

Enumeration of Bent Boolean Functions by Reconfigurable Computer

J. L. Shafer

ECE Department

US Naval Academy

Annapolis, MD 21402 U.S.A.

shafer@usna.edu

S. W. Schneider

Department of ECE

Naval Postgraduate School

Monterey, CA 93943 U.S.A.

stu2k@msn.com/jbutler@nps.edu

J. T. Butler

P. Stănică

Department of Applied Math.

Naval Postgraduate School

Monterey, CA 93943 U.S.A.

pstanica@nps.edu

Abstract

We show that there is significant benefit to using a reconfigurable computer to enumerate bent Boolean functions for cryptographic applications. Bent functions are rare, and the only known way to generate all bent functions is by sieve techniques in which many prospective functions are tested. The speed-up achieved depends on the number of variables n ; for $n = 8$, we show that the reconfigurable computer achieves better than a $60,000\times$ speed-up over a conventional computer. Further, we introduce the transeunt triangle as a means to reduce the number of functions that must be considered. For $n = 6$, this reduction is better than $1,000,000$ to 1 .

Previously, the transeunt triangle had been used only in the design exclusive OR logic circuits; it converts a truth table to the algebraic normal form. However, this fact has never been proven rigorously, and that shortcoming is removed in this paper. Our proof provides a practical benefit; it yields a new realization of the transeunt triangle that has less complexity and delay. Finally, we show computational results from a reconfigurable computer.

1 Introduction

Shannon [17] introduced the concepts of confusion and diffusion as a fundamental technique to achieve security in cryptographic systems. The confusion principle is reflected in the nonlinearity of Boolean functions, since most linear systems are easily breakable. There are various criteria that imply nonlinearity, one of them being bentness. Bent functions were first introduced by Rothaus in 1976 [14], as functions having maximum distance away from the set of affine functions.

Bent Boolean functions have the highest nonlinearity possible, which makes them useful in the design of block and stream ciphers. Maximum length sequences based on bent functions have cross-correlation and autocorrelation properties that are close to the ones of Gold and Kasami

codes [11], which have applications in spread spectrum communication [5].

While we can mathematically define bent functions precisely, to generate them it is a different matter. One needs sophisticated mathematical (like invariant theory) and computational tools to list all n -variable bent functions (this has been achieved for $n \leq 8$). Some of these methods cannot be easily parallelized, and do not offer a significant improvement in a reconfigurable environment.

Using the SRC-6 reconfigurable computer, we have tested millions of Boolean functions. Specific sets of Boolean functions were chosen based on their specific properties, including degree, homogeneity, and symmetry. These groups were evaluated for relationships between nonlinearity and specific properties. The objective is to find groups of Boolean functions that are rich in bent functions [1]. These groups, if small enough, can be tested exhaustively. Testing across the *entire* set of functions, even for small numbers of variables, e.g., $n = 6$ or more, is infeasible because of the large number of functions. The use of the transeunt triangle enables functions to be generated easily in one form, converted to another form and then tested for certain characteristics. Without the transeunt triangle [2, 3], important groups of functions could not be tested efficiently.

2 Background and Definitions

Definition 2.1. A Boolean function f in n variables is a map from the n -dimensional vector space $\mathbb{V}_n = \mathbb{F}_2^n$ to \mathbb{F}_2 , the two-element field. For a function f , let $f_0 = f(0, 0, \dots, 0)$, $f_1 = f(0, 0, \dots, 1)$, \dots , and $f_{2^n-1} = f(1, 1, \dots, 1)$. $TT = (f_0 f_1 \dots f_{2^n-1})$ is the **truth table representation** of f .

Example 2.1. $f = x_1x_2x_3x_4$ has the truth table representation $TT = (0000000000000001)$. $g = x_1x_2 \oplus x_3x_4$ has the truth table representation $TT = (0001000100011110)$. (End of Example)

Definition 2.2. A linear function is the constant 0 function or the exclusive-OR of one or more variables. An **affine**

function is a linear function or the complement of a linear function.

Example 2.2. There are 16 linear functions on 4 variables, $0, x_1, x_2, x_3, x_4, x_1 \oplus x_2, x_1 \oplus x_3, x_1 \oplus x_4, x_2 \oplus x_3, x_2 \oplus x_4, x_3 \oplus x_4, x_1 \oplus x_2 \oplus x_3, x_1 \oplus x_2 \oplus x_4, x_1 \oplus x_3 \oplus x_4, x_2 \oplus x_3 \oplus x_4$, and $x_1 \oplus x_2 \oplus x_3 \oplus x_4$. These functions and their complements comprise the 32 4-variable affine functions. (End of Example)

Affine functions, when used in encrypting a plaintext message, are susceptible to a linear attack. We seek functions that are as “far” away as possible from affine functions.

Definition 2.3. The **Hamming distance** $d(f, g)$ between two functions f and g is the number of places where their truth table representations differ.

Definition 2.4. The **nonlinearity** NL_f of a function f is the minimum Hamming distance between f and an affine function.

Example 2.3. $f = x_1x_2x_3x_4$ has nonlinearity 1, since converting the single 1 to a 0 in its truth table representation creates the truth table representation of the constant 0 function, which is affine. $g = x_1x_2 \oplus x_3x_4$ has a distance 6 or 10 from any affine function. Thus, its nonlinearity is 6. (End of Example)

Definition 2.5. Let f be a Boolean function on n -variables, where n is even. f is a **bent function** if its nonlinearity is maximum among n -variable functions.

