Generalised Berlekamp–Massey algorithm

M.-H. Cheng

Abstract: The Berlekamp–Massey algorithm is revisited and proven again by using the matrix representation. This approach makes the derivation and proof of the algorithm straightforward, simple and easily understood. It further enables the presentation of a generalised Berlekamp–Massey algorithm, including the conventional algorithm and the inversion-free algorithm as two special cases.

1 Introduction

The Berlekamp–Massey algorithm (BMA) [1, 2] is an efficient method for determining the error-locator polynomial in decoding the Reed–Solomon and BCH codes. Massey [2] formulated the problem of finding the error-locator polynomial from a sequence of syndromes as the problem of finding a linear feedback shift register (LFSR) for generating the sequence. The properties of LFSR are then employed for developing the famous BMA. The algorithm, however, needs to evaluate a sequence of inversion operations which usually require high realisation complexity, and therefore, the inversion-free BMA [3, 4] is proposed to overcome this drawback. In this paper we revisit the derivation of the BMA in [2], and formulate the problem of obtaining an LFSR for generating the syndrome sequence using the matrix representation. This approach makes the derivation of the BMA straightforward and easily understood, and the proof much more concise; we further obtain, from the proof, a generalised BMA which includes the conventional BMA [2] and the inversion-free BMA [3, 4] as two special cases. Note that previously the matrix approach [5] has been taken for describing the BMA, but it focuses merely on the conventional BMA and does not fully explore the benefits of this representation.

2 LFSR

Consider a general linear feedback shift register (LFSR) of length L with $c_0 \neq 0$ shown in Fig. 1, which generates an infinite data sequence $s_0, s_1, \cdots, s_{L-1}, s_L, \cdots$. The first L -data, $s_0, s_1, \cdots, s_{L-1}$, of the sequence are the initial data stored in each of the L cascaded registers; the data sequence after S_{L-1} should satisfy the following relation:

$$\sum_{i=0}^{L} c_i s_{j-i} = 0, \quad j = L, L+1, \cdots \tag{1}$$

Therefore, to design an LFSR of length L for generating a sequence with the first N data equal to $s_0, s_1, \cdots, s_{N-1}$, if $N \leq L$, using initial data setting can readily realise the function. If $N > L$, however, in addition to initial data setting, realising the function further requires the LFSR with coefficients $c_0, c_1, \cdots, c_L$, $c_0 \neq 0$, satisfying the equation below, in a matrix form:

$$\begin{bmatrix} s_0 & s_1 & \cdots & s_L \\ s_1 & s_2 & \cdots & s_{L+1} \\ \vdots & \vdots & \ddots & \vdots \\ s_{N-1-L} & s_{N-L} & \cdots & s_{N-1} \end{bmatrix} \begin{bmatrix} c_L \\ c_{L-1} \\ \vdots \\ c_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{2}$$

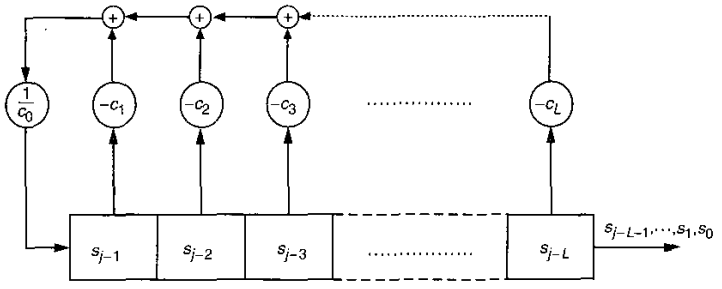

Fig. 1 *Linear feedback shift register*

One simple, important and useful property of an LFSR for developing the BMA is shown and presented in theorem 1 of [2], restated here and proven again using the matrix representation. The proof below, in essence, is identical to that in [2]; this proof, in my opinion, is much more concise, straightforward and clear, and thus more easily grasped.

