2014

Generator Polynomial Formulation for Parallel Counters with Applications

Lee A. Belfore II
Old Dominion University

Original Publication Citation

Belfore, L. A., II. (2014). Generator polynomial formulation for parallel counters with applications. *International Journal of Computers and Their Applications, 21*(1), 4-13.

Generator Polynomial Formulation for Parallel Counters with Applications

Lee A. Belfore, II *
Old Dominion University, Norfolk, VA 23529, USA

Abstract

Parallel counters have been studied for several decades as a component in high speed multipliers and multi-operand adder circuits. Using a generator polynomial as a formalism for describing parallel counters in the general case, parallel counter properties can be derived and inferred. Furthermore, the structure and decomposition of the generator polynomial can suggest different implementation strategies. These include simple implementations of (7,3) and (15,4) parallel counters. By grouping factors, the design of a fast (7,3) parallel counter is presented. Finally, the generator polynomial is extended to permit factors of different weights. This extension provides a means for describing the design of the (5,5,4) and (4,5,5,5) multicolumn parallel counters.

Keywords: Parallel counter, generator polynomial, GF(2) algebra, logic synthesis, fast multipliers.

1 Introduction

A parallel counter is a combinational logic circuit that, in its most basic form, inputs a collection of equally weighted binary literals and outputs a binary number reporting the number of these literals that are one. Parallel counters are used in applications which are sped up when multiple addends can be concurrently added to form a result. This occurs in such applications as fast combinational multipliers and signal processing applications. Methods for describing parallel counters usually take the form of heuristics that build upon the basic description of a parallel counter. As such, no known general formalism exists for representing parallel counters algebraically.

Because applications vary, thereby changing the number of operands, a generalized approach for synthesizing parallel counters is of value. Many of the approaches build general parallel counters from (3,2) parallel counters (e.g., full adders) as well as higher order parallel counter building blocks. Indeed, the references [6, 7] provide good overviews of the design and specification of parallel counters. Applications of parallel counters in fast multipliers were first formulated in [3, 8]. Because parallel counters are used in high performance applications, they are often designed with high speed and compactness in mind to manage the complementary issues of performance and implementation complexity.

This paper is organized into five sections including an introduction, a short tutorial of parallel counter multiplication applications, a development of the parallel counter formalism, a presentation of several circuit synthesis examples based on the formalism, and a summary.

2 Basic Overview of the use of Parallel Counters

Parallel counters have an application where the sum of a collection of values is required. In particular, much effort has been devoted to devising fast multipliers for high performance computation. In this section, we provide a basic introduction to the use of parallel counters in fast multipliers. For more information, the interested reader is encouraged to consult [4, 5]. A simplified schematic of a fast multiplier strategy is presented in Figure 1.

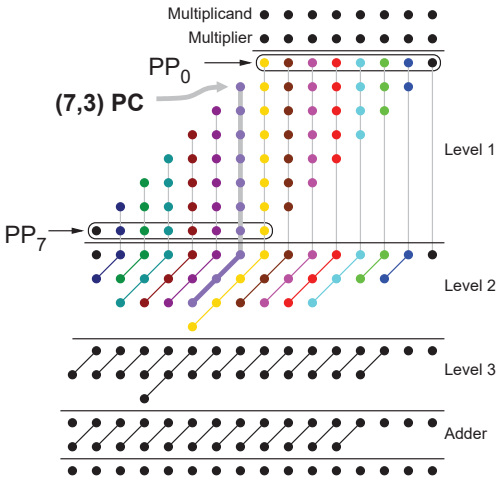


Figure 1: Strategy for fast multiplication

One strategy for fast multiplication is to perform all operations using combinational logic implementations of parallel counters, employ a carry save strategy, and then ripple carries in the very last phase of the computation. The multiplication example shown above effectively goes through four phases. In the first phase, the partial products are generated which is simply the products of each bit in the

*Department of Electrical and Computer Engineering, Email: lbelfore@odu.edu

multiplier with the multiplicand, placing each partial product in the appropriately shifted position depending on the bit of the multiplier. This gives the level 1 partial products. Adding columns and saving the carries results in the Level 2 and 3 partial products. Finally, in the last phase, a fast adder adds the results to form the product. Noted also in this diagram are the parallel counters in Level 1 along with the results produced in Level 2. Highlighted is a (7,3) parallel counter and its result. In addition, other organizations are possible using multiple column parallel counters as will be noted later in this paper.

3 Generator Polynomial Formulation

Before introducing the generator polynomial formulation, we first state some assumptions. First, the GF(2) algebra is used in the formulation. Recall that GF(2) has two operators, *Exclusive-OR* (XOR) and *AND*, which will be denoted by $+$ and $\cdot$ (often implied) respectively. Literals in the GF(2) system nominally take on the values 0 and 1. The GF(2) system serves as the mathematical system that is used in a variety of applications including, for example, encryption and error correcting codes. In this section, we explore a generator polynomial formulation for parallel counters [1].

3.1 Properties of the Generator Polynomial

Rather than formulating a fixed coefficient generator polynomial, the generator polynomial coefficients described here are determined by literals. A literal a can be represented by the following factor

$$\mathcal{F}(r) = 1 + a \cdot r, \tag{1}$$

where r is the generator polynomial indeterminate variable. Representing literals using (1) suggests a mathematical formulation for combining literal factors into higher order generator polynomials. For example, the product of the factors for a_1, a_2, and a_3 is

$$\begin{aligned} \mathcal{F}^{(3)}(r) &= (1 + a_1 r)(1 + a_2 r)(1 + a_3 r) \\ &= 1 + (a_1 + a_2 + a_3)r \\ &\quad + (a_1 a_2 + a_1 a_3 + a_2 a_3)r^2 \\ &\quad + (a_1 a_2 a_3)r^3 \\ &= 1 + F_1^{(3)} r + F_2^{(3)} r^2 + F_3^{(3)} r^3. \end{aligned} \tag{2}$$

Note that the degree for each coefficient matches the number of literals in the contributing product terms, or cubes, included in the coefficient. Furthermore, and curiously, the second and first degree coefficients, respectively, are the sum and carry functions for the full adder function with input literals a_1, a_2, and a_3. Indeed, these also define the outputs for the (3,2) parallel counter:

$$\begin{aligned} F_2^{(3)} = S_1 = &\quad \text{Carry} \quad = a_1 a_2 + a_1 a_3 + a_2 a_3 \\ F_1^{(3)} = S_0 = &\quad \text{Sum} \quad = a_1 + a_2 + a_3. \end{aligned} \tag{3}$$

Traditionally, the full adder carry function is expressed as sum of products form with AND and OR primitives. While in GF(2) '+' is XOR, it is easy to confirm the correctness of the full adder carry function.

