Efficient Exhaustive Search for Optimal-Peak-Sidelobe Binary Codes

GREG COXSON
Lockheed Martin Maritime Systems & Sensors

JON RUSSO
Lockheed Martin Advanced Technology Laboratories

The work presented here describes an exhaustive search for minimum peak sidelobe level (PSL) binary codes, combining several devices for efficiency. These include combinatoric tree search techniques, the use of PSL-preserving symmetries to reduce search space, data representations and operations for fast sidelobe computation, and a partitioning scheme for parallelism. All balanced 64-bit minimum PSL codes are presented, and the upper limit on known consecutive lengths to have PSL = 4 **codes is extended to 70. In addition, a technique for determining balance properties of a code for given PSL-preserving transformations is developed.**

Manuscript received February 14, 2004; revised August 6 and August 26, 2004; released for publication September 16, 2004.

IEEE Log No. T-AES/41/1/841946.

Refereeing of this contribution was handled by E. S. Chornoboy.

Authors' addresses: G. Coxson, Mail stop 137-228, Lockheed Martin Maritime Systems & Sensors, 199 Borton Landing Road, Moorestown, NJ 08057, E-mail: (gregory.e.coxson@lmco.com); J. Russo, Lockheed Martin Advanced Technology Laboratories, 3 Executive Campus, 6th floor, Cherry Hill, NJ 08002.

0018-9251/05/$17.00 © 2005 IEEE

INTRODUCTION

Binary phase codes with low autocorrelation sidelobe levels are useful in such diverse applications as radar pulse compression [11], communication systems [10], and theoretical physics ([6, 8]). Knowledge of minimum achievable sidelobe levels for given code lengths can help inform searches for low-sidelobe-level codes. Sets of codes that achieve the minima can provide new dimensions to trade studies supporting code selection tasks.

This paper adds to available knowledge for the particular case of aperiodic autocorrelation function (ACF) peak sidelobe level (PSL). In particular, it is aimed at continuing in the tradition of Lindner [4], Cohen et al. [3], and Coxson et al. [1], in advancing the upper bound on code lengths for which the optimal PSL is known. It also corrects errors in Coxson et al. [1], which attempted to reach 69 for optimal PSL values; while the minimum PSLs listed up to $N = 69$ were correct, the sample codes listed were optimal only for consecutive lengths up to $N = 60$. In addition it provides a full set of balanced optimal-PSL codes for length $N = 64$. Each of these tasks involve significant computational challenge. Militzer et al. [9] liken the search for minimum-sidelobe-level codes of the aperiodic ACF to that of searching for needles in a haystack.

This paper extends the length for best known minimum PSL codes to $N = 70$, and enumerates the balanced minimum-PSL codes of order $N = 64$, a length important for radar and communications applications. The search algorithm developed for these tasks combines five devices to achieve performance:

1) a back-tracking branch-and-bound search strategy;
2) search logic that reduces search space by leveraging PSL-preserving symmetries;
3) a fast method for computing the aperiodic autocorrelation of binary sequences;
4) additional optimizations made possible by symmetry constraints;
5) an effective partitioning and parallelization of the search.

A description of the algorithm is included with attention to implementation details.

The full search of order $N = 64$ yielded of 1859 optimal-PSL codes, not including redundancies due to PSL-preserving operations, or "PSL preservers." The optimal PSL for $N = 64$ is 4. This corresponds to a peak-sidelobe-to-peak ratio 3 dB better than that expected for pseudo-random noise (PRN) codes of the same length. In addition, the search turned up an optimal-PSL code with relatively high merit factor of 9.8.

A subset of the optimal-PSL codes of length 64 also have the balance property, meaning that they have equal numbers of 1 and -1 elements. An equivalence relation can be defined between pairs of codes by whether or not one can be mapped to the other by combinations of PSL preservers, in which case the two are "balance-equivalent." The search found 142 such equivalence classes for $N = 64$. A list of representatives of each class is given in Table I.