Example 2.4. Rothaus [14] showed that bent functions have nonlinearity $2^{n-1} - 2^{\frac{n}{2}-1}$. Thus, $f = x_1x_2x_3x_4$ is not bent ($NL_f = 1$), and $g = x_1x_2 \oplus x_3x_4$ is bent ($NL_g = 6$). (End of Example)

Along with the truth table representation, the algebraic normal form is an important tool in the study of bent functions.

Definition 2.6. The **algebraic normal form (ANF)** of a function f is $f = \sum_{\mathbf{a} \in \mathbb{F}_2} c_{\mathbf{a}} x_1^{a_1} x_2^{a_2} \dots x_n^{a_n}$, where \sum is the exclusive OR sum, $\mathbf{a} = (a_1, a_2, \dots, a_n)$, $c_{\mathbf{a}}, a_i \in \mathbb{F}_2$, $x_i^0 = 1$, and $x_i^1 = x_i$. $ANF = (c_0 c_1 \dots c_{2^n-1})$ is the **ANF representation** of f .

Example 2.5. $f = x_1x_2x_3x_4$ has the ANF $f = x_1x_2x_3x_4$ and the ANF representation $ANF = (0000000000000001)$. $g = x_1x_2 \oplus x_3x_4$ has the ANF $g = x_1x_2 \oplus x_3x_4$ and the ANF representation $ANF = (0001000000001000)$. (End of Example)

Definition 2.7. The **degree of a product term** is the number of variables in that term. The **degree of a function** f is the maximum of the degrees among the product terms in the ANF of f .

Example 2.6. $f = x_1x_2x_3x_4$ has degree 4 and $g = x_1x_2 \oplus x_3x_4$ has degree 2. (End of Example)

Definition 2.8. Functions f and h belong to the same **affine class** if and only if $f = h \oplus a$, where a is an affine function.

Example 2.7. $f = x_1x_2x_3x_4$, a non-bent function, belongs to an affine class of 32 functions. $g = x_1x_2 \oplus x_3x_4$, a bent function, belongs to an affine class of 32 functions. (End of Example)

Certainly, each affine class contains the same number of functions, namely 2^{n+1} . Also, all functions in the same affine class as a non-affine function f have the same degree.

Definition 2.9. A function f is **homogeneous** of degree d if and only if all terms in the ANF of f have degree d .

Example 2.8. $f = x_1x_2x_3x_4$ is homogeneous of degree 4, and $g = x_1x_2 \oplus x_3x_4$ is homogeneous of degree 2. (End of Example)

Xia, Seberry, Pieprzyk, and Charnes [18] considered homogeneity in the context of bent functions and showed the next result.

Theorem 2.1. When $n > 6$, no n -variable homogeneous bent function has degree $\frac{n}{2}$.

Because of $f = x_1x_2 \oplus x_3x_4$, Theorem 2.1 does not hold for $n = 4$. Qu, Seberry, and Pieprzyk [13] found 30 homogeneous 6-variable bent functions of degree 3, and so, Theorem 2.1 does not hold for $n = 6$. Therefore, from [13, 18], for $n > 6$, degree- $\frac{n}{2}$ n -variable bent functions exist, but none are homogeneous. More recently, Meng et al. [10] showed (purely combinatorially) that, for any nonnegative integer k , there exists a positive integer N , such that for $n \geq N$, there do not exist $2n$ variable homogeneous bent functions having degree $n - k$ or more, where N is the least integer satisfying $2^{N-1} > \binom{N+1}{0} + \binom{N+1}{1} + \dots + \binom{N+1}{k+1}$.

3 Architecture of Bent Function Enumerator

A reconfigurable computer allows one to adapt the architecture to the problem. Fig. 1 shows the architecture to enumerate bent functions based on the ANF of the tested functions. This and other variations yield the data we present later. In all cases, a counter was used to enumerate prospective functions. This is shown on the left. This is applied to a block labeled **Transeunt Triangle**. In this case, the counter enumerates ANFs; each bit of the counter determines the presence or absence of a term in the ANF. The transeunt triangle produces the corresponding truth table. This is then applied to a block that computes the function’s nonlinearity, NL . If NL is maximum, the function is bent, and it is stored.

