

G.A. Jullien

VLSI Research Group, University of Windsor, Windsor, ON, Canada N9B 3P4

ABSTRACT

In this review paper we discuss selected issues associated with the implementation of arithmetic for VLSI Digital Signal Processors. We start with a Silicon Technology Roadmap view of the next decade, in order to grasp some of the issues facing the next generation of VLSI designers, particularly associated with high performance DSP systems. We use this roadmap to open the discussion on the role basic arithmetic operations play in the construction of DSP systems; in particular we look at the interplay between algorithms, architecture, arithmetic representation and circuit implementation. Many of the illustrative examples are taken from work conducted in the VLSI Research Group, University of Windsor over the past few years, including on-going work.

Keywords: VLSI; Digital Signal Processing; Computer Arithmetic; Number Systems; Pipelined Circuits

1. INTRODUCTION

Although this paper is intended to concentrate on certain issues in arithmetic for DSP; it is appropriate, however, to briefly review the time line from the 1960's until the first decade of the new millennium. Digital Signal Processing was born in the 1960's. The (re)discovery of the Fast Fourier Transform¹, marked the birth and the field of digital signal processing became a mature discipline within the next decade. Digital Signal Processing has always received a great impetus from electronic technological advances; it often rides the crest of that wave and sometimes is responsible for pushing it.

General purpose DSP chips first appeared in the late 1970's, and were essentially microcontroller technology with most of the area given up to an array multiplier. The architectural technology was pushed by the requirement for large numbers of multiplications to be performed at high speeds. As technology improved, the multiplier shrank in relative area, to the point where the multiplier grew to a complete floating point ALU. The need to amortize development costs over large volumes of production chips, led DSP chip design in similar directions to that of single-chip general processor design; instruction based bus architectures with a handful of 'fast' floating point ALUs on board. The chips are relatively easy to use, and DSP algorithms can be 'firm-wired' by OEM houses in a very short time. The raw processing power of the silicon, however, is sacrificed for the convenience of programmability over a wide variety of algorithms.

Special purpose DSP chips also saw their birth in the late 1970's. Special purpose architectures are programmable only in a local algorithmic sense (e.g. a FIR filter fixed architecture with programmable coefficients) but the potential processing speeds can be 1 to 2 orders of magnitude greater than those obtained by general purpose programmable designs. Quite often this class of chips uses special arithmetic structures and arrays which trade-off general use (e.g. 32 bit IEEE standard floating point ALUs) against lower *Area* and *Power*, and higher *Speed* (e.g. an array of redundant binary integer multiplier/accumulators). It is this latter class of chip architectures that have driven much of the work in our VLSI Research Group.

The fabrication densities available in the 1990's have seen a merger of these two architectures into programmable DSP chips with special hardware functions. A good example of this is the TMS320C8x generation that contain several DSP cores with a master RISC controller and an IEEE standard floating point ALU. Such DSP chips are 'semicustom' configured based on special peripheral circuitry, such as video controllers. DSP *cores*, CAD files that place full DSP layouts onto the OEM CAD system, are often used in fast turn-around special DSP chip design.

Since the 1970's, programmable logic devices have been used to implement 'glue' logic in digital systems. These devices started as arrays of AND and OR gates, with programmable interconnection points (e.g. PLAs) and were used to implement combinational logic or even complete finite-state-machines. Field Programmable Gate Arrays (FPGAs), a much more flexible extension of this idea, came onto the market in the 1980s. These devices are now ubiquitous, with RAM based designs (i.e. logic and interconnect programmed by downloading bit maps into control RAM) the most popular. Although there is a heavy price to pay for programma-

bility, in terms of a large reduction in gates/mm² compared to full custom silicon, this is acceptable when a very fast turn-around is required for a new product. Often there is a product migration path from FPGA implementation to full custom silicon as product volume increases. The densities of FPGAs have now reached the point where they are serious contenders for implementing small DSP arrays, and we are beginning to see programmable arithmetic logic implemented within the FPGA structure.

The fabrication technology that drives these advances is nothing short of remarkable. It is probably true to state that the fabrication technology has advanced much faster than the design tools that allow us to use it. Clearly we always have tools to use the latest fabrication technology, but these are certainly not optimum in terms of ease of use, and automation and verification of design. It is expected that current trends in fabrication technology will continue until the year 2010, after that there may be some slow down as totally new technology starts to take over. Perhaps this will allow the software to catch up!

The state-of-the-art at 2010 promises fabrication densities that allow 2 orders of magnitude increase in DRAM and SRAM bits, and over an order of magnitude increase in logic transistors per unit area. Table 1 shows several features, with an emphasis on high performance processors, from a combined Process Technology Roadmap².

	1995	1998	2001	2004	2007	2010	Driver
Year of first DRAM shipment	1995	1998	2001	2004	2007	2010	
Feature Size (µm)	.35	.25	.18	.13	.10	.07	
DRAM bits/Chip	64M	256M	1G	4G	18G	64G	D
SRAM (cache) Bits/cm ²	2M	6M	20M	50M	100M	300M	L (µP)
Logic Transistors/cm ² (packed)	4M	7M	13M	25M	50M	90M	L (µP)
On-chip clock (MHz) (high perf.DSP)	400	600	800	1100	1500	1900	DSP
Wiring Levels (Logic)	4-5	5	5-6	6	6-7	7-8	µP
Power Supply (Desktop)	3.3	2.5	1.8	1.5	1.2	0.9	µP
Power Supply (Battery)	2.5	1.8-2.5	0.9-1.8	0.9	0.9	0.9	A
Maximum Power W (high perf. w. heat-sink)	80	100	120	140	160	180	µP

Table 1: Combined Process Technology Roadmap

On the basis that single-chip DSP devices contain a mix of custom logic and memory (mainly SRAM), we would expect between one and two orders of magnitude decrease in the area taken up by current designs. Future DSP chips will clearly contain many fully programmable blocks with surrounding structures of special purpose arrays designed for a specific task (e.g. video compression, coding/de-coding, 3D rendering etc.) The 5-fold expected increase in clock rate and the 20-fold increase in processor complexity brings a 2 order of magnitude increase in processor computational power. There are many potential applications, however, that will require this increase in processing power; some obvious examples are virtual reality processors, real-time rendering, model based video compression, PCS controllers etc.

