Application Specific Serial Arithmetic Arrays

K. Winters, D. Mathews and T. Thompson
Department of Electrical Engineering
Montana State University
Bozeman, Montana

Abstract– **High performance systolic arrays of serial- parallel multiplier elements may be rapidly constructed for specific applications by applying hardware description language techniques to a library of full-custom CMOS building blocks. Single clock precharged circuits have been implemented for these arrays at clock rates in excess of 100Mhz using economical 2-micron (minimum feature size) CMOS processes, which may be quickly configured for a variety of applications. A number of application-specific arrays are presented, including a 2-D convolver for image processing, an integer polynomial solver, and a finite-field polynomial solver.**

1 Introduction

Parallel arrays of serial arithmetic processor modules have been proposed since the late 1950s [13] for high performance signal and information processing. Only since the development of very large scale integrated (VLSI) circuit technologies nearly 20 years later, has it been possible to fully exploit the speed and modularity of this architecture type. This paper presents an approach to the design of special-purpose systolic arrays that attempts to maximize performance and cost effectiveness and minimize design time. A family of high-speed CMOS circuits has been tailored to take advantage of the inherent locality of connection of serial processor arrays, while modern hardware description language (HDL) and logic synthesis techniques are employed to enable very rapid development of simulation models, standard-cell prototype components, and high-performance full custom CMOS implementations.

A design methodology has been adopted for the development of these arrays that combines bottom-up and top-down approaches. First, a hardware description language (HDL) model is written that reflects the structure of the array to the cell level and functional behavior at the cell level. This model is simulated and debugged, becoming the defined specification of the array. At this point, a standard-cell semi-custom implementation may be quickly generated using logic synthesis tools. Standard-cell realizations may be sufficient in performance for many applications, or optionally may serve as prototypes for full-custom components developed later.

Finally, full-custom CMOS versions of the datapath cells are designed to the HPL specification, verified with SPICE, and laid out with a physical artwork editor. The artwork is checked for design rule violations, and extracted for simulation using the same test

vectors used to verify the HDL models. Parasitic capacitances are also extracted, and SPICE timing models annotated. Once the datapath cell dimensions are known, the HDL models used to define the topology of the array can, with very little modification, be used to construct the tiled datapath artwork. Composite artwork layout, padring generation, and module place-and-route are performed automatically, using the same tools as for standard-cell prototypes.

It is crucial that there be a close coupling between the array architecture and the CMOS circuit design if clock speed is to be maximized. A family of dynamic logic circuits was developed that avoids precharge race conditions, clock skew, and hold time considerations, and minimizes series fet and static inverter stage delays. The resulting differential synchronous logic elements are clocked by a single uncomplemented signal. Conversely, array architectures are constrained to highly systolic configurations with very local connectivity between synchronous elements.

The OCT toolset from the University of California, Berkeley, is used for top-down behavioral and structural HDL modeling, logic and switch simulation, CMOS standard-cell prototype synthesis, and custom layout. The toolset is an integrated system for VLSI design, including tools and libraries for multi-level logic synthesis, standard cell placement and routing, programmable logic array and gate matrix module generation, custom cell design, and utility programs for managing design data. Most tools are integrated with the OCT data-base manager and the X-based VEM graphical user interface.

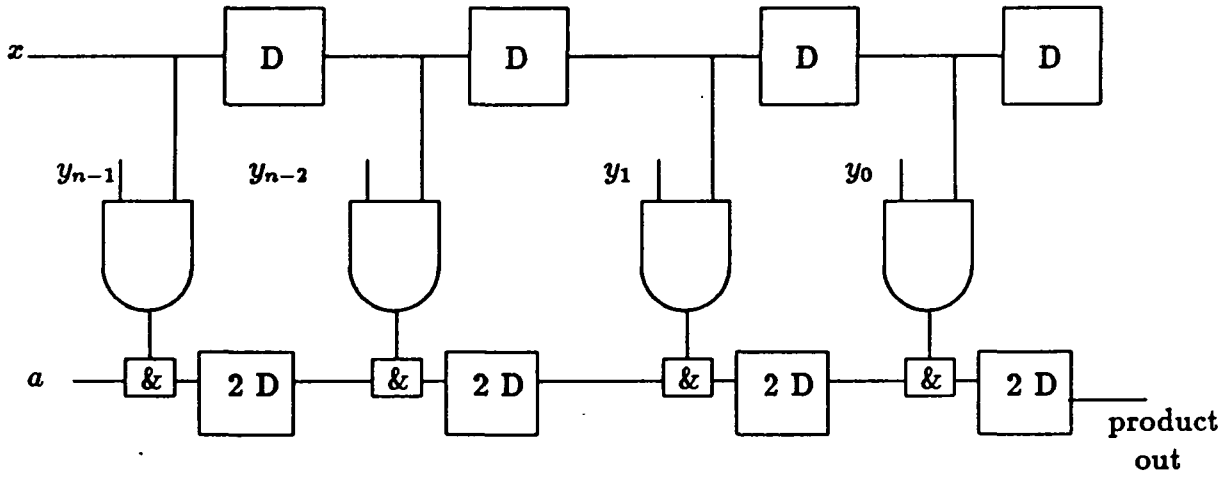

Figure 1: Fully Systolic Serial-Parallel Multiplier

2 Single-Clock Serial-Parallel Multipliers

A single-clock dynamic multiplier circuit has been developed in 2 micron CMOS technology using differential single-clock NORA techniques. This multiplier, shown in Figure 1, is a

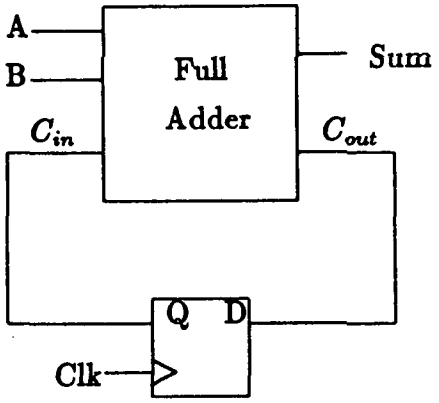

Figure 2: Bit Serial Adder

systolic adaptation of the serial-parallel multiplier introduced by Yaohan Chu in 1962 [3] and later by Daniel Hampel [7]. The boxes labeled with ampersands represent bit-serial adders, shown in Figure 2. "D"s represent delay flip-flops.

