ARITHMETIC FOR HIGH SPEED FFT IMPLEMENTATION

Earl E. Swartzlander, Jr.
TRW Defense Systems Group
Redondo Beach, California

John Eldon
TRW LSI Products
La Jolla, California

ABSTRACT

This paper describes recent progress in the implementation of high speed spectrum analysis systems with state-of-the-art commercial and semi-custom VLSI circuits. Initial efforts are producing Fast Fourier Transform (FFT) and inverse FFT processors that operate at data rates of up to 40 MHz (complex). The current implementation computes transforms of up to 16,384 points in length by means of the radix 4 pipeline FFT algorithm. The interstage reordering is performed by delay commutators implemented with semi-custom VLSI, while the arithmetic is performed by commercial single chip 22 bit floating point adders and multipliers. This paper explains the pipeline FFT implementation and focuses attention on the arithmetic used to realize the design.

INTRODUCTION

Although the Cooley-Tukey FFT algorithm developed in 1965 has made it possible to apply digital techniques to many signal analysis applications, many others (e.g., radar and sonar beam forming, adaptive filtering, communications spectrum analysis, etc.) require combinations of flexibility and speed necessitating computational performance that exceeds the present state of the art. Current data acquisition technology demands analysis bandwidths of 25-50 MHz for real time operation. Currently, there are three approaches: software implementation on general purpose computers, software implementation on a general purpose computer augmented with a Programmable Signal Processor (PSP), and custom hardware development. Software only and Software-PSP implementations are adequate when the spectral bandwidth is under 1 MHz. Custom processors achieve analysis bandwidths of 1-10 MHz but most are optimized for a specific application and require extensive (and expensive) redesign to modify them to suit other applications. Thus general purpose computers with or without PSP augmentation are too slow while custom processors are too expensive and lack the required flexibility.

Current signal processing systems require many diverse functions: transform processors, time and frequency domain vector processors, and general purpose computers. Work is underway to produce a growing family of building block modules to facilitate the development of such systems on a semi-custom basis. The result is the ability to quickly develop high performance signal processing systems for a wide variety of algorithms. The use of predesigned and precharacterized modules reduces cost, development time, and most importantly, risk.

A critical design decision in signal processing concerns the arithmetic implementation. There are three somewhat contradictory requirements: 1) high speed to accommodate increasing signal bandwidths, 2) high precision, to minimize computational error, and 3) wide dynamic range, to accommodate various signal and noise levels without overflow. A wide variety of techniques are employed in signal processing due to the relative importance of these requirements for specific applications. At the highest speeds, analog techniques, such as SAW devices, are often employed. In audio and geophysical applications, where high accuracy is needed but speed is less critical, minicomputers can perform high precision operations in firmware. In most digital signal processing applications, both accuracy and speed are important. Many users have compromised on 16-bit fixed-point arithmetic. Although fast, fixed point arithmetic can lead to overflow errors or loss of precision, unless complex data dependent scaling is provided. Recently floating point arithmetic has become feasible, in part due to the attainment of high levels of integration via VLSI. Most floating point digital signal processing systems have used 32 bit formats which provide far more precision and dynamic range (and circuit complexity) than is justified by the data. The recent availability of 22 bit floating point single chip adders and multipliers provides the opportunity to use a simpler implementation that is more consistent with signal data.

SIGNAL PROCESSING MODULES

The initial set of signal processing modules includes a data acquisition module, building block elements that are replicated to realize pipeline FFT and inverse FFT modules, a frequency domain filter module, a power spectral density computational module, and an output interface module [1,2]. The modules all have separate data and control interfaces. All data interfaces satisfy a common interface protocol so that modules can be connected together to form architectures that match the data flow of each specific system. The separation of the data and control is analogous to the Harvard mainframe computer architecture which uses separate data and instruction memories to eliminate the "von Neumann bottleneck." In

CH2146-9/85/0000/0223$01.00 © 1985 IEEE

signal processing the separation of data and control allows the simple data interfaces to operate at high speed while the more flexible (and complex) control interfaces operate at a slower rate.

Due to its importance in most signal processing applications, the FFT module was selected for initial development. The FFT processor uses the radix 4 pipeline algorithm developed a decade ago at Lincoln Laboratory [3] as an extension of the radix 2 pipeline FFT algorithm [4]. With the radix 4 pipeline algorithm, 4 data pass in parallel through a pipeline network comprised of computational elements and delay commutators as shown on Figure 1. An important feature of this algorithm and architecture is that only two types of elements are used: computational elements and delay commutators. Only minor changes are required to implement forward and inverse transforms of lengths that are powers of 4. The changes involve varying the number of stages connected in series, changing the counter sequence and step size on the computational elements, and changing the length of the delays on the delay commutator. The computational element performs a four point discrete Fourier transform. In this implementation 22 bit floating point arithmetic is performed with single chip adders, subtractors, and multipliers [5]. The delay commutator reorders the data between computational stages as required for the FFT algorithm. Data rates of 40 MHz are achieved using 10 MHz clock rates since the radix 4 architecture processes four data streams concurrently.

