A Reed–Solomon Product-Code (RS-PC) Decoder Chip for DVD Applications

Hsie-Chia Chang, C. Bernard Shung, *Member, IEEE*, and Chen-Yi Lee

***Abstract*—In this paper, a Reed–Solomon Product-Code (RS-PC) decoder for DVD applications is presented. It mainly contains two frame-buffer controllers, a (182, 172) row RS decoder, and a (208, 192) column RS decoder. The RS decoder features an area-efficient key equation solver using a novel modified *decomposed* inversionless Berlekamp–Massey algorithm.**

The proposed RS-PC decoder solution was implemented using 0.6-μm CMOS single-poly double-metal (SPDM) standard cells. The chip size is 4.22×3.64 mm^2 with a core area of 2.90×2.88 mm^2, where the total gate count is about 26K. Test results show that the proposed RS-PC decoder chip can support $4\times$ DVD speed with off-chip frame buffers or $8\times$ DVD speed with embedded frame buffers operating at 3 V.

***Index Terms*—Reed–Solomon Product-Code decoder, DVD, decomposed inversionless Berlekamp–Massey algorithm.**

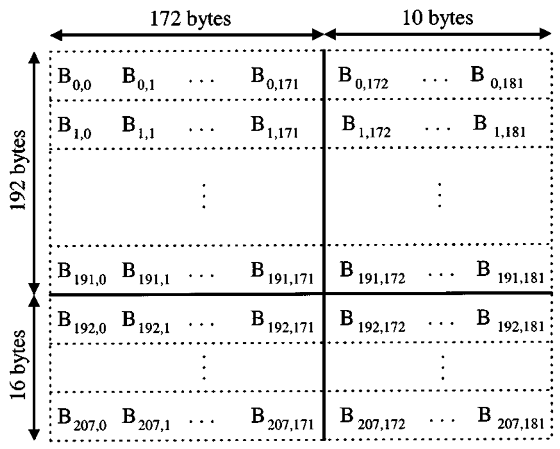

172 bytes
10 bytes
192 bytes
16 bytes
$B_{0,0}$ $B_{0,1}$ ⋯ $B_{0,171}$ $B_{0,172}$ ⋯ $B_{0,181}$
$B_{1,0}$ $B_{1,1}$ ⋯ $B_{1,171}$ $B_{1,172}$ ⋯ $B_{1,181}$
$B_{191,0}$ $B_{191,1}$ ⋯ $B_{191,171}$ $B_{191,172}$ ⋯ $B_{191,181}$
$B_{192,0}$ $B_{192,1}$ ⋯ $B_{192,171}$ $B_{192,172}$ ⋯ $B_{192,181}$
$B_{207,0}$ $B_{207,1}$ ⋯ $B_{207,171}$ $B_{207,172}$ ⋯ $B_{207,181}$

Fig. 1. DVD RS-PC frame structure.

I. Introduction

DUE TO increasing demand for high-quality video and audio consumer products, the digital versatile disc (DVD) was standardized in 1995 to provide higher storage capacity by leading industrial consortion. In order to mitigate the errors that may be introduced during manufacturing or by user damage, a Reed–Solomon Product-Code (RS-PC) is used in DVD for error correction. In this paper, we report a RS-PC decoder chip for DVD applications.

As illustrated in Fig. 1, the DVD RS-PC is composed of a (182, 172) RS code in the row direction and (208, 192) RS code in the column direction. We will refer to the 208×182 matrix in Fig. 1 as a *frame*. A (N, K) RS code contains K message symbols and $N - K$ parity checking symbols, and is capable of correcting up to $t = \lfloor (N-K)/2 \rfloor$ symbol errors. For (182, 172) and (208, 192) RS codes, each symbol is one byte.

The most popular RS decoder architecture today, [1], [2] can be summarized into four steps: 1) calculating the *syndromes* from the received codeword; 2) computing the *error locator polynomial* and the *error evaluator polynomial*; 3) finding the error locations; and 4) computing error values. The second step in the four-step procedure involves solving the *key equation*[1] [1], which is

$$S(x)\sigma(x) = \Omega(x) \bmod x^{N-K} \tag{1}$$

where $S(x)$ is the syndrome polynomial, $\sigma(x)$ is the error locator polynomial, and $\Omega(x)$ is the error evaluator polynomial.