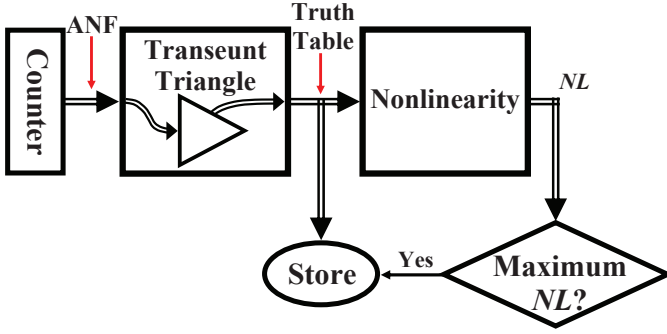


Figure 1. Bent function enumeration circuit

In the SRC-6 reconfigurable computer, this circuit is implemented on a Xilinx Virtex2 Pro FPGA. It is pipelined and runs at 100 MHz. Specifically, one function is tested every clock cycle. We used this to enumerate all 6-variable bent functions [15]. If we had to enumerate all $2^{2^6} = 1.85 \times 10^{19}$ 6-variable functions, this would take 5,849 years. However, Rothaus [14] showed that no bent function has degree greater than $\frac{n}{2}$. By eliminating functions with degree greater than $\frac{n}{2}$, it is only necessary to enumerate $2^{\binom{6}{3} + \binom{6}{2} + \binom{6}{1} + \binom{6}{0}} = 2^{42}$ functions. A function in an affine class is bent if and only if all functions in the same affine class are bent. As a result, the number of bent functions is found by multiplying the number of affine classes by the number of functions in each class, $2^{\binom{6}{1} + \binom{6}{0}} = 2^{128}$. The number of affine classes with degree 3 or less is $2^{\binom{6}{3} + \binom{6}{2}} = 2^{35}$. At one class (function) per 100 MHz clock period, this enumeration takes only 5.7 minutes. That is, by enumerating only the affine classes corresponding to functions of degree 3 or less, we achieve a reduction of $\frac{1}{2^{10}} = \frac{1}{1,048,576}$. However, this requires that we quickly convert between the ANF of a function and its truth table. For this, we need the Transeunt Triangle of Fig. 1, which we discuss in the next section.

Fig. 2 shows the circuit that realizes the **Nonlinearity** block of Fig. 1. The truth table representation of the function f is applied on the left to 2^{n+1} sets of exclusive OR gates to compute 2^{n+1} distance vectors. The number of 1's in these vectors is the distance from f to each affine function. The **Ones_Count** circuit produces a binary number that is the distance between f and an affine function. Then, a **Minimum** circuit computes the overall minimum distance. This is the nonlinearity NL .

Both the **Ones_Count** and the **Minimum** circuit in Fig. 2 are trees. Fig. 3 shows that, in the case of the **Ones_Count** circuit, adders of various sizes form the circuit. Fig. 4 shows that, in the case of the **Minimum** circuit, two-input one-output minimum circuits are used.

The circuit to compute nonlinearity is combinational. However, its delay is larger than the period of the SRC-6's

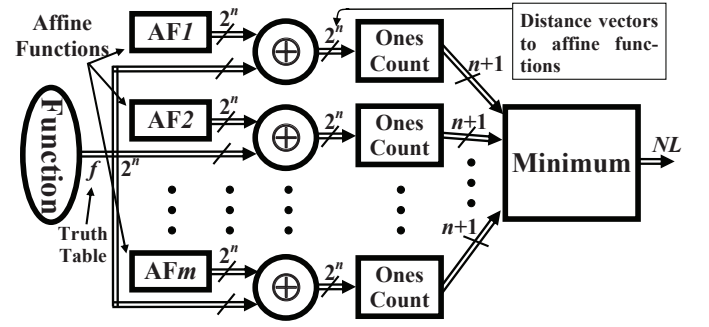


Figure 2. Nonlinearity circuit

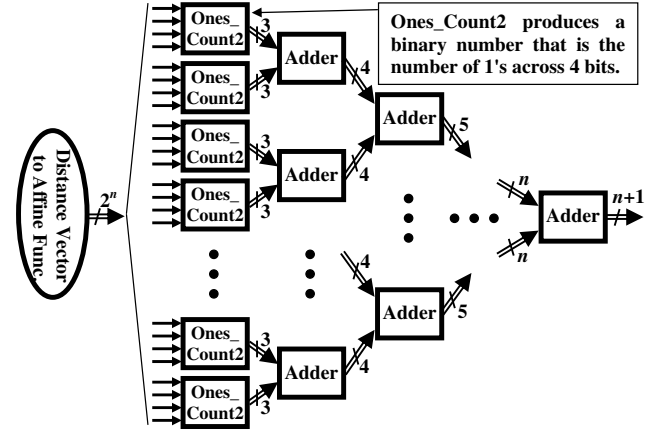


Figure 3. Ones_Count circuit

100 MHz clock, and so it is pipelined. The added latency is not a problem; it is much less than the total computation times we experienced. The FPGA is ideal for this computation, since the many small additions that are needed are done simultaneously. This is to be compared with the PC which has limited adders. Similarly, in the Minimum Circuit, many comparators are used simultaneously. This is to be compared with the PC again, which has limited comparators.

Most of the code was written in Verilog. The code that generated the functions was written in C, and created circuits on the FPGA. Code on the PC, for example, for input-output, was also written in C.

4 The Transeunt Triangle

4.1 Definition

Green [8] and others [2, 3, 7] proposed the transeunt triangle as a means to derive the ANF from the truth table of a given function and, in so doing, produce compact exclusive OR sum-of-products circuits. In this paper, we show the benefit of the transeunt triangle in a computational ap-

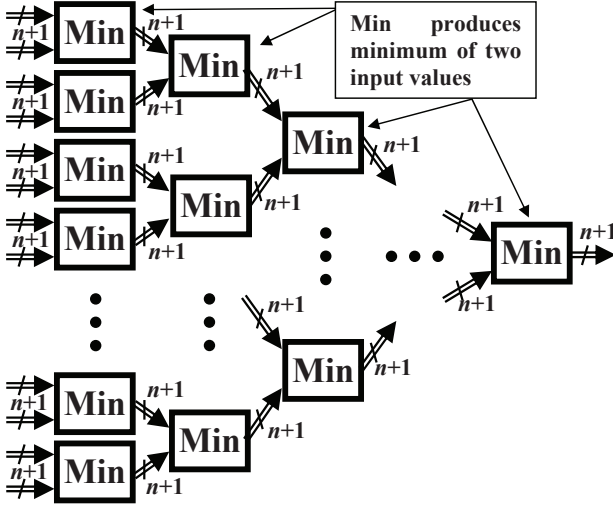


Figure 4. Minimum circuit

plication. Not only can the ANF be computed from the truth table, but the truth table can be computed from the ANF by using the same algorithm. This yields a significant computational advantage.