THE DELAY COMMUTATOR CIRCUIT

Careful examination of the FFT module design revealed that much of the complexity was due to the delay commutator element. Initial complexity estimates are 80 commercial integrated circuits for the computational element and 180 circuits for the delay commutator. The disparity in complexity arises because of the commercial unavailability of shift registers of lengths 1, 2, 3, 4, 8, 12, 16, 32, 48, 64, 128, 192, 256, 512, and 768 as needed to implement the delays in the delay commutator. An alternative approach is to develop a delay that can be programmed to an arbitrary length. The most efficient approach as shown on Figure 2 involves simulating a delay line by using a RAM as a

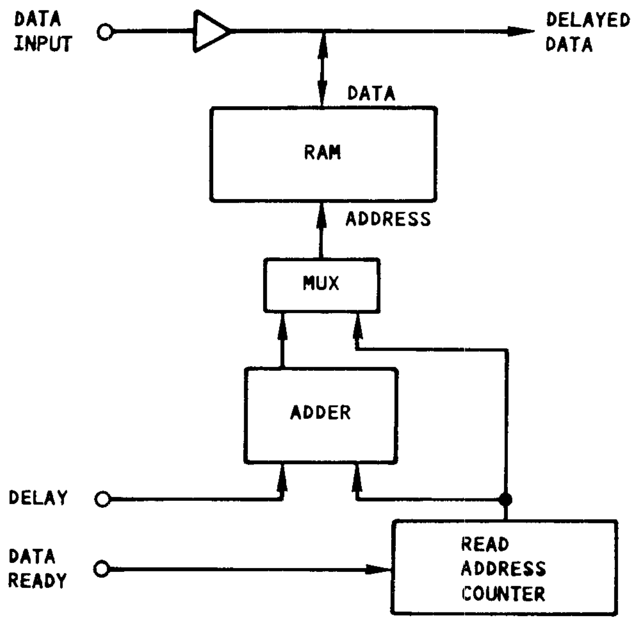

Figure 2. Variable Length Delay Implementation with Circular Buffer

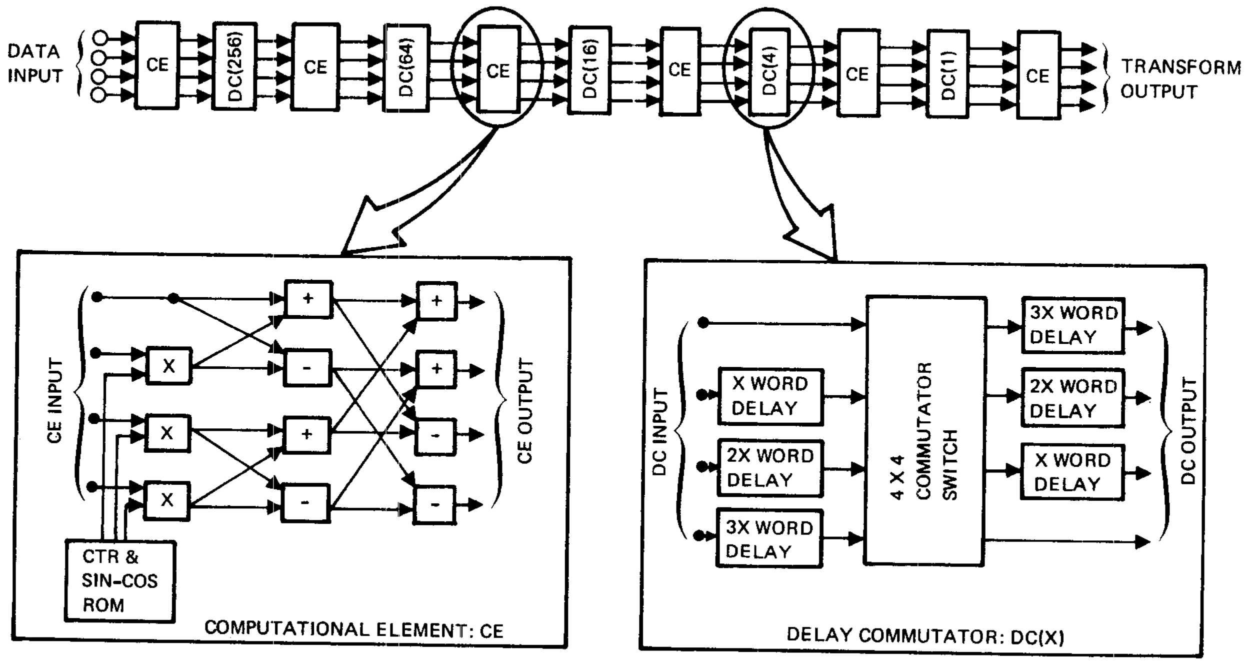

Figure 1. Pipeline FFT Architecture

circular buffer with write and read addresses displaced by a constant (i.e., the length of the simulated delay line). Given the high complexity of the commercial implementation of the delay commutator, alternative approaches were examined. A B bit wide data slice of a delay commutator that can be programmed for X = 1, 4, 16, 64, and 256 requires approximately 400 (B+1) logic gates and 3072 B shift register stages. Since a shift register stage is comparable in complexity to 3 random logic gates, this reduces to 400 + 9616 B gates. Table 1 compares VLSI versions based on gate arrays, standard cells, and custom technology. For these technologies (as of 1984), maximum achievable data bit slice widths are limited to 1, 4, and 10, respectively. For these widths 44, 11, or 5 delay commutator circuits would be required per computational stage of the pipeline FFT. Given the desire to minimize system complexity and to avoid an expensive custom VLSI development activity, the standard cell approach was selected. The resulting delay commutator circuit is a 4 bit wide slice that uses programmable length shift registers and a 4 x 4 switch as shown on Figure 3. Data enters through shift registers with taps and multiplexers to set the delay at 1, 4, 16, 64, or 256 (=X) in the uppermost input register and multiples of 2X and 3X in the middle and lower registers, respectively. Four 4:1 multiplexers implement the commutator function under the control of the programmable rate counter. The final 2 bit counter/decoder that controls the multiplexer settings can be reset and held to disable the commutator switch function. In this mode the chip provides fixed length registers with delays which can be used to expand the delay commutator for transform lengths greater than 4096 points. Data from the 4:1 multiplexers are output through programmable length shift registers that are similar to the input registers.

TABLE 1. DELAY COMMUTATOR IMPLEMENTATION OPTIONS

	GATES/CHIP	MAX B	CHIPS/STAGE	RELATIVE DEVELOPMENT COST
Commercial	-	-	179	1
Gate Array	10K	1	44	2
Standard Cell	40K	4	11	3
Custom	100K	10	5	10

Operation of the delay commutator to reorder data is shown on Figure 4 where the data flow for a 64 point transform is shown [6]. The input data (with a spacing of 16) are applied to a radix 4 butterfly producing output data with a spacing of 16. The reordering necessary to produce a data spacing of 4 is accomplished with delays of 0, 4, 8, and 12; commutation at a rate of one fourth the