This multiplier requires 2n clock cycles to multiply two n-bit operands. Essentially, the parallel multiplicand, y, is multiplied by the serial multiplier, x, on each clock cycle and accumulated in the product pipeline. The multiplier input is pipelined through n stages, which reduces the maximum fanout of the multiplier input to 2 gate inputs. Commensurate delay elements are added to the product pipeline, totaling two delay elements per bit-slice. A primary advantage of this systolic multiplier circuit is that is has the proper ratio of operand to product storage. That is, it contains n bits of storage in the multiplier pipeline, and 2n bits of storage in the product pipeline [15].

In operation, the multiplier is shifted into the module in n cycles, followed by n cycles of zeroes. In some applications, it is desirable to utilize the full bandwidth of the multiplier pipeline during these "dead" cycles. This is accomplished by ANDing the multiplicand, y, with the output of an n-bit Johnson-Ring or Mobius [12] counter, as illustrated later.

The product pipeline can accumulate external addends with its partial products without additional adder logic. The external addend is shifted serially in the *a* input and must be pre-shifted n bits into the product pipeline before the LSB of the multiplier is entered. Thus, the lower n bits of the addend occupy the high n bits of the product pipeline at the beginning of a multiply sequence. Then, n multiplier bits are shifted into the multiplier pipeline, leaving the LSB of the accumulated product in the LSB of the product pipeline. During the next n clock cycles, the multiplier is shifted out (replaced by zeroes), the low n bits of the product are shifted out of the product pipeline, the high n bits of the product are left in the low half of the product pipeline, while the low n bits of the next addend are pre-shifted into the high product pipeline half.

This multiplier configuration operates on positive integers. For signed two's complement operands, the sign bit of the multiplier operand, MSB, must be repeated, or sign extended, and shifted into the multiplier pipeline during the second n cycles of the mul-

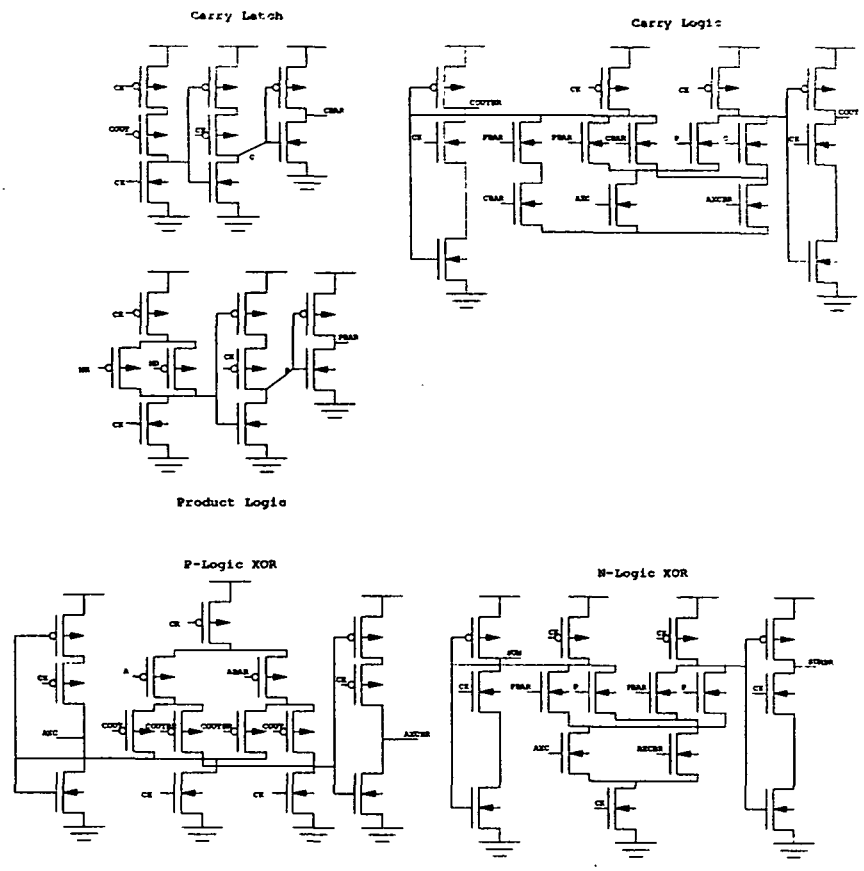

Figure 3: Single Clock Multiplier Cell Schematic

tiply. This can be implemented simply with a sign hold flip-flop preceding the multiplier pipeline that is load-disabled during the second n cycles of the multiply [7,11].

A circuit diagram for the systolic multiplier bit-slice is shown in Figure 3. This circuit was designed using a single-phase, noncomplemented clocking scheme, capable of higher clock rates than conventional dual-clock or complemented-clock techniques, since clock skew is less of a problem [8], [9], [1]. Here, synchronous systems may be constructed from alternating precharged P-fet and N-fet logic stages separated by clocked inverters. This scheme, like its predecessors NORA [4] and Domino [10] logic, is free of precharge race failures, yet eliminates the need for a complemented clock and associated skewing problems at very high clock rates.

The P-logic stages of the circuit at left precharge low during clock high and evaluate during clock low, while the N- logic stages in the center precharge high during clock low and evaluate during clock high. These two stages provide the product of the operands, and also one of the bits of storage in the product pipeline. The product of the operands is generated in the following manner. The product logic is a P-logic AND gate, which multipliers mr and md to produce p. The addend a is an input from the previous stage of the multiplier, and is XORed with c in the P-logic XOR. The Carry logic generates the carry which will be used in the succeeding clock cycle's multiplication. The accumulated product is generated in the N Logic by XORing *a-XOR-c* and p. This product is used as the addend for the next cell.