For the code lengths 61–63 and 65–70, incomplete searches were performed to establish PSL = 4 as the optimal PSL in each case. This was done in each case by running the routine only up to the point where a PSL-4 code was found, since it is known that $N = 51$ is the greatest length for which a code with PSL = 3 exists (see, e.g., [5]).

MATHEMATICAL PRELIMINARIES

Binary codes x of length N are represented by vectors $x = (x_1, x_2, \ldots, x_N)$ for which $x_i = \pm 1$ for $i = 1, \ldots, N$.

The autocorrelation of x is

$$\mathrm{ACF} = x * \bar{x}$$

where $\bar{x}$ is the reversal, or left-right transposition, of x and * represents the usual convolution operation for sequences. Aperiodic autocorrelation is assumed. The reversal is defined element by element by

$$\bar{x}_i = x_{N+1-i}$$

for $i = 1, 2, \ldots, N$. The kth element of the autocorrelation may be written

$$\mathrm{ACF}(k) = \sum_{i=1}^{k} x_i x_{N+i-k}$$

for $k = 1, \ldots, 2N-1$, where it is understood that $x_i = 0$ for $i > N$. The peak of the autocorrelation is $\mathrm{ACF}(N) = N$; the elements of ACF for k not equal to N are called sidelobes, and $\mathrm{ACF}(k)$ is the kth sidelobe for $k = 1, \ldots, 2N-1$. The autocorrelation is symmetric with respect to its center element, or peak.

The imbalance λ of x is the number of elements equal to 1 minus the number of elements equal to -1. It is easy to derive the relationship

$$\lambda = \sum_{i=1}^{N} x_i.$$

A code having $\lambda = 0$ is said to be balanced; otherwise it is unbalanced.

An alternative sidelobe measure is the rms of the sidelobes, which is computed by the ratio of the rms of $\mathrm{ACF}(k)$ for k not equal to N to the autocorrelation peak N. RMS is proportionally related to the integrated sidelobe level (ISL) and to the merit factor (MF) of a code, an increasingly popular measure of sidelobe level which defined as

$$\mathrm{MF} = \frac{N^2}{2\sum_{k=1}^{N-1}(\mathrm{ACF}(k))^2}$$

(see [2], [7], [8]). Note that larger values of MF are associated with lower sidelobe levels. For certain code classes of interest it has a finite non-zero asymptotic value as code length grows. For example, the asymptotic value for PRN codes is 3 [2].

There are three operations on binary codes that preserve both PSL and rms. These operations are referred to as PSL-preserving operations, or PSL preservers. The three operations are negation of x to get $-x$, reversal of x to get $\bar{x}$, and negation of every other element of x, or xA, where A is an $N \times N$ diagonal matrix with diagonal entries $(-1)^i$ for $i = 1, \ldots, N$ (see [1]). For any given length N, these operations induce a partition of the total search space into equivalence classes. Two codes are equivalent if one can be found from the other by applying some combination of these operations. Furthermore, the entire set of codes can be represented by a subset of roughly one-eighth the size, containing a single code from each equivalence class. To find all the optimal-sidelobe level codes of length N, it suffices to search over this smaller subset.

These PSL-preservers don't necessarily preserve imbalance. To be specific, let λ_e and λ_o represent the imbalance of the elements of code x with, respectively, even and odd indices, i.e.,

$$\lambda_e = \sum_{k=1}^{\lceil N/2 \rceil} x_{2k}$$

and

$$\lambda_o = \sum_{k=1}^{\lceil N/2 \rceil} x_{2k-1}$$

where $\lceil N/2 \rceil$ involves the ceiling operation, giving the closest integer greater than or equal to $N/2$. Starting from a code x with imbalance $\lambda = \lambda_e + \lambda_o$ and applying combinations of these operations results in codes with imbalance values

$$\pm\lambda_e \pm \lambda_o.$$

Hence, this set of codes, which is an equivalence class relative to the PSL-preserving operations, will contain a balanced code if and only if

$$|\lambda_e| = |\lambda_o|.$$

If it does contain a balanced code, all codes in the equivalence class will be deemed "balance-equivalent."