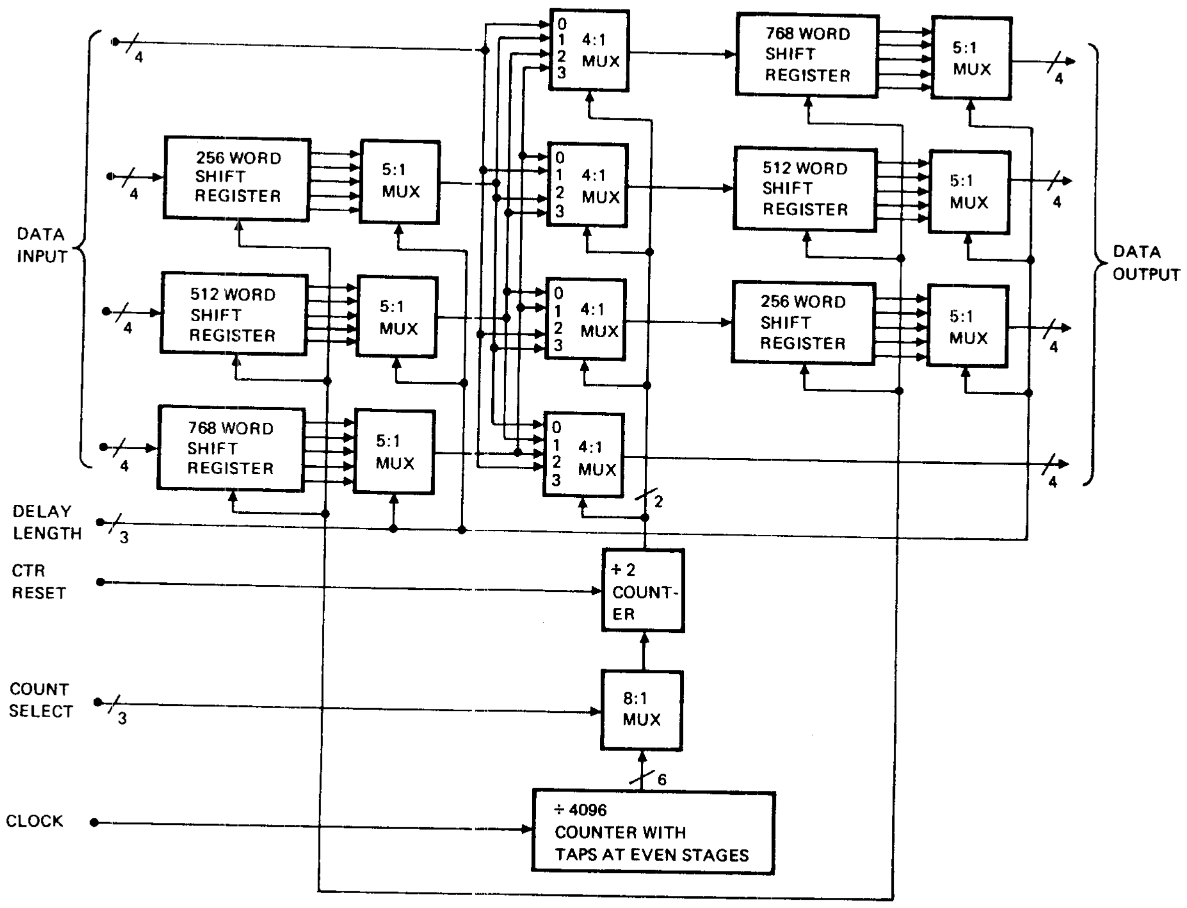

Figure 3. Delay Commutator Circuit

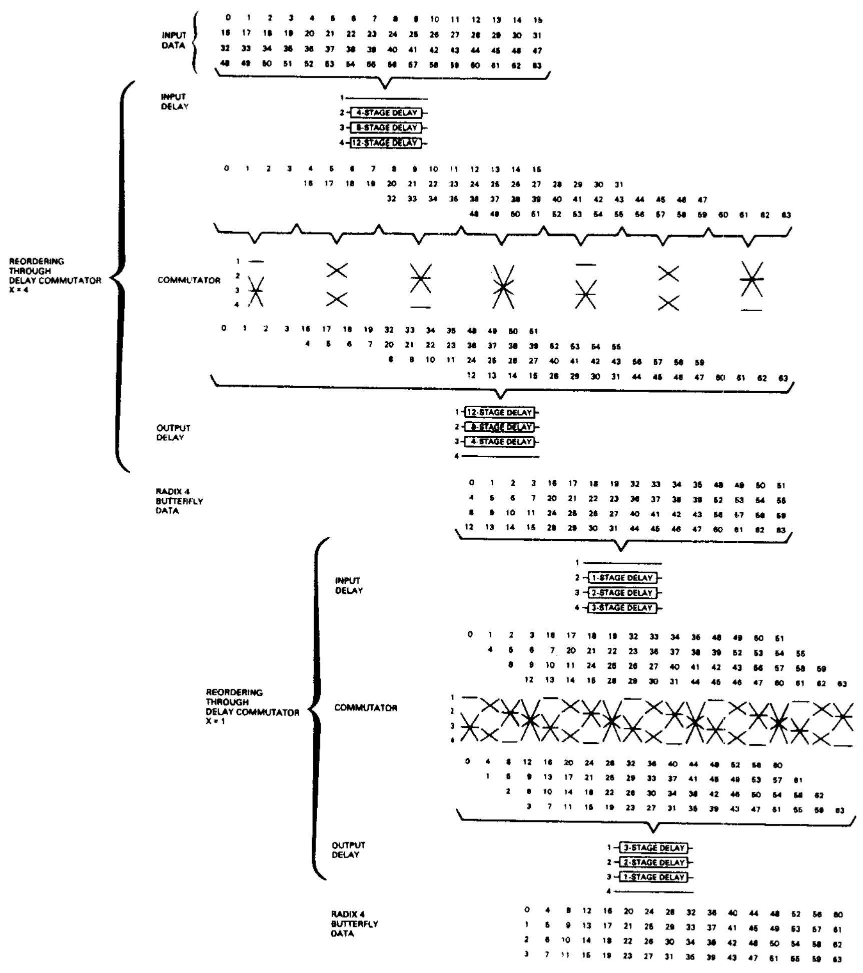

Figure 4. 64 Point FFT Data Flow Through the Delay Commutator (After [6])

data rate; and delays of 12, 8, 4, and 0. The data (with a spacing of 4) are applied to a radix 4 butterfly. The resulting data are reordered by use of delays of 0, 1, 2, and 3; commutation at a rate equal to the data rate; and delays of 3, 2, 1, and 0. The final data have a spacing of 1 as required for the final radix 4 butterfly.