While there has been a lot of research work reported on RS decoder designs, there has been little on RS-PC decoders. The architectural design of the RS-PC decoders is different from that of the RS decoders in the following ways. First, each symbol in the RS-PC decoder is subject to a row RS decoding and a column RS decoding. Since the column RS decoding cannot proceed until all the row RS decodings in the frame are finished, a *frame buffer* is required to parallelize the row and column decoding. Second, in most RS decoder designs, *line buffers* in the form of shift registers are used to store the received symbols when the error locations and error values are computed. When the code size is large, these line buffers constitute a major portion of the hardware complexity. In a RS-PC decoder, however, we can exploit the frame buffers by cleverly arranging the accessing pattern and eliminating the need of the line buffers. Third, it is a design choice whether to implement a *programmable* RS decoder which can serve as row RS decoding and column RS decoding at different times, or implement one *dedicated* row RS decoder and one dedicated column RS decoder. All these design considerations will be explained in more detail in this paper.

Manuscript received March 17, 2000; revised August 15, 2000. This work was supported in part by the NSC of Taiwan, R.O.C., under Grant NSC89-2215-E-029-053. This paper was presented at the International Solid-State Circuits Conference, San Francisco, CA, February 1998.

H.-C. Chang and C.-Y. Lee are with the Department of Electronics Engineering, National Chiao Tung University, Hsinchu, Taiwan, R.O.C.

C. B. Shung was with the Department of Electronics Engineering, National Chiao Tung University, Hsinchu, Taiwan, R.O.C. He is currently with Allayer Communications Corporation, San Jose, CA 95134 USA.

Publisher Item Identifier S 0018-9200(01)00929-5.

[1]In fact, the key equation defined in [1] was $(1 + S(x))\sigma(x) = \Omega(x) \bmod x^{N-K}$, where the syndrome polynomial was defined to be $S(x) = \sum S_j x^j$. In our notation which follows [4], $S(x) = \sum S_j x^{j-1}$, and hence our key equation is slightly different.

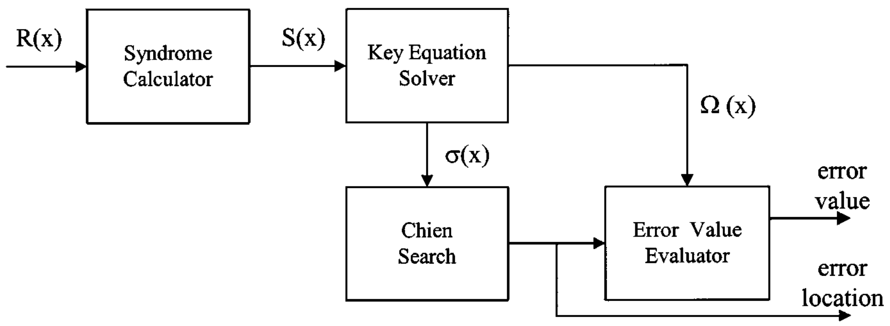


Fig. 2. Reed–Solomon decoding flowchart.

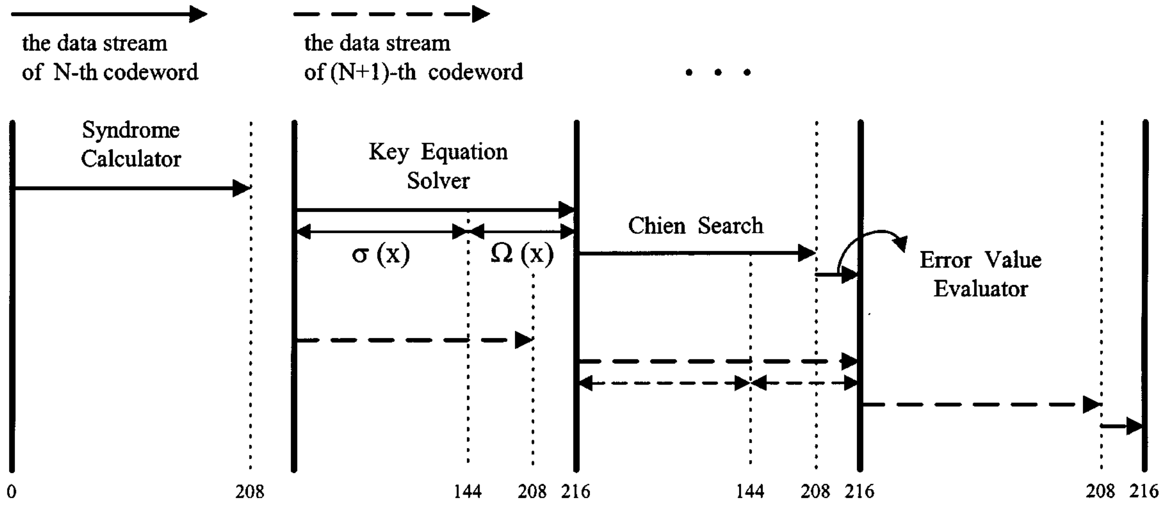


Fig. 3. Three-stage pipelining of the RS decoder using the column RS decoder as the example.