The dominant technology will probably remain as CMOS. BiCMOS is currently relegated to the high end market where speed and versatility advantages outweigh the cost penalty; it has failed to oust CMOS as the dominant silicon process. The early promises of GaAs replacing silicon in standard high speed digital applications appear to be unfulfilled. There will be increased emphasis on optical data transmission, routing etc. where GaAs comes into its own. GaAs will also probably continue to be used in applications, such as wireless communications, where silicon based processes cannot, at present, meet the performance requirements³. The fate of other technologies, such as SiGe are still in some doubt.

2. DESIGN ISSUES

There are many issues in DSP array design that present themselves; in general these will be driven by selected entries in the Roadmap. Some of the technology design issues, that are affected at the circuit and architecture level, are: maximum power; clock rate; current spikes and ground bounce; interconnect complexity; design for testability; verification.

In no particular order of importance the design issues associated with the technology issues include:

- Algorithm design
- Architecture design
- Implementation technology
- Verification and test
- Library circuit design
- Arithmetic implementation

Ideally all of these issues should be considered at the same time (as optimizing variables for a silicon cost function), but this is rarely the case since this makes the design space too large for quick solutions. Often these issues are each considered an area of expertise in their own right, and we end up with the ‘brick wall’ design scenario: the algorithm is designed by the algorithm experts and thrown over the ‘brick wall’ to the architecture team; the architecture is designed by the DSP architecture experts and thrown over the brick wall to the VLSI design team; the... etc. In this scenario the final design never returns to the algorithm team; anyway, they probably do not have the expertise to understand the VLSI concepts! This is an extreme scenario, but elements of it exist in all product design. To some extent, the use of programmable DSP chips has embraced this philosophy, since the designer’s task mainly consists of writing software for a fixed architecture; most of the software being written in a high level language that completely hides the architectural structure of the hardware. The advantage, of course, of this programmable approach, is a very fast design cycle with the ability to correct errors and update features without a change of hardware; thus the sacrifice in inefficiency is acceptable.

For established algorithms that will form peripheral circuitry on future DSP chips, it makes sense to implement the best silicon design possible. A good, classical, example is a video encoder/decoder. A typical design for a H.261 video coder^{4,5} (designed for ISDN visual telephony applications) is shown in Fig. 1.

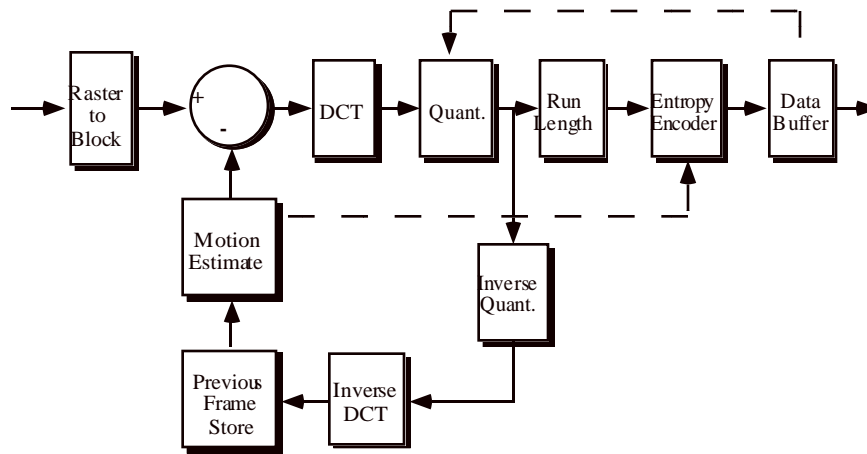


Fig. 1. Embedded Video Coder

Consider the DCT block. At the algorithm level we have the basic definition of an N -th order DCT as given by matrix eqn. (1).

$$\mathbf{X} = \mathbf{C}_N \mathbf{x} \quad \text{where} \quad \mathbf{C}_N = \left[\rho_k \cos \left(k \left(n + \frac{1}{2} \right) \frac{\pi}{N} \right) \right] \quad (1)$$

ρ_k is unity except for the first value (at $k=0$) for which case it has the value $1/(\sqrt{2})$. Using a technique to factor \mathbf{C}_N into lower order matrices, without multiplications, (see eqn. (2) for N even⁶)

$$\mathbf{C}_{2M} = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 & 1 \\ 0 & 1 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & 0 & \dots & 0 & 1 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & 1 & 0 & 0 & \dots & 0 \\ 0 & \dots & 0 & 1 & 0 & \dots & 0 \end{bmatrix} \begin{bmatrix} \mathbf{C}_M & 0 \\ 0 & \hat{\mathbf{I}}_M \mathbf{C}_M^{IV} \hat{\mathbf{I}}_M \end{bmatrix} \begin{bmatrix} \mathbf{I}_M & \hat{\mathbf{I}}_M \\ \hat{\mathbf{I}}_M & -\mathbf{I}_M \end{bmatrix} \quad (2)$$

we can implement a 15-point DCT using a ‘fast’ algorithm with only 1 multiplication in each data path⁷ (see Fig. 2 a)). This now allows us to reduce the scaling requirements on an integer number system used to implement the arithmetic. Using some rather exotic direct product ring mappings⁸, we can restructure the architecture, as shown in Fig. 2 b). We have reduced the interconnection requirement and relaxed the synchrony requirement for the clock over the entire *DCT algorithm* computation (there is an overhead associated with the *mapping* which does not respond to these reductions⁸.)