This circuit was designed and implemented with Bell Laboratory's polycell (standard cell) CMOS technology. This technology was selected because it is well suited to the development of VLSI with high density shift registers and random logic. The chip contains 12,288 shift register stages and about 2000 gates of random logic, for a total complexity of 108,000 transistors [7]. At a clock rate of 10 MHz, the power dissipation is under 1/2 watt. The 340 x 376 mils chip is packaged in a 48 pin dual-in-line ceramic package. The chip is shown on Figure 5. Each of the four bit slices is constructed with input registers in a column, switching logic in a second "random logic" column, and output register in a column. The four nearly identical slices are about four times as tall as they are wide, producing a roughly square chip when they are properly stacked. There is minor variation in the random logic of each bit slice to account for sharing of the counters, decoders, clock drivers, etc. Although a radix 2 delay commutator chip has been developed by NEC [8], it is limited to clock rates of 5 MHz, which results in data rates of 10 MHz. In contrast, the radix 4 delay commutator operates at twice the clock rate and simultaneously processes twice as many data streams which quadruples the data rate.

Development of the delay commutator chip reduces the complexity of the 40 MHz 4096 point FFT from 1375 commercial integrated circuits to 546 circuits (of which 66 are delay commutator chips). This is a 60% complexity reduction achieved through the development of a single semi-custom chip. For larger transforms, the complexity of a 16,384 point FFT processor is reduced from 1634 integrated circuits to 670 circuits with the VLSI delay commutator circuit. Such a reduction greatly improves system reliability since connections between circuits represent the dominant failure mechanism in modern systems [9]. With 60% fewer circuits (and a corresponding reduction in the number of interconnections) the reliability is greatly improved.