The two flip-flop circuits at the far right complete the pipelines. The mr flip-flop

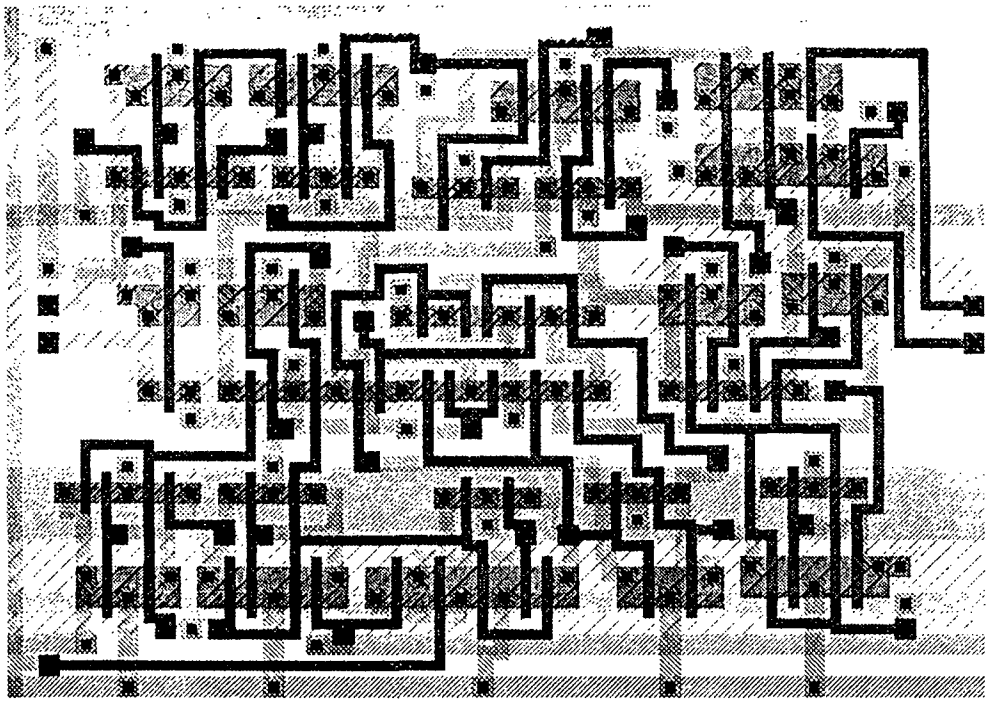

Figure 4: Single clock multiplier cell artwork

provides the storage in the multiplier pipeline, and the sumbar flipflop provides the second bit of storage in the product pipeline.

The following constraints are imposed to maximize clock rate in array architectures with very high locality of connection. A maximum of two series logic fets are used in any dynamic stage. Differential logic configurations are used wherever there are two logic fets in series and complemented outputs are required, such as the P-logic and N-logic differential XOR stages. Discrete output inverters are allowed only where one series logic fet exists in a dynamic stage, such as the *md-AND-mr* P-logic stage. N-logic stages are used to drive the cell outputs to take advantage of higher N mobility.

In state machines built from differential or dual-rail logic, there exist possible lockup states where differential outputs are stuck both ones or both zeroes. To address lockup states in the multiplier cell, the additional term *pbar-AND-cbar* is added to the logic equation for coutbar. This ensures that after a clock cycle the circuit will break out of the case where cout and coutbar both are stuck at zero. The case where cout and coutbar are stuck at one corrects itself when p becomes a one. This suggests that a power-up procedure should be run to ensure that the circuit has broken out of the stuck cases and also to initialize the pipelines.

The layout of the systolic multiplier cell is shown in Figure 4. This cell is 130 by 183 microns in size. The multiplier was simulated [1] at clock rates greater than 100 MHz using MOSIS 2-micron design rules, assuming that the output load of the cell was an identical cell.

[1] Using Tektronix, Inc. Tekspice, Tektronix MFET Level 2 device models, and slow speed device parameters for a MOSIS 2-micron (drawn gate length) SCMOS process, Revision 6 design rules, at T=100 degrees C and Vdd = 4.5V.

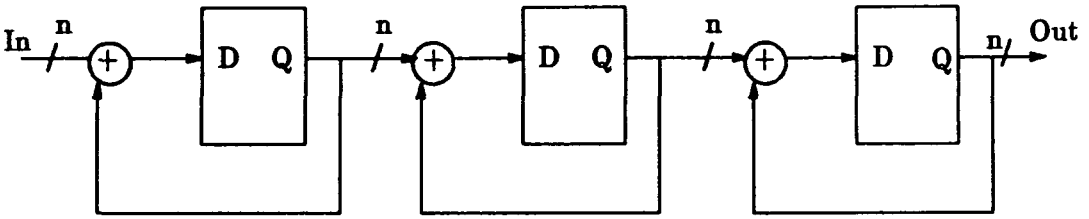

Figure 5: Babbage polynomial solver

3 Babbage Polynomial Solver

The Difference Engine, proposed by Charles Babbage in 1822, can be easily adapted to serial arithmetic, providing a simple example of the proposed design methodology and circuit implementation. Once initialized for a known value of polynomial p(X), the Babbage polynomial solver evaluates p(X+1), p(X+2), and so on, requiring only one clock cycle per evaluation. This method of finite differences is based on the binomial theorem, which guarantees that a polynomial p(X), when evaluated at p(X+1), will yield a new polynomial of X: the sum of p(X) and a remainder polynomial, r(X). Thus, polynomials evaluated at successive values may be decomposed recursively.

The general form of the Babbage Difference Engine is shown in Figure 5. It consists of only adders and pipeline registers.

Since the Babbage machine is systolic by column, the performance bottleneck is in the adder carry propagation logic. This can be reduced by pipelining each adder bit by row, replacing each parallel adder stage with a bit-serial adder. The resulting datapath architecture, shown in Figure 6, consists of a single cell.

The first step is to define the behavior and structure of the Babbage array cell, called babit. This is done in the Berkeley OCT environment using an HDL called BDS to define behavior, Table 1, and BDNET to define structure as in Table 2.

Next, a BDNET description of the datapath topology is constructed,

The general form of the Babbage Difference Engine is shown in Figure 5. It consists of only adders and pipeline registers.

Since the Babbage machine is systolic by column, the performance bottleneck is in the adder carry propagation logic. This can be reduced by pipelining each adder bit by row, replacing each parallel adder stage with a bit-serial adder. The resulting datapath architecture, shown in Figure 6, consists of a single cell.

The first step is to define the behavior and structure of the Babbage array cell, called babit. This is done in the Berkeley OCT environment using an HDL called BDS to define behavior, Table 1, and BDNET to define structure as in Table 2.

Next, a BDNET description of the datapath topology is constructed, configured here for a degree-3 by 4-bit engine in Table 3. This process required less than one afternoon, excluding simulation test vector generation.