Extending this result, a generator polynomial can be formulated in the general case for n literals $a_1, a_2, ..., a_n$

$$\begin{aligned} \mathcal{F}^{(n)}(r) &= \prod_{i=1}^{n} (1 + a_i r) \\ &= 1 + F_1^{(n)} r + F_2^{(n)} r^2 + \cdots + F_n^{(n)} r^n \\ &= 1 + \sum_{d=1}^{n} F_d^{(n)} r^d. \end{aligned} \tag{4}$$

From (4), several useful properties can be elicited.

Theorem 1. Each coefficient $F_d^{(n)}$ in $\mathcal{F}^{(n)}(r)$ is the XOR sum of all $\binom{n}{d}$ distinct degree d cubes.

Proof: The binomial expansion,

$$\mathcal{F}(r) = (x_0 + x_1)^n, \tag{5}$$

can be expressed as

$$\mathcal{F}(r) = \prod_{i=1}^{n} (x_0 + x_1). \tag{6}$$

In (6), the product is composed of 2^n terms where each can be uniquely identified by an n bit codeword, $\mathcal{S}_m$, representing the contribution of either x_1 or x_0 from each of the n factors at position i and is expressed as

$$\mathcal{S}_m = \Big\|_{i=1}^{n} \begin{cases} x_1 & \text{for} \quad m_{i-1} = 1 \\ x_0 & \text{for} \quad m_{i-1} = 0 \end{cases} \tag{7}$$

where $\|$ is the concatenation operator and m is the "value" of this codeword. The codeword value is determined by associating a 1 for x_1, and 0 for x_0 in the following expression

$$m = \sum_{i=1}^{n} \begin{cases} 2^{i-1} & \text{for} \quad x_1 \\ 0 & \text{for} \quad x_0. \end{cases} \tag{8}$$

Generalizing (6), we substitute $x_0 = 1$ and $x_1 = a_i r$, resulting in the unexpanded and expanded polynomial forms given in (4). Since it has been shown that all codewords, and therefore all terms, are generated by the binomial product, (4) is exhaustive and includes all unique cubes. In this formulation, each polynomial coefficient i will be the sum of all different cubes composed from exactly i literals, because cubes having greater or fewer literals will be included in higher or lower degree coefficients. □

Lemma 1. The number of degree d cubes in (4) is $N_d = \binom{n}{d}$.

Proof: The Binomial Theorem can be used to determine the number of product terms fitting a particular composition, i.e. the multiplicity of x_1 and x_0. Consider the expression

$$\mathcal{F}(r) = \prod_{i=1}^{n}(1 + a_i r). \tag{9}$$

In (9), 2^n cubes result, and each can be associated with an encoding, m, that reflects the contribution of the literals, a_i. Each cube is unique because it reflects of a different selection of literals from each of the respective n factors. Furthermore, each cube's degree, d, is the number of literals in the product and is also the the number of ones in the encoding m. Thus, the number of cubes for a particular degree follows from the binomial coefficient

$$N_d = \binom{n}{d}, \tag{10}$$

where N_d is the number of degree d cubes. Each degree d coefficient, $F_d^{(n)}$, is the XOR of the N_d different degree d cubes. □

3.2 Power of Two Combinatorics

In §3.1, the first and second degree coefficients from $\mathcal{F}^{(3)}(r)$ were the sum and carry functions for the full adder. In order to generalize the formulation to parallel counters of arbitrary size, the number of cubes contributing to a coefficient will be determined to study coefficient properties. Since the polynomial is GF(2), the focus will be determining whether the number of cubes contributing to specific coefficients is even or odd.

Lemma 2. For an integer $k > 1$, $(2^k)!$ is divisible by two with multiplicity $(2^k - 1)$.

Proof: The lemma can be proven by repeatedly factoring out two. Note that the number of even terms in $(2^k)!$ is $\frac{2^k}{2}$, the number divisible by four are $\frac{2^k}{4}$, the number divisible by 2^i for $i \leq k$ are $\frac{2^k}{2^i}$. Let p be the number of twos that can be factored out and is expressed as

$$\begin{aligned} p &= \sum_{i=1}^{k} \tfrac{2^k}{(2^i)} \\ &= \sum_{i=1}^{k} 2^{k-i} \\ &= \sum_{j=0}^{k-1} 2^j \\ &= 2^k - 1. \end{aligned} \tag{11}$$

Equation (11) accounts for all factors of two and the final step in (11) establishes that $(2^k)!$ is divisible by two with multiplicity $(2^k - 1)$. □

This property is generally attributed to Legendre.

Lemma 3. For $k > j$, $(2^k - 2^j)!$ is divisible by two with multiplicity $(2^k - 2^j - (k - j))$.

Proof: By Lemma 2, $(2^k)!$ is divisible by two with multiplicity $(2^k) - 1$. Note that $(2^k - 2^j)!$ has 2^j fewer factors and will thus reduce the multiplicity. Noting further that $(2^k - 2^j)!$ is on a 2^j boundary, the multiplicity is reduced by $(2^j) - 1$. The very last factor, 2^k has only been reduced by 2^j, and so multiplicity must be further reduced by $(k - j)$. We can express the power of two divisibility in $(2^k - 2^j)!$ as

$$\begin{aligned} p_2 &= (2^k - 1) - (2^j - 1) - (k - j) \\ &= 2^k - 2^j - (k - j). \end{aligned} \tag{12}$$

□

Lemma 4. For $k > j$, $\binom{2^k}{2^j}$ is divisible by two with multiplicity $k - j$.

Proof: Because

$$\binom{2^k}{2^j} = \frac{(2^k)!}{(2^k - 2^j)!(2^j)!} \tag{13}$$

and using the previous Lemmas, (13) is divisible by two with multiplicity

$$\begin{aligned} p_2 &= 2^k - 1 - ((2^k - 1) - (2^j - 1) - (k - j)) - \\ &\quad (2^j - 1) \\ &= (k - j). \end{aligned} \tag{14}$$

Thus, $\binom{2^k}{2^j}$ is divisible by 2^{k-j}. □

Lemma 5. The quantity $2^j + 2^m$, for $m < j$, is divisible by 2^m.

This can easily be confirmed by noting that $(2^j + 2^m) = (2^{j-m} + 1)2^m$. □

Lemma 6. $\binom{2^k+i}{2^j}$ is even for $j < k$ and $i < 2^j$.