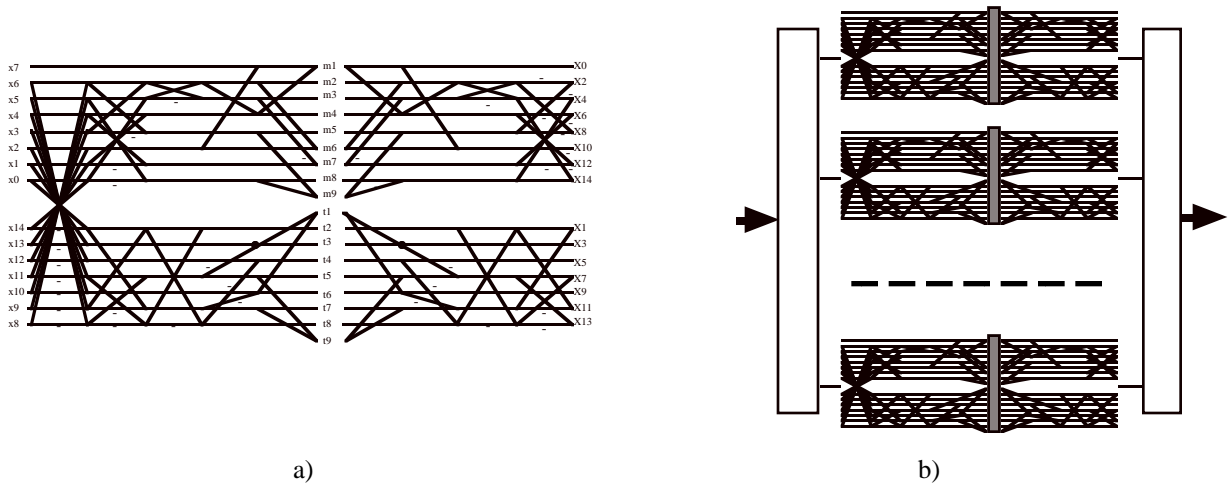


Fig. 2. Mapping a 15-point DCT to a Direct Product Ring

Relaxing the synchrony requirement means that we may be able to purposely skew the clock in order to reduce the associated current spike. We have therefore produced positive results in both the interconnection and current spike technology issues by simultaneously manipulating the algorithm, architecture and arithmetic. This solution is, of course, somewhat off the mainstream; the number of samples into the DCT does not fall into the standard powers of two used by current DCT implementations, and the arithmetic and mappings are somewhat exotic. This does not mean, however, that we should not consider such approaches for denser technologies.

In order to obtain a completely different architecture, we can use the recurrence formulas for computing Chebychev polynomials⁹, in order to produce recursive computational procedures for the DCT¹⁰. We now write the DCT from eqn. (1) as eqn. (3), where V_n is obtained from a definition of Chebychev polynomials of the third kind¹¹.

$$Y(k) = \frac{2}{N} \gamma_k (-1)^k \cos\left(\frac{\theta_k}{2}\right) \sum_{n=0}^{N-1} y(N-1-n) V_n(x_k) \quad (3)$$

Using the recurrence relations for computing V_n , we indirectly obtain the recursive architecture shown in Fig. 3¹⁰. This is no longer a ‘fast’ architecture (i.e. providing a reduction in multiplications compared to computing the DCT directly), but it has the advantage of providing a filter like structure for each coefficient. If we wish to only compute some of the DCT coefficients (often required for data compression) then each coefficient can have its own fixed coefficient filter, and we may use the efficiencies available when we have a priori knowledge of multiplier values. The disadvantage is that the computation is recursive, so that there are

N cascaded integer multiplications required to generate the output coefficient. This will involve scaling the arithmetic because of the large integer number growth. This architecture is clearly a candidate for implementation using programmable DSP chips.

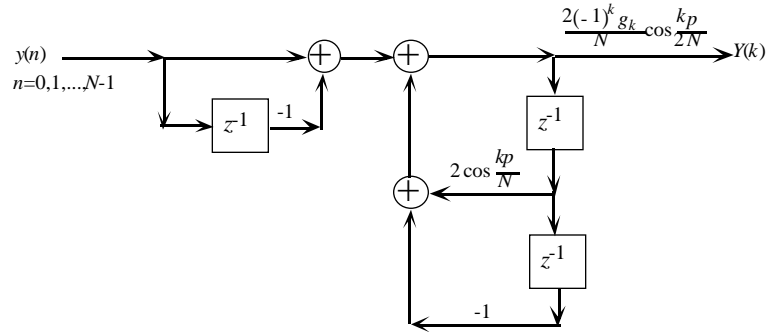


Fig. 3. Recursive Architecture for the DCT

3. NUMBER REPRESENTATION

The key to manipulating arithmetic implementation techniques, is in the number representation used. In this section we will briefly discuss the role of the representation of integers in computation over rings and the efficiency of integer computation using redundant representations. We have purposely avoided floating point arithmetic, since it will probably not be used in the types of processing arrays we are considering here, and we also wish to be brief. For the same reason we have not included many other types of number representation used in computing over the past few decades.

3.1. Computing over rings

Undergraduate education concentrates on using the binary number system to implement computer arithmetic. A B-bit binary number system representation of a positive number, x where $x_i \in \{0, 1\}$, is shown in eqn. (4).

$$x = \sum_{i=0}^{B-1} x_i 2^i \quad (4)$$

We often think of the binary number system as a product of the computer age because the digit set $\{0,1\}$ is easily implemented by logic gates. The binary number representation is, in fact, about four thousand years old¹². The Harrâpan culture of the Indus River used balance stones, for weighing, in the sequence 1,1,2,4,8,... The sequence of stones is easily found by selecting a stone as the basic weight 'digit', then finding another stone of the same weight using the balance. By placing these two stones on one side of the balance, a stone of twice the basic 'digit' weight can be selected, and so on. The sequence is immediately recognizable as that used in a classical D/A converter, and the technique of choosing a stone of twice the weight of a previous stone can be thought analogous to the classical R/2R ladder network principle. The fact that the binary number system also allows a fairly easy and quite regular implementation of arithmetic, using Boolean algebra and logic gates, is, however, a finding from our computer age.

A 'closed' arithmetic system is directly obtained from the binary number representation by computing the operation of addition and multiplication modulo 2^B . This forms a finite ring in which a negative number, $[-x]$, can be defined using the additive inverse: $[-x] = 2^B - x$; the additive identity is 0. Using only a B-bit binary representation, we are unable to form the number 2^B , and so the inverse is computed as $[-x] = [(2^B - 1) - x] + 1$. The RHS expression in square parentheses is found by simply inverting each bit of x , the ones complement; we then add 1 to this result, the twos complement. We use the following notation, expressed in eqn. , to define the ring, modulo 2^B .

$$R_{2^B} = \left\{ 0, 1, \dots, 2^B - 1 \mid \oplus_{m_{2^B}}; \otimes_{2^B} \right\} \quad (5)$$

where $\oplus_{m_{2^B}}$ and \otimes_{2^B} represent addition and multiplication modulo 2^B .