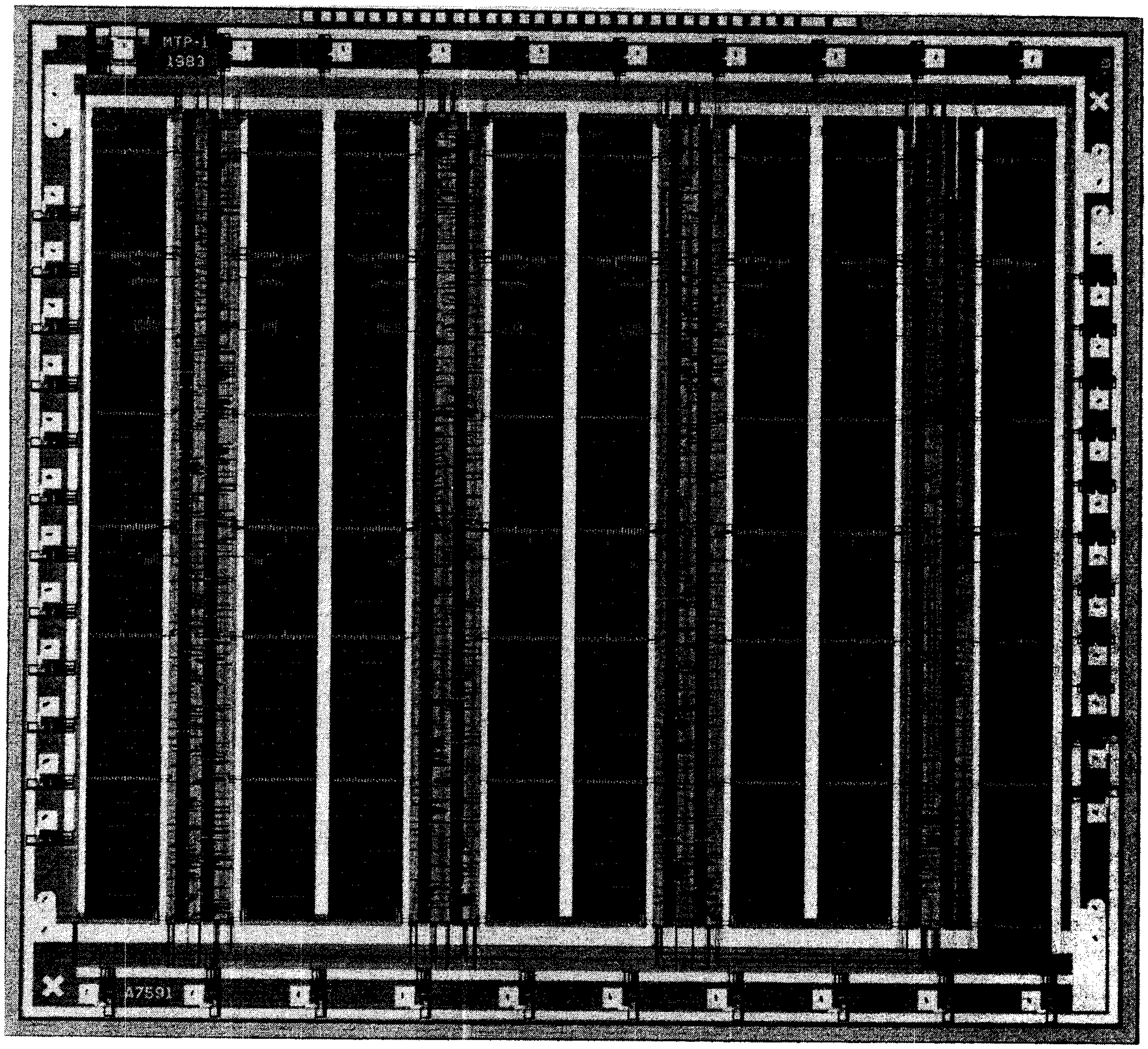

Figure 5. Delay Commutator Integrated Circuit Photograph

ARITHMETIC REALIZATION

Until recently, most real time signal processing has been performed with fixed point arithmetic, due to size, cost, and speed limitations of available hardware. The increased dynamic range and automatic normalization of floating point are desirable but have been prohibitively complex. Recently single chip adders and multipliers using the 22 bit floating point format (16 bit fraction, 6 bit exponent), have been developed. This section describes the use of these components to perform the arithmetic required for the computational element of Figure 1 (a radix 4 butterfly).

The 22-bit format, with a 16-bit two's complement fraction and a 6-bit two's complement exponent, is a reasonable compromise among performance, speed and size. Although single chip 32-bit floating point devices are commercially available, for a given technology the 22-bit format will always produce chips that are simpler and as a result cheaper and faster, with adequate dynamic range and precision for most applications. 32 bit arithmetic is useful in scientific computation when inverting matrices, evaluating eigenvectors, etc. but these operations are usually performed at much lower rates than those required for signal processing where input data are often limited in precision to 8 bits or less.

The chief advantage of floating point is increased dynamic range. As shown on Table 2, 22 bit floating point arithmetic provides 96 dB of precision (i.e., equivalent to 16 bit fixed point arithmetic) over a dynamic range of 476 dB [5]. Although this dynamic range is less than that of 32 bit floating point arithmetic, it is more than adequate for most high speed signal processing applications. The 16-bit fixed point format has a dynamic range of 15 x 20 log2 = 90 dB; the 22-bit floating point format provides (2^6+15) x 20log2 = 480 dB. With proper normalization, 15-bit precision is available over a dynamic range of 2^6 x 6 = 384 dB. In contrast, 22-bit fixed point would offer 15-bit precision over only a 7-bit (42 dB) range.

