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
DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.
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
   $C(D) - d \cdot b^{-1} D^x B(D) \rightarrow C(D)$
   $x + 1 \rightarrow x$
   and go to 6).

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

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 \( s_n = s_0, s_1, s_2, \ldots, s_{n-1} \) 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 \( d \).
       \( d \leftarrow (s_n + \sum_{i=1}^{L} c_is_{n-i}) \mod 2. \)
   2.2 If \( d = 1 \) then do the following:
       \( T(D) \leftarrow C(D), C(D) \leftarrow C(D) + B(D) \cdot D^{N-m}. \)
       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

<table>
<thead>
<tr>
<th>cell numbers</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>cells</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>taps</td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 sequence \(^3\)

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 sequence \(^4\)

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:

```c
/*
Berlekamp-Massey Linear Feedback Shift-Register 
Synthesis Algorithm.

Adapted from item 6.30 on page 201 of Menezes' text.

We assume the following:

1. that the function get_s (not shown) fills the integer array 
s with the input binary sequence, where s[i] is set to either 
0 or 1, and s[0] holds the first bit generated, s[1] holds the 
next bit generated, etc.

2. that the function get_s_length (not shown) sets s_length to 
the length of the input sequence s;

3. and that MAX_LFSR_LENGTH >= MAX_S_LENGTH >= s_length;

(The function print_line (not shown) prints a line of dashes.)
*/

static int 
s_length,
s [MAX_S_LENGTH],
d, k, m, L, N, p, i, j, bound,
b [MAX_LFSR_LENGTH],
c [MAX_LFSR_LENGTH],
t [MAX_LFSR_LENGTH],
```

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, 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 )
{  
  bound = N+1;
  p = N - m;

  for ( i = 1 ; i < bound ; i++ )
    t[i] = c[i];

  for ( i = p, j = 0 ; i < bound ; i++, j++ )
  {
    if ( b[j] )
    {
      if ( c[i] )
        c[i] = 0;
      else
        c[i] = 1;
    }
  }

  if ( L <= (N/2))
    printf("\tL \leq \frac{N}{2}\n");
  else
    printf("\tL > \frac{N}{2}\n");

  if ( L <= (N/2) )
  {
    L = N + 1 - L;
    m = N;

    for ( i = 1 ; i < bound ; i++ )
      b[i] = t[i];
  }

  /* step 2.3. N <-- N+1 */
  N++;

  printf("\tat end of iteration:  \n");
  printf("\tt\n");
  if ( N < s_length )
    printf("s[N] = %d, ", s[N]);
  else
    printf("s[N] = ?, ");

  printf("N = %d, ", N);
  printf("L = %d, ", L);
  printf("m = %d, ", m);
  printf("p = %d; ", p);
  printf("n\n");

  printf("\ttb:  ");
  for ( i = 0 ; i < s_length ; i++ )
  
}
134     printf("%d", b[i]);
135     printf("\n");
136     printf("\t\tc: ");
137     for ( i = 0 ; i < s_length ; i++ )
138         printf("%d", c[i]);
139     printf("\n");
140     printf("\t\tt: ");
141     for ( i = 0 ; i < s_length ; i++ )
142         printf("%d", t[i]);
143     printf("\n");
144 }
145
146     print_line ();
147     convert_results ();
148     print_lfsr ();
149 }
150
151 /*-----------------------------------------------*/
152 /
153 /* the initial values of the lfsr cells are always the initial
154 bits from the input binary sequence;
155 the tap sequence is the c array shifted once to the left,
156 that is:
157     tap[i] = c[i+1];
158 */
159
160 static convert_results ()
161 {
162     int i;
163     lfsr_length = L;
164     num_taps = 0;
165     for ( i = 0 ; i < lfsr_length ; i++ )
166         { lfsr[i] = s[L-i-1];
167             taps[i] = c[i+1];
168             if ( taps[i] == 1 )
169                 num_taps++;
170         }
171 }
172
173 /*-----------------------------------------------*/
174 /
175 static print_lfsr()
176 {
177     int i;
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 begin iteration 0 = N:
5 d = 0
6 at end of iteration:
7 s[N] = 0, N = 1, L = 0, m = 0, p = 0;
8 b: 100000000
9 c: 000000000
10 t: 000000000
11 begin iteration 1 = N:
12 d = 0
13 at end of iteration:
14 s[N] = 1, N = 2, L = 0, m = 0, p = 0;
15 b: 100000000
16 c: 000000000
17 t: 000000000
begin iteration 2 = N:
    d = 1
    L <= (N / 2)
    at end of iteration:
        s[N] = 1, N = 3, L = 3, m = 2, p = 2;
        b: 100000000
        c: 001000000
        t: 000000000

begin iteration 3 = N:
    d = 1
    L > (N / 2)
    at end of iteration:
        s[N] = 0, N = 4, L = 3, m = 2, p = 1;
        b: 100000000
        c: 011000000
        t: 001000000

begin iteration 4 = N:
    d = 0
    at end of iteration:
        s[N] = 1, N = 5, L = 3, m = 2, p = 1;
        b: 100000000
        c: 011000000
        t: 001000000

begin iteration 5 = N:
    d = 0
    at end of iteration:
        s[N] = 1, N = 6, L = 3, m = 2, p = 1;
        b: 100000000
        c: 011000000
        t: 001000000

begin iteration 6 = N:
    d = 0
    at end of iteration:
        s[N] = 1, N = 7, L = 3, m = 2, p = 1;
        b: 100000000
        c: 011000000
        t: 001000000

begin iteration 7 = N:
    d = 1
    L <= (N / 2)
    at end of iteration:
        s[N] = 0, N = 8, L = 5, m = 7, p = 5;
        b: 111000000
        c: 011001000
        t: 011000000
begin iteration 8 = N:
    d = 1
    L > (N / 2)
    at end of iteration:
    s[N] = ?, N = 9, L = 5, m = 7, p = 1;
    b: 111000000
    c: 000101000
    t: 011001000
lfsr (length 5, taps 2):
    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 bit sequence.
The lfsr generates bits from right to left.