Section II describes the RS decoder architecture. Our RS decoder architecture is shown in Fig. 2 which contains a syndrome calculator, a key equation solver, a Chien search and an error value evaluator. We present a novel implementation of the key equation solver which helps to reduce the hardware complexity significantly. Section III describes the dual-frame-buffer architecture of the RS-PC decoder. We explain the control flow and data flow of the two frame-buffer controllers. Section IV shows the chip implementation and chip testing. Finally, in Section V we conclude the paper.

II. RS Decoder Design

As shown in Fig. 2, we divide the decoding process into four steps. The syndrome calculator calculates a set of syndromes from the received codewords. From the syndromes, the key equation solver produces the error locator polynomial $\sigma(x)$ and the error value evaluator polynomial $\Omega(x)$, used by the Chien search and the error value evaluator to produce the error locations and error values, respectively. In Fig. 3, we illustrate the three-stage pipelining used in (208, 192) column RS decoder.

In our RS decoder, an inversionless Berlekamp–Massey algorithm is adopted which not only eliminates the finite-field inverter (FFI) but also introduces additional parallelism. We discover a clever scheduling of *three* finite-field multipliers to implement the algorithm, which is named as *decomposed* inversionless Berlekamp–Massey algorithm here. Because of the decomposed algorithm, a specified sequence is added to the syndrome calculator, and we will illustrate the modification in Section II-A. In Sections II-C and II-D, we introduce how to calculate error locations and error values.

A. Syndrome Calculator

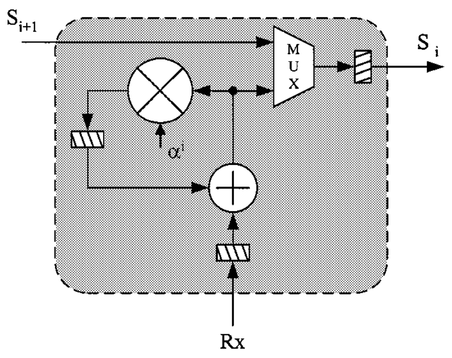
By definition the syndrome polynomial is $S(x) = \sum_{i=1}^{2t} S_i x^{i-1}$, $S_i = R(\alpha^i)$, where $R(x) = R_0 + R_1 x + R_2 x^2 + \cdots + R_{N-1} x^{N-1}$ is a received polynomial and R_{N-1} is the first received symbol into a syndrome cell illustrated in Fig. 4(a).

As shown in Fig. 4(a), at each cycle, the partial syndrome is multiplied with α^i and accumulated with the received symbol. After all the received symbols are processed, the accumulated result is the ith syndrome. The upper side of Fig. 4(a) indicates a way to connect multiple syndrome cells to generate a controllable sequence of syndrome results.

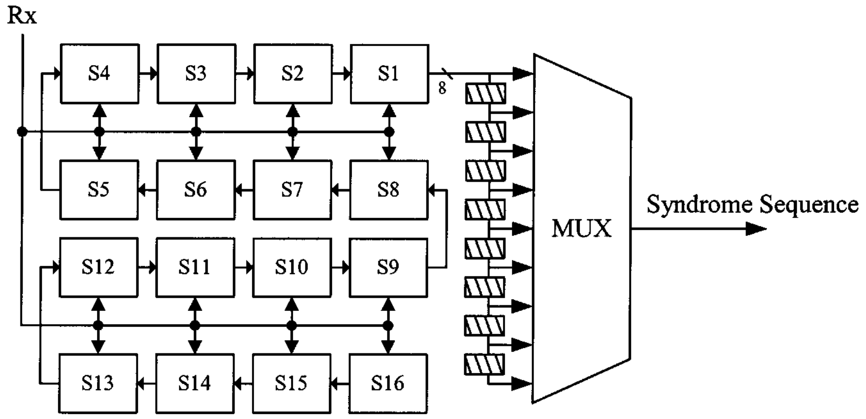
Fig. 4(b) shows how the 16 syndrome cells (for $t = 8$) are organized in our chip. By controlling the multiplexer in Fig. 4(b), we can generate different syndrome sequences for the calculation of the discrepancy $\Delta^{(i)}$ in the key equation solver. Table I shows all 16 different syndrome sequences.