Definition 4.10. The **transeunt triangle*** is a set of $2^n - 1$ rows of adjacent 2-input 1-output exclusive-OR gates, where adjacent exclusive-OR gates connect to the same point. The input is a set of 2^n binary values that applies to the first (bottom) row of the triangle. That is, the inputs connect to a row of $2^n - 1$ adjacent exclusive-OR gates, whose outputs connect to a row of $2^n - 2$ adjacent exclusive-OR gates, etc.. The apex of the transeunt triangle is a row of just one exclusive-OR gate. The output of the transeunt triangle consists of the leftmost input bit and the outputs of the leftmost gates in each row. The inputs are indexed by the binary tuples $00 \dots 000$, $00 \dots 001$, $00 \dots 010$, ..., and $11 \dots 111$ from left to right. Similarly, the outputs are indexed from the lower left corner to the apex by the binary tuples $00 \dots 000$, $00 \dots 001$, $00 \dots 010$, ..., and $11 \dots 111$.

Example 4.9. Fig. 5a, shows the transeunt triangle for $n = 3$. In this case, there are eight inputs and eight outputs.

Table 1 shows an example of the output values for given input values. Specifically, if the input truth table representation, $TT = (01101001)$, which corresponds to the minterm canonical form $\bar{x}_1\bar{x}_2x_3 \vee \bar{x}_1x_2\bar{x}_3 \vee x_1\bar{x}_2\bar{x}_3 \vee x_1x_2x_3$, is applied to the bottom, then the left side of the transeunt triangle corresponds to the output ANF representation of this function, $ANF = (01101000)$, which is $x_3 \oplus x_2 \oplus x_1$. Conversely, Table 1 also shows that if the input ANF representation, $ANF = (01101001)$, is ap-

*Green [8] and others [3] define the transeunt triangle to be the logic values at the inputs and outputs of the 2-input 1-output exclusive-OR gates. We define it to be a circuit of exclusive-OR gates.

plied to the bottom, then the truth table representation of that function, $TT = (01101000)$, is produced on the left side of the transeunt triangle. (End of Example)

4.2 The Transeunt Triangle Proof

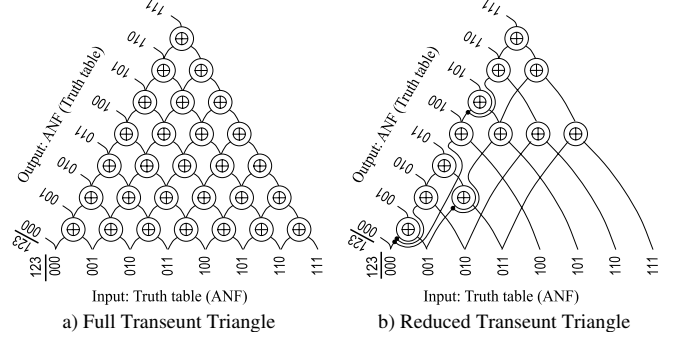


Figure 5. Comparing the full and reduced transeunt triangle for $n = 3$.

Green [8] did not prove that the transeunt triangle converts a truth table representation to an ANF representation. We do so now. The following result from [9, p. 68] will be used in our proof.

Theorem 4.2 (Lucas). Let p be a prime number, and two integers represented in base p , namely $n = n_s p^s + \dots + n_1 p^1 + n_0$ and $r = r_s p^s + \dots + r_1 p^1 + r_0$, with $0 \leq n_i, r_i \leq p - 1$. Then, $\binom{n}{r} \equiv \binom{n_s}{r_s} \dots \binom{n_1}{r_1} \binom{n_0}{r_0} \pmod{p}$.

The main result of this section is as follows.

Theorem 4.3. If the input to the transeunt triangle is the truth table representation of an n -variable function f , then the output is the ANF representation of f . Conversely, if the input to the transeunt triangle is the ANF representation of an n -variable function f , then the output is the truth table representation of f .

Proof: The second statement follows from the first because the logic values in the transeunt triangle are unchanged if all exclusive-OR gates are rotated 120 degrees clockwise (thus

Table 1. Example transeunt triangle in/out

$x_1 x_2 x_3 =$	000	001	010	011	100	101	110	111
Output ANF Repr. =	(0	1	1	0	1	0	0	0)
Output Expression	$x_3 \oplus x_2 \oplus x_1$							
Input TT Repr. =	(0	1	1	0	1	0	0	1)
Input Expression	$\bar{x}_1\bar{x}_2x_3 \vee \bar{x}_1x_2\bar{x}_3 \vee x_1\bar{x}_2\bar{x}_3 \vee x_1x_2x_3$							
Output TT Repr. =	(0	1	1	0	1	0	0	0)
Output Expression	$\bar{x}_1\bar{x}_2x_3 \vee \bar{x}_1x_2\bar{x}_3 \vee x_1\bar{x}_2\bar{x}_3$							
Input ANF Repr. =	(0	1	1	0	1	0	0	1)
Input Expression	$x_3 \oplus x_2 \oplus x_1 \oplus x_1x_2x_3$							

exchanging the input with the output). We prove the first statement by induction.

Fig. 6a shows that the first statement is true for all functions on $n = 1$ variable.