EXHAUSTIVE SEARCH METHODOLOGY

The search method works with codes and partial codes having elements 0 and +1, rather than +1 and −1. The results translate directly to codes with +1 and −1 elements by replacing 0 s with −1 s and leaving +1 s unchanged.

The approach is a depth-first search which progresses from the extreme elements of the code

TABLE I
Representatives of All Optimal-PSL Balance-Equivalent Codes for $N = 64$

Code in Hexadecimal	λ_e	λ_o	Merit Factor	Code in Hexadecimal	λ_e	λ_o	Merit Factor
514d38913efc37a1	+4	−4	6.5641	512826bde83e71b3	−2	+2	4.9231
204edb4b9dc1ea8e	−4	+4	6.4000	260d09bcdea1df54	+2	−2	4.9231
0ff28a258b967672	−4	+4	6.2439	2101f3d2ee2ed1ae	−4	+4	4.9231
70aa8f009f6b5b33	−4	+4	6.2439	7bc0e7aae44c134b	−2	+2	4.9231
0e7066a149f6fa9a	−4	+4	6.2439	10cd2d6447c5cabf	+4	−4	4.9231
3dfce0760a654d52	+4	−4	6.0952	57074924ec433af7	+4	−4	4.9231
47c156ac890dbce7	+2	−2	5.9535	1bc80695e3d7564e	+4	−4	4.9231
4c6b0578128faf9b	−2	+2	5.9535	72a0623af0d6db27	−4	+4	4.9231
1558e072d219b37f	+4	−4	5.8182	4f69e1d9c4a902ee	−2	+2	4.9231
55006db92f33d38d	+4	−4	5.8182	100f559b23f3969e	+2	−2	4.9231
1118eed9a1fa1a7a	−4	+4	5.8182	333c367fa1942aca	−4	+4	4.9231
364c661e12d0abfe	−2	+2	5.8182	40a5a7544cf70fb3	+4	−4	4.8302
15ea86b0b0336df3	−2	+2	5.8182	40c5cb965586f27b	+4	−4	4.8302
763250ea83a469fe	−4	+4	5.8182	55ff84b069386665	+4	−4	4.8302
13fe6b48f0e25532	+0	+0	5.6889	6ea2dcded0270e13	−2	+2	4.8302
08f5834b6fcba8cc	−4	+4	5.6889	520a47b1bb28f3e5	−2	+2	4.8302
00dccdb2d6ba870f	−2	+2	5.6889	6ad664c03e30daeb	−4	+4	4.8302
155ad600f30d9f9b	+4	−4	5.6889	1496aac489f0cfe7	−2	+2	4.8302
33cd9c7a3ea6b402	−4	+4	5.6889	00b5acd5d661e4cf	+4	−4	4.8302
24213bd55f30f27c	+4	−4	5.6889	1e8d4cc935047beb	+2	−2	4.7407
754ed896d808c7e3	+2	−2	5.6889	5ecee91a4516f1c1	+4	−4	4.7407
3334a47a0175df3c	+4	−4	5.5652	388f4aa025f367e6	−4	+4	4.7407
0849cf72acbe48fc	−4	+4	5.5652	6450537ae317e876	+4	−4	4.7407
06f98e9bb9e8d052	−4	+4	5.5652	576dd2310d2be307	+4	−4	4.7407
5050ce7a542fe4db	+4	−4	5.5652	4db007aaf1e96333	−2	+2	4.7407
4c279a0789397bab	−4	+4	5.5652	2d3d5081dcc7d9b2	+4	−4	4.7407
0c75f6d2a5f02ccc	+2	−2	5.4468	7e6ed5261a29ec21	−2	+2	4.7407
02a8f34e969e466f	−4	+4	5.4468	7a022b51de34d979	+2	−2	4.7407
39c00b5978bcbb66	−2	+2	5.4468	43fbcae3943a4991	−2	+2	4.6545
64f8b015c67d62d6	+4	−4	5.4468	0571986d89bd7af0	+2	−2	4.6545
0287ceccb6a3dc53	−2	+2	5.4468	0c5463eee16c9bd2	+0	+0	4.6545