B. Key Equation Solver

The techniques frequently used to solve the key equation include the Berlekamp–Massey algorithm [1], [5], the Euclidean



(a)



(b)

Fig. 4. (a)Syndrome cell S_i. (b) Syndrome calculator cell structure and its buffer. Note that for simplification, we take $t = 8$.

TABLE I
THE 16 SYNDROME SEQUENCES REQUIRED TO CALCULATE $\Delta^{(i)}$

$\Delta^{(i)}$	Syndrome Sequence
$i = 0$	S_1
$i = 1$	S_2, S_1
$\vdots$	$\vdots$
$i = 7$	$S_8, S_7, \cdots, S_1$
$i = 8$	$S_9, S_8, \cdots, S_2, S_1$
$i = 9$	$S_{10}, S_9, \cdots, S_3, S_2$
$\vdots$	$\vdots$
$i = 15$	$S_{16}, S_{15}, \cdots\cdots, S_8$

algorithm [2], and the continuous-fraction algorithm [3]. Compared to the other two algorithms, the Berlekamp–Massey algorithm is generally considered to be the one with the least hardware complexity [6]. Another advantage of the Berlekamp–Massey algorithm is that it can be formulated to compute $\sigma(x)$ only and the computation of $\Omega(x)$ is similiar to discrepancy Δ, thus saving a portion of the hardware used to compute $\Omega(x)$.

Existing architectures to implement the Berlekamp–Massey algorithm in hardware were proposed by Berlekamp [7], Liu [8], and Oh and Kim [9]. These proposals require $2t \sim 3t$ finite-field multiplications (FFMs) where t is the number of correctable errors. In addition, they all require an FFI to implement the division operation. An inversionless Berlekamp–Massey algorithm was proposed by Burton [10] for BCH decoders, and was implemented by Reed, Shih, and Truong [6] for BCH and RS codes. However, more FFMs are required in the existing implementation of the inversionless Berlekamp–Massey algorithm [6].

1) Decomposed Inversionless Berlekamp–Massey Algorithm: An inversionless Berlekamp–Massey algorithm is adopted in our architecture that is a $2t$-step iterative algorithm, as shown in the following:

```
Initial condition:
        $D^{(-1)} = 0, \quad \delta = 1,$
        $\sigma^{(-1)}(x) = \tau^{(-1)}(x) = 1,$
        $\Delta^{(0)} = S_1$
 for ($i = 0$ to $2t - 1$)
    $\sigma^{(i)}(x) = \delta \cdot \sigma^{(i-1)}(x) + \Delta^{(i)} x \tau^{(i-1)}(x),$
    $\Delta^{(i+1)} = S_{i+2}\sigma_0^{(i)} + S_{i+1}\sigma_1^{(i)} + \cdots + S_{i-t+2}\sigma_t^{(i)}$
   If ($\Delta^{(i)} = 0$ or $2D^{(i-1)} \geq i + 1$) then
      $D^{(i)} = D^{(i-1)}, \quad \tau^{(i)}(x) = x\tau^{(i-1)}(x);$
   else
  $D^{(i)} = i + 1 - D^{(i-1)}, \quad \delta = \Delta^{(i)}, \quad \tau^{(i)}(x) = \sigma^{(i-1)}(x)$
```

where $\sigma^{(i)}(x)$ is the ith step error locator polynomial and $\sigma_j^{(i)}$'s are the coefficients of $\sigma^{(i)}(x)$; $\Delta^{(i)}$ is the ith step discrepancy and δ is a previous nonzero discrepancy; $\tau^{(i)}(x)$ is an