Theorem 1 of [2]: If some LFSR of length L generates the sequence $s_0, s_1, \cdots, s_{N-1}$, but not the sequence

IEE Proceedings online no. 20020413

DOI: 10.1049/ip-com:20020413

Paper first received 20th June 2001 and in revised form 14th March 2002

The author is with the Department of Electrical and Control Engineering, National Chiao Tung University, 1001 Ta Hsueh Road, Hsinchu 300, Taiwan, Republic of China

$s_0, s_1, \cdots, s_{N-1}, s_N$, then any LFSR that generates the latter sequence has length L', satisfying

$$L' \geq N + 1 - L \tag{3}$$

Proof: For $L \geq N$, the theorem is trivially true. Hence, assume that $L < N$. Let the former and the latter LFSRs be with coefficients $c_0, c_1, \cdots, c_L$, $c_0 \neq 0$, and $c'_0, c'_1, \cdots, c'_{L'}$, $c'_0 \neq 0$, respectively. The first LFSR will satisfy the equation

$$\begin{bmatrix} s_0 & s_1 & \cdots & s_L \\ s_1 & s_2 & \cdots & s_{L+1} \\ \vdots & \vdots & \ddots & \vdots \\ s_{N-1-L} & s_{N-L} & \cdots & s_{N-1} \\ s_{N-L} & s_{N-L+1} & \cdots & s_N \end{bmatrix} \begin{bmatrix} c_L \\ c_{L-1} \\ \vdots \\ c_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ d \neq 0 \end{bmatrix} \tag{4}$$

Note that d is nonzero, because otherwise the LFSR generates $s_0, s_1, \cdots, s_N$. Since the second LFSR generates $s_0, s_1, \cdots, s_N$, it satisfies the following relation,

$$\begin{bmatrix} s_0 & s_1 & \cdots & s_{L'} \\ s_1 & s_2 & \cdots & s_{L'+1} \\ \vdots & \vdots & \ddots & \vdots \\ s_{N-1-L'} & s_{N-L'} & \cdots & s_{N-1} \\ s_{N-L'} & s_{N-L'+1} & \cdots & s_N \end{bmatrix} \begin{bmatrix} c'_{L'} \\ c'_{L'-1} \\ \vdots \\ c'_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix} \tag{5}$$

Assume first that $L' < N+1-L$, or equivalently $N-L' \geq L$. When $N-L' \geq L$, the matrix in (5) has its number of rows larger than L; hence we can multiply both sides of (5) on the left by the following row vector:

$$\underbrace{[0, \cdots, 0,}_{N-L'-L} c_L, c_{L-1}, \cdots, c_0]$$

This yields 0 on the right-hand side and, using the result of (4) dc'_0 on the left side. However, dc'_0 is not equal to zero, and the assumption $L' < N+1-L$, is therefore not valid, so the result $L' \geq N+1-L$ is proven.

Let L_n be the minimum of the lengths of all the LFSRs that can generate $s_0, s_1, \cdots, s_{n-1}$. Obviously, L_n is monotonically nondecreasing with increasing n. With this property and the result of theorem 1, it is obtained in [2] that if some LFSR of length L_n generates $s_0, s_1, \cdots, s_{n-1}$ but not $s_0, s_1, \cdots, s_{n-1}, s_n$, then

$$L_{n+1} \geq \max[L_n, n + 1 - L_n] \tag{6}$$

3 Berlekamp–Massey algorithm

The conventional BMA is presented in [2] not only to provide a way for computing an LFSR of the minimum length L_k for generating $s_0, s_1, \cdots, s_{k-1}$ for any $k \geq 0$, but also to show that the inequality in (6) is in fact an equality, i.e.

$$L_{k+1} = \max[L_k, k + 1 - L_k] \tag{7}$$

The proof presented here follows the procedure used in [2] but uses the matrix representation. The resulting algorithm is more general to include the conventional BMA and the inversion-free BMA as two special cases.