4940d93f543af339	+4	−4	5.4468	055225b93b0f6f3c	+2	−2	4.6545
34923c077778d1ab	+2	−2	5.4468	5a2f4bbac71d9809	−2	+2	4.6545
082dace2e57cc9f2	−4	+4	5.3333	0ffe338322d51696	+0	+0	4.6545
5ad1024d758eee0f	+2	−2	5.3333	514c368d1b3bf90b	+2	−2	4.6545
003e2e2d6ae734db	−4	+4	5.3333	1b0765b56bbb810e	+0	+0	4.6545
50bd0e28ccd24bfb	−2	+2	5.3333	7bee28b448b42763	−4	+4	4.6545
2220ddd6c1f4e35e	+2	−2	5.3333	726520f0afb50ceb	−2	+2	4.6545
2a816c4bd98c3cfe	−4	+4	5.3333	56d47a3873024df6	+4	−4	4.6545
30f184f2154bfbb2	+0	+0	5.2245	746f8ae6012bd34b	−2	+2	4.5714
3b9dd3a740578692	+2	−2	5.2245	1312fdf507962b86	+2	−2	4.5714
5ab777413078c279	+4	−4	5.2245	7367e1945fa294a1	+2	−2	4.5714
007f15c6a6d3a72e	+0	+0	5.2245	645492b5fb078e63	+2	−2	4.5714
44157a9873697e1b	+4	−4	5.2245	104ae998e9c2f4bf	−4	+4	4.5714
79b2ce4f4ac6a807	−4	+4	5.2245	4023f0dd38e9adab	−4	+4	4.5714
72ab8d62d26240ff	−4	+4	5.2245	01ba886c9f9e795a	−4	+4	4.4912
6ea26d2970c41b9f	−2	+2	5.2245	0e83ed52dfd30c4c	+2	−2	4.4912
3ab5c6484a3ec87e	−4	+4	5.2245	33690baa1461fcfc	−2	+2	4.4912
203433f638dda6ab	−4	+4	5.2245	036650f63950eb6f	+2	−2	4.4912
150ac1e72da26fec	−2	+2	5.1200	458f9358135a98ef	+2	−2	4.4138
454233fe25cb4f0d	+4	−4	5.1200	463296aa03c7ec9f	−4	+4	4.4138
42817f333cb2ae2d	−4	+4	5.1200	3a2ffc4c964a51c6	+0	+0	4.4138
057918e1c8b97db6	+2	−2	5.1200	040eeae35ac6d92f	−4	+4	4.4138
20c33d5c5a7f44ec	+4	−4	5.1200	55fdc98634962f05	+6	−6	4.4138
044af5974c6f2c2f	+2	−2	5.1200	79293813aa30cdf7	−2	+2	4.4138
49a63152971707fe	+4	−4	5.1200	03c22f5e29af6653	−2	+2	4.4138
60f133c8529a2bf7	−2	+2	5.1200	149aa72ee7a7c01e	−4	+4	4.3390
245835717b0b3de6	+4	−4	5.1200	12407cd1ed58aee7	+2	−2	4.3390
49bdf6253b07382a	−2	+2	5.1200	602e22fe70d9ae56	−4	+4	4.3390
0186f24dd8eac6af	−4	+4	5.1200	02faf55b4c723670	+2	−2	4.3390
1891c9b6a6a1dfe2	−4	+4	5.1200	59c6d86a03b706af	−2	+2	4.3390
2e892bd1098df71e	−2	+2	5.1200	154a7a0273e3edb2	−2	+2	4.2667
10d6e82b876bb363	−4	+4	5.0196	53adaca8f7cc8381	−4	+4	4.2667
03b068f3a8a35b7b	−6	+6	5.0196	7f9e251748696227	+2	−2	4.2667
54d07a253b0873ef	+2	−2	5.0196	1153d2b841ecd33f	+4	−4	4.2667
081de9c4f29dd176	+4	−4	5.0196	41bb058c58debe9a	−2	+2	4.2667
4057986b459b1f9b	+4	−4	5.0196	40c5db94dca6be43	+2	−2	4.2667
3c63f7e1724948aa	−2	+2	5.0196	27da3fab8118b634	−4	+4	4.1967
6060f6de22f519d6	+2	−2	5.0196	72d535dfa23602c3	+2	−2	4.1290
6f827cf09052acee	−4	+4	5.0196	1114a9ef893cbe1b	−2	+2	4.0635
56afaec909bc0ce1	−4	+4	4.9231	3ad79ea600dc4d36	+0	+0	3.6571