After simulation and optional mask release of the standard-cell prototype IC, custom versions of the bit cell are designed employing the single-clock dynamic circuits described

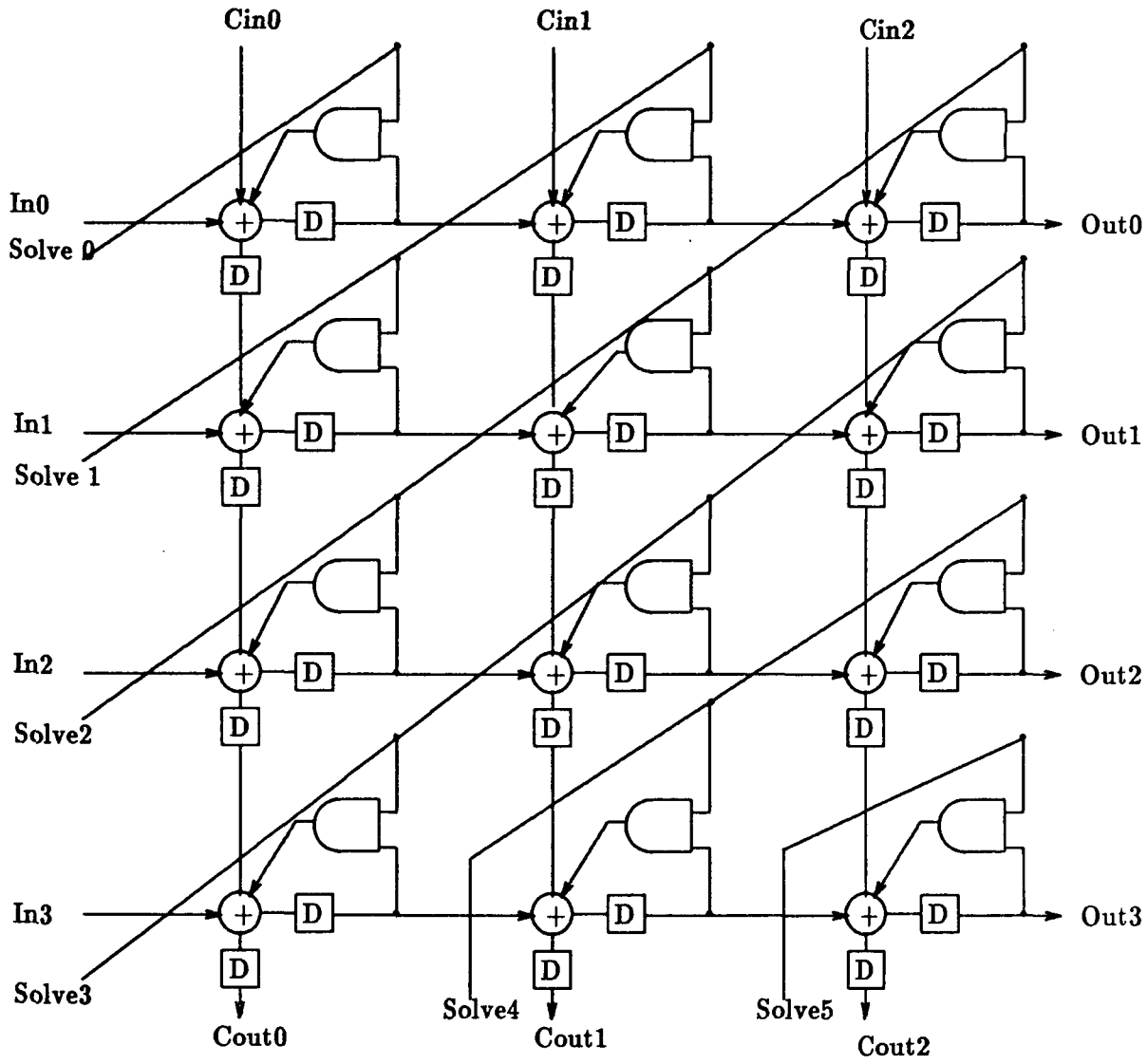

Figure 6: Systolic Babbage polynomial solver

8.1.8

```
! BABIT difference engine bit module BDS description
! FILE: babit.bds
! Kel Winters
! Rev: 5-22-90

!Vars:          in -addend input, solve - add enable
!               cin - carry in, f - feedback input, p - solve AND f
!               sum - sum output, c - carry out

model babit sum<0>,c<0>=in<0>,solve<0>,f<0>,cin<0>;
routine cycle;
state p<0>;
p = solve AND f;
sum = in XOR p XOR cin;
c = (in AND p) OR (inAND cin)OR (pANDcin);
ENDROUTINE;
ENDMODEL;
```

Table 1: Babit Behavioral Description

earlier. Starting with a basic set of CMOS "building blocks" for constructing serial adder cells with delay elements, full custom babit realization required about two engineer-weeks. The same BDNET description used to construct the standard cell datapath is then configured to build the tiled custom version, Figure 7.

4 Two-Dimensional Integer Convolver

The two-dimensional convolver discussed in this section is essentially a digital filter for two-dimensional integer-valued image data. The purpose of the convolver is to filter out specific information, or data patterns, which are defined by the convolution mask at high speeds. The use of the convolver assumes that the processed data will be subtracted from the image so that the desired data is the only information which is retained. This module is intended for configuration for applications such as LaPlacian pyramid image compaction [2], [6], feature extraction, and enhancement/restoration.

The organization of the convolver is now row parallel, so that a column of image data enters the convolver at the same time. The structure is basically an array of serial-parallel multiplier/accumulators with a few modifications to the circuit discussed previously.

A block diagram of the convolver module is shown in Figure 8. The convolver moves the raw data through its registers, accumulating the convolved result as it travels. The input is multiplied by the first values in the positive-valued convolution mask, and is piped to the next multiplier/accumulator register (column) as the addend for its operation. The partially accumulated result, for a given location in the data, is piped through the

```
! BABIT difference engine bit cell BDNET description
! File: babit.bdnet
! Kel Winters
! Rev: 5-23-90
! ! Variables:    in - addend input, solve - add enable
!                cin - carry in, cout - carry out delayed
!                out - sum output delayed
!
MODEL            babit:unplaced;
TECHNOLOGY       scmos;
VIEWTYPE         SYMBOLIC;
EDITSTYLE        SYMBOLIC;

INPUT            clk,in<0>,solve<0>,cin<0>;
OUTPUT           cout<0>,out<0>;
SUPPLY           Vdd;
GROUND           GND;

INSTANCE         babit:logic PROMOTE;
                 f<0>:out<0>;
INSTANCE         "~/octtools/lib/technology/scmos/msu/stdcell2_0/dfnf311":physical
                 DATA1:sum<0>;
                 CLK3:clk;
                 Q:out<0>;
                 Qb:UNCONNECTED;
                 "Vdd!":Vdd;
                 "GND!":GND;

INSTANCE         "~/octtools/lib/technology/scmos/msu/stdcell2_0/dfnf311":physical
                 DATA1:c<0>;
                 CLK3:clk;
                 Q:cout<0>;
                 Qb:UNCONNECTED;
                 "Vdd!":Vdd;
                 "GND!":GND;

ENDMODEL;
```

