nanoseconds. We selected a representative example of a FIR filter with binary weights, and verified using simulation results that the neural network yields weights that enable the filter to perform very close to the theoretical peak performance that one can obtain from the given filter. We showed that the conventional LMS approach is unable to match the performance of the neural network since it cannot select the correct minimum from all the possible minima of the error function.

We showed that the FIR filter with binary weights coupled with the proposed iterative neural network procedure for obtaining the optimal weights, forms an extremely simple filter to implement in hardware with good performance characteristics.

REFERENCES

[1] B. Widrow and S. Sterns, *Adaptive Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1986.
[2] M. P. Kennedy and L. O. Chua, "Neural networks for nonlinear programming," *IEEE Trans. Circuits Syst.*, vol. 35, pp. 554–562, May 1988.
[3] G. M. Ewing, *Calculus of Variations with Applications*. New York: Dover, 1985.

A Systolic Architecture for Modulo Multiplication

Khaled M. Elleithy and Magdy A. Bayoumi

***Abstract*—With the current advances in VLSI technology, traditional algorithms for Residue Number System (RNS) based architectures should be reevaluated to explore the new technology dimensions. In this brief, we introduce A $\theta(\log n)$ algorithm for large moduli multiplication for RNS based architectures. A systolic array has been designed to perform the modulo multiplication Algorithm. The proposed modulo multiplier is much faster than previously proposed multipliers and more area efficient. The implementation of this multiplier is modular and is based on using simple cells which leads to efficient VLSI realization. A VLSI implementation using 3 micron CMOS technology shows that a pipelined n-bit modulo multiplication scheme can operate with a throughput of 30 M operation per second.**

I. INTRODUCTION

Traditional weighted number system suffers from the carry propagation from low to high significance digits. As a consequence to this phenomenon, there is a slowdown in arithmetic operations like addition and multiplication. The carry can be accelerated using special techniques at the expense of additional hardware. Residue Number System (RNS) is a carry-free system with the capability to support high-speed concurrent arithmetic [1]. Applications such as fast Fourier transform, digital filtering, and image processing utilize the high speed RNS arithmetic operations; addition and multiplication, they do not require the difficult RNS operations such as division and magnitude comparison. The technological advantages offered by VLSI have added a new dimension in the implementation of RNS-based architectures [2]. Several high speed VLSI special purpose digital signal processors have been successfully implemented [3]–[5].

The scope of this brief is modulo multiplication. In general, look-up tables and PLA's [6], [7] have been the main logical modules used when the data granularity is the word. It has been found that such structure is only efficient for small size moduli. For medium and large size moduli, bit-level structures are more efficient, where the data granularity is the bit [17]. Most of the reported works for large moduli multipliers are based on using special set of moduli. In [6], [8] the moduli should be prime numbers. In [9], [10] the moduli should be from the set $(2^n-1,\ 2^n,\ 2^n+1)$. Due to the constraints imposed on the chosen moduli, such approaches have limited applications.

In this brief, we present a modulo multiplier for medium size and large moduli. The multiplier is based on using a $\theta(1)$ modulo adder [11]. It is configured as a two dimensional array of very simple cells (modulo adder). The modulo multiplication is performed in $\theta(\log n)$ steps with no constraints imposed on the chosen moduli.

Manuscript received December 31, 1991; revised June 15, 1994. The work of K. M. Elleithy was supported by King Fahd University of Petroleum and Minerals. This paper was recommended by Associate Editor K. Yao.

K. M. Elleithy is with the Computer Engineering Department, King Fahd University of Petroleum and Minerals, Dhahran 31261, Saudi Arabia.
M. A. Bayoumi is with The Center for Advanced Computer Studies, University of SW Louisiana, Lafayette, LA 70504 USA.

IEEE Log Number 9413888.

II. RESIDUE NUMBER SYSTEM

In RNS, an integer, X, can be represented by N-tuple of residue digits,

$$X = (r_1,\ r_2,\ \cdots,\ r_N)$$

where $r_i = |X|_{m_i}$, with respect to a set of N moduli $\{m_1,\ m_2,\ \cdots,\ m_N\}$. In order to have a unique residue representation, the moduli must be pairwise relatively prime, that is,

$$GCD(m_i,\ m_j) = 1, \quad \text{for } i \neq j$$

then it is shown that there is a unique representation for each number in the range of $0 \leq X < \prod_{i=1}^{N} m_i = M$, where N is the number of moduli.