TABLE II
DATA DEPENDENCY OF THE INVERSIONLESS BERLEKAMP–MASSEY ALGORITHM AFTER DECOMPOSITION

cycle	$\Delta^{(i+1)}$	$\sigma^{(i)}(x)$
$j=0$	$\Delta^{(i)} = \Delta_t^{(i)} + S_{i-t+1}\sigma_t^{(i-1)}$	
	$\Delta_0^{(i+1)} = 0$	$\sigma_0^{(i)} = \delta \cdot \sigma_0^{(i-1)}$
$j=1$	$\Delta_1^{(i+1)} = S_{i+2}\sigma_0^{(i)}$	$\sigma_1^{(i)} = \delta \cdot \sigma_1^{(i-1)} + \Delta^{(i)} \cdot \tau_0^{(i-1)}$
$j=2$	$\Delta_2^{(i+1)} = \Delta_1^{(i+1)} + S_{i+1}\sigma_1^{(i)}$	$\sigma_2^{(i)} = \delta \cdot \sigma_2^{(i-1)} + \Delta^{(i)} \cdot \tau_1^{(i-1)}$
$\vdots$	$\vdots$	$\vdots$
$j=t$	$\Delta_t^{(i+1)} = \Delta_{t-1}^{(i+1)} + S_{i-t+3}\sigma_{t-1}^{(i)}$	$\sigma_t^{(i)} = \delta \cdot \sigma_t^{(i-1)} + \Delta^{(i)} \cdot \tau_{t-1}^{(i-1)}$

auxiliary polynomial and $D^{(i)}$is an auxiliary degree variable in ith step.

Define

$$\sigma_j^{(i)} = \begin{cases} \delta \cdot \sigma_0^{(i-1)}, & \text{for } j=0 \\ \delta \cdot \sigma_j^{(i-1)} + \Delta^{(i)}\tau_{j-1}^{(i-1)}, & \text{for } 1 \le j \le t \end{cases} \tag{2}$$

$$\Delta_j^{(i+1)} = \begin{cases} 0, & \text{for } j=0 \\ \Delta_{j-1}^{(i+1)} + S_{i-j+3} \cdot \sigma_{j-1}^{(i)}, & \text{for } 1 \le j \le t \end{cases} \tag{3}$$

where $\sigma^{(i)}(x) = \sigma_0^{(i)} + \sigma_1^{(i)}x + \cdots + \sigma_t^{(i)}x^t$, $\tau_j^{(i)}$'s are the coefficients of $\tau^{(i)}(x)$, and $\Delta_j^{(i+1)}$'s are the *partial results* in computing $\Delta^{(i+1)}$. At the first cycle of $(i+1)$th step, we get

$$\begin{aligned} \Delta^{(i+1)} &= \Delta_t^{(i+1)} + S_{i-t+2}\sigma_t^{(i)} \\ &= \cdots \\ &= S_{i+2}\sigma_0^{(i)} + S_{i+1}\sigma_1^{(i)} + \cdots + S_{i-t+2}\sigma_t^{(i)}. \end{aligned} \tag{4}$$

In other words, we can *decompose* the ith iteration into $t+1$ *cycles*. In each cycle $\sigma_j^{(i)}$ requires at most two FFMs and $\Delta_j^{(i+1)}$ requires only one FFM. The data dependency of the decomposed algorithm can be seen in Table II.

It is evident from Table II that, at cycle j, the computation of $\Delta_j^{(i+1)}$ requires $\sigma_{j-1}^{(i)}$ and $\Delta_{j-1}^{(i+1)}$, which have been computed at cycle $(j-1)$. Similarly, at cycle j, the computation of $\sigma_j^{(i)}$ requires $\Delta^{(i)}$ and $\sigma_j^{(i-1)}$, which have been computed at cycle 0 and the $(i-1)$th step, respectively. Note that the original Berlekamp–Massey algorithm cannot be scheduled as efficiently because the computation, $\sigma^{(i)}(x) = \sigma^{(i-1)}(x) + (\Delta^{(i)}/\delta) \cdot x \cdot \tau^{(i-1)}(x)$, $\sigma_j^{(i)}$ requires two *sequential* multiplications and one inversion. The inversionless Berlekamp–Massey algorithm provides the necessary parallelism to allow our efficient scheduling. The scheduling and data dependency of the *decomposed* algorithm are further illustrated in Fig. 5.

The decomposed algorithm shown above suggests a three-FFM implementation of the inversionless Berlekamp–Massey algorithm, which is shown in Fig. 6. Compared to the previously proposed parallel architectures [6]–[9] our architecture reduces the hardware complexity significantly. Compared to a previously proposed serial architecture [11], our architecture reduces the time complexity significantly because of the reduction of cycle time and the number of clock cycles. Therefore, our proposed architecture achieves an optimization in the area-delay product.