Table 2: Babit Structural Description

```
! BABAR3x4, difference engine array BDNET description
! File: babar.bdnet
! Kel Winters
! Rev: 5-24-90
!
! Variables: same as Babit

!                    NOTE:  this file is written to be extensible in array height,
!                           4, and width, 3. Copy this file into the actual bdnet
!                           bdnet file and substitute the desired values for 4 and
!                           3 before running bdnet

MODEL          babar3x4:unplaced;
TECHNOLOGY     scmos;
VIEWTYPE       SYMBOLIC;
EDITSTYLE      SYMBOLIC;

INPUT          clk,in<(4-1):0>,cin<(3-1):0>,solve<(4+3-2):0>;
OUTPUT         out<(4-1):0>:in<((3+1)*4-1):(4*3)>;
OUTPUT         OUTPUT cout<(3-1):0>:cin<(3*(4+1)-1):(3*4)>;
SUPPLY         Vdd;
GROUND         GND;

ARRAY          %x FROM 0 to (3-1) OF
ARRAY          %y FROM 0 to (4-1) OF

INSTANCE       babit:unplaced
                        clk : clk;
                        in<0>:in<%x*4+%y>;
                        out<0>:in<(%x+1)*4+%y>;
                        cin<0>:cin<%y*3+%x>;
                        cout<0>:cin<(%y+1)*3+%x>;
                        solve<0>:solve<%x+%y>;
                        Vdd : Vdd;
                        GND : GND;

ENDMODEL;
```

Table 3: BABAR 3 × 4 Structural Description

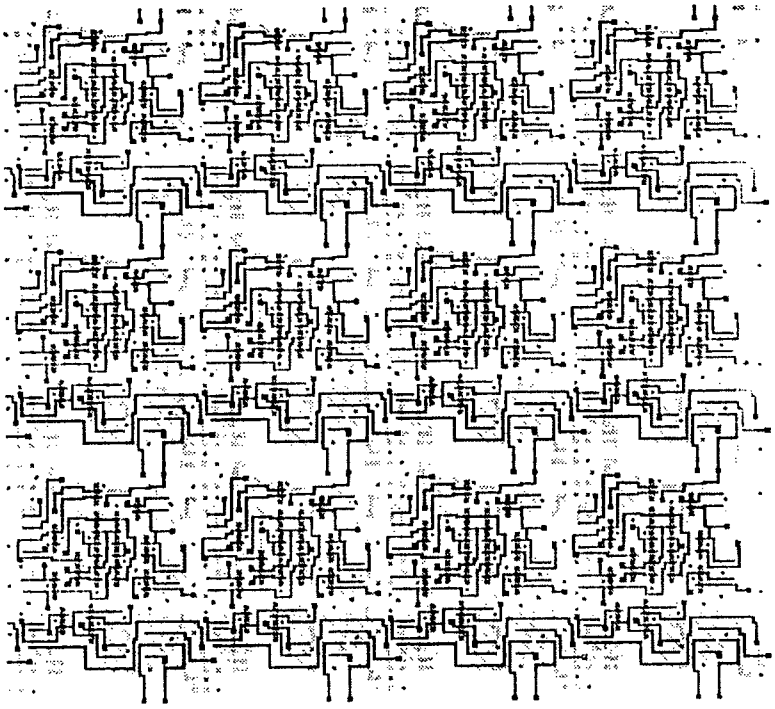

Figure 7: Custom single clock polynomial solver module

array in parallel with data at vertical neighbor locations. It is piped to the multiplication operation with which it must be added next (this includes diagonal/vertical routing.) The last register operation produces a convolved value for each point in one column of the two-dimensional input data.

Three modifications have been made on the systolic multiplier circuit to build the convolver circuit. A second multiplier (mr) pipeline has been added, along with a Johnson ring counter and a control cell which routes data through the multiplier/accumulator registers.

The multiplier storage line, or pipeline, consists of two flip-flops per bit slice (multiplier/accumulator cell.) This serves to provide a pipeline path for the multiplier (image) data that is matched in length to the product accumulator pipe so that the product sums from neighboring rows can be synchronized then combined.

This array configuration calculates a convolution matrix for every odd column of the input matrix. Therefore, each systolic multiplier stage receives an n-bit serial operand, then ignores the next as it completes the second n clock cycles of the multiply sequence. This second operand must not be lost, however, as it will be used be the next multiplier stage downstream in the succeeding n cycles. Therefore, an n-bit Johnson Ring counter, described previously, shifted through the multiplier pipeline.

The Johnson ring counter is effectively an n-bit shift register with its output fed back, inverted to its input. A serial reset is provided to initialize the ring. Again, the ring counter serves to allow a second set of n-bit words to share the mr pipeline, disabling the