The proof of (7) is achieved by induction. Assume that L_k and its corresponding LFSR have been found and satisfy (7) for $k = 1, 2, \cdots, n$. Then we shall prove that L_{n+1} also satisfies (7) by showing that the corresponding LFSR can be computed. Let an LFSR of the minimum length L_n with coefficients $c_0^{(n)}, c_1^{(n)}, \cdots, c_{L_n}^{(n)}$ be denoted by a row vector

$$\boldsymbol{c}^{(n)} = [c_0^{(n)}, c_1^{(n)}, \cdots, c_{L_n}^{(n)}] \tag{8}$$

This LFSR generates $s_0, s_1, \cdots, s_{n-1}$; it also generates $s_0, s_1, \cdots, s_{n-1}, s_n$ if d_n in the following equation is equal to zero:

$$\begin{bmatrix} s_0 & s_1 & \cdots & s_{L_n} \\ s_1 & s_2 & \cdots & s_{L_n+1} \\ \vdots & \vdots & \ddots & \vdots \\ s_{n-1-L_n} & s_{n-L_n} & \cdots & s_{n-1} \\ s_{n-L_n} & s_{n-L_n+1} & \cdots & s_n \end{bmatrix} \begin{bmatrix} c_{L_n}^{(n)} \\ c_{L_n-1}^{(n)} \\ \vdots \\ c_0^{(n)} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ d_n \end{bmatrix} \tag{9}$$

Hence, if d_n is equal to zero, then $L_{n+1} = L_n$ and $\boldsymbol{c}^{(n+1)} = \boldsymbol{c}^{(n)}$. If d_n is not equal to zero, we shall provide below a way for computing $\boldsymbol{c}^{(n+1)}$ such that (7) is proven. The following information is needed for computing the LFSR. Let m be the sequence length before the *last length change* in the minimal-length registers, i.e.

$$L_m < L_n \tag{10}$$

$$L_{m+1} = L_n \tag{11}$$

$$= m + 1 - L_m \tag{12}$$

Note that the above relation is obtained by using (7) and because the length is changed. Moreover, because a length change is required, the LFSR of length L_m with coefficients $c_0^{(m)}, c_1^{(m)}, \cdots, c_{L_m}^{(m)}$, $c_0^{(m)} \neq 0$, generates $s_0, s_1, \cdots, s_{m-1}$, but not $s_0, s_1, \cdots, s_{m-1}, s_m$; thus, in matrix form:

$$\begin{bmatrix} s_0 & s_1 & \cdots & s_{L_m} \\ s_1 & s_2 & \cdots & s_{L_m+1} \\ \vdots & \vdots & \ddots & \vdots \\ s_{m-1-L_m} & s_{m-L_m} & \cdots & s_{m-1} \\ s_{m-L_m} & s_{m-L_m+1} & \cdots & s_m \end{bmatrix} \begin{bmatrix} c_{L_m}^{(m)} \\ c_{L_m-1}^{(m)} \\ \vdots \\ c_0^{(m)} \end{bmatrix}$$

$$= \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ d_m \neq 0 \end{bmatrix} \tag{13}$$

We then show in the following that for either $L_{n+1} = L_n$ or $L_{n+1} = n+1-L_n$, the corresponding LFSR $\boldsymbol{c}^{(n+1)}$ can be computed.

(i) $L_{n+1} = L_n$: When $L_{n+1} = L_n$, by (7) it means that $L_n \geq n+1-L_n$. Since $L_n = m+1-L_m$, as shown in (12), $m-L_m \geq n-L_n$. Hence, we can write down the following equation:

$$\begin{bmatrix} s_0 & s_1 & \cdots & s_{L_n} \\ s_1 & s_2 & \cdots & s_{L_n+1} \\ \vdots & \vdots & \ddots & \vdots \\ s_{n-1-L_n} & s_{n-L_n} & \cdots & s_{n-1} \\ s_{n-L_n} & s_{n-L_n+1} & \cdots & s_n \end{bmatrix}.$$

$$\begin{bmatrix} \left.\begin{matrix} 0 \\ \vdots \\ 0 \end{matrix}\right\} L_n - L_m + m - n \\ \left.\begin{matrix} c_{L_m}^{(m)} \\ c_{L_m-1}^{(m)} \\ \vdots \\ c_0^{(m)} \end{matrix}\right\} L_m + 1 \\ \left.\begin{matrix} 0 \\ \vdots \\ 0 \end{matrix}\right\} n - m \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ d_m \neq 0 \end{bmatrix} \tag{14}$$