6.2 Second Example

begin iteration 0 = N:
    d = 1
    L <= (N / 2)
    at end of iteration:
    s[N] = 1, N = 1, L = 1, m = 0, p = 0;
    b: 100000000000000
    c: 100000000000000
    t: 000000000000000

begin iteration 1 = N:
    d = 1
    L > (N / 2)
    at end of iteration:
    s[N] = 1, N = 2, L = 1, m = 0, p = 1;
    b: 100000000000000
    c: 110000000000000
    t: 000000000000000

begin iteration 2 = N:
    d = 0
    at end of iteration:
    s[N] = 1, N = 3, L = 1, m = 0, p = 1;
    b: 100000000000000
    c: 110000000000000
    t: 000000000000000

begin iteration 3 = N:
d = 0
at end of iteration:
s[N] = 0, N = 4, L = 1, m = 0, p = 1;
b: 100000000000000
c: 110000000000000
t: 000000000000000

begin iteration 4 = N:
d = 1
L <= (N / 2)
at end of iteration:
s[N] = 1, N = 5, L = 4, m = 4, p = 4;
b: 110000000000000
c: 110010000000000
t: 010000000000000

begin iteration 5 = N:
d = 0
at end of iteration:
s[N] = 0, N = 6, L = 4, m = 4, p = 4;
b: 110000000000000
c: 110010000000000
t: 010000000000000

begin iteration 6 = N:
d = 0
at end of iteration:
s[N] = 1, N = 7, L = 4, m = 4, p = 4;
b: 110000000000000
c: 110010000000000
t: 010000000000000

begin iteration 7 = N:
d = 0
at end of iteration:
s[N] = 1, N = 8, L = 4, m = 4, p = 4;
b: 110000000000000
c: 110010000000000
t: 010000000000000

begin iteration 8 = N:
d = 0
at end of iteration:
s[N] = 0, N = 9, L = 4, m = 4, p = 4;
b: 110000000000000
c: 110010000000000
t: 010000000000000

begin iteration 9 = N:
d = 0
at end of iteration:
begin iteration 10 = N:
\[ d = 0 \]
\[ \text{at end of iteration:} \]
\[ s[N] = 1, N = 11, L = 4, m = 4, p = 4; \]
\[ b: \overline{110000000000000} \]
\[ c: \overline{110010000000000} \]
\[ t: \overline{010000000000000} \]

begin iteration 11 = N:
\[ d = 0 \]
\[ \text{at end of iteration:} \]
\[ s[N] = 0, N = 12, L = 4, m = 4, p = 4; \]
\[ b: \overline{110000000000000} \]
\[ c: \overline{110010000000000} \]
\[ t: \overline{010000000000000} \]

begin iteration 12 = N:
\[ d = 0 \]
\[ \text{at end of iteration:} \]
\[ s[N] = 0, N = 13, L = 4, m = 4, p = 4; \]
\[ b: \overline{110000000000000} \]
\[ c: \overline{110010000000000} \]
\[ t: \overline{010000000000000} \]

begin iteration 13 = N:
\[ d = 0 \]
\[ \text{at end of iteration:} \]
\[ s[N] = 0, N = 14, L = 4, m = 4, p = 4; \]
\[ b: \overline{110000000000000} \]
\[ c: \overline{110010000000000} \]
\[ t: \overline{010000000000000} \]

begin iteration 14 = N:
\[ d = 0 \]
\[ \text{at end of iteration:} \]
\[ s[N] = ?, N = 15, L = 4, m = 4, p = 4; \]
\[ b: \overline{110000000000000} \]
\[ c: \overline{110010000000000} \]
\[ t: \overline{010000000000000} \]

1fsr (length 4, taps 2):
1fsr: 1111
taps: 0 3

where the leftmost cell is number 0.
The initial cell values are the reverse of the leftmost 4 values of the input bit sequence.
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


## Distribution List

<table>
<thead>
<tr>
<th>Count</th>
<th>Line</th>
<th>Code</th>
<th>Name</th>
<th>Org.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
<td>0449</td>
<td>P. L. Campbell</td>
<td>6237</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>0449</td>
<td>R. S. Tamashiro</td>
<td>6237</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>0451</td>
<td>R. E. Trellue</td>
<td>6238</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>0612</td>
<td>Review &amp; Approval Desk For DOE/OSTI</td>
<td>4912</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>0741</td>
<td>S. G. Varnado</td>
<td>6200</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>0806</td>
<td>L. B. Dean</td>
<td>4616</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>0806</td>
<td>L. G. Pierson</td>
<td>4616</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>0806</td>
<td>T. D. Tarman</td>
<td>4616</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>0806</td>
<td>M. O. Vahle</td>
<td>4616</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>0806</td>
<td>E. L. Witzke</td>
<td>4616</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>0874</td>
<td>P. J. Robertson</td>
<td>1716</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>0874</td>
<td>K. L. Gass</td>
<td>6421</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>0899</td>
<td>Technical Library</td>
<td>4916</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>8903</td>
<td>P. W. Dean</td>
<td>9011</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>9018</td>
<td>Central Technical Files</td>
<td>8940-2</td>
</tr>
</tbody>
</table>