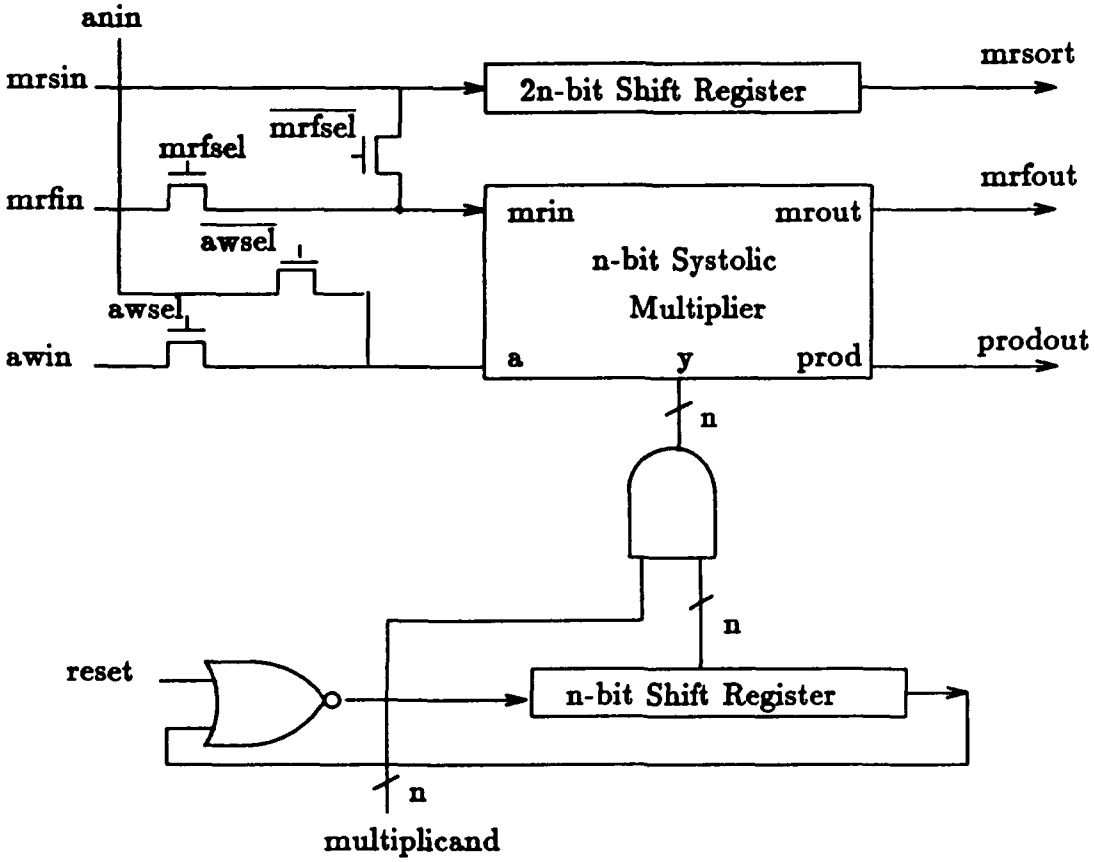

Figure 8: Two dimensional systolic convolver register

unused odd words from affecting the multiplier logic. For the twos-complement capability, the odd words in the mr pipeline should be filled with sign-extension bits. The Johnson ring counter would then be replaced with a second mr pipeline. Naturally, overflows from the sign-extension field must be cleared between multiply-accumulate operations.

The complete convolver bit cell contains 119 fets and is 150 by 160 microns in size, using 2-micron MOSIS CMOS design rules. Artwork is shown in Figure 9.

The control cell includes four pass gates, the reset for the Johnson ring counter, and the power, ground and clock routing. Two of the pass gates switch the addend input from the west or the northwest neighbors upstream. The other two pass gates determine whether the incoming multiplier is processed through the multiplier pipeline or detoured through the slow (twice delayed) multiplier pipeline. This allows adjacent row products to be summed, with the multiplier pipelines matched to the product pipelines. Power, ground and control signals are routed vertically, by column.

