SEP 9 1999

RECEIVED
OCT 20 1999
OSTI

SANDIA REPORT

SAND99-2033

Unlimited Release

Printed August 1999

An Implementation of the Berlekamp-Massey Linear Feedback Shift-Register Synthesis Algorithm in the C Programming Language

Philip L. Campbell

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

Approved for public release; further dissemination unlimited.

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Prices available from (703) 605-6000
Web site: <http://www.ntis.gov/ordering.htm>

Available to the public from
National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Rd
Springfield, VA 22161

NTIS price codes
Printed copy: A03
Microfiche copy: A01

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

SAND99-2033
Unlimited Release
Printed August 1999

An Implementation of the Berlekamp-Massey Linear Feedback Shift-Register Synthesis Algorithm in the C Programming Language

Philip L. Campbell
Secure Networks & Information Systems
Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87185-0449
plcampb@sandia.gov

Abstract:

This report presents an implementation of the Berlekamp-Massey linear feedback shift-register (LFSR) synthesis algorithm in the C programming language. Two pseudo-code versions of the code are given, the operation of LFSRs is explained, C-version of the pseudo-code versions is presented, and the output of the code, when run on two input samples, is shown.

1 Introduction

The Berlekamp-Massey linear feedback shift-register synthesis algorithm produces one of the shortest linear feedback shift-registers (LFSR) that will generate a given input binary sequence. An implementation of that algorithm in the C programming language is presented here, needed because there is not one commonly available.

We give the pseudo-code presentations of the algorithm in Section 2, explain in Section 3 how LFSRs operate, present the algorithm in the C programming language in Section 5, and show the output for two inputs in Section 6. More information on the algorithm, such as a proof of its correctness [1], and an analysis of its running time [2], can be found elsewhere.

2 Pseudo-Code Presentations

The Berlekamp-Massey algorithm is expressed in pseudo-code in Massey's paper [1] and is shown in Figure 2.

Figure 1. Berlekamp-Massey algorithm in pseudo-code (from Massey [1])

[The input is in s ; the tap sequence is in c .]

1. $1 \rightarrow C(D)$, $1 \rightarrow B(D)$, $1 \rightarrow x$
 $0 \rightarrow L$, $1 \rightarrow b$, $0 \rightarrow N$

2) If $N = n$, stop. Otherwise, compute
 $d = s_N + \sum_{i=1}^L c_i s_{N-i}$

3) If $d = 0$, then $x + 1 \rightarrow x$, and go to 6).

4. If $d \neq 0$ and $2L > N$, then
 $C(D) - d b^{-1} D^x B(D) \rightarrow C(D)$
 $x + 1 \rightarrow x$
and go to 6).

5) If $d \neq 0$ and $2L \leq N$, then
 $C(D) \rightarrow T(D)$ [temporary storage of $C(D)$]
 $C(D) - d b^{-1} D^x B(D) \rightarrow C(D)$
 $N + 1 - L \rightarrow L$
 $T(D) \rightarrow B(D)$
 $d \rightarrow b$
 $1 \rightarrow x$.

6) $N+1 \rightarrow N$ and return to 2).

The algorithm is stated in slightly different terms in Gustavson's paper [2]. The implementation provided in this report, shown in Section 5, is derived from the pseudo-code in the text by Menezes et al. [3] and is shown in Figure 2.

Figure 2. Berlekamp-Massey algorithm in pseudo-code (This is item 6.30 on page 201 from Menezes et al. [3])

```

INPUT: a binary sequence  $s^n = s_0, s_1, s_2, \dots, s_{n-1}$  of length  $n$ .
OUTPUT: the linear complexity  $L(s^n)$  of  $s^n$ ,  $0 \leq L(s^n) \leq N$ .

1. Initialization:  $C(D) \leftarrow 1$ ,  $L \leftarrow 0$ ,  $m \leftarrow -1$ ,  $B(D) \leftarrow 1$ ,  $N \leftarrow 0$ .
2. While ( $N < n$ ) do the following:
    2.1 Compute the next discrepancy  $d$ .
         $d \leftarrow (s_N + \sum_{i=1}^L c_i s_{N-i}) \bmod 2$ .
    2.2 If  $d = 1$  then do the following:
         $T(D) \leftarrow C(D)$ ,  $C(D) \leftarrow C(D) + B(D) * D^{N-m}$ .
        If  $L \leq N/2$  then  $L \leftarrow N+1-L$ ,  $m \leftarrow N$ ,  $B(B) \leftarrow T(D)$ .
    2.3  $N \leftarrow N+1$ .
3. Return( $L$ ) .