Assume the first statement is true for n , and consider an $n + 1$ -variable transeunt triangle. Fig. 6b shows that there are two n -variable transeunt triangles embedded in this transeunt triangle. Applied as an input to the lower one is $f_{0 \rightarrow x_1}$, shown as f_0 in Fig. 6b. By the inductive assumption, the output of this transeunt triangle is the ANF representation of $f_{0 \rightarrow x_1}$.

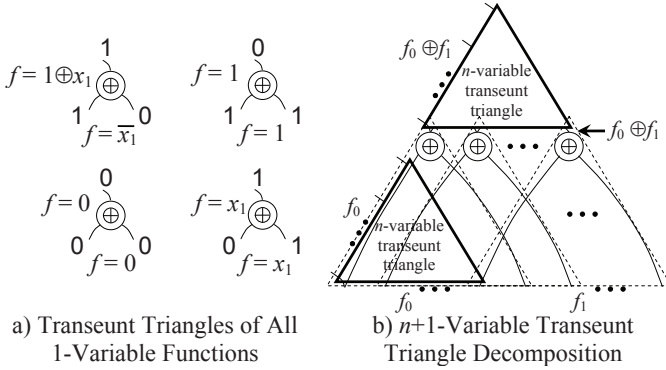


Figure 6. Transeunt triangle composition

We now show that $f_{0 \rightarrow x_1} \oplus f_{1 \rightarrow x_1}$ is applied as an input to the upper transeunt triangle. Let α be an assignment of values to x_2, x_3, \dots , and x_n . Then, each input to the upper triangle is driven by a $(2^{n-1} + 1) \times (2^{n-1} + 1) \times (2^{n-1} + 1)$ transeunt triangle whose inputs assignments range from 0α through 1α . This is shown in Fig. 6b as a dotted-line triangle. For example, the left input is driven by a transeunt triangle whose $2^{n-1} + 1$ inputs are $00 \dots 000, 00 \dots 001, \dots, 01 \dots 111$, and $10 \dots 000$ where $\alpha = 0 \dots 000$. Consider one triangle, and index its inputs by i , for $0 \leq i \leq 2^{n-1}$. The output of this triangle is the exclusive-OR of some number of its inputs. The number of times an assignment appears in the exclusive-OR expression of the inputs is the number of paths from that input to the output. This is just $\binom{2^{n-1}}{i}$. For $i = 0$ and $i = 2^{n-1}$, $\binom{2^{n-1}}{i} = 1$; i.e. there is exactly one path to the triangle's output, and these two inputs appear once in the exclusive-OR expression. Consider i , such that $0 < i < 2^{n-1}$. We use Theorem 4.2.

Since $n = 2^{k-1}$, $n_i = 0$ for all i , except that $n_{k-1} = 1$. For $0 < r < n = 2^{k-1}$, there is at least one j such that $\binom{n_j}{r_j} = \binom{0}{1} = 0$. Thus, for $0 < r < n = 2^{k-1}$, $\binom{2^{k-1}}{r} \equiv 0 \pmod{2}$. Thus, the number of paths from any assignment of values in the truth table input to the root is even. It follows that the only terms that occur are 0α and 1α . We can conclude, therefore, that the input to the upper transeunt triangle in Fig. 6b is the truth table representation of $f_{0 \rightarrow x_1} \oplus f_{1 \rightarrow x_1}$, shown in this figure as $f_0 \oplus f_1$.

By the inductive hypothesis, the output of the upper transeunt triangle is the ANF representation of $f_{0 \rightarrow x_1} \oplus f_{1 \rightarrow x_1}$. The input to the $n + 1$ -variable transeunt triangle in Fig. 6b is the truth table representation of $f_{0 \rightarrow x_1} \bar{x}_1 \vee f_{1 \rightarrow x_1} x_1$. The output is the ANF representation of $f_{0 \rightarrow x_1} \oplus (f_{0 \rightarrow x_1} \oplus f_{1 \rightarrow x_1})x_1$, which represents the same function. ■

4.3 Reduced Transeunt Triangle

We note that, in Fig. 6b, only one of the dotted-line triangles embeds a transeunt triangle (left dotted-line triangle). That is, all but one of these triangles can be replaced by a single 2-input 1-output exclusive-OR gate. Doing this yields the *reduced transeunt triangle*. Fig. 5b shows the reduced transeunt triangle for $n = 3$. In this case, only 12 2-input 1-output exclusive-OR gates are needed, compared to 28 gates for the full transeunt triangle.

Definition 4.11. A transeunt triangle is **balanced** if, for every output f , the path length to all inputs on which f depends is the same.

Example 4.10. Both the full and reduced transeunt triangles are balanced. A transeunt triangle in which all outputs are driven by a cascade of 2-input 1-output exclusive OR gates is not a balanced transeunt triangle.

Lemma 4.1. The full transeunt triangle for n -variable functions requires $(2^n - 1)2^{n-1}$ 2-input 1-output exclusive-OR gates, while the reduced transeunt triangle requires $n2^{n-1}$, which is the smallest possible among all balanced transeunt triangles using only 2-input 1-output exclusive-OR gates.

Proof: The number of gates in the full transeunt triangle is $f_n = 1 + 2 + 3 + \dots + 2^n - 1 = \frac{2^n(2^n - 1)}{2}$. The number of gates r_n in the reduced transeunt triangle is given by the recurrence relation $r_n = 2r_{n-1} + 2^{n-1}$, with initial condition $r_1 = 1$. Solving yields $r_n = n2^{n-1}$. The fact that this is the smallest possible can be seen as follows.