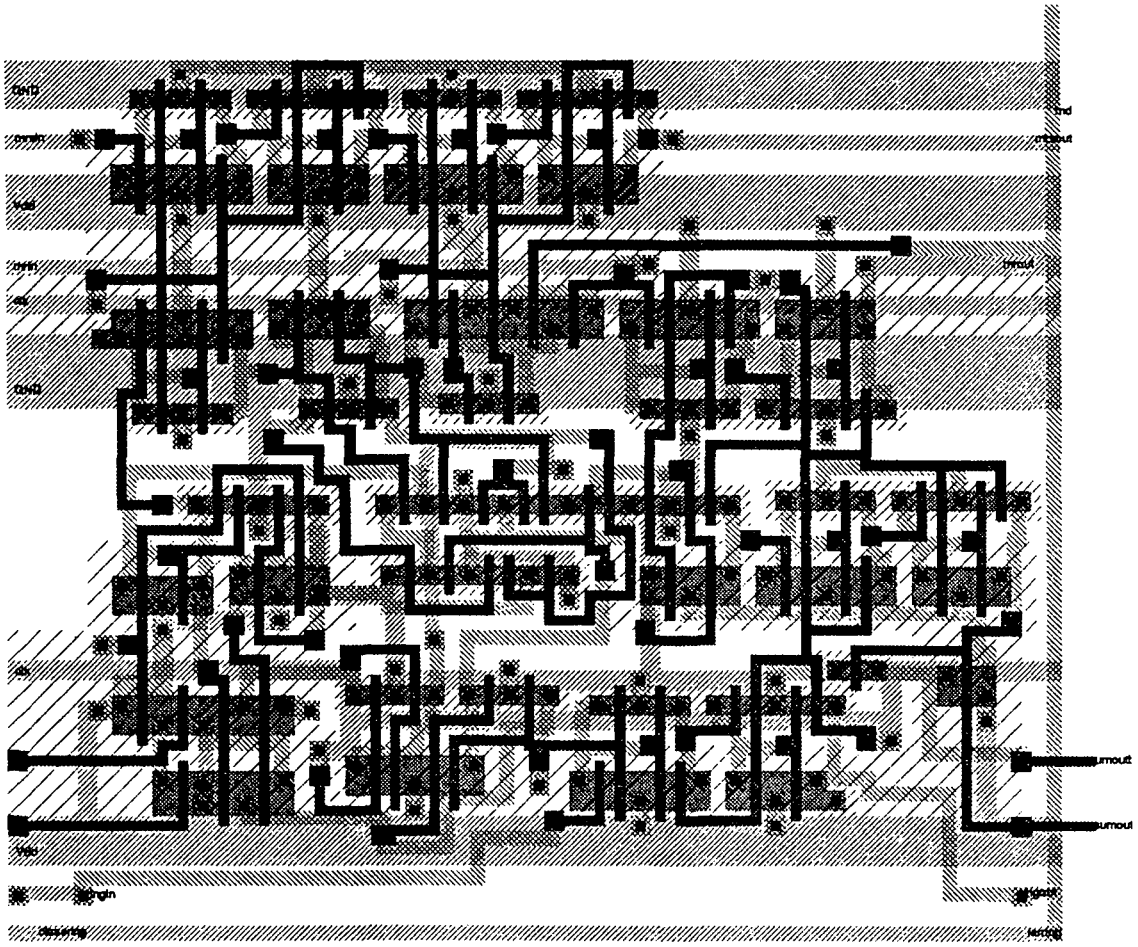

Figure 9: Two dimensional convolver bit cell

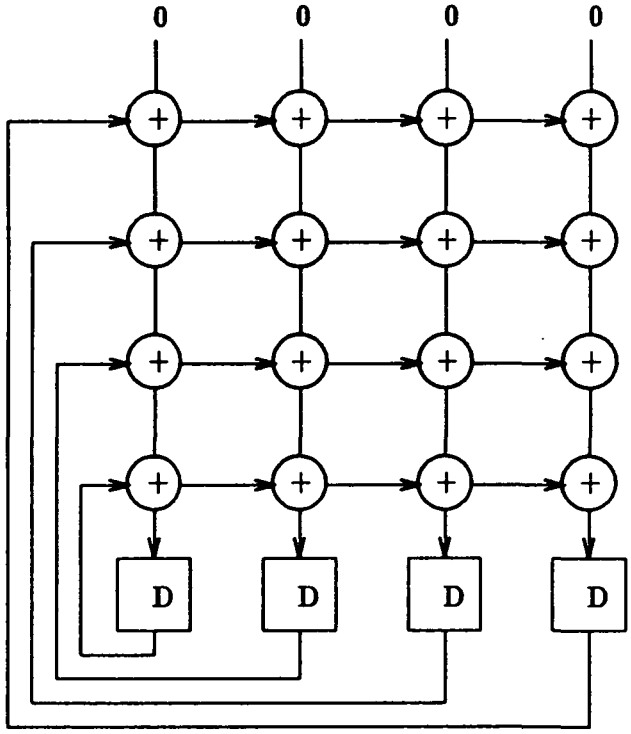

Figure 10: Finite field constant multiplier

5 Two-Dimensional Systolic State Machines

Two-dimensional systolic state machines are defined here as automata where the present-state feedback is organized as a set of n row pipelines and next-state equations organized as n column pipelines, where n is the width of the state variable vector. Rows and columns must be ordered so that all loops from a state-variable to itself contain n delays. The intent here is to maximize clock speed independently of the state vector width or functional complexity.

For example, a finite-field constant multiplier-accumulator consists of a synchronous register to store the present-state and an array of exclusive-or (XOR) gates, representing modulo-2 adders. Next-state equations are sum-of-product functions of the present-state vector. In its simplest organization [14], the present-state register outputs are fed back as row lines, and the next-state equations are organized as columns of XOR gates or crossovers as needed, Figure 10. Here, XOR gate sites are represented by "+," and flip-flops by "D." Note that the XOR gate outputs to the bottom; the row input is passed through the row. The critical timing path in this organization is the fanout of the state register, up to n XOR gates across n columns, and up to n XOR gate delays routed through n rows.

A 2-D systolic finite-field multiplier is shown in Figure 11. Here, the present-state feedback is through n n-stage shift registers, one per row. Column XOR gates are also pipelined, one stage per row. Again, rows and columns are organized so that all loops from feedback (present-state) signals to themselves contain n delays. This configuration

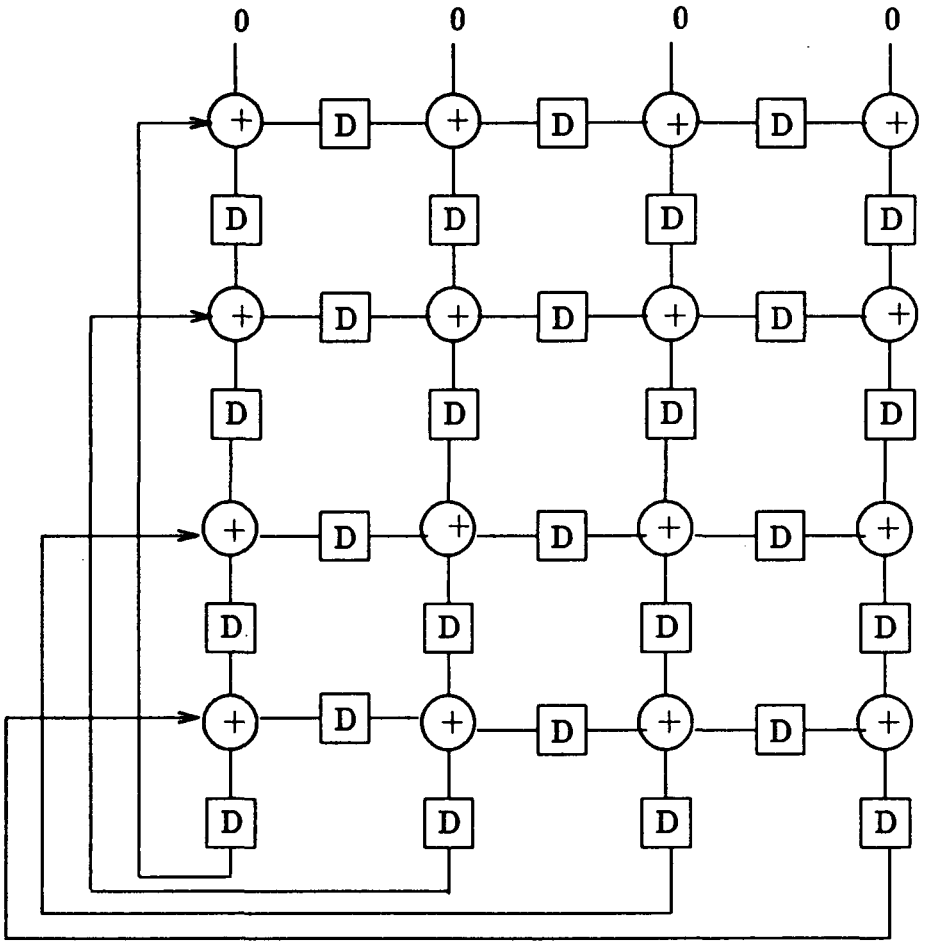

Figure 11: Systolic finite field multiplier

Configuration	Flip-Flops	XOR Sites	n-mult cycles	Fanout Max.	Series Delays
n Conventional	n^2	n^3	1	n	n
Systolic	$2n^2$	n^2	n	1	1

Table 4: Finite-Field Constant Multiplier Time-Area Considerations

performs the function of n conventional multipliers operating in parallel. The time-area considerations for n conventional multipliers and the proposed systolic array are compared in Table 4. The fourth column gives the number of clock cycles required to perform n multiplications of n bits.