The arithmetic operation on two integers A and B is equivalent to the arithmetic operation on their residue representation, that is,

$$|A \cdot B|_M = (||A|_{m_1} \cdot |B|_{m_1}|_{m_1},\ ||A|_{m_2} \cdot |B|_{m_2}|_{m_2},\ \cdots,\ ||A|_{m_N} \cdot |B|_{m_N}|_{m_N})$$

where "$\cdot$" can be addition, subtraction, or multiplication. It is desired to convert binary arithmetic on large integers to residue arithmetic on smaller residue digits in which the operations can be parallelly executed, and there is no carry chain between residue digits.

A. The Modulo Multiplication

Generally, multiplication modulo m has $2^n - m (n = \lceil \log m \rceil)$ incorrect residue states. These states are in the range $[m,\ 2^n - 1]$ which may be called overflow states. The corrected residue numbers can be obtained by two methods; employing a binary adder or a correction table. In the first method, a constant $(2^n - m)$ is added to correct the overflow residue states (generalized end-round carry) as shown in Fig. 1. The multiplication is performed as follows:

$$y = |x_1 * x_2|_m = \begin{cases} x_1 * x_2 & \text{if } x_1 * x_2 < m \\ x_1 * x_2 - m & \text{if } x_1 * x_2 \geq m \end{cases}.$$

A n-bit multiplier and an inner-product cell are used; the multiplier computes $x_1 * x_2$, while the inner-product cell computes $x_1 * x_2 - m$.

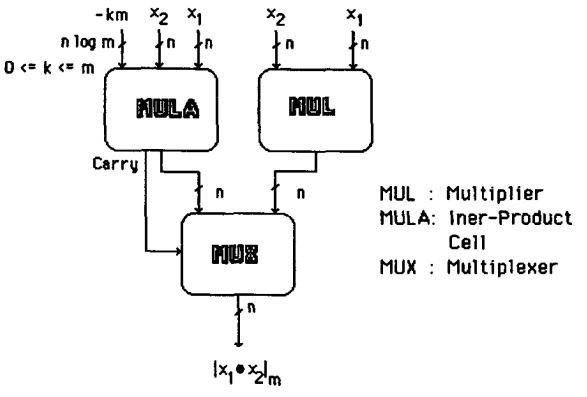

Fig. 1. Modulo multiplication using a multiplier and an inner-product cell.

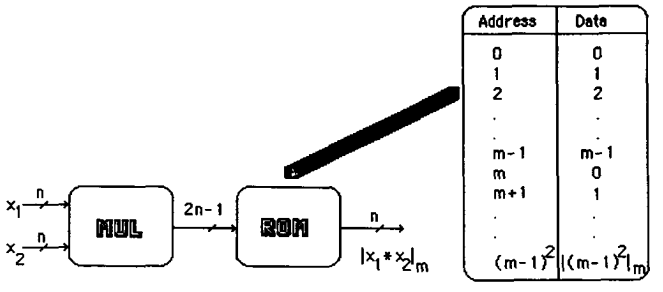

Fig. 2. Modulo multiplication using look-up table.

The carry bit generated from the second unit indicates whether or not $x_1 * x_2$ is greater than m. A multiplexer, controlled by the carry, selects the correct output. In the second method, a look-up table is used to correct the incorrect residue states $(2^n - m)$, Fig. 2. The first algorithm of modulo multiplication is slow, and the second algorithm is not suitable for medium and large moduli.