Order the inputs so that they are in lexicographical order, $00 \dots 00, 00 \dots 01, \dots$, and $11 \dots 11$, and construct a minimal balanced transeunt triangle so that the outputs are in lexicographical order. Each output bit indexed by $o_1 o_2 \dots o_n$ is the exclusive OR of all input bits indexed by i_1, i_2, \dots, i_n , such that $i_j \leq o_j$. For example, output bit $00 \dots 00$ is driven by input bit $00 \dots 00$, and no gate is needed. Output bit $00 \dots 01$ is driven by a gate with input bits $00 \dots 00$ and $00 \dots 01$, and one gate is needed. Specifically, each output bit is the root of a full binary tree, where the leaves are driven by input bits whose index is the same as the output node's index where some 1's may be changed to 0's. Input bit $00 \dots 00$ is in the binary tree of every output. Let $wt(o_1 o_2 \dots o_n)$ be the Hamming weight of $o_1 o_2 \dots o_n$ (the number of o_j 's that are 1). Consider the number of nodes

added to the transeunt triangle constructed so far by output bit $o_1 o_2 \dots o_n$. The fewest nodes added are those in the binary tree associated with output bit $o_1 o_2 \dots o_n$ that are along a path from the output bit $o_1 o_2 \dots o_n$ to the input bit $i_1 i_2 \dots i_n$, such that $i_j = o_j$, for all $1 \leq j \leq n$. None of these nodes are part of the transeunt triangle constructed so far. Because the transeunt triangle is balanced, there are $wt(o_1 o_2 \dots o_n)$ added nodes. Because of the lexicographical order of the inputs, all arcs from these nodes must go toward that part of the transeunt triangle constructed so far.

It follows that the number of gates in a balanced transeunt triangle is bounded below by the total number of 1's among all binary n -tuples, which is $n2^{n-1}$. \blacksquare

In addition, the reduced transeunt triangle yields smaller delay than the full transeunt triangle. It is straightforward to show the following.

Lemma 4.2. *The full transeunt triangle for n -variable functions requires $2^n - 1$ gate delays, while the reduced transeunt triangle requires n gate delays, where one gate delay is the delay associated with a 2-input 1-output exclusive-OR gate.*

Since the full and reduced transeunt triangles are balanced, the delay to an output from any of the inputs is identical. We can specify the gate delays for all outputs of the reduced transeunt triangle, as follows.

Definition 4.12. *Let i and j be non-negative integers. $i \Rightarrow j$ if and only if the binary representation of j has a 1 everywhere there is a 1 in the binary representation of i .*

The next result appears in [4].

Lemma 4.3. *Let $f(x_1, x_2, \dots, x_n)$ be a function. Let t_i be the i -th truth table entry for f , and let a_j be the j -th ANF entry for f . Then, $a_j = \sum_{k \Rightarrow j} t_k$, where \sum is modulo 2 sum. Conversely, $t_i = \sum_{k \Rightarrow i} a_k$.*

It follows from Lemma 4.3 that the outputs of the transeunt triangle are just the exclusive OR of all inputs whose indices are subsumed by the index of the output. For example, from Fig. 5b the output 101 is the exclusive OR of the inputs 000, 001, 100, and 101. From this, it can be seen that each output depends on $2^{\text{wgt}(i)}$ inputs, where i is the index of the output and $\text{wgt}(i)$ is the number of 1's in the binary representation of i . The reduced transeunt triangle realizes each output as a balanced binary tree of exclusive OR gates.

5 Experimental Results

5.1 Speed-up Achievable by the Reconfigurable Computer

We compare the computation time required by an SRC-6 reconfigurable computer with the time required by a conven-

tional computer. In our case, this is an Intel Xeon processor running at 2.8 GHz., which is one of two conventional microprocessors associated with the SRC-6. The program, written in C, computes the nonlinearity of n -variable functions, forming the distribution of functions to nonlinearity. Similarly, the time it takes to do the exact same calculation on the SRC-6 can be calculated since the throughput is one function per clock period. The results are shown in Table 2.

Table 2. Speed-up obtained by the SRC-6 reconfigurable computer

n	PC Compute Time (@2.8 GHz.)	SRC-6 Compute Time (@100 MHz)	Speed-up Factor
2	6.38 $\mu\text{sec.}$	0.16 $\mu\text{sec.}$	$39.9 \times$
3	457.0 $\mu\text{sec.}$	2.56 $\mu\text{sec.}$	$178.5 \times$
4	0.388 sec.	655.4 $\mu\text{sec.}$	$592.0 \times$
5	25.338 hours	42.9 sec.	$2,126.3 \times$
6	39,807,788 years	5,840 years	$6,805.9 \times$
7	2.05×10^{27} years	1.08×10^{23} years	$19,005 \times$
8	2.28×10^{66} years	3.67×10^{61} years	$62,111 \times$

Speed-up factors range from $39.9 \times$ for $n = 2$ to $62,111 \times$ for $n = 8$. Note that the speed-up factor should increase by approximately 4 for each increase in n by 1. On the PC, the computation time doubles for each increase in n because the number of affine functions doubles. Similarly, the number of Ones_Count operations also doubles. However, on the SRC-6, the circuit size increases; the throughput of one function per clock cycle remains the same. The computation times for $2 \leq n \leq 5$ shown in Table 2 were achieved by programs on the PC and on the SRC-6 that enumerated all 2^{2^n} n -variable functions. The computation times for $6 \leq n \leq 8$ for the PC were obtained by running the C program over a fraction of the functions and then prorating to compute the time had all functions been enumerated. Although the computation time on the SRC-6 for these values of n is much less, it is still excessive, and this computation could not be done. However, the speed-up applies when we enumerate a sufficiently small subset of all functions. For example, we enumerated all 6-variable functions with degree 3 or less and, in so doing, enumerated all bent functions [15] using the theorem by Rothaus [14]. As discussed in Section 3, this computation required 5.7 minutes. Had this computation been done on the PC, it would have taken $6805.9 \times$ longer or 27 days. We achieved the $62,111 \times$ speed-up associated with $n = 8$ in computing the distribution of rotation symmetric functions, as described in Section 5.4.