Proof: For $i = 0$, $\binom{2^k}{2^j}$ can easily be shown to be even by applying Lemma 4. The remaining cases can be examined by rearranging the factors in the numerator and denominator as follows:

$$\begin{aligned} \tbinom{2^k+i}{2^j} &= \tfrac{(2^k+i)!}{(2^k+i-2^j)!(2^j)!} \\ &= \left(\tfrac{(2^k+i)\cdots(2^k+1)}{(2^k-2^j+i)\cdots(2^k-2^j+1)}\right)\left(\tfrac{(2^k)!}{(2^k-2^j)!(2^j)!}\right) \\ &= \tfrac{(2^k+i)\cdots(2^k+1)}{(2^k-2^j+i)\cdots(2^k-2^j+1)}\tbinom{2^k}{2^j}. \end{aligned} \tag{15}$$

Because the numerator of $\frac{(2^k+i)\cdots(2^k+1)}{(2^k-2^j+i)\cdots(2^k-2^j+1)}$ starts on a 2^k boundary and the denominator starts on a 2^j boundary not divisible by 2^{j+1}, each is divisible by the same number of factors of two by Lemma 5 and thus is odd. By Theorem 2, the second term, $\binom{2^k}{2^j}$ is divisible by two with multiplicity $k - j$. Thus, $\binom{2^k+i}{2^j}$ is even for $k > j$ and $i < 2^j$. □

Lemma 7. $\binom{2^k+i}{2^k}$ is odd for $i < 2^k$.

Proof: Note that

$$\begin{aligned}\binom{2^k+i}{2^k} &= \frac{(2^k+i)!}{(2^k)!(i)!} \\ &= \frac{((2^k+i)\cdots(2^k+1))(2^k)!}{(2^k)!(i)!} \\ &= \frac{(2^k+i)\cdots(2^k+1)}{(i)!}.\end{aligned} \tag{16}$$

Since $((2^k+i)\cdots(2^k+1))$ and $(i)!$ begin on 2^k boundaries, each is divisible by two in the same multiplicity and $\binom{2^k+i}{2^k}$ is thus odd. □

3.3 Properties of Generator Polynomial Coefficients

In this subsection, the properties of generator polynomial coefficients will be studied in terms of their relation to the number of literals that are one. In the previous section, a collection of combinatorial relations were developed to ascertain whether a particular combination specification was even or odd. This is relevant to studying of coefficients because if we can relate the number of literals that are one to cubes in polynomial coefficients, the coefficient values can determined as well as other useful relationships.

We will begin with the following definitions.

Definition 1. *An instance, ι, of a degree d cube is denoted by C_d^ι and is an instance ι of a product term composed of d different literals.*

Definition 2. *A cube of degree d, C_d^ι* **covers** *a cube C_e^s, $e \leq d$ when the literals that form C_e^s are a subset of the literals used to form to C_d^ι.*

Definition 3. *The quantity $|\mathbf{A}|$ is the number of literals in the vector $\mathbf{A} = a_1 a_2 \cdots a_n$ that are one.*

From these definitions, we can state the following two lemmas.

Lemma 8. If the cube $C_{2^k}^\iota = 1$ and $j < k$, then each covered cube $C_{2^j}^\zeta = 1$.

Proof: If $C_{2^k}^\iota = 1$, then all of the literals in the cube must be one. Furthermore, any cube ζ formed by a subset of these literals must also be one so $C_{2^j}^\zeta = 1$. □

Lemma 9. The cube $C_{2^k}^\iota$ covers $\binom{2^k}{2^j}$ degree 2^j cubes when the 2^j literals are subset of the 2^k literals used to form $C_{2^k}^\iota$.

Proof: Since literals in the degree 2^j are a subset of the literals formed from $C_{2^k}^\iota$, the total number of different cubes is the different 2^j degree cubes selected from a pool of 2^k, or $\binom{2^k}{2^j}$. □

Next, the following theorem considers the case where $|\mathbf{A}| = 2^k$ and the impact on the degree 2^j, $j \leq k$ coefficients $F_{2^j}^{(n)}$.

Theorem 2. If $|\mathbf{A}| = 2^k$, exactly one cube $F_{2^k}^{(n)}$ evaluates to one and results in $F_{2^k}^{(n)} = 1$. All other coefficients, $F_{2^j}^{(n)} = 0$ for $j < k$.

Proof: If $|\mathbf{A}| = 2^k$, then exactly one the cube, $C_{2^k}^\iota$ is one and it is included in $F_{2^k}^{(n)}$, so therefore $F_{2^k}^{(n)} = 1$.

For $j < k$, the coefficient $F_{2^j}^{(n)}$ includes the contribution of $\binom{2^n}{2^j}$ cubes, all covered by $C_{2^k}^\iota$, or $\binom{2^k}{2^j}$ are one. By Lemma 4, $\binom{2^k}{2^j}$ is divisible by 2^{k-j} and thus is even. Since $\binom{2^k}{2^j}$ is even, $F_{2^j}^{(n)} = 0$. □

One interpretation of Theorem 2 is that if $|\mathbf{A}| = 2^k$, all smaller cubes $C_{2^j}^\zeta$ that are formed from the same literals are covered and effectively masked out because the total number of these smaller cubes is even. Next, we generalize this result for cases where $|\mathbf{A}| \neq 2^k$ in the following two corollaries.

Corollary 1. For $0 \leq i < 2^k$, if $|\mathbf{A}| = (2^k + i)$, then $F_{2^k}^{(n)} = 1$.

Proof: Since $(2^k + i)$ ones cover $\binom{2^k+i}{2^k}$ degree 2^k cubes and each of these 2^k cubes is one, then the number of degree 2^k cubes are odd by Lemma 7, and $F_{2^k}^{(n)} = 1$. □

Corollary 2. If $|\mathbf{A}| = (2^k, 2^{k+1})$, one cube $C_{2^k}^\iota$ covers 2^k literals and all cubes composed of these literals. The remaining $|\mathbf{A}| - 2^k$ literals are disjoint from $C_{2^k}^\iota$ and uncovered.

Proof: By Corollary 1, an odd number of cubes are one in the coefficient $F_{2^k}^{(n)}$. Without loss of generality, we select one representative cube $C_{2^k}^\iota$ and the remaining even number of cubes effectively cancel one and other out. By Theorem 2, all lower order cubes covered by $C_{2^k}^\iota$ make no contribution to their respective lower order generator polynomial coefficients. The remaining $|\mathbf{A}| - 2^k$ uncovered literals, $\mathbf{A^d}$, are disjoint from the representative 2^k literals. □

Corollary 3. The disjoint literals from Corollary 2, $\mathbf{A^d}$, form a lower order generator polynomial representative of its constituent disjoint literals.