```

The meaning of the variables and exactly how the algorithm works should be clear from the presentation in C, shown in Section 5.

3 How LFSRs Operate

An LFSR consists of n "cells," $0 \leq n$, and k "taps," $0 \leq k \leq n$. The value in each cell is initialized to either 0 or 1. For example, the following

cell numbers →	0	1	2	3	4
cells →	0	1	1	0	0
taps →			*		*

represents an LFSR of length 5 (i.e., with 5 cells) and two taps. The taps are at cell numbers 2 and 4, where the leftmost cell is cell number 0, as numbered in the example. This LFSR generates a binary sequence that begins with 001101110, where the leftmost bit is the first bit generated.¹

The Berlekamp-Massey algorithm presumes the Fibonacci configuration.² That is, an LFSR generates a bit by proceeding through the following three steps:

1. The binary values in the cells corresponding to the taps are added up, mod 2, to produce a new binary value that we will call, for the nonce, the "shift-in bit."

1. This LFSR and input binary sequence are taken from the example that accompanies the statement of the pseudo-code of the algorithm in Menezes' text. [3]

2. Other configurations are possible, such as the Galois configuration, in which the output of the register is XORed with the cell corresponding to each tap

2. The LFSR is shifted to the right. The bit shifted out is the generated bit.

3. The leftmost cell in the LFSR is set to the value of the shift-in bit.

Note that if there are zero taps, then the shift-in bit is always zero. If there is only one tap, then the shift-in bit is always the value of the cell corresponding to the tap. And if there are at least two taps, then the shift-in bit is the result of the EXCLUSIVE-OR function.

4 Sample LFSRs

Given the generating steps shown in Section 3, note that

- binary sequences of the form 0^* , where $*$ denotes zero or more appearances of the preceding symbol (Kleene closure), can be generated by the LFSR of length zero;
- binary sequences of the form 1^+ , where $^+$ denotes at least one appearance of the preceding symbol, can be generated by an LFSR of length one, the one cell initialized to 1, with one tap;
- binary sequences of the form 10^* can be generated by an LFSR of length one, the one cell initialized to 1, with zero taps;
- binary sequences of the form 0^+1 require an LFSR that is the same length as the input binary sequence, with any arrangement of taps. (The arrangement of taps is of no consequence to this LFSR because the entire input binary sequence is in the LFSR when the LFSR begins to "generate" the sequence.)

When the code that is shown below is given the following input binary sequence³

```
001101110
```

the code will generate the following output:

```
lfsr: 01100
```

```
taps: 2 4
```

where the leftmost cell is number 0.

The initial cell values are the reverse of the leftmost 5 values of the input binary sequence.

The lfsr generates bits from right to left.

As a second example, when the code is given the following input binary sequence⁴

```
111101011001000
```

it will generate the following output:

```
lfsr: 1111
```

```
taps: 0 3
```

where the leftmost cell is number 0.

The initial cell values are the reverse of the

3. This is the same binary sequence shown further above.

4. This binary sequence is taken from an example in Schneier [4] p. 374. Note that the tap numbering convention here is different than Schneier's.

leftmost 4 values of the input binary sequence.
The lfsr generates bits from right to left.

The two example inputs shown above are run through the code shown in Section 5. The output of these two runs are shown in Section 6.1 and Section 6.2 respectively.

5 The Code in C

In this section we present the code in C. The code was run on all binary sequences of length 0 through 16, inclusive, and the results were checked to confirm that the LFSR produced by the algorithm generates the input binary sequence and that there are no shorter LFSRs that produce the same string.⁵

The code:

```
1  /*
2  Berlekamp-Massey Linear Feedback Shift-Register
3  Synthesis Algorithm.
4
5  Adapted from item 6.30 on page 201 of Menezes' text.
6
7  We assume the following:
8
9  1. that the function get_s (not shown) fills the integer array
10 s with the input binary sequence, where s[i] is set to either
11 0 or 1, and s[0] holds the first bit generated, s[1] holds the
12 next bit generated, etc.
13
14 2. that the function get_s_length (not shown) sets s_length to
15 the length of the input sequence s;
16
17 3. and that MAX_LFSR_LENGTH >= MAX_S_LENGTH >= s_length;
18
19 (The function print_line (not shown) prints a line of dashes.)
20 */
21
22 static int
23     s_length,
24     s      [MAX_S_LENGTH],
25
26     d, k, m, L, N, p, i, j, bound,
27
28     b      [MAX_LFSR_LENGTH],
29     c      [MAX_LFSR_LENGTH],
30     t      [MAX_LFSR_LENGTH],
31
```