III. Modulo Adder

The modulo adder is the basic kernel used in performing the modulo multiplication. The modulo adder is based on representing a number as a *Carry* and a *Sum* to obtain a scheme that has a constant speed which does not depend on the number of bits [11]. The modulo adder is used to add two numbers A and B in modulo m. Fig. 3 shows that A is represented as a pair of numbers (A_S, A_C), B is also represented as (B_S, B_C), and the output C is represented as (C_S, C_C). Each number is represented as a group of *Sum* bits and *Carry* bits. There is no unique representation for A_S and A_C. The condition that need to be satisfied is:

$$|A_S + A_C|_m = |A|_m.$$

One possible representation is:

$$A_S = |A|_m$$
$$A_C = 0$$

The choice of a representation has no implication on the complexity of the design. With such representation, four numbers (A_S, A_C, B_S, B_C) need to be added, two steps of CSA are required. After the addition process we need to detect if $-M$ or $2*(-M)$ is required to adjust the result. The adjusting process takes at most three steps. Since the adder has a fixed number of steps; five; no matter how long is A and B, it can be used in a multioperand

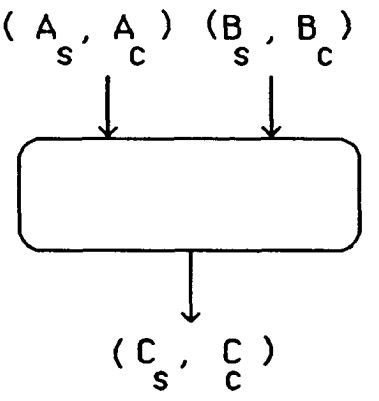

Fig. 3. A modulo sum adder.

pipelined addition scheme [13] and in the implementation of a pipelined architecture of the CRT [14].

A. The Modulo Addition Algorithm

The proposed algorithm for modulo m addition of two numbers can be described as follows:

Algorithm modulo_add (A, B, Result):

Input: Two variables A and B in modulo m, A is represented as A_S and A_C. B is represented as B_S and B_C. All variables are n bit numbers $(2^{n-1} \leq m \leq 2^n)$.

Output: Variable $Result$ represented as $Result_C$ and $Result_S$. The relation between A, B, and $Result$ is $Result = |A + B|_m$.

Procedure:

```
begin
  Do in parallel
    begin
      Call Sum(temp1, AS, AC, BS)
      Call Carry(temp2, AS, AC, BS)
    end
  Do in parallel
    begin
      Call Sum(temp3, temp1, temp2, BC)
      Call Carry(temp4, temp1, temp2, BC)
    end
  Case (temp2[n + 1] + temp4[n + 1]) of
    0: Do in parallel
       begin
         ResultS := temp3
         ResultC := temp4
       end
       exit
    1: Do in parallel
       begin
         Call Sum[temp5, temp3, temp4,
         (2^n - m)]
         Call Carry[temp6, temp3, temp4,
         (2^n - m)]
       end
    2: Do in parallel
       begin
         Call Sum[temp5, temp3, temp4,
         2 * (2^n - m)]
         Call Carry[temp6, temp3, temp4,
         2 * (2^n - m)]
       end
  end case
```

Case $[temp_6(n+1)]$ of
0: Do in parallel
begin
$Result_S := temp_5$
$Result_C := temp_6$
end
exit
1: Do in parallel
begin
Call Sum[$temp_7$, $temp_5$, $temp_6$, $(2^n - m)$]
$(2^n - m)$]
Call Carry[$temp_8$, $temp_5$, $temp_6$, $(2^n - m)$]
end
end case
Case $(temp_8[n+1])$ of
0: Do in parallel
begin
$Result_S := temp_7$
$Result_C := temp_8$
end
1: Do in parallel
begin
Call Sum[$temp_9$, $temp_7$, $temp_8$, $(2^n - m)$]
Call Carry[$temp_1 0$, $temp_7$, $temp_8$, $(2^n - m)$]
end
Do in parallel
begin
$Result_S := temp_9$
$Result_C := temp_{10}$
end
end case
end.
Sum (A, B, C, D)
begin
Do in parallel $(1 \le i \le n)$
$A[i] := (B[i] \wedge C[i])$
$\vee (B[i] \wedge D[i]) \vee (C[i] \wedge D[i])$
end
Carry (A, B, C, D)
begin
$A[1] := 0$
Do in parallel $(1 \le i \le n)$
$A[i+1] := B[i] \ominus C[i] \ominus D[i]$
end.