Proof: The original general polynomial was given in (4). Factoring out the contribution of the 2^k literals that were one leaves a generator polynomial with $|\mathbf{A^d}| = |\mathbf{A}| - \mathbf{2^k}$ factors. □

Corollaries 2 and 3 can be used as a sieve to associate literals that are one with power of two degree generator polynomial coefficients. The following definition provides a convenient way to state the starting point for this process.

Definition 4. *k_m is the largest power of two such that 2^{k_m} does not exceed n or $2^{k_m} \leq n < 2^{(k_m-1)}$.*

Pulling all this together, one of the principle results of this paper is captured in the following theorem.

Theorem 3. For a given assignment of the literals $a_1 a_2 \cdots a_n$, the coefficients from $F_{2^k}^{(n)}$ for $k \in \{k_m, ..., 0\}$ encode a binary number which is the number literals that are one and is expressed as

$$|\mathbf{A}| = \sum_{k=0}^{k_m} F_{2^k}^{(n)} 2^k. \tag{17}$$

Proof: Given $0 \leq |\mathbf{A}| \leq n$, where $n = 2^{(k_m+1)} - 1$. In the trivial case where $|\mathbf{A}| = 0$, all literals are zero and therefore all $F_{2^k}^{(n)} = 0$.

Next, we will assume that $|\mathbf{A}| = 2^k$. In this case, the k^{th} bit of the encoding is one and the rest are zero by Theorem 2 and $F_{2^k}^{(n)} = 1$.

The general case can be shown by repeated application of Corollaries 2 and 3. If $|\mathbf{A}| = (2^k, 2^{k+1})$, for higher contributions, $[k+1, k_m]$ are zero because the cubes for the respective generator polynomial coefficients must be zero. By Corollary 2, $F_{2^k}^{(n)} = 1$ and it covers the included 2^k literals in the lower order generator coefficients and contributes 2^k to the literal count. By excluding the covered cubes by Corollary 2 and specifying a reduced order generator polynomial of the remaining literals by Corollary 3, the process is repeated until all literals are covered, resulting in the literal count specified by (17). □

From Theorem 3 it is useful to define the binary number that gives the literal count as follows.

Definition 5. *Concatenating the coefficients from* $F_{2^k}^{(n)}$ *for* $k \in \{k_m, ..., 0\}$ *results in the binary string*

$$\mathcal{S} = F_{2^{k_m}}^{(n)} \parallel F_{2^{k_m-1}}^{(n)} \parallel \cdots \parallel F_{2^0}^{(n)}. \tag{18}$$

4 Parallel Counter Synthesis

In addition to being able to express parallel counting function mathematically, it is also exceedingly valuable to show how the generator function formulation can be used to influence implementations. Because the operations in the GF(2) algebra are XOR and AND, any expressions can lead directly to implementation. In this section, several implementation examples are presented. The first example shows how parallel counters can be implemented directly from the polynomial. The second example uses a recursive tree formulation that identifies parallelism in the process to result in an implementation with shorter propagation delays. The third example extends the basic formulation to show how multiple column parallel counters can be implemented.

4.1 Simple Synthesis

To this point, we have developed a mathematical framework for describing parallel counters in the general case. In this subsection, some basic implementation issues are addressed [1, 2]. Because the generator polynomial is formed with AND and XOR operations, in the context of the generator polynomial coefficients, we can specify AND/XOR logic circuits to implement the respective logic functions.

Theorem 4. The expressions that specify $\mathcal{S}$ can be implemented with combinational logic circuits.

Proof: This is shown by implementing the logic functions defined in Theorem 3 with XOR gates and two-input AND gates.

Assume that we start with the generator polynomial $\mathcal{F}^{(j-1)}$. We can express $\mathcal{F}^{(j)}$ as

$$\begin{aligned} \mathcal{F}^{(j)} &= \mathcal{F}^{(j-1)}(1 + a_j r) \\ &= \mathcal{F}^{(j-1)} + a_j \mathcal{F}^{(j-1)} r. \end{aligned} \tag{19}$$

Equation (19) implies three different relations describing the determination of the coefficients in $\mathcal{F}^{(j)}$ depending on the degree of the associated coefficient. For the first degree coefficient,

$$F_1^{(j)} = F_1^{(j-1)} + a_j. \tag{20}$$

Indeed, in the general case, the first degree coefficient is the XOR of all literals. For the highest degree, degree j case,

$$F_j^{(j)} = F_{j-1}^{(j-1)} a_j. \tag{21}$$

In this case, the highest degree coefficient is the AND of all literals. In the remaining cases, the coefficients are

$$F_k^{(j)} = F_k^{(j-1)} + a_j F_{k-1}^{(j-1)}, \tag{22}$$

where $k < j$. Equation (22) indicates a primitive unit consisting of an AND gate and an XOR gate can be used to construct arbitrary sized parallel counters.

Equations (20)-(22) describe how the j^{th} degree generator polynomial can be determined from the $(j-1)^{\text{th}}$ generator polynomial and literal a_j. Furthermore, each additional literal contributes a layer to a combinational logic circuit to give the implementation for the next higher order polynomial. Beginning with the first degree generator polynomial, we can construct the circuit for the n^{th} degree generator polynomial. The resulting parallel counter outputs are simply the coefficients from the generator polynomial identified in Definition 5. □

Theorem 4 can be used to describe all the circuitry necessary to build the generator polynomial coefficients in the general case. Indeed, the actual circuit can be pruned by making a couple of observations. First, given $2^{k_m} \leq n < 2^{k_m-1}$, only the coefficients up to 2^{k_m} need be

considered. In addition, coefficients not contributing to coefficients identified in $\mathcal{S}$ may be pruned as well.