The Chinese also developed a form of computation over rings, which dates back approximately two thousand years. In this case several rings are used quite separately and independently. The difference between the two uses of the rings is that, in the binary case, the ring has enough elements to contain the final answer in any closed computation (if it does not then the ring is simply extended to a larger power of two), in the Chinese case we simply add more rings (carefully!...see^{13,14}). A major difference in the representation is that the value of the binary number is immediately available from its digits, in the Chinese case a mapping has to be invoked in order to find the value. The binary system is therefore more natural to us in terms of always providing an immediate value for the number; the Chinese system should not be neglected, however, since it is responsible for the DCT architecture in Fig. 2 b).

3.2. Representation Redundancy

Both the binary number system and the Chinese system discussed in section 3.1 provide unique representations for every element in the ring. If the rings are not big enough to contain the result of a computation, then we obtain a residue of the result. There are, of course, an infinite number of computational results that produce the same residue; this is sometimes referred to as a system of residual classes¹⁵. Redundant representations provide the opposite condition; they allow the same number to have several different representations. Although, on the surface, it would appear that redundant representations are inefficient, the opposite quite often happens. We can demonstrate this using the Roman number system! The year 1995 can be written down many different ways: MDCCCCLXXXV, MCMXCV, MCMVC, MXMV, MVM. Ordered sequences of characters representing the same, or reducing, values are added. A sequence of two increasing values implies a subtraction of the first from the second. We can see that redundancy can have its advantages if we are allowed to choose the representation from the list. Applying redundancy to binary systems has both the advantages of canonicity (a representation with a minimum number of non-zero terms...equivalent to the last item in the 'Roman' list above since there are no zeros in the Roman system), and flexible order of computation (we may compute from the most significant bit (MSB) down to the least significant bit (LSB)...the opposite of the computational algorithm required for the unique binary number system).

Redundancy in a weighted, positional number system, such as the binary number system, is obtained by providing a mechanism for overlapping the magnitude associated with adjacent digits. This does not happen with binary because each successive digit has a magnitude that is more than the sum of the preceding digit magnitudes (the extra '1' in the Harrâpan sequence guarantees that!) If we allow, for example, 2 different digits other than zero (or perhaps a negative digit) to be associated with each weight, then the uniqueness disappears. The binary-like number system, where we use the same weight sequence $\{2^i \mid 0 \leq i < B\}$ but allow the digits to be in the set $\{\bar{1}, 0, 1\}$, where $\bar{1} = -1$, is a good example of a redundant number system; this is sometimes referred to as a signed-digit number system, and corresponds to a minimally redundant set (number of digit values one more than the radix) after Avizienis¹⁶. Both the redundancy and the canonicity can be observed by considering the number 127. Since the binary representation is also a valid signed-digit representation then one way of writing the number is 01111111 (assuming $B=8$). We may also write the number as $1000000\bar{1}$ which is the canonic form. A signed-digit system is sometimes referred to as a canonic signed-digit system (CSD) if all the numbers use their canonic form. One property of the signed-digit system is that the evaluation of digits for additions and subtractions only requires a knowledge of the lower 2 digits, not all of the digits as in the binary system. This means that the time to perform addition or subtraction (hence multiplication) is independent of B . For fixed coefficient systems (e.g. when there is a priori knowledge of the multipliers in a filter) we can eliminate the partial product hardware associated with the zero terms in the CSD representation of the multiplier. This idea had been published by A.D. Booth in 1951¹⁷, now known as Booth's algorithm, but cast as a serial algorithm for a general multiplier architecture. The algorithm keeps track of the succession of multiplier digits from LSB to MSB. Addition or subtraction only takes place if adjacent digits are different.

We can also overlap digit magnitudes by overlapping the weights, and retaining the original binary digit set, $x_i \in \{0, 1\}$. We can also use an extended digit set, such as $\{\bar{1}, 0, 1\}$, we simply increase the amount of redundancy. Consider writing a binary number as a polynomial in several indeterminates, say $X_1 = 2^1$; $X_2 = 2^2$; $X_3 = 2^4$; $X_4 = 2^8$ and allowing the coefficient of the polynomial to be within one of the above sets, as shown in eqn. (6)¹⁸.

$$Y = \sum_{i_1, i_2, i_3, i_4} y_{i_1, i_2, i_3, i_4} X_1^{i_1} X_2^{i_2} X_3^{i_3} X_4^{i_4} \quad (6)$$

The weights are now $X_1^{i_1} X_2^{i_2} X_3^{i_3} X_4^{i_4}$, and clearly there is considerable overlap, in terms of the powers of 2 magnitude, between adjacent polynomial ‘weights’, because we can always write a given power of with different products and powers of the indeterminates. As an example of the redundancy, using only the binary digit set $y_i \in \{0, 1\}$, consider representing the number $Y=305$. Two possible representations are shown in eqn. (7). -305, for example, can be represented using the signed-digit set and simply negating the coefficients. In this representation it is more difficult to determine the canonic form, in fact we may not be able to uniquely define one! The answer depends on the arithmetic implementation procedure, as discussed in section 4. We can, however, immediately make the observation that although both representations use the same number of digits, the first representation requires fewer indeterminates but uses powers greater than one.

$$305 = \begin{cases} X_3^2 + X_2^2 X_1 + X_2^2 + 1 \\ X_4 + X_3 X_2 - X_3 + 1 \end{cases} \quad (7)$$

There are many other redundant representations based on sequences of weights. An exotic example is the Zeckendorf representation¹⁹ which uses the Fibonacci numbers as weights. An intriguing example is the Double-Base Number System²⁰ where a number is represented using weights formed from the double-base $3^i 2^j$, and digits from the binary set. Unlike the polynomial scheme above, there is a unique minimal representation which is very sparse, but finding it is an exponentially complex problem. It is possible to use close to minimal representations which are sparse enough for additions to take place. The reduction rules operate over the two-dimensional space, where the two dimensions are based on $\{2,3\}$. Borrowing from morphological image processing, it may be possible to use analog cellular neural networks to perform both the arithmetic and the canonic reduction²⁰.

4. IMPLEMENTING THE ARITHMETIC

The circuit and architectural search space that is generated from the different number representations is extremely large. Here will pick a tortuous path through a very few diverse implementations of the closed operations (addition and multiplication). Until very recently, most arithmetic designs concentrated on speed; recently the concentration has moved to power reduction as fabrication technology advances have made designing for speed less of a critical issue. For DSP applications is important to define the widely used terms *speed* and *fast*. They are often used synonymously, but *fast* implies a measurement in time only, whereas *speed* connotes a rate. For DSP that rate is surely samples per second, and a high speed design should refer to a circuit that processes large numbers of samples per second. For fast designs we look at minimum critical path circuits. For high speed designs, we need to look at pipelined computations, rather than pipelined data that is the result of fast computations.