which is obtained by extracting $n+1-L_n$ equations in the lower part of (13), enlarging the resulting matrix to make it identical to the matrix in (9), and concatenating zero vectors under and over the column vector. Note that (14) can be formed to include equations in the lower part of (13) because $m-L_m \geq n-L_n$. Since the matrix in (9) and the matrix in (14) are identical, multiplying (9) by k_n and multiplying (14) by k_m, and then adding them together yield

$$\begin{bmatrix} s_0 & s_1 & \cdots & s_{L_n} \\ s_1 & s_2 & \cdots & s_{L_n+1} \\ \vdots & \vdots & \ddots & \vdots \\ s_{n-1-L_n} & s_{n-L_n} & \cdots & s_{n-1} \\ s_{n-L_n} & s_{n-L_n+1} & \cdots & s_n \end{bmatrix}$$

$$\left\{ k_n \begin{bmatrix} c_{L_n}^{(n)} \\ c_{L_n-1}^{(n)} \\ \vdots \\ c_0^{(n)} \end{bmatrix} + k_m \begin{bmatrix} 0 \\ \vdots \\ 0 \\ c_{L_m}^{(m)} \\ c_{L_m-1}^{(m)} \\ \vdots \\ c_0^{(m)} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \right\} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix} \tag{15}$$

when the scalars k_n, k_m satisfy the relation,

$$k_n d_n = -k_m d_m \tag{16}$$

Therefore, an LFSR with length $L_{n+1} = L_n$ for generating $s_0, s_1, \ldots, s_n$ can be obtained,

$$\boldsymbol{c}^{(n+1)} = k_n \boldsymbol{c}^{(n)} + k_m [\underbrace{0, \cdots, 0}_{n-m}, \boldsymbol{c}^{(m)}, \overbrace{0, \cdots, 0}^{L_n - L_m + m - n}] \tag{17}$$

Note that since d_n and d_m are all non-zero, the only requirement for choosing k_n and k_m is that the condition (16) be met. Therefore, any choice of k_n, k_m satisfying $k_n/k_m = -d_m/d_n$ results in a method for computing the LFSR; this flexibility, discussed later, makes the developed algorithm more generalised.

(ii) $L_{n+1} = n+1-L_n$: Since $L_n = m+1-L_m$, we thus have $n-L_{n+1} = m-L_m$. Note that the LFSR in this case is of length L_{n+1}; (13) is then modified by enlarging the matrix of size from $(1+m-L_m) \times (1+L_m)$ to $(1+m-L_m) \times (1+L_{n+1})$ and concatenating the zeros in the column vectors, yielding the equation:

$$\begin{bmatrix} s_0 & s_1 & \cdots & s_{L_{n+1}} \\ s_1 & s_2 & \cdots & s_{L_{n+1}+1} \\ \vdots & \vdots & \ddots & \vdots \\ s_{n-1-L_{n+1}} & s_{n-L_{n+1}} & \cdots & s_{n-1} \\ s_{n-L_{n+1}} & s_{n-L_{n+1}+1} & \cdots & s_n \end{bmatrix} \begin{bmatrix} \left.\begin{matrix} c_{L_m}^{(m)} \\ c_{L_m-1}^{(m)} \\ \vdots \\ c_0^{(m)} \end{matrix}\right\} L_m + 1 \\ \left.\begin{matrix} 0 \\ \vdots \\ 0 \end{matrix}\right\} n-m \end{bmatrix}$$

$$= \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ d_m \neq 0 \end{bmatrix} \tag{18}$$

Also extracting $1+n-L_{n+1}$ equations in the lower part of (9), enlarging the resulting matrix to make it identical to the matrix in (18), and concatenating zeros in the column vector, we have:

$$\begin{bmatrix} s_0 & s_1 & \cdots & s_{L_{n+1}} \\ s_1 & s_2 & \cdots & s_{L_{n+1}+1} \\ \vdots & \vdots & \ddots & \vdots \\ s_{n-1-L_{n+1}} & s_{n-L_{n+1}} & \cdots & s_{n-1} \\ s_{n-L_{n+1}} & s_{n-L_{n+1}+1} & \cdots & s_n \end{bmatrix}$$

$$\begin{bmatrix} \left.\begin{matrix} 0 \\ \vdots \\ 0 \end{matrix}\right\} L_{n+1} - L_n \\ \left.\begin{matrix} c_{L_n}^{(n)} \\ c_{L_n-1}^{(n)} \\ \vdots \\ c_0^{(n)} \end{matrix}\right\} 1 + L_n \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ d_n \neq 0 \end{bmatrix} \tag{19}$$