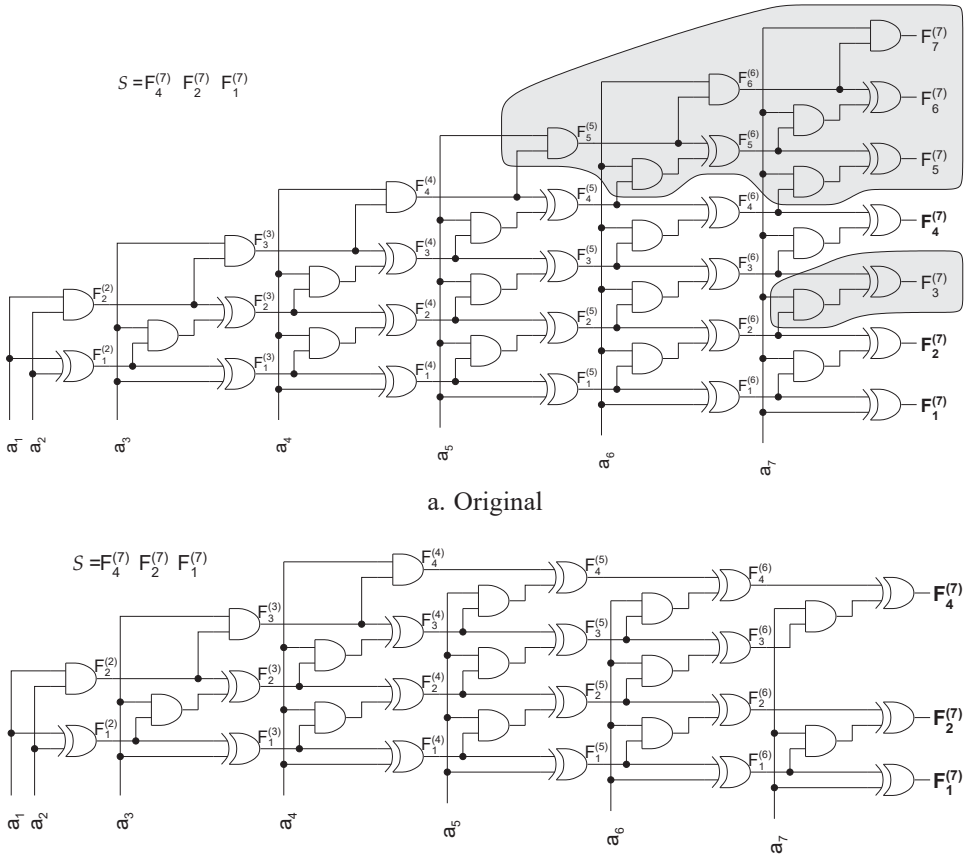
The mathematical formulation just presented forms the basis for the synthesis of parallel counters. Because the mathematics employs AND and XOR operations for the expressions, these can be mapped directly to the implementation. Figure 2 gives a gate level schematic for a (7,3) parallel counter. In addition, this figure is labeled to show the generator polynomial coefficients used to synthesize the circuit. Parts of the circuit that do not contribute to the count are highlighted in gray and can be pruned from the circuit representing the generator polynomial. Extending the process, the circuit for a (15,4) parallel counter can be derived and is shown in Figure 3. Note that the (7,3) parallel counter is embedded in the (15,4) counter reflecting the observation that this can be algebraically factored out.

4.2 Recursive Formulation of Minimum Delay Parallel Counters

Because multiplication of factors is commutative, the order and composition of factor products do not affect the resulting final polynomial. As presented, Equation (4) presumes the product is computed from left to right, resulting in the relatively long chain of operations as can be observed in Figures 2 and 3. Furthermore, these results presume a serial multiplication process where each factor is successively multiplied by the result of the previous intermediate polynomial. This structure does not take advantage of any parallelism that might exist.

Consider a (3,2) parallel counter. Its generator polynomial can be expressed as

$$\begin{aligned}\mathcal{F}^{(3)}(r) &= (1+a_1r)(1+a_2r)(1+a_3r)\\ &= 1+(a_1+a_2+a_3)r\\ &\quad +(a_1a_2+a_1a_3+a_2a_3)r^2\\ &\quad +(a_1a_2a_3)r^3\\ &= 1+F_1^{(3)}r+F_2^{(3)}r^2+F_3^{(3)}r^3.\end{aligned} \tag{23}$$



a. Original

b. Reduced

Figure 2: (7,3) Parallel counter. In Figure 2a, note the pruned components (gray) and the generator polynomial coefficients. The final circuit is shown in Figure 2b

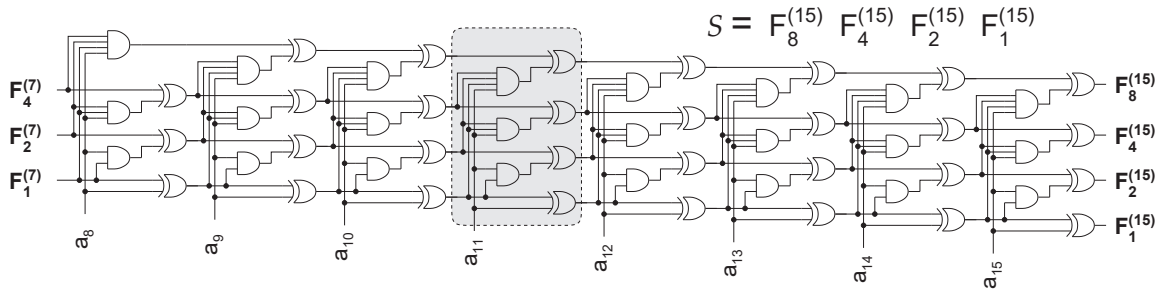


Figure 3: (15,4) Parallel counter with the factor contributing $(1 + a_{11}r)$ highlighted

Grouping the first two factors, the polynomial can be expressed as

$$\begin{aligned}\mathcal{F}^{(3)}(r) &= ((1+a_1r)(1+a_2r))\,(1+a_3r)\\ &= \left(1+(a_1+a_2)r+(a_1a_2)r^2\right)(1+a_3r)\\ &= \left(1+F_1^{(2)}r+F_2^{(3)}r^2\right)(1+a_3r)\\ &= 1\\ &\quad+\left(F_1^{(2)}+a_3\right)r\\ &\quad+\left(F_2^{(2)}+F_1^{(2)}a_3\right)r^2\\ &\quad+\left(F_2^{(2)}a_3\right)r^3.\end{aligned} \tag{24}$$

The result of (24) shows the manner in which polynomials may be combined with other terms, in this case a single factor. Further, because the polynomial coefficients are determined by AND and XOR operations, implementations of circuits follow from the mathematics. The (3,2) parallel counter can be implemented with the circuit given in Figure 4. With close study of this circuit, the reader will correctly conclude that this factoring will provide no benefit over the originally proposed implementation strategy.