4.1. Fast Binary Adders

The binary adder is probably the most re-designed circuit, after the basic logic gates, in any digital circuit technology. The basic circuit is a full adder, but producing word-wide addition involves selecting from a wide-variety of algorithms, all promising to mitigate the problem of the ripple carry. Most of the algorithms are referred to by their manipulation of the carry (e.g. Carry Lookahead; Carry Skip; Carry Save; Manchester carry chain etc.) In order to squeeze the ‘last ounce’ of performance out of the silicon, these circuits are designed right at the transistor level using full-custom layout. We will take one of the popular algorithms as an example. The carry lookahead adder (CLA) was recently identified as being one of the most efficient in terms of logic transitions; this directly leads to efficiency in power dissipation. The CLA defines the sum and carry at bit position i in terms of propagate, p_i , and generate, g_i , functions and a special operator, \circ ²², as shown in eqn. .

$$c_i = (g_{1,i} p_{1,i}) \circ c_0 \quad s_i = x_i \oplus c_{i-1} \quad (8)$$

Most often we see the design of such adders defined at the gate level, but in order to extract the highest performance from the silicon, we should look at the architecture from the transistor circuit-level. A recent example of this is to couple a dynamic multiple-output transistor circuit approach with an architectural search²³. The use of transistor sizing techniques completes the ‘performance squeezing’ approach. In this example the adder is designed for minimum critical path delay, but similar techniques can be used for

power reduction. Fig. 4 shows an example of a dynamic logic circuit for the CLS and also the irregular architecture that results from the circuit design approach²³. A 32-bit adder produced from this design has a measured 2.7ns critical path for a very mature 1.2μ CMOS process!

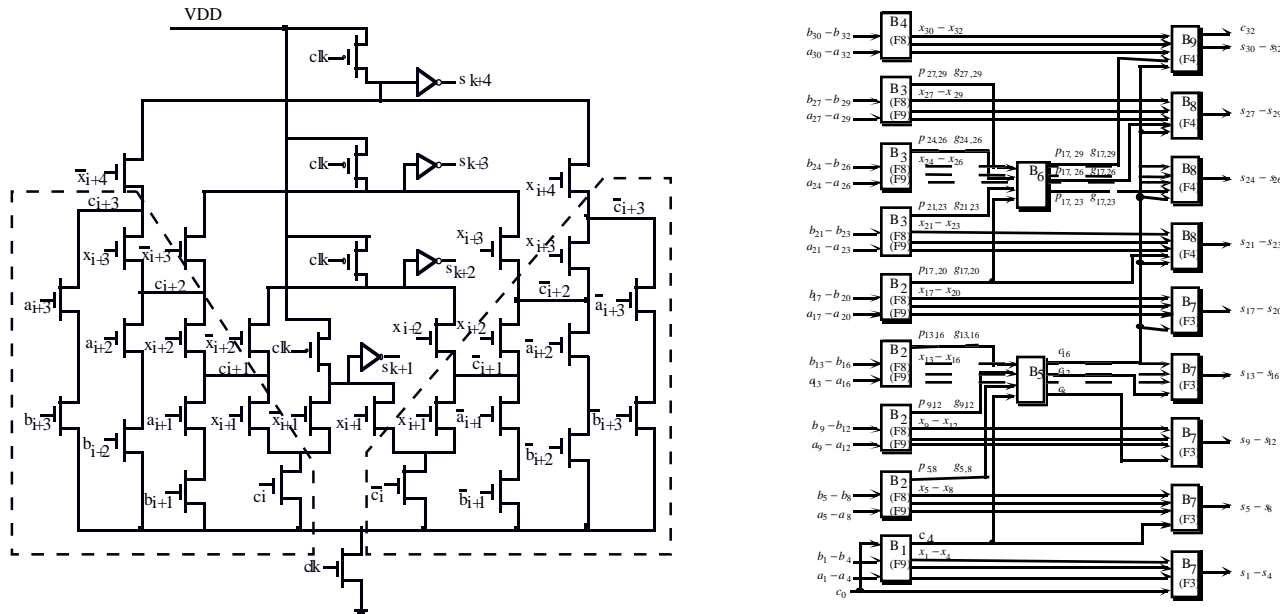


Fig. 4. A Dynamic Logic CLA Adder (typical circuit and the irregular architecture)

4.2. Fast Binary Multipliers

Most multipliers used in DSP chips are array multipliers. Essentially they implement the classical partial product addition we associate with performing multiplication using paper and pencil (when our calculator batteries are flat!) An interesting issue is the selection of the basic logic block to be used in the multiplier. Traditionally this is a carry save adder (which allows the carry to simultaneously propagate down the partial products and across the adders) but we can also consider higher order blocks...essentially multiple-bit adders. These adders can either add bits of the same weight (often these blocks are referred to as compressors) or they can add bits of different weights²⁴. By carefully examining the way that bits of the same weight are ‘compressed’ we can arrive at an optimum sequence for applying the compression circuits²⁵. The difference between the two approaches is shown in Fig. 5. Note that both arrays use the same number of adders (shaded blocks are half adders...2-inputs, and white blocks are full adders...3-inputs). Note that both multiplier structures are arrays and that each compresses bits, even though they are known as ‘array’ and ‘column compression’ multipliers separately.