Note again that (19) can be formed to include equations in the lower part of (9) because in this case $n-L_n \geq n-L_{n+1}$. Similarly, choosing scalars k_n and k_m that satisfy (16), multiplying (19) by k_n, multiplying (18) by k_m, and adding them together, we have:

$$\begin{bmatrix} s_0 & s_1 & \cdots & s_{L_{n+1}} \\ s_1 & s_2 & \cdots & s_{L_{n+1}+1} \\ \vdots & \vdots & \ddots & \vdots \\ s_{n-1-L_{n+1}} & s_{n-L_{n+1}} & \cdots & s_{n-1} \\ s_{n-L_{n+1}} & s_{n-L_{n+1}+1} & \cdots & s_n \end{bmatrix}$$

$$\left\{ k_n \begin{bmatrix} 0 \\ \vdots \\ 0 \\ c_{L_n}^{(n)} \\ c_{L_n-1}^{(n)} \\ \vdots \\ c_0^{(n)} \end{bmatrix} + k_m \begin{bmatrix} c_{L_m}^{(m)} \\ c_{L_m-1}^{(m)} \\ \vdots \\ c_0^{(m)} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \right\} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix} \tag{20}$$

Therefore, an LFSR with length L_{n+1} for generating $s_0, s_1, \cdots, s_n$, in this case, can be computed as follows,

$$\boldsymbol{c}^{(n+1)} = k_n [\boldsymbol{c}^{(n)}, \underbrace{0, \cdots, 0}_{L_{n+1}-L_n}] + k_m [\underbrace{0, \cdots, 0}_{n-m}, \boldsymbol{c}^{(m)}] \tag{21}$$

The result (7) is thus proven and a generalised algorithm for computing an LFSR for generating the syndrome sequence is provided.

Table 1: Illustrative example

Iteration	Conventional BMA	Inverse-free BMA	New approach
$n=0$	$\boldsymbol{c}^{(0)}=[1]$	$\boldsymbol{c}^{(0)}=[1]$	$\boldsymbol{c}^{(0)}=[1]$
$n=1$	$[1, 0]+(\alpha^0/\alpha^3)[0, 1]$ $=[1, \alpha^4]=\boldsymbol{c}^{(1)}$	$\alpha^3[1, 0]+\alpha^0[0, 1]$ $=[\alpha^3, \alpha^0]=\boldsymbol{c}^{(1)}$	$\alpha^3[1, 0]+\alpha^0[0, 1]$ $=[\alpha^3, \alpha^0]=\boldsymbol{c}^{(1)}$
$n=2$	$[1, \alpha^4, 0]+(\alpha^0/\alpha^3)[0, 0, 1]$ $=[1, \alpha^4, \alpha^5]=\boldsymbol{c}^{(2)}$	$\alpha^3[\alpha^3, \alpha^0, 0]+\alpha^4[0, 0, 1]$ $=[\alpha^6, \alpha^3, \alpha^4]=\boldsymbol{c}^{(2)}$	$\alpha^4[\alpha^3, \alpha^0, 0]+\alpha^5[0, 0, 1]$ $=[\alpha^0, \alpha^4, \alpha^5]=\boldsymbol{c}^{(2)}$
$n=3$	$[1, \alpha^4, \alpha^5]+(\alpha/\alpha)[0, 1, \alpha^4]$ $=[1, \alpha^5, 1]=\boldsymbol{c}^{(3)}$	$\alpha^4[\alpha^6, \alpha^3, \alpha^4]+\alpha^0[0, \alpha^3, \alpha^0]$ $=[\alpha^3, \alpha, \alpha^3]=\boldsymbol{c}^{(3)}$	$\alpha^3[\alpha^0, \alpha^4, \alpha^5]+\alpha^0[0, \alpha^3, \alpha^0]$ $=[\alpha^3, \alpha, \alpha^3]=\boldsymbol{c}^{(3)}$