inward. At the top level, some equal number of elements at the left end and right end of the code are set by the user. At each successive level, the two symmetrically placed elements which are one position closer to the middle of the code are set. The idea is to exploit the property of the ACF that sidelobes on either end depend on elements near the ends of the code. If a given sidelobe depends on element i of an N-length code then it also depends on element $N + 1 - i$. Hence, element choices are made in pairs. With each pair of new element selections (with the exception of the last), one more sidelobe on each side of the ACF can be computed and tested. Since the autocorrelation sequence is symmetric with respect to the peak, the search needs only to check the sidelobes on one side of the sequence. Hence, at each level, with the exception of the last, one sidelobe is computed and tested. If it surpasses a specified maximum value (e.g., the optimal PSL, if only the optimal codes are desired), then the search retreats to the previous level.

The process is refined to take into account redundancies with respect to the three PSL-preserving operations. The idea is to search over a reduced search space including only a single code from each equivalence class (two codes are equivalent if one can be reached from the other by some combination of the PSL-preserving operations). For example, in the selection of elements 1 and N, redundancy due to the negation operation is avoided by allowing only a setting of 0 for the first element. The alternating-sign operation is accounted for by setting the last element to 0 as well. This can be seen as follows. Let w_1 and w_2 be unknown bits, with $y_1 = -w_1$ and $y_2 = -w_2$. Recall the alternating-element operation xA detailed earlier, and observe that

$$-(0\ w_1\ w_2\ 0) = (1\ y_1\ y_2\ 1)$$
$$(0\ w_1\ w_2\ 0)A = (0\ w_1\ y_2\ 1)$$
$$-(0\ w_1\ w_2\ 0)A = (0\ y_1\ w_2\ 1).$$

Hence, an exhaustive search over the inner two elements with the two outer elements set to 0 achieves a full exhaustive search of all length-4 binary codes. Note that this reduced search would still perform almost twice the necessary checks, since the reversal operation has not been accounted for. This final symmetry is handled in the search logic, as explained later. The essential idea is that at any given point in the search, the code can be represented by a partial right string r and partial left string l of equal length. No code will be considered if its bit-reversed right string r represents a decimal number greater than the decimal number represented by the left string l. This can be anticipated in selecting the top-level branches to prosecute in the search. For example, if two elements instead of one are set on each side at the top search level, then only three of 16 branches need be checked; they are (0 1 … 0 0), (0 1 … 1 0) and (0 0 … 0 0).

The possibilities for parallelizing the search are evident. Each choice for code elements to be set at the top search level yields an identifiable list of branching choices. Each choice may be assigned to a separate processor. Furthermore, experience with this process has shown that some branches tend to produce larger numbers of promising sub-branches, which means longer processing times. These more fruitful branches can be assigned to faster processors.

It is the nature of the autocorrelation sequence that a number of sidelobes (those near the center of the sequence) are not specified until every element of the code is filled in. This means that at the lowest level of the search, several sidelobes are computed and tested.

The device responsible for the greatest efficiency is the use of bit operations for the sidelobe calculations. The program uses unsigned integer registers containing numbers representing binary codes. A register shift-and-XNOR operation performs all multiplies required for a single element of the ACF, or sidelobe. XNOR is an abbreviation for exclusive NOR. The summation of the multiplies for a single sidelobe is found by consulting a look-up table in which the bit count is given for any binary number that results from the process. The computer program is designed to be compiled with full optimization, using loop-unrolling if needed.

The look-up table contains the bit count, or number of bits equal to 1, for the integers from 0 up to a maximum value set for the code length, to facilitate fast calculation of the bit imbalance of any binary number of that length. The table is premultiplied by 2, eliminating one of the operations needed to compute the imbalance. The bit counts (times 2) of the upper and lower bits are looked up, summed, and then the total number of bits is subtracted to give the final imbalance value. The imbalance values serve two roles: screening codes for a particular imbalance, and computing sidelobes.