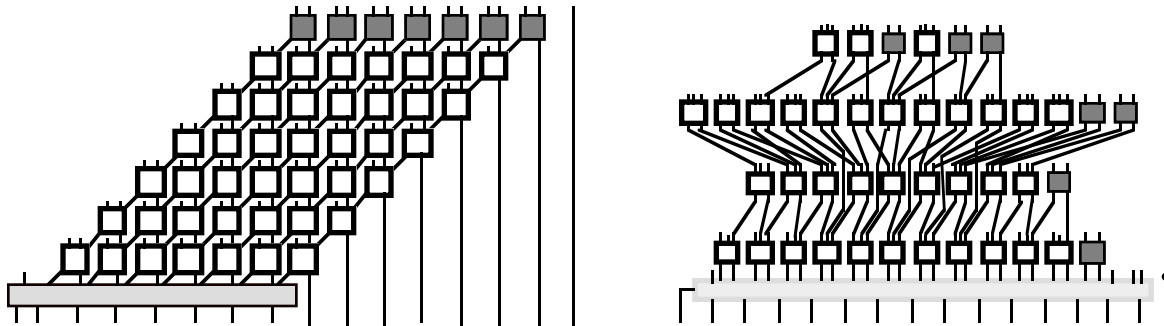


Fig. 5. Comparison between an array (left) and column compression (right) multiplier

The column compression multiplier has long been overlooked at the VLSI level because of the irregularity of the interconnect

structure, but recently we have shown that it is possible to obtain more regularly interconnected arrays while retaining both the minimum number of adders and optimum critical path²⁶. Two designs for a regularly interconnected 8×8 2's complement multiplier are shown in Fig. 6. The design in Fig. 6 b) demonstrates the use of a technique to reduce the final fast adder²⁶.

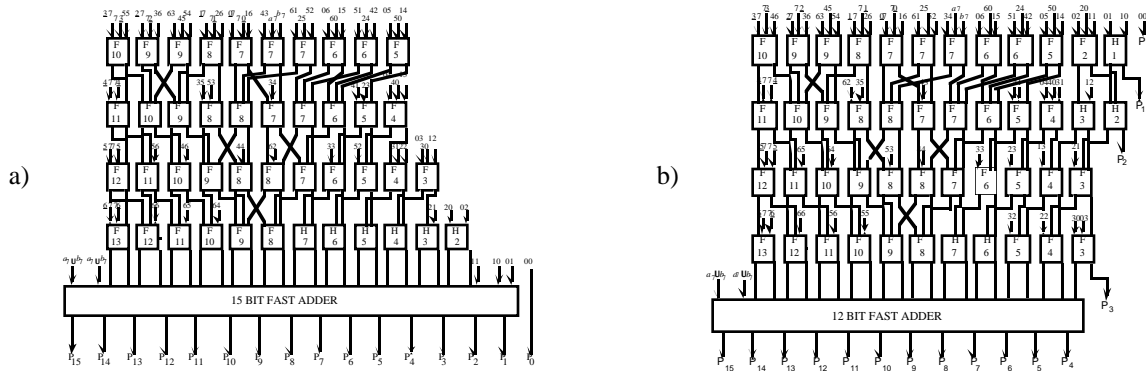


Fig. 6. Two Designs for an 8×8 Regular Interconnection Column Compressor Multiplier

4.3. Redundant Arithmetic Blocks

For brevity we will only discuss blocks for signed-digit binary arithmetic. Because 3 elements are used in the digit set, standard logic gate implementation uses 2-bits for each digit. Since 2-bits can represent 4 elements, we have a choice of mapping the 3 elements to the four 2-bit combinations. There are 9 possible mapping schemes²⁷ as shown in Table 2.

	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	d	0	0	$\bar{1}$
0	1	1	1	1	1	1	1	1	1
1	0	$\bar{1}$	d	0	$\bar{1}$	0	1	$\bar{1}$	0
1	1	0	$\bar{1}$	$\bar{1}$	d	$\bar{1}$	$\bar{1}$	$\bar{1}$	$\bar{1}$

Table 2: The 9 possible mapping schemes for 2-bit signed-digit binary

Mapping 2 in Table 2 is interesting because it allows an implementation using only full adders²⁸. In order to map from signed-digit to binary (all 'exotic' number representations are usually required to interface to the binary number system) an on-the-fly scheme has also been suggested for this redundant format²⁹. The ability to compute from MSB to LSB using signed-digit binary is demonstrated in the DSP ALU array shown in Fig. 8³⁰. For division and square root, the ALU is able to compute these finite recursive algorithms using pipelined blocks around the recursion with no clock speed reduction due to pipelined ripple carries.

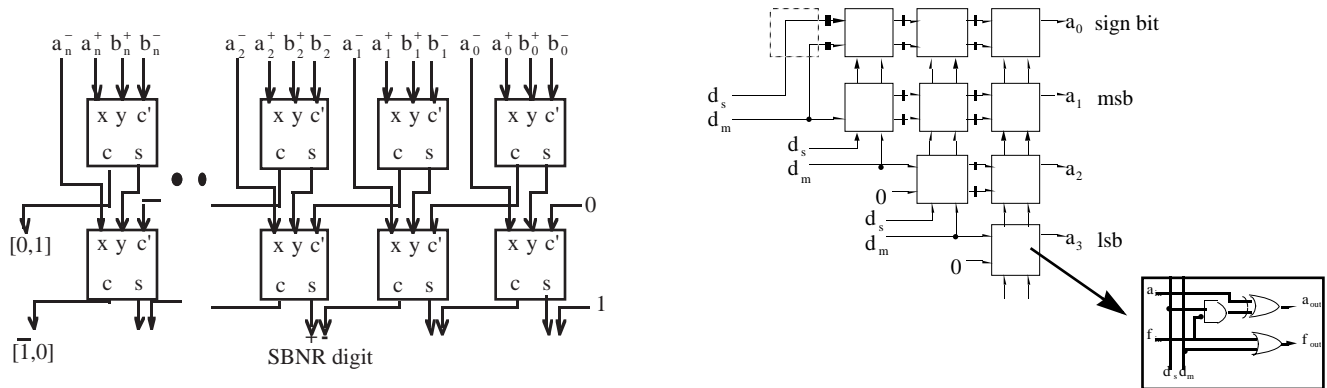


Fig. 7. Signed-digit adder and on-the-fly converter

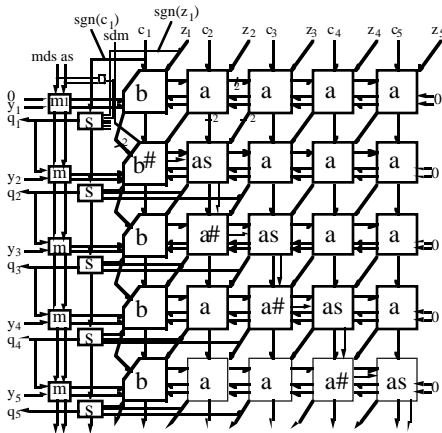


Fig. 8. Pipelined DSP ALU using recursive pipelines for division and square root functions

4.4. High-Speed Pipelined Circuits

Most DSP arrays employ pipelined circuitry. The true single-phase clock (TSPC) dynamic latch is one of the more important pipeline circuits that have been explored over the past decade³¹. Although not traditionally favoured by industry, dynamic logic is inherently faster than static CMOS logic because of the lower gate capacitance³². The disadvantages are associated with the floating evaluation node and the potential for charge sharing causing invalid logic levels; there are also issues concerned with the need to precharge the circuit at every clock pulse independent of the previous logic evaluation. Using arithmetic circuits with a low fan-in (2), Svensson's group at Linköping in Sweden were able to pipeline the arithmetic at many hundreds of MHz using very mature 2 μ CMOS technology³³.