An implementation of the algorithm is shown in Fig. 4. The proof that the modulo adder scheme for adding two n-bit numbers in modulo m has an asymptotic time complexity $\theta(1)$ is shown in [11] with an example for the addition operation.

IV. The Modulo Multiplier

The modulo adder presented in the previous section is the main kernel of the modulo multiplier. The multiplier consists of two stages. In the first stage (Fig. 5) an array of AND gates is used to obtain the partial products. The second stage of the adder is a binary tree of modulo adders used to perform the addition of the n partial products (Fig. 6). The correctness of the operation is established by the following theorem.

Theorem 1: Adding n numbers $(y_1, y_2, \cdots, y_n)$ in modulo m can be performed in $\theta(\log n)$ time complexity using modulo adders.

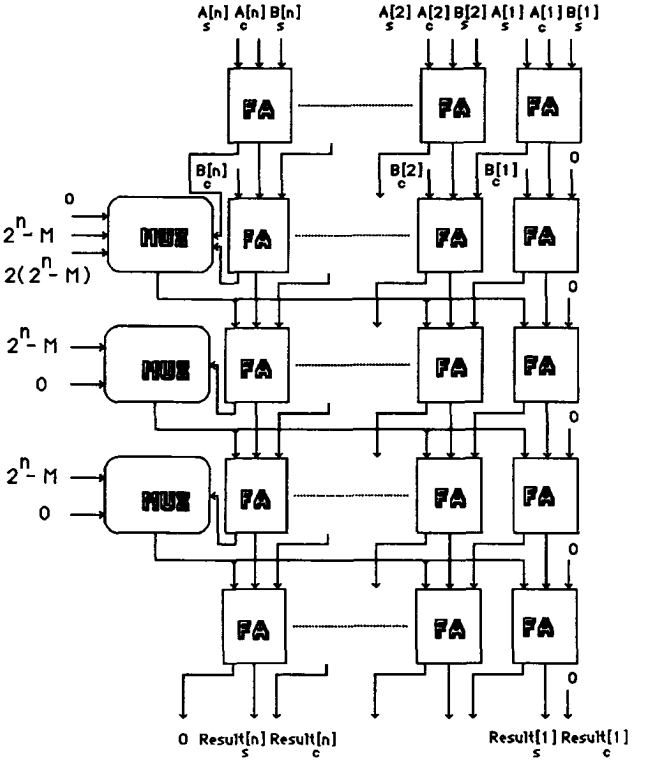

Fig. 4. Different stages of the modulo adder.

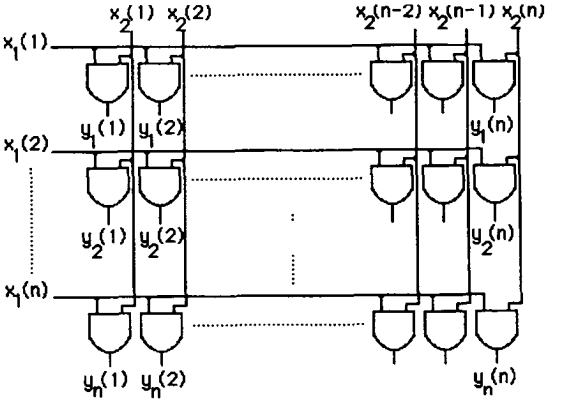

Fig. 5. Implementation of partial product generator.

Proof: The addition of n numbers modulo m can be performed as follows:

1) Adding (y_1, y_2) modulo $M, \cdots, (y_i, y_{i+1}), \cdots,$ and (y_{n-1}, y_n) gives $y_{12}, \cdots, y_{(n-1)n}$.
2) Step (1) is repeated on $(y_{12}, y_{34}), \cdots, [y_{(n-3)(n-2)}, y_{(n-1)n}]$.
3) Step (2) is repeated for $\lceil \log N \rceil - 2$ times to obtain one final output represented as a sum and carry.