For an example of how imbalance can be used to compute the sidelobes, consider the binary code $(1\ 1\ -1\ 1)$, which is translated into the binary 0-1 code (1 1 0 1). This is a code of length 4 having autocorrelation sidelobes of index 1, 2, and 3 equal to, respectively, 1, 0, and -1. Sidelobe 1 is computed by the XNOR of (1 1 0 1) with itself shifted $N - 1$, or in this case 3, spaces to the right, which gives (1). The imbalance of this result sequence is 1, which is taken to be the value for the first sidelobe. For the index-2 sidelobe, the XNOR of (1 1 0 1) with itself shifted $N - 2$, or 2, spaces to the right gives (0 1), having an imbalance of 0, and this equates to a sidelobe of 0. Finally, for the index-3 sidelobe, the XNOR of (1 1 0 1) with itself shifted one unit to the right is (1 0 0) which has an imbalance of -1.

Searches were run on four 750 MHz Sun UltraSPARC-III workstations. They were run using full 64-bit mode, which resulted in performance improvement on the order of a factor of 4 over 32-bit mode.

SEARCH ROUTINE

Consider the following bit-string operations on strings of 0 s and 1 s. Some of the operations are designed to be applied to strings which include a partial left string and right half-string of equal length; in such cases the partial strings will be denoted by **l** and **r**, respectively.

RCAT(**x**): Increases length of bit string **x** by 1, by adding a 0 as a new rightmost string element.
LCAT(**x**): Increases length of bit string **x** by 1, by adding a 0 as a new leftmost string element.
CORR(**l**, **r**, thresh): Early-termination correlation calculation. Outputs a 1 if all sidelobes of partial autocorrelation for partial left string **l** and partial right string **r** are no larger than thresh; otherwise, outputs a 0.
INC(**l**, **r**): Increments to the next value-pair in {'00', '01', '10', '11'} the middle two elements of the string formed from the current left half-string **l** and right half-string **r**.
XNOR(**l**, **r**): Bit-wise XNOR operation on two 0-1 strings **l** and **r** of equal length.
BAL(**x**): Balance of bit string **x**, computed as the number of 1 s in **x** minus the number of 0 s.
LEN(**x**): Returns the length of bit string **x**.
REV(**x**): Returns the reversal of bit string **x**.
All operators operate only on the length of the bit string.

The increment operator (INC) is fairly involved. In the case where the middle pair of bits of [**l**, **r**] are both 1, a carry operation is performed. It operates as a retreat, decreasing the sizes of **l** and **r** until a pair is found which can be incremented. In addition, the increment operator eliminates reversal symmetry redundancies. It does this by only pursuing increment operations for which the resulting left and right partial strings have the property that the numerical equivalent of the binary left string is less than or equal to the numerical equivalent of the bit-reversed binary right string.

The table lookup described to in the previous section is handled by the BAL operator. Some machines provide a hardware POPC population count instruction which counts the number of 1 s in a word. If running on such a machine, this can be used to implement the BAL operator for additional speed enhancement over lookup table techniques.

Let n be the desired total string, or code, length, assumed to be even. Initial equal-length bit strings at the left and right ends of the code are specified by the user. Let nn0 represent the number of element in these preset strings, and let **l** stand for the left string, and let **r** stand for the right string. For example, the user might want to specify initial strings of length nn0 = 2 on each side, and set **l** = '00' and **r** = '01', for an initial partial string [**l**, **r**] = '0001'. As mentioned earlier, this allows for parallelism, in that several processes can be set in motion each for a different case of initial string settings. The algorithm proceeds as indicated by the following pseudocode.