5.2 Number of 6-Variable Bent Functions

The computation described in the previous section verified Preneel's [12] result that there are 5,425,430,528 bent

functions on 6 variables. We showed further, that 1,777,664 of these functions or 0.03% have degree 2. All of the remaining have degree 3.

5.3 Nonlinearity of 6-Variable Homogeneous Boolean Functions

Although it is not possible to enumerate all 6-variable functions, it is possible to enumerate all homogeneous 6-variable functions. There are $\sum_{k=0}^6 (2^{\binom{6}{k}} - 1) = 1,114,237$ 6-variable homogeneous functions. Fig. 7 shows the distribution of 6-variable homogeneous functions to nonlinearity and degree. The vertical axis shows the \log_2 number of functions. For example, there are 63 homogeneous functions of nonlinearity 0 and degree 1; these are the linear functions. A vertical bar of length 63 extends just above the tick mark labeled 32 because of compression due to the \log_2 scale. There is one function, $f = 1$, of nonlinearity 0 and degree 0. This is not visible in Fig. 7. Nor is the only function that has degree 6 and nonlinearity 1 ($f = x_1x_2x_3x_4x_5x_6$). The bent functions have nonlinearity 28, and Fig. 7 shows there are two different degrees. 13,888 have degree 2 and 30 have degree 3. The next largest nonlinearity is 23, and again, functions exist with only degrees 2 and 3. For degrees 3, 4, and 5, there is bell-like distribution across nonlinearity.

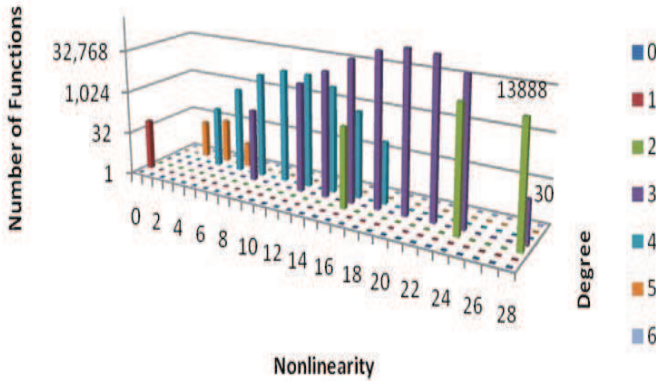


Figure 7. Distribution of homogeneous 6-variable functions by nonlinearity and degree

Table 3 shows the resource usage on the Xilinx Virtex2 Pro.

Table 3. Resources used to compute the nonlinearity of 6-variable homogeneous functions

Number of	Number/Total	Percentage
Slice Flip-Flops	7,959/88,192	9%
4-Input LUTs	12,335/88,192	13%
Occupied Slices	8,724/44,096	19%

5.4 Nonlinearity of 8-Variable Rotation Symmetric Boolean Functions

Definition 5.13. A function f is rotation symmetric if and only if for any $(x_1, x_2, \dots, x_n) \in \mathbb{F}_2^n$

$$f(x_1, x_2, x_3, \dots, x_n) = f(x_n, x_1, x_2, \dots, x_{n-1}).$$

In a rotation symmetric function, “rotating” an assignment of values to the variables leaves the function unchanged. A special case of rotation symmetric functions are the totally symmetric functions, in which *any* permutation of the variables leaves the function unchanged. Rotation symmetric functions have interesting properties [5] and there is evidence to suggest that their class may contain bent functions. It is conjectured [6] that the weight and nonlinearity of a third degree homogeneous rotation symmetric function are identical.

Fig. 8 shows the distribution of 8-variable rotation symmetric functions to nonlinearity. This shows that more rotation symmetric functions have nonlinearity around 110 than other values. Relatively few have low nonlinearity (0 – 75) or high nonlinearity (> 113). This distribution resembles the distribution of nonlinearity to all functions, which is known only for $n = 4$ [1].

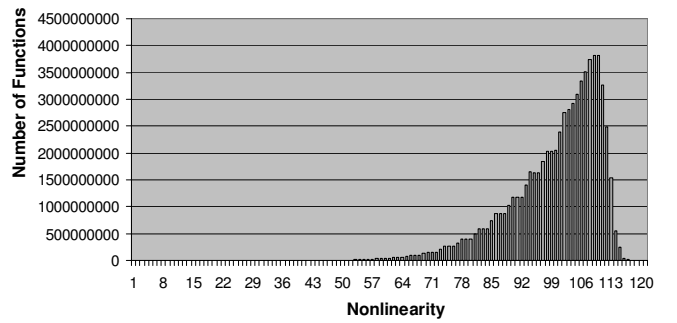


Figure 8. Distribution of 8-variable rotation symmetric functions by nonlinearity

Table 4 shows the resource usage on the Xilinx Virtex2 Pro.