Since the introduction of the TSPC circuit, many variations have been produced. As an example, the original master/slave TSPC latch is shown in Fig. 9 a) alongside a differential dynamic latch³⁴ (Fig. 9 b)) that overcomes the problem of precharge independent of previous logic evaluation.

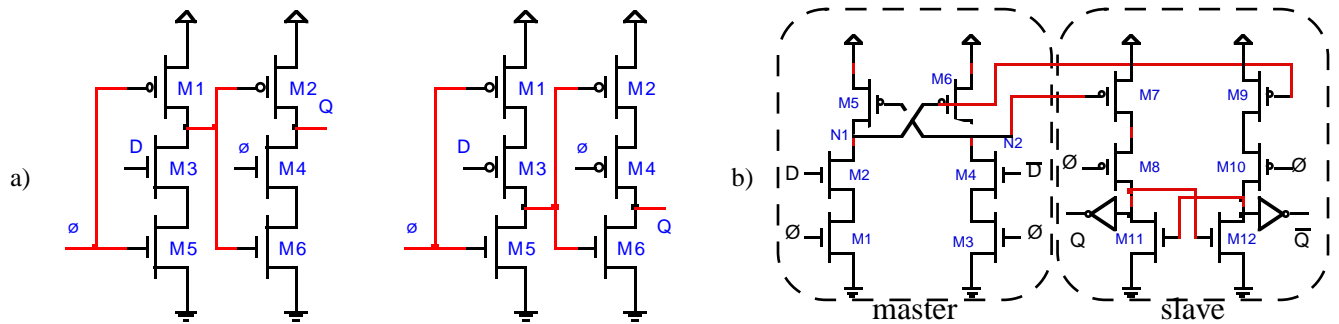


Fig. 9. a) TSPC Latch; b) latched differential cascode voltage switch logic

There have been reported problems with slow clock transitions using the original TSPC latch, our experiments indicate that these problems can be mitigated using appropriately sized transistors. The differential latch shown above can be clocked with very slow clocks, albeit at the expense of many more transistors for the differential logic used to drive the latches.

4.5. Finite Ring Arithmetic

One of the advantages of binary arithmetic is the ability for the associated logic to be easily factored into small logic arrays. Finite ring arithmetic normally does not allow easy factorization, and we often resort to simply storing the truth table (in a ROM) without any attempt at minimization. This is often referred to as the look-up table approach. The choice of rings is often related to ease of implementation, and we often use algebraic tricks, such as finite field logarithms, to help mitigate the implementation problems. A typical example is the Fermat ALU³⁵ which is used to implement the polynomial mapping scheme, discussed in section 3.2, using logarithms and a special representation technique for the polynomial coefficients. The ROM often consumes most of the ALU area, as shown in a Mod 257 Fermat ALU shown in Fig. 10. In this implementation we have used sized TSPC latches and modified dynamic adders (based on the circuit of Fig. 4). The ROM is a reasonably straightforward row/column design, using dynamic sense amp and decoder circuitry. Initial studies show a reasonable reduction in power dissipation compared with a conventional binary arithmetic implementation. We have also experimented with embedding minimized ROMs, built from binary trees, directly into the TSPC latch. We have successfully fabricated current sense latch circuits³⁶ with a fan-in of 14 that have reliably pipelined at 50MHz (0.8 μ BiCMOS process).

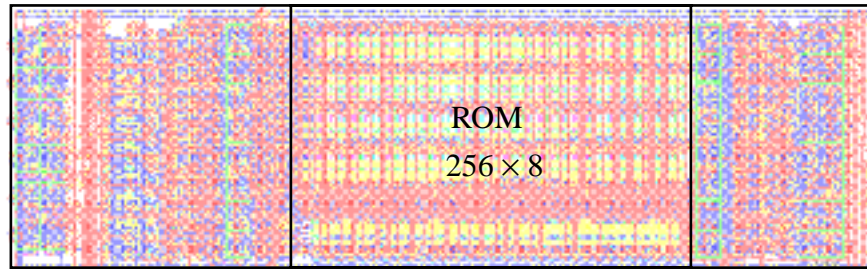


Fig. 10. Fermat ALU

5. OTHER ISSUES

There are many other important issues associated with DSP design for silicon. Design tools, for example, have been produced in proliferation by university groups and government and private research institutions, but much of this work has yet to make its way into the commercial tools market place. The integration of formal verification into DSP design, for example, is still in its infancy even though tools have been in development since the middle 80's. We hope to deal with this issue and some others in the presentation.

6. CONCLUSIONS

In this paper we have taken a rather tortuous path through the field of VLSI DSP. We started with elements from the Semiconductor Industry Association 1994 technology roadmap that will drive DSP VLSI technology over the next decade. Our main discussion opened with the interaction between DSP algorithm and architecture and the affect on the type of arithmetic that can be used for the implementation. The main orientation of the paper has been to focus on arithmetic issues and we have included a reasonably large section on number representation since this has a profound effect on the implementation. We have also taken a few snapshots of arithmetic implementation examples, both at the architectural and circuit level. Many of the snapshot examples have been taken from work performed at the VLSI Research Group, University of Windsor.

7. ACKNOWLEDGMENTS

The author gratefully acknowledges financial support from the Natural Sciences and Engineering Research Council of Canada and the Micronet Network of Centres of Excellence. VLSI design workstations, CAD software and fabrication services are provided by the Canadian Microelectronics Corporation.