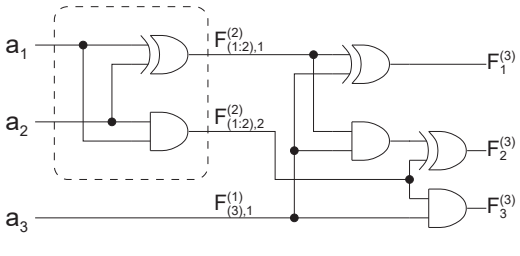


Figure 4: Circuit (3,2) parallel counter

Consider the formulation of the (7,3) parallel counter by partitioning into two smaller polynomials, $\mathcal{F}^{(4)}_{(1:4)}(r)$ & $\mathcal{F}^{(3)}_{(5:7)}(r)$ where the quantity $(1:4)$ refers to the inclusion of literals $a_1, ..., a_4$ and the quantity $(5:7)$ refers to the literals $a_5, ..., a_7$ [2]. The product of these two polynomials will provide the polynomial that results in the formulation of the (7,3) parallel counter

$$\begin{aligned}\mathcal{F}^{(7)}(r) &= \mathcal{F}^{(4)}_{(1:4)}(r)\mathcal{F}^{(3)}_{(5:7)}(r)\\ &= \left(\mathcal{F}^{(2)}_{(1:2)}(r)\mathcal{F}^{(2)}_{(3:4)}(r)\right)\left(\mathcal{F}^{(2)}_{(5:6)}(r)\mathcal{F}^{(1)}_{(7)}(r)\right),\end{aligned} \tag{25}$$

where $\mathcal{F}^{(D)}_{(i:j)}(r)$ is the degree D generator polynomial that includes contributions from literals indexed from i to j. Because the polynomials $\mathcal{F}^{(4)}_{(1:4)}(r)$ & $\mathcal{F}^{(3)}_{(5:7)}(r)$ are independent and share no terms, their logic functions can be evaluated concurrently. Furthermore, the polynomials, $\mathcal{F}^{(4)}_{(1:4)}(r)$ & $\mathcal{F}^{(3)}_{(5:7)}(r)$, can be decomposed themselves into the polynomials $\mathcal{F}^{(4)}_{(1:2)}(r)$, $\mathcal{F}^{(3)}_{(3:4)}(r)$, $\mathcal{F}^{(4)}_{(5:6)}(r)$ & $\mathcal{F}^{(3)}_{(7)}(r)$. Because these lower order polynomials are also independent of one and other, the expressions and any logic that implements their coefficients can operate concurrently as well. The flow implied in computing the result of the parallel count is illustrated in Figure 5. Inspecting how factors are combined in the form of tree clearly highlights the operations that can be performed in parallel. Indeed, operations that can be performed in parallel are nodes on the same row.

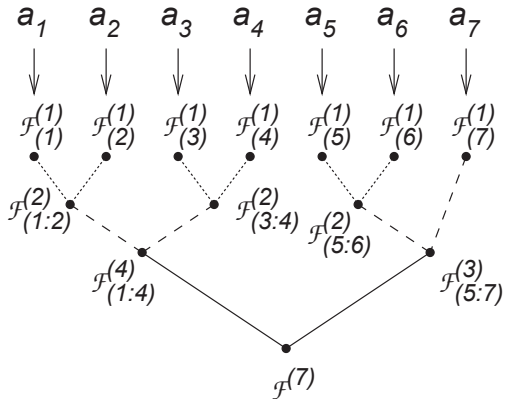


Figure 5: (7,3) Factor products for recursive formulated parallel counter

Circuits for the parallel counter follow directly from the equations that can be derived for each coefficient. First, we

show the polynomial for $\mathcal{F}^{(4)}_{(1:4)}(r)$.

$$\begin{aligned}\mathcal{F}^{(4)}_{(1:4)}(r)=&((1+a_1r)(1+a_2r))\,((1+a_3r)(1+a_4r))\\ =&\left(1+(a_1+a_2)r+(a_1a_2)r^2\right)\\ &\cdot\left(1+(a_3+a_4)r+(a_3a_4)r^2\right)\\ =&\left(1+F^{(2)}_{(1:2),1}r+F^{(2)}_{(1:2),2}r^2\right)\\ &\cdot\left(1+F^{(2)}_{(3:4),1}r+F^{(2)}_{(3:4),2}r^2\right)\\ =&1\\ &+\left(F^{(2)}_{(1:2),1}+F^{(2)}_{(3:4),1}\right)r\\ &+\left(F^{(2)}_{(1:2),2}+F^{(2)}_{(1:2),1}F^{(2)}_{(3:4),1}+F^{(2)}_{(3:4),2}\right)r^2\\ &+\left(F^{(2)}_{(1:2),2}F^{(2)}_{(3:4),1}+F^{(2)}_{(1:2),1}F^{(2)}_{(3:4),2}\right)r^3\\ &+\left(F^{(2)}_{(1:2),2}F^{(2)}_{(3:4),2}\right)r^4,\end{aligned} \tag{26}$$

where each term $F^{(D)}_{(i:j),d}$ is the degree d coefficient from the generator polynomial. From (26), the circuit implementation for the parallel counter with the inputs $a_1a_2a_3a_4$ is given in Figure 6.

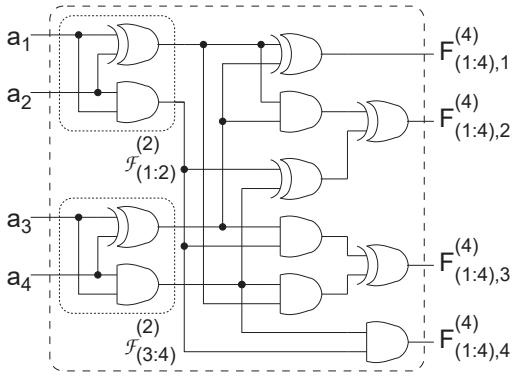


Figure 6: Annotated circuit for (4,3) parallel counter

The degree-3 polynomial that includes literals 5 to 7 follows directly from (24).

$$\begin{aligned}\mathcal{F}^{(3)}_{(5:7)}(r) &= ((1+a_5r)(1+a_6r))\,((1+a_7r))\\ &= \left(1+(a_5+a_6)r+(a_5a_6)r^2\right)\\ &\quad\cdot(1+a_7r)\\ &= \left(1+F^{(2)}_{(5:6),1}r+F^{(2)}_{(5:6),2}r^2\right)\\ &\quad\cdot\left(1+F^{(1)}_{(7),1}r\right)\\ &= 1\\ &\quad+\left(F^{(2)}_{(5:6),1}+F^{(1)}_{(7),1}\right)r\\ &\quad+\left(F^{(2)}_{(5:6),2}+F^{(2)}_{(5:6),1}F^{(1)}_{(7),1}\right)r^2\\ &\quad+\left(F^{(2)}_{(5:6),2}F^{(1)}_{(7),1}\right)r^3.\end{aligned} \tag{27}$$

Modifying the annotations of Figure 4, the implementation for (27) is shown in Figure 7.