Table 4. Resources used to compute the nonlinearity of 8-variable rotation symmetric functions

Number of	Number/Total	Percentage
Slice Flip-Flops	9,531/88,192	10%
4-Input LUTs	8,850/88,192	10%
Occupied Slices	8,540/44,096	19%

6 Concluding Remarks

Our use of the reconfigurable computer focused on a scientific application. We show that the reconfigurable com-

puter is an effective research tool in bent function discovery. Because we adapt the architecture to the problem, we achieve significant efficiencies. Indeed, we show that a reconfigurable computer can achieve better than a $60,000\times$ speed-up over a conventional computer for 8-variable functions. The implementation of the transeunt triangle is beneficial in reducing the number of functions through which we must sieve. We show that the reduction is better than 1,000,000 to 1 for 6-variable functions. Although the transformation produced by the transeunt triangle is generally accepted as correct, no proof is known. We provide such a proof. This proof yields the reduced transeunt triangle, which produces the identical transformation of the full transeunt triangle, but with significantly fewer gates and less delay. We show examples of results obtained from this tool. For other results, see Schneider [15] and Shafer [16].

Nonlinearity is only one type of cryptographic property. Other types include strict avalanche criterion, propagation criterion, correlation immunity, and algebraic immunity. There is significant promise in exploiting the efficiencies of a reconfigurable computer to make new discoveries in these important topics.

7 Acknowledgments

We thank 1) the National Security Agency, which provided support for this project, 2) Daniel P. Zulaica, who provided technical support for the SRC-6, 3) Christopher D. Johnson, who provided comments that improved this paper, and 4) four reviewers, who provided suggestions for improvement.

References

- [1] J. T. Butler and T. Sasao, "Bent functions and their relation to switching circuit theory – A tutorial", *Proc. of the Reed-Muller Workshop 2009*, May 23-24, 2009, Naha, Okinawa, Japan, 127–136.
- [2] J. T. Butler, G. W. Dueck, S. N. Yanushkevich, and V. P. Shmerko, "On the use of transeunt triangles to synthesize fixed-polarity Reed-Muller expansions of symmetric functions", *Proc. of the Reed-Muller Workshop 2009*, May 23-24, 2009, Naha, Okinawa, Japan, 119–126.
- [3] J. T. Butler, G. W. Dueck, S. N. Yanushkevich, and V. P. Shmerko, "On the number of generators for transeunt triangles", *Discrete Applied Math.*, Vol. 108, Issue 3 (2001), 309–316.
- [4] A. Canteaut and M. Videau, "Symmetric Boolean functions", *IEEE Trans. Inf. Theory*, Vol. 51, no. 8 (2005), 2791–2811.
- [5] T. W. Cusick and P. Stănică, *Cryptographic Boolean Functions and Applications*, Academic Press, San Diego, CA, 2009.
- [6] T. W. Cusick and P. Stănică, "Fast evaluation, weight, and nonlinearity of rotation-symmetric functions", *Discrete Math.*, Vol. 258 (2002), 289–301.
- [7] G. W. Dueck, D. Maslov, J. T. Butler, V. P. Shmerko, and S. N. Yanushkevich, "A method to find the best mixed polarity Reed-Muller expression using transeunt triangle", *Proc. of the 5th Int. Reed-Muller Workshop (RM'2001)*, Starkville, MA, USA, 82–92, August 2001.
- [8] D. H. Green, *Modern Logic Design*, Addison-Wesley Publishing Company, 1986.
- [9] D. H. Knuth, *The Art of Computer Programming*, 2nd Ed., Addison-Wesley Publishing Co., Reading, Menlo Park, London, Amsterdam, Don Mills, Sydney, 1973.
- [10] Q. Meng, H. Zhang, M. Yang, and J. Cui, "On the degree of homogeneous bent functions", *Discrete Applied Math.*, Vol. 155, Issue 55 (2006), 665–669.
- [11] J. Olsen, R. Scholtz, and L. Welch, "Bent-Function Sequences", *IEEE Trans. Inf. Theory* Vol. 28, No. 6, (1982), 858-864.
- [12] B. Preneel, *Analysis and Design of Cryptographic Hash Functions*, Ph.D. Thesis, Katholieke Universiteit Leuven, K. Mercierlaan 94, 3001 Leuven, Belgium, 1993.
- [13] C. Qu, J. Seberry, and J. Pieprzyk, "Homogeneous bent functions", *Discrete Applied Math.*, Vol. 102 (2000), 133–139.
- [14] O. S. Rothaus, "On 'bent' functions", *J. Combinatorial Theory, Ser. A*, Vol. 20 (1976), 300–305.
- [15] S. W. Schneider, "Finding bent functions using genetic algorithms", M.S. Thesis, Naval Postgraduate School, Monterey, CA, September 2009.
- [16] J. L. Shafer, "An analysis of bent function properties using the transeunt triangle and the SRC-6 reconfigurable computer", M.S. Thesis, Naval Postgraduate School, Monterey, CA, September 2009.
- [17] C. E. Shannon, "Communication theory of secrecy systems", *Bell System Tech. J.* 28 (1949), 656–715.
- [18] T. Xia, J. Seberry, J. Pieprzyk, and C. Charnes, "Homogeneous bent functions of degree n in $2n$ variables do not exist for $n > 3$ ", *Discrete Applied Math.*, Vol. 142 (2004), 127–132.