5. There may be more than one shortest-length LFSR that can generate a given binary sequence. If there is, the initial cell settings for each LFSR will be the same but their tap sequences will differ. For example, for the binary sequence $0^{n-1}1$, where $n > 1$, there are 2^n generating LFSRs, all of length n , and each is a shortest-length LFSR.


```

32     lfsr_length,
33     lfsr   [MAX_LFSR_LENGTH],
34     num_taps,
35     taps   [MAX_LFSR_LENGTH];
36
37  /*-----*/
38
39  bm()
40  {
41      /* step 1. Initialization */
42
43      s_length = get_s_length ();
44      get_s ( s );
45
46      printf("input string (length %d):\n", s_length);
47      for ( i = 0 ; i < s_length ; i++ )
48          printf("%ld", s[i]);
49      printf("\n");
50      printf("(the leftmost bit was the first input)\n");
51
52      L   = 0;
53      m   = 0;
54      b[0] = 1;
55      N   = 0;
56
57      for ( i = 0 ; i < s_length ; i++ )
58          c[i] = 0;
59
60      /* step 2 */
61
62      while ( N < s_length )
63      {
64          print_line ();
65          printf("begin iteration %d = N:\n", N);
66
67          /* step 2.1. Compute the next discrepancy d. */
68
69          for ( i = 1, k = 0 ; i <= L ; i++ )
70              if ( c[i] )
71                  k += s[N-i];
72
73          d = s[N] != (k % 2);
74
75          if ( d == 1 )
76              printf("\td = 1\n");
77          else
78              printf("\td = 0\n");
79
80          /* step 2.2. If d = 1, then do the following: */
81
82          if ( d == 1 )

```

```

83     {
84         bound = N+1;
85         p = N - m;
86
87         for ( i = 1 ; i < bound ; i++ )
88             t[i] = c[i];
89
90         for ( i = p, j = 0 ; i < bound ; i++, j++ )
91         {
92             if ( b[j] )
93             {
94                 if ( c[i] )
95                     c[i] = 0;
96                 else
97                     c[i] = 1;
98             }
99         }
100
101         if ( L <= (N/2) )
102             printf("\t\tL <= (N / 2)\n");
103         else
104             printf("\t\tL > (N / 2)\n");
105
106         if ( L <= (N / 2) )
107         {
108             L = N + 1 - L;
109             m = N;
110
111             for ( i = 1 ; i < bound ; i++ )
112                 b[i] = t[i];
113         }
114     }
115
116     /* step 2.3. N <-- N+1 */
117
118     N++;
119
120     printf("\tat end of iteration:\n");
121     printf("\t\t");
122     if ( N < s_length )
123         printf("s[N] = %d, ", s[N]);
124     else
125         printf("s[N] = ?, ");
126     printf("N = %d, ", N);
127     printf("L = %d, ", L);
128     printf("m = %d, ", m);
129     printf("p = %d; ", p);
130     printf("\n");
131
132     printf("\t\tb: ");
133     for ( i = 0 ; i < s_length ; i++ )

```

```

134         printf("%d", b[i]);
135     printf("\n");
136
137     printf("\t\tc: ");
138     for ( i = 0 ; i < s_length ; i++ )
139         printf("%d", c[i]);
140     printf("\n");
141
142     printf("\t\tt: ");
143     for ( i = 0 ; i < s_length ; i++ )
144         printf("%d", t[i]);
145     printf("\n");
146 }
147
148     print_line ();
149     convert_results ();
150     print_lfsr ();
151 }
152
153 /*-----*/
154
155 /*
156 the initial values of the lfsr cells are always the initial
157 bits from the input binary sequence;
158 the tap sequence is the c array shifted once to the left,
159 that is:
160     tap[i] = c[i+1];
161 */
162
163 static convert_results ()
164 {
165     int i;
166
167     lfsr_length = L;
168     num_taps = 0;
169
170     for ( i = 0 ; i < lfsr_length ; i++ )
171     {
172         lfsr[i] = s[L-i-1];
173         taps[i] = c[i+1];
174
175         if ( taps[i] == 1 )
176             num_taps++;
177     }
178 }
179
180 /*-----*/
181
182 static print_lfsr()
183 {
184     int i;

```

```

185
186     printf("lfsr (length %d, taps %d):\n",
187           lfsr_length, num_taps );
188
189     printf("lfsr: ");
190     for ( i = 0 ; i < lfsr_length ; i++ )
191         printf("%d", lfsr[i] );
192     printf("\n");
193
194     printf("taps: ");
195     for ( i = 0 ; i < lfsr_length ; i++ )
196         if ( taps[i] == 1 )
197             printf("%d ", i );
198     printf("\n");
199
200     printf("      where the leftmost cell is number 0.\n");
201     printf("The initial cell values are the reverse of the\n");
202     printf("leftmost %d values of the input bit sequence.\n",
203           lfsr_length);
204     printf("The lfsr generates bits from right to left.\n");
205 }

```

6 Sample Output

Shown in this section is the output of the code in Section 5 when run on the two sample input bit sequences shown in Section 3.