TABLE 2. ARITHMETIC COMPARISON

ARITHMETIC SYSTEM	DYNAMIC RANGE	PRECISION
12 Bit Fixed Point	72 dB	72 dB
16 Bit Fixed Point	96 dB	96 dB
22 Bit Fixed Point	132 dB	132 dB
22 Bit Floating Point	476 dB	96 dB
32 Bit Floating Point	1686 dB	144 dB

Adder

Under user control, the 22 bit adder performs floating point addition, accumulation, and conversions between fixed and floating point formats. Rounding and scaling (÷2) are also selectable if desired. For proper operation and maximum accuracy, non-zero floating-point operands must be normalized, with fractions in the range of $-1.0 \leq S < -0.5$ or $0.5 \leq S < 1.0$.

The adder is shown on Figure 6. It performs the three component operations of floating point addition: denormalization (exponent alignment), addition, and renormalization. The first and last steps are hardware intensive, involving shifting of the fraction and compensatory incrementing of the exponent. In the addition mode, the adder first selects the addend with the smaller exponent. It denormalizes this operand by shifting it rightward by the difference between the exponents of the two addends. The denormalized fraction is added to the other addend's unshifted fraction, and their sum passes to the renormalizing section, along with the larger addend's exponent. The sum is normalized by shifting its fraction leftward until the sign bit differs from the next bit, and the exponent is decremented by the number of bit positions of this shift.

Subtraction is identical to addition, except that the fraction of the subtrahend is complemented before the addition is performed. In standard two's complement fashion, the bits are inverted and a "hot one" is introduced at the adder's LSB carry-in position. Fixed-to-floating-point conversion and normalization of floating point numbers is performed by left shifting the fraction as necessary to eliminate redundant leading zeros or ones while decrementing the exponent to compensate.

