the top data port in the form of K and K'. The coefficient can therefore be stored in this form and

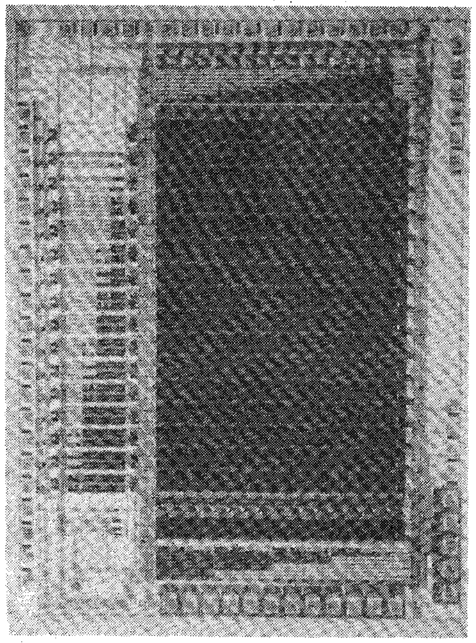


Fig. 8. SOS chip layout.

is not actively computed on the chip. This further lowered the power needed to compute the complex multiply. Unlike the NMOS chip, however, all data in the CMOS-SOS design passes through the complex multiplier. This has several advantages which are:

1) simple time-domain windowing on the first pass if required;

2) lower Butterfly noise caused by amplitude errors in W coefficient; and

3) easier system design with fewer components.

Further, the CMOS-SOS chip can compute a two-point nontrivial DFT, allowing larger DFT's to be built up using a slower external accumulator.

VII. Details and Expected Performance of CMOS-SOS Device

The CMOS-SOS device (Fig. 8) measures 7×8 mm and contains around 30 000 transistors. The device was designed using 4 μm feature CMOS-SOS design rules. This device uses an external clock generator to allow the highest possible clock rates to be achieved. Table II shows the expected performance of the CMOS-SOS device in comparison to the measured performance for the NMOS prototype.

VIII. Conclusions

Two LSI/VLSI chips which use distributed arithmetic to compute the arithmetic requirements of the Radix-2 FFT Butterfly have been described. Each of these devices has the throughput equivalence of two parallel multipliers, allowing very high bandwidths.

Distributed arithmetic offers a highly regular design approach in parallel data systems and also offers lower power consumptions than is possible using conventional arithmetic.

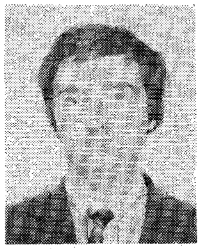
These techniques are thus highly suited to parallel data arithmetic, where an irregular structure can be replaced with a highly regular and compact array which offers a high degree of algorithmic efficiency.

Acknowledgment

The authors would like to thank the Edinburgh Microfabrication Facility for silicon processing and the General Electric Company for providing design rules for their high performance SOS process.

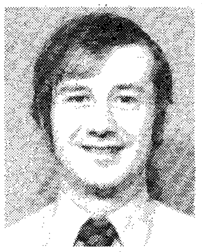
References

[1] S. A. White, "A simple FFT butterfly arithmetic unit," *IEEE Trans. Circuits Syst.*, vol. CAS-28, Apr. 1981.
[2] I. R. Mactaggart and M. A. Jack, "Radix-2 butterfly processor using distributed arithmetic," *Electron. Lett.*, vol. 19, pp. 43–44, Jan. 1983.
[3] L. R. Rabiner and B. I. Gold, *Theory and Application of Digital Signal Processing.* Englewood Cliffs, NJ: Prentice-Hall, 1975.
[4] E. O. Brigham, *The Fast Fourier Transform*, 1974.
[5] D. A. Booth, "A signed binary multiplication technique," *Q. J. Mech. Appl. Maths.*, vol. 4, pp. 236–240, 1951 (Oxford Univ. Press, no. 4, Apr. 1951).
[6] H. T. Kung and C. Leiserson, "Systolic arrays for (VLSI)," in *Introduction to VLSI Systems*, Mead and Conways, Eds. Reading, MA: Addison-Wesley, 1980.



I. Ross Mactaggart received the B.Sc. (Hons.) degree in chemical physics in 1980 and the M.Sc. degree in the design and manufacture of microelectronic systems, both from Edinburgh University.

He is currently a member of the Integrated Systems Group at Edinburgh University where he is involved in semi-custom and full-custom design and research activities.



Mervyn A. Jack was born in Edinburgh, Scotland, on June 20, 1949. He received the B.Sc. degree in electronic engineering and the M.Sc. degree in digital techniques from the Heriot-Watt University, Edinburgh, in 1971 and 1975, respectively, and the Ph.D. degree from the University of Edinburgh in 1978.

From 1971 to 1975 he worked as a project engineer with Microwave and Electronic Systems, Ltd., Edinburgh, where he was responsible for the design and development of security systems based on passive infrared and microwave Doppler intruder detectors. In 1975 he was appointed to a Research Fellowship at the University of Edinburgh to study the design and application of Fourier transform processors based on surface acoustic wave and charge coupled devices. In 1979 he was appointed to a lectureship in the Department of Electrical Engineering at Edinburgh University.

Dr. Jack is a member of the Institution of Electrical Engineers.