8. REFERENCES

1. Cooley, J.W., and Tukey, J.W., "An algorithm for the machine computation of complex fourier series," *Math. Comput.*, vol. 19, pp. 297-301.
2. "The National Technology Roadmap for Semiconductors", Semiconductor Industry Association, 1994.
3. "Micronet Technology Roadmap: Phase I," Internal Publication from the Micronet Network of Centres of Excellence
4. CCITT SG XV, "Draft revision of recommendation H.261," Document 572, Mar. 1990.
5. Schäfer R., and Sikora, T., "Digital video coding standards and their role in video communications," *Proc. IEEE*, vol. 83, No. 6, pp. 907-924.
6. Wang, Z. "Fast algorithms for the discrete W transform and for the discrete Fourier transform," *IEEE Trans. Acoust., Speech, Signal Process.* vol. ASSP-32, pp.803-816, 1984.
7. Wang, Z., Jullien, G. A., and Miller, W. C., "One and two dimensional algorithms for length 15 and 30 discrete cosine transform", *IEEE Trans. Cir. Syst.* vol. 43, No. 2, February, pp. 149-153, 1996.
8. Jullien, G.A., Luo W., and Wigley, N.M., "High Throughput VLSI DSP Using Replicated Finite Rings", *Journal of VLSI Signal Processing* (in print).

9. Fox, L., and Parker, I. B. "Chebychev Polynomials in Numerical Analysis," Oxford University Press, London, 1968.
10. Wang, Z., Jullien, G. A., and Miller, W. C., 1994, "Recursive algorithms for the forward and inverse cosine transform with arbitrary length", IEEE Signal Processing Letters, vol. 1, No. 7, pp. 101-102.
11. Wang, Z., "Two new kinds of the Chebychev polynomials," J. Beijing University of Posts & Telecommunications, Vol. 12, No. 2, pp. 46-54, 1989.
12. Morrison, P. and Morrison, P., "The Physics of Binary Numbers," Scientific American, February 1996, pp. 130-131.
13. Soderstrand, M. A., Jenkins, W. K., Jullien, G. A. and Taylor, F. J., "Residue Number System Arithmetic: Modern Applications in Digital Signal Processing." IEEE Press, 1986.
14. Szabo, N.S. and Tanaka, R. I. (1967). "Residue Arithmetic and Its Applications to Computer Technology." McGraw-Hill Publishing Co.
15. Svoboda, A. (1957). "Rational numerical system of residual classes." Stroje Na Zpracovani Informaci, vol. 5, pp. 9-37, Prague.
16. Avizienis, A., "Signed Number Representations for Fast Parallel Arithmetic", IRE Trans. On Compt., Vol. EC-10, pp. 389-400, 1961.
17. Booth, A.D., "A Signed Binary Multiplication Technique," Quart. J. Mech. Appl. Math., vol. 4, Part 2, pp. 236-240, 1951.
18. Wigley, N.M., Jullien, G.A., and Reaume, D., "Large Dynamic Range Computations over Small Finite Rings." IEEE Trans. on Computers, Vol. 43, No.1, pp. 78-86, 1994.
19. Dimitrov, V. and Donevsky, B., "Faster multiplication of medium large integers via the Zeckendorf representation," The Fibonacci Quarterly, pp.74-77, 1995.
20. Dimitrov, V., Sadeghi-Emamchaie, S., Jullien, G.A., and W.C. Miller, "A Near Canonic Double-Based Number System (DBNS) with Applications in Digital Signal Processing," Proceedings of the 1996 SPIE Conference on Advanced Signal Processing Algorithms, Architectures, and Implementations IV, Denver, August, 1996.
21. Callway, T. K., and Swartzlander, E. E., "Optimizing arithmetic elements for signal processing", in VLSI Signal Processing, V (Edited by K. Yao et al) IEEE Publication, New York NY, pp. 91-100, 1992.
22. Ladner, R. E., and Fischer, M. J., "Parallel prefix computation", J. ACM, vol. 27, pp. 831-838, 1980.
23. Wang, Z., Jullien, G. A., Miller, W. C., Wang, J., and Bizzan, S.S., "Fast Adders Using Enhanced Multiple-Output Domino Logic", IEEE Journal of Solid-State Circuits (in print)
24. Wang, Z., Jullien, G. A., and Miller, W. C., "A New Architecture for Parallel Multipliers", IEE Electronics Letters, Vol. 28, No. 13, June, 1992, pp. 1278-1279, 1992.
25. Dadda, L. "Some schemes for parallel multipliers." *Acta Frequentia*, vol. 45, pp. 574-580, 1966.
26. Wang, Z., Jullien, G. A., and Miller, W. C., "A New Design Technique for Column Compression Multipliers" IEEE Trans. on Computers, Vol. 44, No. 8, pp. 962-970, 1995.
27. Chow, C.Y., and Robertson, J.E., "Logical Design of A Redundant Binary Adder", IEEE Trans. on Computer, Vol. C-27, pp. 109-115, 1978.
28. McQuillan, S.E. "Algorithms and Architectures for High Performance Arithmetic Processors", Ph.D. thesis, Faculty of Engineering, Queen's University of Belfast, September, 1992.
29. Ercegovic, M., and Lang, T., "On-the-fly conversion of redundant into conventional number representations", IEEE Trans.on Computer, C-36, pp. 895-897, 1987.
30. McQuillan, S.E., and McCanny, J.V., "A VLSI Architecture For Multiplication, Division and Square Root", IEEE Intl. Conf. Acoustics, Speech and Signal Processing, May 14-17, pp. 1205-1208, 1991.
31. Yuan, J., Svensson, C. "High-Speed CMOS Circuit Technique." *IEEE J. Solid-State Circuits*. vol. 24 pp. 62-70, 1989.
32. Shoji, M. *CMOS Digital Circuit Technology*, Prentice-Hall, N.J. 1988.
33. Yuan, J. Svensson, C. "Pushing the Limits of Standard CMOS." *IEEE Spectrum*, February, pp. 52-5
34. Hong-Yi Huang and Chung-Yu Wu, "New CMOS Differential Logic Circuits for True-Single-Phase Pipelined Systems", Proc. of the ISCAS Conf. 1994, vol. 4, pp. 15-18, 1994.
35. Jullien, G.A., Luo, W., and Wigley, N.M., "High Throughput VLSI DSP Using Replicated Finite Rings", Journal of VLSI Signal Processing (in print).
36. Czilli, J.C., Zhou, P., Jullien, G.A., and Miller, W.C., "BiCMOS Current Steering Pipeline Circuit Technique.", IEE Electronics Letters, Vol. 30, No. 12, pp. 943-945, 1994.

PostScript error (--nostringval--, --nostringval--)