6.1 First Example

```

1 input string (length 9):
2 001101110
3 (the leftmost bit was the first input)
4 -----
5 begin iteration 0 = N:
6     d = 0
7     at end of iteration:
8         s[N] = 0, N = 1, L = 0, m = 0, p = 0;
9         b: 100000000
10        c: 000000000
11        t: 000000000
12 -----
13 begin iteration 1 = N:
14     d = 0
15     at end of iteration:
16         s[N] = 1, N = 2, L = 0, m = 0, p = 0;
17         b: 100000000
18         c: 000000000
19         t: 000000000
20 -----

```

```

21 begin iteration 2 = N:
22     d = 1
23     L <= (N / 2)
24     at end of iteration:
25         s[N] = 1, N = 3, L = 3, m = 2, p = 2;
26         b: 100000000
27         c: 001000000
28         t: 000000000
29 -----
30 begin iteration 3 = N:
31     d = 1
32     L > (N / 2)
33     at end of iteration:
34         s[N] = 0, N = 4, L = 3, m = 2, p = 1;
35         b: 100000000
36         c: 011000000
37         t: 001000000
38 -----
39 begin iteration 4 = N:
40     d = 0
41     at end of iteration:
42         s[N] = 1, N = 5, L = 3, m = 2, p = 1;
43         b: 100000000
44         c: 011000000
45         t: 001000000
46 -----
47 begin iteration 5 = N:
48     d = 0
49     at end of iteration:
50         s[N] = 1, N = 6, L = 3, m = 2, p = 1;
51         b: 100000000
52         c: 011000000
53         t: 001000000
54 -----
55 begin iteration 6 = N:
56     d = 0
57     at end of iteration:
58         s[N] = 1, N = 7, L = 3, m = 2, p = 1;
59         b: 100000000
60         c: 011000000
61         t: 001000000
62 -----
63 begin iteration 7 = N:
64     d = 1
65     L <= (N / 2)
66     at end of iteration:
67         s[N] = 0, N = 8, L = 5, m = 7, p = 5;
68         b: 111000000
69         c: 011001000
70         t: 011000000
71 -----

```

```

72 begin iteration 8 = N:
73     d = 1
74     L > (N / 2)
75     at end of iteration:
76         s[N] = ?, N = 9, L = 5, m = 7, p = 1;
77         b: 111000000
78         c: 000101000
79         t: 011001000
80 -----
81 lfsr (length 5, taps 2):
82 lfsr: 01100
83 taps: 2 4
84     where the leftmost cell is number 0.
85 The initial cell values are the reverse of the
86 leftmost 5 values of the input bit sequence.
87 The lfsr generates bits from right to left.

```

6.2 Second Example

```

1 input string (length 15):
2 111101011001000
3 (the leftmost bit was the first input)
4 -----
5 begin iteration 0 = N:
6     d = 1
7     L <= (N / 2)
8     at end of iteration:
9         s[N] = 1, N = 1, L = 1, m = 0, p = 0;
10        b: 100000000000000
11        c: 100000000000000
12        t: 000000000000000
13 -----
14 begin iteration 1 = N:
15     d = 1
16     L > (N / 2)
17     at end of iteration:
18         s[N] = 1, N = 2, L = 1, m = 0, p = 1;
19         b: 100000000000000
20         c: 110000000000000
21         t: 000000000000000
22 -----
23 begin iteration 2 = N:
24     d = 0
25     at end of iteration:
26         s[N] = 1, N = 3, L = 1, m = 0, p = 1;
27         b: 100000000000000
28         c: 110000000000000
29         t: 000000000000000
30 -----
31 begin iteration 3 = N:

```

```

32     d = 0
33     at end of iteration:
34         s[N] = 0, N = 4, L = 1, m = 0, p = 1;
35         b: 1000000000000000
36         c: 1100000000000000
37         t: 0000000000000000
38 -----
39 begin iteration 4 = N:
40     d = 1
41     L <= (N / 2)
42     at end of iteration:
43         s[N] = 1, N = 5, L = 4, m = 4, p = 4;
44         b: 1100000000000000
45         c: 1100100000000000
46         t: 0100000000000000
47 -----
48 begin iteration 5 = N:
49     d = 0
50     at end of iteration:
51         s[N] = 0, N = 6, L = 4, m = 4, p = 4;
52         b: 1100000000000000
53         c: 1100100000000000
54         t: 0100000000000000
55 -----
56 begin iteration 6 = N:
57     d = 0
58     at end of iteration:
59         s[N] = 1, N = 7, L = 4, m = 4, p = 4;
60         b: 1100000000000000
61         c: 1100100000000000
62         t: 0100000000000000
63 -----
64 begin iteration 7 = N:
65     d = 0
66     at end of iteration:
67         s[N] = 1, N = 8, L = 4, m = 4, p = 4;
68         b: 1100000000000000
69         c: 1100100000000000
70         t: 0100000000000000
71 -----
72 begin iteration 8 = N:
73     d = 0
74     at end of iteration:
75         s[N] = 0, N = 9, L = 4, m = 4, p = 4;
76         b: 1100000000000000
77         c: 1100100000000000
78         t: 0100000000000000
79 -----
80 begin iteration 9 = N:
81     d = 0
82     at end of iteration:

```

```

83         s[N] = 0, N = 10, L = 4, m = 4, p = 4;
84         b: 1100000000000000
85         c: 1100100000000000
86         t: 0100000000000000
87 -----
88 begin iteration 10 = N:
89     d = 0
90     at end of iteration:
91         s[N] = 1, N = 11, L = 4, m = 4, p = 4;
92         b: 1100000000000000
93         c: 1100100000000000
94         t: 0100000000000000
95 -----
96 begin iteration 11 = N:
97     d = 0
98     at end of iteration:
99         s[N] = 0, N = 12, L = 4, m = 4, p = 4;
100        b: 1100000000000000
101        c: 1100100000000000
102        t: 0100000000000000
103 -----
104 begin iteration 12 = N:
105     d = 0
106     at end of iteration:
107         s[N] = 0, N = 13, L = 4, m = 4, p = 4;
108         b: 1100000000000000
109         c: 1100100000000000
110         t: 0100000000000000
111 -----
112 begin iteration 13 = N:
113     d = 0
114     at end of iteration:
115         s[N] = 0, N = 14, L = 4, m = 4, p = 4;
116         b: 1100000000000000
117         c: 1100100000000000
118         t: 0100000000000000
119 -----
120 begin iteration 14 = N:
121     d = 0
122     at end of iteration:
123         s[N] = ?, N = 15, L = 4, m = 4, p = 4;
124         b: 1100000000000000
125         c: 1100100000000000
126         t: 0100000000000000
127 -----
128 lfsr (length 4, taps 2):
129 lfsr: 1111
130 taps: 0 3
131     where the leftmost cell is number 0.
132 The initial cell values are the reverse of the
133 leftmost 4 values of the input bit sequence.

```


134 The lfsr generates bits from right to left.

7 Conclusion

This paper presented an implementation of the Berlekamp-Massey linear feedback shift-register synthesis algorithm in the C programming language. Two pseudo-code forms of the algorithm were given. The output of the algorithm was shown for two sample inputs.

Acknowledgments

I would like to thank Lyndon Pierson and Anne Van Arsdall for their review of this report.

References

- [1] J. L. Massey, "Shift-Register Synthesis and BCH Decoding," *IEEE Tran. Information Theory* IT-15, 122, 1969.
- [2] F. G. Gustavson, "Analysis of the Berlekamp-Massey Linear Feedback Shift-Register Synthesis Algorithm," *IBM Journal of Research and Development*, 20:204-12, May 1976.
- [3] A. J. Menezes, P. C. van Oorschot, S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, Boca Raton, Florida, 1997.
- [4] B. Schneier, *Applied Cryptography*, 2nd ed., John Wiley & Sons, New York, 1996.

Distribution List

2	MS	0449	P. L. Campbell	Org.	6237
1		0449	R. S. Tamashiro		6237
1		0451	R. E. Trelue		6238
1		0612	Review & Approval Desk For DOE/OSTI		4912
1		0741	S. G. Varnado		6200
1		0806	L. B. Dean		4616
1		0806	L. G. Pierson		4616
1		0806	T. D. Tarman		4616
1		0806	M. O. Vahle		4616
1		0806	E. L. Witzke		4616
1		0874	P. J. Robertson		1716
1		0874	K. L. Gass		6421
2		0899	Technical Library		4916
1		8903	P. W. Dean		9011
1		9018	Central Technical Files		8940-2