Multiplier

The floating point multiplier shown in Figure 7 is basically a 16-bit two's complement fixed point multiplier, a 6-bit adder, and a very simple normalizer. It does not require an operand conditioner, such as the adder's denormalizer. Furthermore, its output conditioning requirements are minimal: if the input operands are normalized, then the product is at most one shift left or right from normalization. Hence, the 16-bit half-barrel shifter of the adder is replaced by a small 16-bit, 3-position multiplexer. The only "communication" between fraction and exponent occurs in the final product normalization step, where the exponent must be incremented or decremented to compensate for any shift in the fraction. With the normalizer defeated, the chip performs 16-bit two's complement, fixed point multiplications.

If not properly accommodated, overflows and underflows can adversely affect the accuracy of a signal processing system or the stability of a digital filter. The floating point chips include

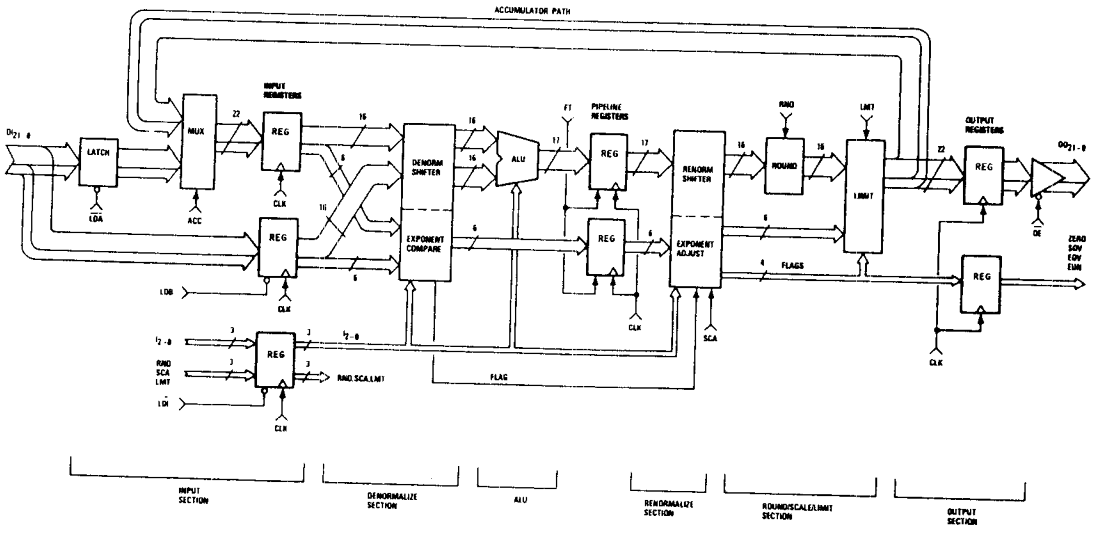

Figure 6. Single Chip 22 Bit Floating Point Adder Block Diagram

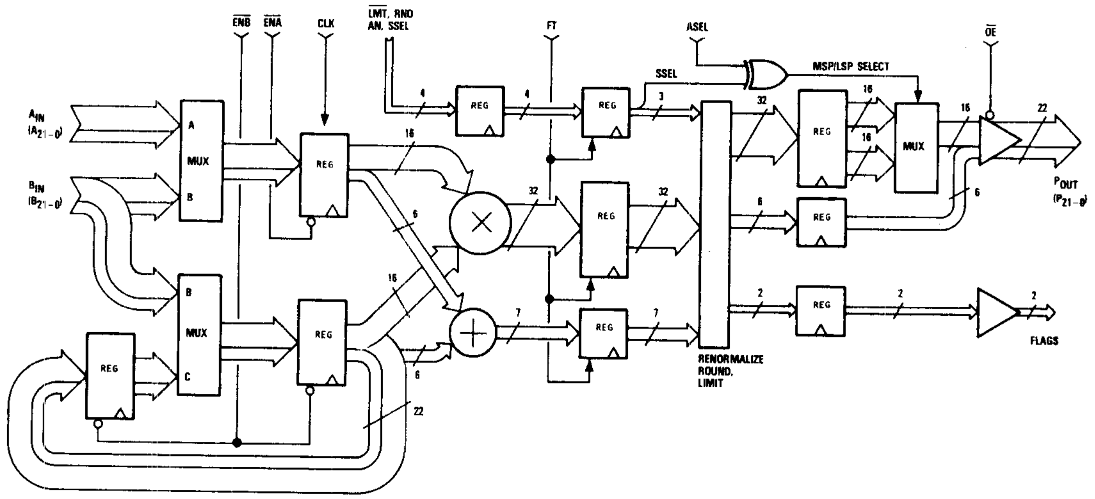

Figure 7. Single Chip 22 Bit Floating Point Multiplier Block Diagram

user-controllable limit circuits, which implement "hard limiting" when enabled. In the limit mode, overflowing positive quantities are replaced with a full-scale positive value while overflowing negatives cause a full-scale negative output. The limiter also operates on underflows, which are quantities too small to be properly normalized. When the limiter is enabled, such numbers are replaced with a true floating-point zero. If the user turns off the limiter, overflows will "wrap around" in the normal two's complement fashion, producing extremely wrong results. (For example, full-scale positive plus one LSB equals full-scale negative, due to the carry into the sign bit.) Underflows are far less catastrophic - the output, though unnormalized, will be nearly correct. However, it is usually safest to limit these underflows to true floating point zero, since most floating point hardware is designed to work on normalized operands only. The TRW floating point devices generate flags to alert the user to overflow conditions, to allow their explicit handling off-chip. To avoid round-triggered overflows (of positive numbers) or underflows (negative numbers), limiting occurs after rounding. Although the limiter exacts a slight speed, power, and chip size penalty, it is a valuable feature in most DSP work.

Over the years various floating-point rounding schemes have been proposed, attacked, and defended. The procedures described herein and used in the adder and multiplier are simple, logical, and relatively unbiased. The rounding scheme is based on the standard "round toward positive infinity," in which both positive and negative numbers are incremented by 1/2 LSB, then truncated. This operation effectively gives a "round to nearest quantum" result, but with a small positive bias for quantities exactly halfway between two quanta. These values are always raised to the next higher quantum (i.e., negatives become less negative and positives become more positive). To eliminate this small bias, "half-way" quantities would have to be rounded upward half the time and rounded downward (truncated) half the time, as suggested in the IEEE's 32-bit floating point standard. (The IEEE format's "sticky bit", necessary for this "unbiased rounding," complicates the hardware significantly.)