If the ratio of flip-flop area to XOR gate area is assumed to be 1:3, an area figure of merit for the n conventional constant multipliers is given by $n + n^2/3$ compared to $7n/3$ for the systolic case. This implies that the area is comparable at $n = 4$. At $n = 8$, common to Reed-Solomon error correction codes, the n conventional constant multipliers would be about 50% larger than the systolic version. Thus, the systolic version would have to operate at about five times the clock rate of the n conventional multipliers to reach time-area parity (not difficult given the difference in fanout and series delays).

This thumbnail sketch is an oversimplification, given that each technique tends to favor different circuit realizations, but serves to illustrate the potential of 2-D systolic state machines. Practical applications in finite field constant multipliers would likely involve an optimal compromise: a systolic array of non-systolic sub-arrays.

The advantages of this technique are more apparent in applications where the value of n grows large. One such application under investigation is the simulation of Hopfield-type neural networks [5], where the next-state of n neurons is a function of the present-state of n neurons described by an n-by-n matrix. Here, neurons would be realized by systolic columns in the state machine array, dendrites by row pipelines, and synaptic connections by systolic serial-parallel multipliers (described previously) at the intersection of each row and column. This array would emulate n neural networks operating in parallel.

6 Conclusions and Future Directions

A technique for designing high-performance systolic serial VLSI arithmetic arrays has been presented, combining a new variation of differential single-clock dynamic CMOS circuits, a systolic adaptation of Chu's serial-parallel multiplier, and a structured design methodology that combines recent advances in hardware description language synthesis and modeling with full-custom IC design.

For the class of applications that can be decomposed into a regular array of arithmetic modules, it is proposed that HDL models may be developed in hours, standard-cell prototype ICs using logic synthesis techniques implemented to mask release in days, and high-speed full-custom implementations to mask release in weeks.

A standard-cell implementation of the Babbage style polynomial solver has been re-

leased for fabrication in a 2 micron (drawn gate length) CMOS process through the National Science Foundation MOSIS silicon brokerage service. VLSI Technology, Inc. will provide the fabrication. A full-custom prototype of the same polynomial solver has been designed and will be released shortly in the same process.

Standard-cell and full-custom implementations of the two-dimensional convolver are in development for release in a 2 micron process from the Massachusetts Micro-electronics Center and a 1.2 micron process from the Hewlett-Packard Co. through the MOSIS service. These convolver arrays are configured for hierarchical or Laplacian pyramid image compression, a technique proposed for the Mars Rover Sample Return mission [6]. Other configurations for space telescope correction/enhancement and digital signal processing are also under investigation.

Preliminary work on a finite-field polynomial solver based on 2-D systolic constant multipliers has recently begun. It is anticipated that rapid development of custom high-speed arithmetic arrays will provide practical solutions for a wide array of image, signal, and data processing applications.

Acknowledgement

This work was supported by grants from the NASA Space Engineering Research Center at the University of Idaho, Moscow, and the Montana State University Engineering Experiment Station as well as equipment donations from the Helwett-Packard Co. and Tektronix, Inc. The authors would especially like to thank the following people for their valuable assistance to this work: Gary Maki and Jon Gibson of the NASA SERC at the University of Idaho, Jaye Mathisen and Sanjay Mitra of Montana State University, Rick Spickelmier of UC Berkeley, Paul Cohen of the Massachusetts Microelectronics Center, and Fred Huck of the NASA Langley Research Center.

References

[1] M. Afghahi and C. Svensson, "A Unified Single-Phase Clocking Scheme for VLSI Systems," *IEEE J. Solid-State Circuits*, Vol. 25, no. 1, Feb. 1990, pp. 225-233.

[2] P. Burt and E. Adelson, "The Laplacian Pyramid as a Compact Image Code," *IEEE Trans. on Communications*, Vol. COM-31, No. 4, April 1983.

[3] Y. Chu, *Digital Computer Design Fundamentals*, McGraw-Hill, 1962.

[4] N. Goncalves and H. De Man, "NORA: A Racefree Dynamic CMOS Technique for

Pipelined Logic Structures," *IEEE J. Solid-State Circuits*, Vol. SC-18, no. 3, June 1983, pp. 261-266.

[5] J. Hopfield and D. Tand, "Computing with Neural Circuits: A Model," *Science*, Vol. 233, pp. 625-633, August 1986.

[6] F. Huck, "Rover Imaging Systems for the Mars Rover/Sample Return Mission," Proposal to the NASA Planetary Instrument Definition and Development Program, NASA Langley Research Center, Hampton, VA, March, 1989.

[7] D. Hampel, K McGuire, and K. Prost, "CMOS/SOS Serial-Parallel Multiplier," *IEEE J. on Solid-State Circuits*, Vol. SC-10, No. 5, October 1975.

[8] Y. Ji-Ren, I. Kaarlson, and C. Svensson, "A True Single- Phase-Clock Dynamic CMOS Circuit Technique," *IEEE J. Solid-State Circuits*, Vol. SC-22, no. 5, October 1987, pp. 899-901.

[9] Y. Ji-Ren and C. Svensson, "High-Speed CMOS Circuit Technique," *IEEE J. Solid-State Circuits*, Vol. 24, no. 1, Feb. 1989, pp. 62-70.

[10] R. Krambeck, C. Lee, and H. Law, "High-Speed Compact Circuits with CMOS," *IEEE J. Solid-State Circuits*, Vol. SC-17, June 1982, pp. 614-619.

[11] R. Lyon, "Two's Complement Pipeline Multipliers," *IEEE Trans. on Communications*, COM-24, April, 1976, pp. 418-425. 97-107.

[12] H. Taub and D. Schilling, *Digital Integrated Electronics*, McGraw-Hill Inc., New York, 1977, pp. 349-355.

[13] S. Unger, "A Computer Oriented Toward Spatial Problems," *Proceedings of the IRE*, Vol. 46, no. 10, pp. 1744-1750, October, 1958.

[14] K. Winters, "A VLSI Error Location Processor for a High Performance Reed-Solomon Decoder," Master's Thesis, University of Idaho, Moscow, ID, June, 1984.

[15] K. Winters, "Serial Multiplier Arrays for Parallel Computation," *NASA SERC Symposium on VLSI Design*, Moscow, ID, Jan. 1990.