The previous method needs $\log n$ step to be performed. We need to prove that the previous procedure returns the required result. To add two numbers a and b in modulo M we have the following cases:

a) $a < M$ and $b < M$ then $a = |a|_M$ and $b = |b|_M$, then:

$$|a + b|_M = |a_M + b_M|_M. \tag{1}$$

b) $a > M$ and $b < M$ then $b = |b|_M$ and $a = M + x$, then:

$$|a + b|_M = |M + x + b|_M = |x + b|_M. \tag{2}$$

Since $x < M$ and $b < M$, then from (1) and (2):

$$|a + b|_M = |a_M + b_M|_M. \tag{3}$$

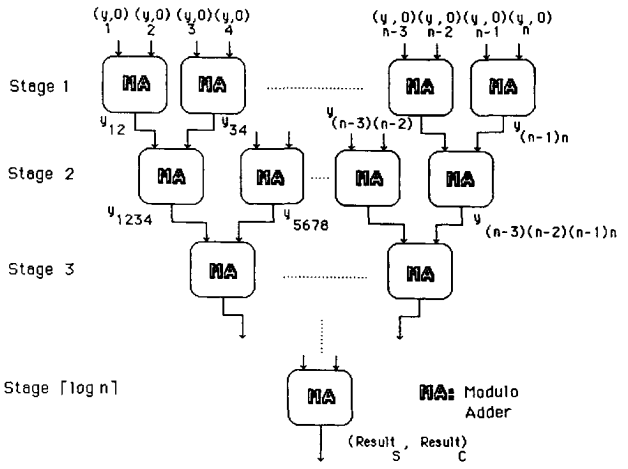

Fig. 6. Addition of partial products using modulo adders.

c) $a > M$ and $b < M$ like case.

d) $a > M$ and $b > M$ then $a = M + x$ and $b = M + y$, then:

$$|a+b|_M = |M + x + M + y|_M = |x+y|_M = |a_M + b_M|_M. \tag{4}$$

From the previous four cases:

$$|a+b|_M = |a_M + b_M|_M. \tag{5}$$

Using (5) we have:

$$|a+b|_M = \left||y_1 + \cdots + y_{n/2}|_M + |y_{n/2+1} + \cdots + y_n|_M\right|_M.$$

We can further expand this expression using the same method to get the addition process in the right hand side in terms of only two operands added in modulo M. □

Theorem 1 means that adding n numbers in modulo M can be performed using a binary tree consists of units that are capable of adding only two numbers in modulo M. The modulo adders are used as those building blocks to perform the addition process. Since the modulo addition requires that inputs be represented in the form of sum and carry, then this form should be enforced at all levels. The form will be enforced automatically for levels ≥ 2, because the outputs of the previous levels are in the correct form. For first level we have the following:

$$Y_{iS} = y_i.$$
$$Y_{iC} = 0 \quad \forall\, 1 \leq i \leq n.$$

For the second stage the output is in the form of sum and carry which is exactly the same form we have using the CSA's.

V. Modulo Multiplier Evaluation

A. Asymptotic Complexity

Using the VLSI model of computation for asymptotic complexity [15], a comparative study for the proposed multiplier is analyzed. For multiplier I (Fig. 1) the complexity measures will be as follows:

$$A = \theta(n^2)$$
$$T = \theta(n)$$
$$AT^2 = \theta(n^4).$$

TABLE I
Comparison Between Different Modulo Multipliers

Type	Area (A)	Time (T)	AT^2
Type I	n^2	n	n^4
Type II	$n2^n$	n	$n^3 2^n$
Proposed	n^2	$\log n$	$n^2(\log n)^2$

For multiplier II (Fig. 2), using the complexity analysis of the correction table of [15]:

$$A = O(n^2 + 2^n n) = O(n2^n)$$
$$T = O(n)$$
$$AT^2 = O(n^3 2^n).$$

For the proposed multiplier, the first stage consists of n^2 AND gates. The area of this stages is $\theta(n^2)$. The partial products are obtained after a constant time (The AND gate delay). For the second stage number of adders required to perform this stage is:

$$\frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \cdots + 1 = n - 1 = \theta(n).$$

Since each modulo adder has $\theta(n)$ full adders, then the total area required for this stage is $\theta(n^2)$. The time required to perform the addition process is the delay of the modulo adder multiplied by the depth of the tree $\theta(\log n)$.

$$A = \theta(n^2)$$
$$T = \theta(\log n)$$
$$AT^2 = \theta[n^2(\log n)^2].$$

From the previous analysis it is clear that the proposed multiplier is superior than previously proposed schemes for medium and large moduli. Table I shows a comparison between the three schemes.