The rounding procedure performed in the adder operates as follows: During denormalization, all bits of the smaller operand which are shifted beyond the other operand's LSB are truncated. In the addition mode, this truncation introduces a slight downward bias, which will be off-set later by the rounding procedure, which is always upward. The result of the addition (or subtraction) will have a 16- or 17-bit fraction, depending on word growth. If the result is 16 bits, then no rounding is done, since the rounding position (17th bit) is unoccupied. In contrast, if the result is 17 bits long, the lowest bit is available for 1/2 LSB rounding, while the upper 16 bits comprise the fraction LSB through sign bit.

CONCLUSIONS

This paper shows the high payoff of synergistic use of commercial and semi-custom integrated circuits. Specifically, a 40 MHz pipeline FFT processor implementing 22 bit floating point arithmetic has been developed. The processor complexity was decreased by 60% through the development of a standard cell VLSI delay commutator circuit. The arithmetic is performed with single chip adders and multipliers that use a 22 bit floating point format. The processor is simpler and correspondingly lower in power, size, and cost than designs using 32 bit floating point arithmetic. Similar improvements can be achieved in a wide variety of signal processing systems by carefully tailoring the algorithms, processor architecture, arithmetic precision, and technology selection.

REFERENCES

[1] E. E. Swartzlander, Jr. and G. Hallnor, "Frequency-Domain Digital Filtering with VLSI," in VLSI and Modern Signal Processing, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1985, Chapter 19.

[2] E. E. Swartzlander, Jr., L. S. Lome, and G. Hallnor, "Digital Signal Processing with VLSI Technology," Proceedings IEEE International Conference on Acoustics, Speech, and Signal Processing, pp. 951-954, 1983.

[3] B. Gold and T. Bially, "Parallelism in Fast Fourier Transform Hardware," IEEE Transactions on Audio and Electroacoustics, Vol. AU-21, pp. 5-16, 1973.

[4] H. L. Groginsky and G. A. Works, "A Pipeline Fast Fourier Transform," IEEE Transactions on Computers, Vol. C-19, pp. 1015-1019, 1970.

[5] J. A. Eldon and C. Robertson, "A Floating Point Format for Signal Processing," Proceedings IEEE International Conference on Acoustics, Speech, and Signal Processing, pp. 717-720, 1982.

[6] L. R. Rabiner and B. Gold, Theory and Applications of Digital Signal Processing, Englewood Cliffs, Prentice-Hall, Inc., p. 611, 1975.

[7] E. E. Swartzlander, Jr., W. K. W. Young, and S. J. Joseph, "A Radix 4 Delay Commutator for Fast Fourier Transform Processor Implementation," IEEE Journal of Solid-State Circuits, pp. 702-709, 1984.

[8] A. Kanemasa, R. Maruta, K. Nakayama, Y. Sakamura, and S. Tanaka, "An LSI Chip Set for DSP Hardware Implementation," Proceedings IEEE International Conference on Acoustics, Speech, and Signal Processing, pp. 644-647, 1981.

[9] G. W. Preston, "The Very Large Scale Integrated Circuit," American Scientist, Vol. 71, pp. 466-472, 1983.