3.1 Generalised algorithm

The detailed flow of the generalised algorithm is not listed here because it is mainly identical to that in [2], except that in computing the LFSR using (17) or (21), the flexibility for choosing k_n and k_m is provided. Therefore, the algorithm is turned into the conventional BMA [2] if we select $k_n=1$ and $k_m=-d_n/d_m$; this choice, of course, should satisfy (16). Note that the inversion operation required in the conventional BMA is in evaluating the scalar k_m. Also, the algorithm becomes the inversion-free BMA [3, 4] when $k_n=d_m$ and $k_m=-d_n$; this algorithm avoids the inversion operation, but requires, however, a larger number of multiplications in evaluating (17) or (21). Note that in each of the above two special cases, k_n and k_m are chosen from d_n and d_m at each iteration according to only one fixed relation. The generalised algorithm, however, allows at each iteration a varying choice of k_n and k_m only if their values satisfy (16). We give an example below to illustrate that this flexibility may enable further simplification for realisation or implementation of the algorithm.

3.2 An example

Consider a (7; 3) Reed–Solomon code over GF(2^3) using $p(x)=1+x+x^3$ as the primitive polynomial, and α as its primitive field element. The error polynomial in this example is given by

$$e(x)=\alpha^0x+\alpha x^6 \tag{22}$$

Hence, the syndromes can be obtained:

$$s_0=e(\alpha)=\alpha^3 \tag{23}$$

$$s_1=e(\alpha^2)=1 \tag{24}$$

$$s_2=e(\alpha^3)=\alpha^2 \tag{25}$$

$$s_3=e(\alpha^4)=0 \tag{26}$$

The main operations in the iteration of the conventional BMA, inverse-free BMA, and one realisation of the proposed approach for the above example are listed in Table 1. Each of the three algorithms results in a correct error-locator polynomial. Note that in the new approach $d_m=\alpha^3$ and $d_n=\alpha^4$ at iteration $n=2$; here we choose $k_n=\alpha^4$, $k_m=-\alpha^5$ not only to satisfy (16) but also to save one multiplication in $k_n[\alpha^3, \alpha^0, 0]$ because $k_n\alpha^3=\alpha^7=1$. Similarly, $d_m=\alpha^4$, $d_n=\alpha$ at iteration $n=3$, then $k_n=\alpha^3$; $k_m=-\alpha^0$ are chosen to save one multiplication in $k_n[\alpha^0, \alpha^4, \alpha^5]$ because $k_n\alpha^4=1$. Therefore, a sensible choice of k_n and k_m in the iteration may lower the realisation complexity of the algorithm.

4 Conclusions

The BMA is revisited and shown by using the matrix representation. This treatment is straightforward, concise and clear, making the algorithm easily understood. The approach further enables us to present a generalized BMA, including the conventional BMA and the inversion-free BMA as two special cases. Therefore, the approach of using the matrix representation and the developed algorithm, with its simplicity and generalisation, are useful for understanding and implementing the BMA.

5 References

1 BERLEKAMP, E.R.: 'Algebraic coding theory' (McGraw-Hill, New York, 1968)

2 MASSEY, J.L.: 'Shift-register synthesis and BCH decoding', *IEEE Trans. Inf. Theory*, Jan. 1969, **IT-15**, (1), pp. 122–127

3 YOUZHI, X.: 'Implementation of Berlekamp-Massey algorithm without inversion,' *IEE Proc. I, Commun. Speech Vis.*, June 1991, **138**, (3), pp. 138–140

4 REED, I.S., SHIH, M.T., and TRUONG, T.K.: 'VLSI design of inverse-free Berlekamp–Massey algorithm,' *IEE Proc. E, Comput. Digit. Tech.*, Sept. 1991, **138**, (5), pp. 295–298

5 HENKEL, W.: 'Another description of the Berlekamp-Massey algorithm,' *IEE Proc. I, Commun. Speech Vis.*, 1989, **136**, (3), pp. 197–200