B. Layout Complexity

An 8-b multiplier is implemented based on Domino Logic using a double metal 3-micron CMOS technology. CRYSTAL program is used to analyze the VLSI layout. The input consists of a circuit description extracted from the mask layout using MEXTRA program. Crystal determines each clock phase length and the circuit's critical paths. This helps in performance tuning by optimizing the critical paths. Although Crystal is designed for systems using multiple nonoverlapping clocks by determining each clock phase length, it checks neither clock skew nor set-up/hold times. Crystal can efficiently analyze the systolic multiplier due it its simplicity and regularity. Table II summarizes the area and time of different components of the design.

For the modulo adder we use three cell types. Type I is used in the implementation of stages one and two, Type II is used in the implementation of stage three, and Type III is used in the implementation of stages four and five. Type I consists of n full-adders, Type II consists of a three input multiplexer and n full-adders, and Type III consists of two input multiplexer and n full-adders. The longest delay among the three types is for type III. The length of type III delay determines the clock period. The clock period is 32.40 ns, which gives a throughput of 31 M modulo multiplication operation per second.

TABLE II
AREA AND TIME DELAY OF THE 8 b MULTIPLIER

Cell	Area	Time Delay
And array	37 * 452	1.45 ns
Latch	232 * 73	5.30 ns
Type I	154 * 164	15.83 ns
Type II	215 * 164	32.40 ns
Type III	298 * 164	17.20 ns
Modulo adder	1557 * 3397	124.96 ns
Multiplier	7569 * 14547	376.33 ns

TABLE III
TIME DELAY AND THROUGHPUT OF n-bit MODULO MULTIPLIERS

Number of bits	Time Delay (ns)
8	376.330
16	501.290
32	626.250
64	751.210
128	876.170

We can generalize the previous figures for n-bit modulo multiplication. The total delay consists of the AND gates array delay and the tree's delay. The array has a delay of 1.45 ns and each adder has a delay of 124.9 ns, then:

$$\text{Total delay} = 124.9 * \log\, n + 1.45\,\text{ns}.$$

The clock period is constant regardless of n, then:

$$\text{Clock period} = 32.40\,\text{ns}.$$

$$\text{Throughput} = 30M \text{ modulo-multiplication/s}.$$

Table III shows the total delay for different multiplier sizes.

VI. CONCLUSION

The modulo multiplier introduced in this brief has a total time-delay complexity of $\theta(\log\, n)$ for multiplying two n-bit numbers in modulo m. Based on the analysis of Section V-A, this adder is the fastest and the most area efficient for large moduli. The VLSI implementation using double metal 3 micron shows that the pipelined multiplier can operate with a clock rate of 32.4 ns, which leads to a throughput of 30 M modulo multiplication operation per second. The proposed design has the following advantages:

1) It does not have any limitation on the size of the modulus.
2) It is quite modular, it is a 2-D array of two cell types (modulo adder, AND gates).
3) It is easy to pipeline yielding a very high throughput.
4) It is very fast and area-efficient compared with other schemes.

REFERENCES

[1] F. J. Taylor, "Residue arithmetic: A tutorial with examples," *IEEE Comp. Mag.*, pp. 50–62, May 1984.

[2] M. A. Bayoumi, G. A. Jullien, and W. C. Miller, "A look-up table VLSI design methodology for RNS structures used in DSP applications," *IEEE Trans. Circuits Syst.*, vol. CAS-34, pp. 604–616, June 1987.

[3] M. A. Bayoumi, "A high speed VLSI complex digital signal processor based on quadratic residue number system," in *VLSI Signal Processing II.* New York: IEEE Press, 1986, pp. 200–211.