It is straightforward to modify the algorithm for odd lengths. Let the "extra" element be the one in the middle. Note that this last element only comes in if the search makes it to the lowest level. At that level, simply test both possibilities for this bit and accommodate the extra bit in the computations of the middle sidelobes.

```
/* initialization */

nn = nn0 + 1;
n2 = n/2;

l = RCAT(l); // add a zero to the right end of the left half-string l
r = LCAT(r); // add a zero to the left end of the right half-string r
s = BAL(XNOR(l,r)); // compute the newly specified sidelobe

/* main loop */

while (LEN(l) > nn0) {
    while (|s| ≤ thresh && LEN(l) < (n/2)) {
        l = RCAT(l)R; // add a zero on the right of left half-string l
        r = LCAT(r); // add a zero on the left of half-string r
        s = BAL(XNOR(l,r)); // compute this sidelobe
    }
    if (LEN(l)==n2) { /* do final middle bits 00, 01, 10, 11 */
        if (|s| ≤ thresh) { if (CORR(l,r,thresh)) REPORT([l,r]);) INC(l,r); }
        if (|s − 2| ≤ thresh) {
            if (CORR(l,r,thresh)) REPORT([l,r]);
            INC(l,r);
            if (CORR(l,r,thresh)) REPORT([l,r]);
            INC(l,r);
        } else {
            INC(l,r);
            INC(l,r);
        }
    if (|s − 4| <=
    thresh) { if (CORR(l,r,thresh)) REPORT([l,r]); }
    while (abs(s) > thresh ||LEN(l)==n2) {
        INC(l,r);
        if (l > REV(r)) INC(l,r);
        s = BAL(XNOR(l,r)); // compute this sidelobe
    }
}
```

SEARCH RESULTS

One of the tasks to which the search algorithm was applied was a search to find all the optimal-PSL binary codes of length 64. The search was done using two separate approaches, one by the algorithm described in the previous sections, and a second employing functional recursion to cover the search space. These searches each found the same set of 1859 codes with lowest PSL, after redundancies due to the three PSL-preserving symmetries were eliminated. The equivalence class for each of these codes contains 8 distinct codes, so the total number of distinct optimal-PSL codes is 8 times this tally of 1859, or 14872, of which 600 are balanced.

Table I lists the 142 codes which represent all balanced optimal-PSL binary codes of length 64. For all entries, the negation and reversal PSL preserving operations may be applied, resulting in four codes for each listed, or 568. The eight entries with $\lambda_e = 0$ and $\lambda_o = 0$ each have an additional four balanced sister codes arrived at by applying sign alternation, for a total of 600.

Another result of the search for $N = 64$ is an optimal-PSL code 26C9FD5F5A1D798C having merit factor 9.8462. This value should be compared with the asymptotic value of 3 for PRN codes and for the maximum known value which is 14.08, achieved by the Barker codes of length 13 [2]. Unlike PSL, for which low values are desirable, higher values of MF are preferable.

A second task to which the algorithm was applied was to establish PSL=4 as the optimal PSL for code lengths 61 to 69. It is known that no codes exist for these lengths that achieve PSL ≤ 3, so it was necessary only to run the search until a single code with PSL of 4 was found. Table II gives optimal-PSL codes for each code length from 61 to 70 in hexadecimal format. In cases where the code length is not an exact multiple of 4, the extra bits appear in the more significant bits, on the left side of the code representation. In each case, the PSL is 4. The example for $N = 63$ is reprinted from [1]; the others are results of the new searches and replace codes erroneously claimed in [1] to have PSL = 4.

TABLE II
Example Optimal-PSL Codes for $N = 61$ to $N = 69$

Code Length N	Code in Hexadecimal	PSL
61	005B44C4C79EA350	4
62	002D66634CB07450	4
63	04CF5A2471657C6F	4
64	55FF84B069386665	4
65	002DC0B0D9BCE5450	4
66	0069B454739F12B42	4
67	007F1D164C62A5242	4
68	009E49E3662A8EA50	4
69	0231C08FDA5A0D9355	4
70	1A133B4E3093EDD57E	4

SUMMARY

An efficient exhaustive search routine is given which is designed to find all binary codes of a given length having autocorrelation PSL under a given size. The routine was applied to two tasks, to prove its effectiveness and efficiency. The first task was to find all optimal-PSL binary codes of length 64. The search found 1859 optimal-PSL code representatives, each representing eight codes for a total of 14872. Of these, 142 are balance-equivalent (that is, can be transformed to at least one balanced code by some combination of PSL-preserving operations), representing a total of 600 balanced codes. The validity of these results was checked by using an independent implementation. The second task to which the algorithm was applied was to find one PSL-4 binary code for each code length from 61 to 70, to establish 4 as the optimal PSL for those lengths.