Computing the product of the polynomial results in the following coefficients

$$\begin{aligned}F^{(7)}_1 &= F^{(4)}_{\alpha1}+F^{(3)}_{\beta1}\\ F^{(7)}_2 &= F^{(4)}_{\alpha2}+F^{(4)}_{\alpha1}F^{(3)}_{\beta1}+F^{(3)}_{\beta2}\\ F^{(7)}_3 &= F^{(4)}_{\alpha3}+F^{(4)}_{\alpha2}F^{(3)}_{\beta1}+F^{(4)}_{\alpha1}F^{(3)}_{\beta2}+F^{(3)}_{\beta3}\\ F^{(7)}_4 &= F^{(4)}_{\alpha4}+F^{(4)}_{\alpha3}F^{(3)}_{\beta1}+F^{(4)}_{\alpha2}F^{(3)}_{\beta2}+F^{(4)}_{\alpha1}F^{(3)}_{\beta3}\\ F^{(7)}_5 &= F^{(4)}_{\alpha4}F^{(3)}_{\beta1}+F^{(4)}_{\alpha3}F^{(3)}_{\beta2}+F^{(4)}_{\alpha2}F^{(3)}_{\beta3}\\ F^{(7)}_6 &= F^{(4)}_{\alpha4}F^{(3)}_{\beta2}+F^{(4)}_{\alpha3}F^{(3)}_{\beta3}\\ F^{(7)}_7 &= F^{(4)}_{\alpha1}F^{(3)}_{\beta1},\end{aligned} \tag{28}$$

where $\alpha=(1:4)$ and $\beta=(5:7)$.

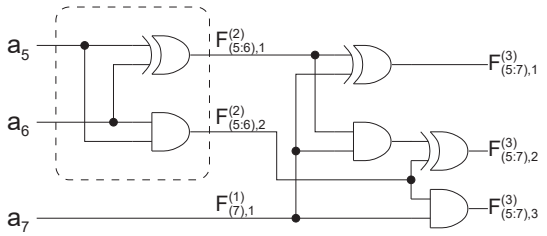


Figure 7: Annotated circuit for (3,2) parallel counter

Note that the parallel counter result is selected power of two coefficients

$$\mathcal{S}=F^{(7)}_4F^{(7)}_2F^{(7)}_1 \tag{29}$$

and the expressions for these coefficients can be mapped directly to the implementation for each. Alternatively, the result can be combined by using a two-stage ripple carry adder to combine $\mathcal{S}_\alpha=F^{(4)}_{\alpha4}F^{(4)}_{\alpha2}F^{(4)}_{\alpha1}$ and $\mathcal{S}_\beta=F^{(4)}_{\beta2}F^{(4)}_{\beta1}$. This alternate implementation would result in a slower circuit requiring more gates.

Combining the circuits shown in Figures 6 and 7 and using the expressions given in (28) results in the circuit for a (7,3) parallel counter shown in Figure 8.

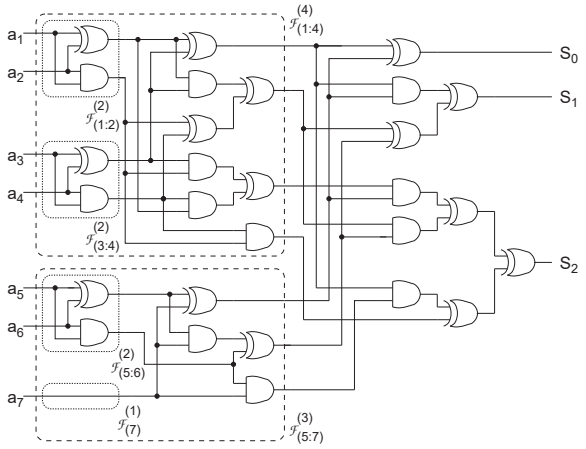


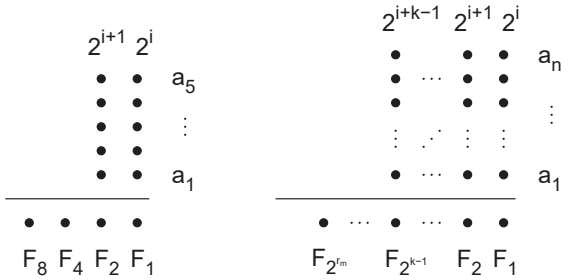
Figure 8: (7,3) Circuit for recursive formulated parallel counter

4.3 Multiple Column Counter Formulation

A multiple column counter adds several columns, with weights that increase by a factor of two for each additional column. For example, a (5,5,4) parallel counter adds five inputs from the first column with a weight of 2^j and five from the second with a weight of 2^{j+1} and produces a four bit sum. The resulting parallel counter can be formulated as

$$\mathcal{F}(r) = \prod_{i=1}^{5}(1 + a_{i,j}r)(1 + a_{i,j+1}r^2), \tag{30}$$

where $a_{i,j}$ is an input from the first column, and $a_{i,j+1}$ is an input from the second column. The largest possible sum is $5\times 2^1+5\times 2^0 = 15$. Thus, the count result is the concatenation of the coefficients $S = F_8F_4F_2F_1$. Figure 9 gives a generalized overview of multiple column counters.



(a) (5,5,4) parallel counter (b) (n,$\cdots$,n,r_m) parallel counter

Figure 9: Generalized block counter

To maintain the proper weights in the contributions to the generating function, the degree of the generator variable for each factor must be the weight of the input and must thus be a power of two. A general formulation of the multiplier column counter can be expressed in the following relation

$$\mathcal{F}(r) = \prod_{i=1}^{n}\prod_{k=1}^{c}(1 + a_{i,j+k-1}r^{(2^k)}), \tag{31}$$

where c is the number of columns, $r^{(2^k)}$ represents the weighted contribution of the input $a_{i,j+k-1}$ [2]. The base two logarithm of highest degree of the generating polynomial that must be computed is

$$k_m = \left\lfloor log_2\left(\sum_{k=1}^{c} n * 2^{k-1}\right)\right\rfloor \tag{32}$$

Following from (32), 2^{k^m} is the largest power of two coefficient in $\mathcal{F}(r)$.

As with the other parallel counters specified by generator polynomials, expressions can be found for the (5,5,4) block parallel counter outputs described by (31). Note that we are assuming for simplicity that $j = 0$.

$$\begin{aligned}\mathcal{F}(r) &= \prod_{i=1}^{5}(1 + a_{i,0}r)(1 + a_{i,1}r^2)\\ &= \left[\prod_{i=1}^{5}(1 + a_{i,0}r)\right]\left[\prod_{i=1}^{5}(1 + a_{i,1}r^2)\right]\\ &= \left[\mathcal{F}_0^{(5)}(r)\right]\left[\mathcal{F}_1^{(5)}(r^2)\right]\end{aligned} \tag{33}$$