[4] ——, "Digital filter VLSI systolic arrays over finite fields for DSP applications," in *Proc. 6th IEEE Ann. Phoenix Conf. Comp. Commun.*, Feb. 1987, pp. 194–199.

[5] W. Jenkins and E. Davidson, "A custom-designed integrated circuit for the realization of residue number digital filters," in *Proc. ICASSP 1985,* Mar. 1985, pp. 220–223.

[6] M. A. Soderstrand and C. Vernia, "A high-speed low-cost modulo p_i multiplier with RNS arithmetic application," *Proc. IEEE,* vol. 68, pp. 529–532, Apr. 1980.

[7] M. A. Soderstrand and E. L. Fields, "Multipliers and residue number arithmetic digital filters," *Electron. Lett.*, vol. 13, no. 6, pp. 164–166, Mar. 1977.

[8] G. A. Jullien, "Implementation of multiplication modulo a prime number with applications to number theoretic transforms," *IEEE Trans. Comput.*, vol. C-29, pp. 899–905, Oct. 1980.

[9] F. J. Taylor, "Large moduli multipliers for signal processing," *IEEE Trans. Circuits Syst.*, vol. CAS-28, July 1981.

[10] F. J. Taylor and C. H. Huang, "An autoscale residue multiplier," *IEEE Trans. Comput.*, vol. C-31, pp. 321–325, Apr. 1982.

[11] K. M. Elleithy and M. A. Bayoumi, "A $\theta(1)$ algorithm for modulo addition," *IEEE Trans. Circuits Syst.*, vol. 37, pp. 628–631, May 1990.

[12] K. Hwang, *Computer Arithmetic: Principles, Architecture, and Design.* New York: Wiley, 1978.

[13] K. M. Elleithy, "On the bit-parallel implementation for the Chinese remainder theorem," VLSI Tech. Rep. TR87-8-1, The Center for Advanced Computer Studies, University of Southwestern Louisiana, 1987.

[14] K. M. Elleithy and M. A. Bayoumi, "Fast and flexible architectures for RNS arithmetic decoding," *IEEE Trans. Circuits Syst. II,* vol. 39, pp. 226–235, Apr. 1992.

[15] M. A. Bayoumi, "Lower bounds for VLSI implementation of residue number system architectures," *Integration, The VLSI J.*, vol. 4, no. 4, pp. 263–269, Dec. 1986.

[16] K. M. Elleithy and M. A. Bayoumi "A $\theta(\log\, n)$ algorithm for modulo multiplication," in *Proc. 32nd Midwest Symp. Circuits Syst.*, Aug. 1989.

[17] K. M. Elleithy, "On bit-parallel processing for modulo arithmetic," VLSI Tech. Rep. TR86-8-1, The Center for Advanced Computer Studies, University of Southwestern Louisiana, 1986.

The Synthesis of Transient Processes

Dean J. Schmidlin

***Abstract*—This brief presents a procedure for synthesizing a transient process by exciting a causal linear shift-variant system with an orthonormal random sequence. The output process has finite average energy and is described by its average energy density spectrum. This spectrum is equal to the ensemble average of the magnitude-squared of the Fourier transform process associated with the transient process. A prototypical bandpass average energy density spectrum is introduced, and a more general spectrum is obtained by adding together a collection of prototypical spectra each having different values for the center frequency, bandwidth, and peak amplitude. Realizations of the transient process are obtained by a matrix transformation applied to an ensemble of MATLAB-generated random input vectors. The average of the magnitude-squared discrete Fourier transform of the resulting random output vectors approximates the average energy density spectrum of the transient process.**

I. INTRODUCTION

This brief is concerned with the synthesis of discrete-time random processes that have finite average energy. These processes are the stochastic counterpart of causal deterministic sequences with finite energy. Robinson [1] coined the term "wavelets" to denote such deterministic finite-energy signals. He also defined and considered

Manuscript received October 6, 1993; revised June 22, 1994 and October 26, 1994. This paper was recommended by Associate Editor W.-S. Lu.

The author is with the Department of ECE, University of Massachusetts Dartmouth, North Dartmouth, MA 02747 USA.

IEEE Log Number 9414329.