REFERENCES

[1] Coxson, G. E., Cohen, M. N., and Hirschel, A.
New results on minimum-PSL binary codes.
Presented at the IEEE National Radar Conference, Atlanta, GA, May 2001.

[2] Golay, M. J. E.
Sieves for low autocorrelation binary sequences.
IEEE Transactions on Information Theory, **IT-23**, 1 (Jan. 1977), 43–51.

[3] Cohen, M. N., Fox, M. R., and Baden, J. M.
Minimum peak sidelobe pulse compression codes.
In *Proceedings of the 1990 IEEE International Radar Conference*, Arlington, VA, 1990, 633–638.

[4] Lindner, J.
Binary sequences up to length 40 with best possible autocorrelation function.
Proceedings of IEEE, **11**, 21 (Oct. 1975).

[5] Kerdock, A. M., Mayer, R., and Bass, D.
Longest binary pulse compression codes with given peak sidelobe levels.
Proceedings of the IEEE, **74**, 2 (Feb. 1988), 366.

[6] Mertens, S.
Exhaustive search for low-autocorrelation binary sequences.
Journal of Physics A: Mathematical and General, **29** (1996), L473-L481.

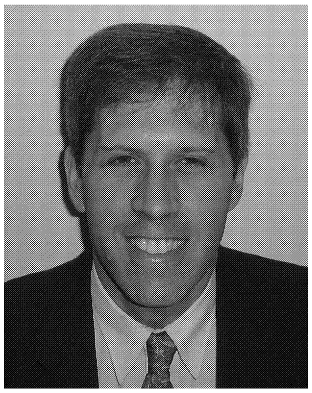
[7] Mertens, S.
On the ground state of the Bernasconi model.
Journal of Physics A: Mathematical and General, **31** (1998), 3731–3749.

[8] Golay, M. J. E.
The merit factor of long low autocorrelation binary sequences.
IEEE Transactions on Information Theory, **28** (May 1982), 543.

[9] Militzer, B., Zamparelli, M., and Beule, D.
Evolutionary search for low autocorrelated binary sequences.
IEEE Transaction on Evolutionary Computation, **2**, 1 (Apr. 1998).

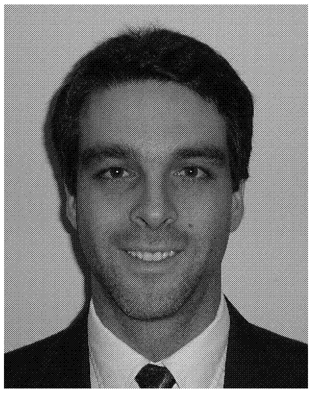
[10] Schroeder, M. R.
Number Theory in Science and Communications, New York: Springer Verlag, 1990.

[11] Skolnik, M.
Radar Handbook.
New York: McGraw-Hill, 1990.



Gregory E. Coxson was born in Guayaquil, Ecuador in 1958. He received the M.A. degree in mathematics in 1987, the M.S. degree in electrical engineering in 1989, and the Ph.D. degree in electrical engineering in 1993, all from the University of Wisconsin at Madison.

He has been a radar systems engineer at Lockheed Martin Maritime Systems and Sensors since 1996, involved in ECM, pulse compression, ship-based missile defense and other radar projects. He was an engineer at Hughes Radar Systems in El Segundo, CA from 1994 to 1996.



Jon C. Russo was born in Geneva, NY. He graduated with distinction from Cornell University School of Electrical Engineering and received his M.S. in 1993.

He joined the research team at Lockheed Martin Advanced Technology Labs, where he has worked in signal processing, radar, hardware design, reconfigurable computing, and compiler technologies. His research interests include cognitive architectures, radar and communications processing.