Each polynomial factor can be expressed as fifth degree polynomials, but can be factored to reduce the delays associated with the implementation of the respective networks. In the following

$$\begin{aligned}\mathcal{F}^{(5)}(r)=&\left(\mathcal{F}^{(1)}_{(1)}(r)\mathcal{F}^{(1)}_{(2)}(r)\right)\left(\mathcal{F}^{(1)}_{(4)}(r)\mathcal{F}^{(1)}_{(5)}(r)\right)\mathcal{F}^{(1)}_{(5)}(r)\\ =&\mathcal{F}^{(2)}_{(1:2)}(r)\left(\mathcal{F}^{(2)}_{(3:4)}(r)\mathcal{F}^{(1)}_{(5)}(r)\right)\\ =&\mathcal{F}^{(2)}_{(1:2)}(r)\mathcal{F}^{(2)}_{(3:5)}(r)\\ =&\mathcal{F}^{(2)}_{(1:2)}(r)\mathcal{F}^{(2)}_{(3:5)}(r)\end{aligned} \tag{34}$$

Note that (34) can be used for the first and second columns in the block parallel adder by expressing the polynomials in r and r^2 respectively. Because the second column polynomial is degree 10 and half the coefficients are zero, the expressions for the resulting degree 15 polynomial are simpler compared with the general case polynomial. The result is further simplified by recognizing that only the degree 2^k coefficients are necessary

$$\begin{aligned}F_1 &= F^{(5)}_{0,1}\\ F_2 &= F^{(5)}_{0,2} + F^{(5)}_{1,1}\\ F_4 &= F^{(5)}_{0,4} + F^{(5)}_{0,2}F^{(5)}_{1,1} + F^{(5)}_{1,2}\\ F_8 &= F^{(5)}_{1,4} + F^{(5)}_{0,4}F^{(5)}_{1,2}.\end{aligned} \tag{35}$$

Expressing the (5,5,4) block parallel counter in this manner suggests two possible implementations. The first is to provide the implementations for the generator polynomials for the weights 2^0 and 2^1 respectively. From the coefficients of the polynomials $\mathcal{F}_0^{(5)}(r)$ and $\mathcal{F}_1^{(5)}(r^2)$, the expressions in (35) can be implemented directly. The second is to distribute the terms to form expressions from the parallel counter inputs. This second approach provides a result that has a minimum number of layers of logic but the expressions that result are more complicated.

The relative sparseness of the coefficients in (35) is a result of the higher degree coefficients from the second column. Since the second column polynomial has no odd degree terms, only half of the coefficients from the first column contribute to resulting higher order (degree two and higher) polynomial coefficients. A higher order block parallel counter of order (4,5,5,5), selected because of its naturally count dense maximal sum of $2^5 - 1 = 31$, further highlights the simpler coefficient expressions that result from this induced

sparseness. The (4,5,5,5) counter sums three successive columns of 4, 5, and 5 columns respectively. Note that one can study the properties of this counter as a (5,5,4) counter embedded within the (4,5,5,5) counter which, viewed from the perspective of the generator polynomial, is just a term that is factored out. The polynomial for the (4,5,5,5) counter can be expressed as coefficients for the (5,5,4) polynomial can be expressed as

$$\mathcal{F}_{(4,5,5,5)}(r) = \left[\mathcal{F}_2^{(5)}(r^4)\right]\left[\mathcal{F}_{(5,5,4)}(r)\right]. \tag{36}$$

The unity term from $\mathcal{F}_2^{(5)}(r^4)$ includes the polynomial coefficients from $\mathcal{F}_{(5,5,4)}(r)$ with no other coefficients from the former contributing. Further, the higher order coefficients are only necessary in so much as they contribute to the power of two coefficients of the (4,5,5,5) counter. Noting this feature, the coefficients for the (4,5,5,5) counter can be shown to be

$$\begin{aligned} F_1 &= F_{(5,5,4),1} \\ F_2 &= F_{(5,5,4),2} \\ F_4 &= F_{(5,5,4),4} + F_{2,1}^{(5)} \\ F_8 &= F_{(5,5,4),8} + F_{(5,5,4),4}F_{2,1}^{(5)} + F_{2,2}^{(5)} \\ F_{16} &= F_{(5,5,4),12}F_{2,1}^{(5)} + F_{(5,5,4),8}F_{2,2}^{(5)} + \\ &\quad F_{(5,5,4),4}F_{2,3}^{(5)} + F_{2,4}^{(5)}. \end{aligned} \tag{37}$$

5 Summary and Future Work

In this paper, a formalism for describing parallel counters using a GF(2) generator function was presented. An important result is that the power of two coefficients from the polynomial directly provide the parallel counter sum. Furthermore, because GF(2) operands are AND and XOR, implementation strategies can also be studied. Several parallel counter examples were studied and implemented using the mathematics that followed from different manipulations and decompositions of the generator polynomial. In particular, a minimum delay parallel counter was formulated by recursively building the final sum from the constituent factors. In addition, a general expression was provided for describing multiple column parallel counters and two specific examples were examined. Future work will include identifying methods to identify and exploit parallelism and other circuit optimizations. In addition, identifying new applications for this formalism will be pursued.

References

[1] L. A. Belfore II, "Parallel Counter Applications Based on a Generator Polynomial Formulation," *The 2012 International Conference on Computer Design*, 2012.

[2] L. A. Belfore II, "Generator Polynomial Expansion Applied to Parallel Counter Applications," *International Conference on Computers Applications in Industry and Engineering (CAINE-13)*, 2013.

[3] L. Dadda, "Some Schemes for Parallel Multipliers," *Acta Frequenza*, 45:574–580, 1965.

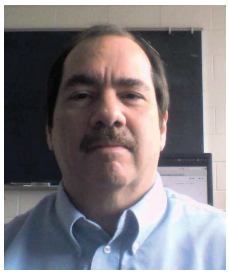
[4] M. J. Flynn and S. F. Oberman, *Advanced Computer Arithmetic Design*, Wiley Interscience, 2001.

[5] I. Koren, *Advanced Computer Arithmetic Design*, A K Peters, 2002.

[6] E. E. Swartzlander, Jr, "Parallel Counters," *IEEE Transactions on Computers*, C-22(11):1021–1024, Nov. 1973.

[7] E. E. Swartzlander, Jr, "A Review of Large Parallel Counter Designs," *Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI'04)*, pp. 89–98, 2004.

[8] C. S. Wallace, "A Suggestion for a Fast Multiplier," *IEEE Transactions on Electronic Computers*, EC-13(1):14–17, 1964.



Lee Belfore is presently an Associate Professor of Electrical and Computer Engineering at Old Dominion University, in Norfolk, Virginia, USA. He received his BS in Electrical Engineering from Virginia Tech in 1982, his MSE in Electrical Engineering & Computer Science from Princeton University in 1983, and his Ph.D. in Electrical Engineering from the University of Virginia in 1990. His research interests include computer arithmetic, low power digital design, and medical modeling and simulation.