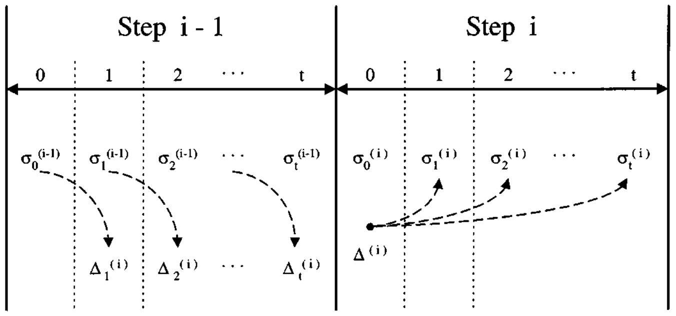


Fig. 5. Scheduling and data dependency of the decomposed inversionless Berlekamp–Massey algorithm. The dotted line represents the data dependency.

Dual-basis finite-field arithmetic is adopted in the key equation solver for lower gate count [12]. A dual-basis FFM takes one input in standard basis and the other input in dual basis to produce a dual-basis output. In Fig. 6, the dotted lines correspond to the data symbols in dual basis while the solid lines correspond to the data symbols in standard basis, and D2S is a dual-to-standard basis converter.

2) Efficient Computation of $\Omega(x)$: The conventional way to compute the error evaluator polynomial $\Omega(x)$ is to do it in parallel with the computation of $\sigma(x)$. Using the Berlekamp–Massey algorithm, this involves an iterative algorithm to compute $\Omega^{(i)}(x)$, $i = 0, \ldots, 2t-1$. However, if $\sigma(x)$ is first obtained, from the key equation and the Newton's identity we could derive $\Omega(x)$ as follows:

$$\begin{aligned} \Omega(x) &= S(x)\sigma(x) \bmod x^{2t} \\ &= (S_1 + S_2x + \cdots + S_{2t}x^{2t-1}) \\ &\quad \cdot (\sigma_0 + \sigma_1x + \cdots + \sigma_tx^t) \bmod x^{2t} \\ &\triangleq \Omega^{(0)} + \Omega^{(1)}x + \cdots + \Omega^{(t-1)}x^{t-1} \end{aligned} \tag{5}$$

$$\begin{aligned} \Omega^{(i)} &= S_{i+1}\sigma_0 + S_i\sigma_1 + \cdots + S_1\sigma_i, \\ i &= 0, 1, \ldots, t-1. \end{aligned} \tag{6}$$

That is, the computation of $\Omega(x)$ can be performed *directly* after $\sigma(x)$ is computed. Note that the direct computation requires fewer multiplications than the iterative algorithm which computes many *unnecessary* intermediate results. The penalty of this efficient computation is the additional latency because $\sigma(x)$ and $\Omega(x)$ are computed in sequence.

Furthermore, it can be seen that the computation of $\Omega^{(i)}$ is very similar to that of $\Delta^{(i)}$ except for some minor differences. Therefore, the same hardware used to compute $\sigma(x)$ can be *reconfigured* to compute $\Omega(x)$ after $\sigma(x)$ is computed. Like $\Delta_j^{(i)}$,

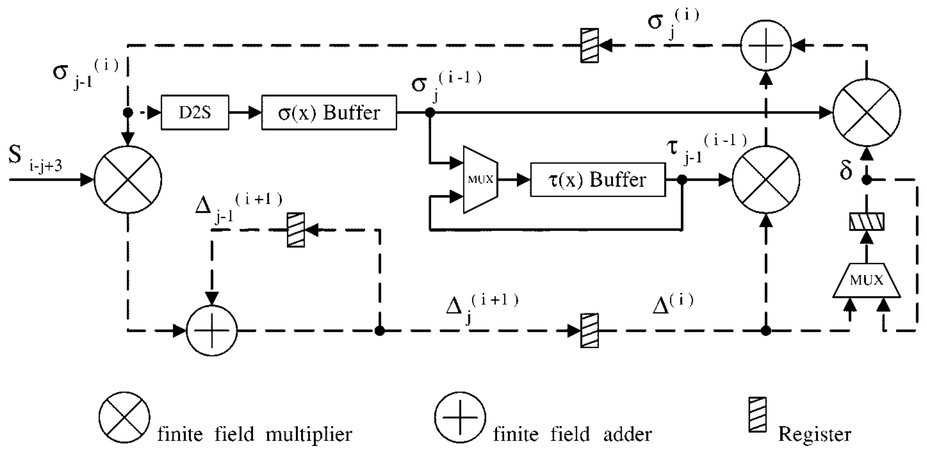


Fig. 6. Three-FFM architecture for implementing the decomposed inversionless Berlekamp–Massey algorithm. Note that the $\sigma(x)$ buffer stores final coefficients of $\sigma(x)$ for the standard basis.

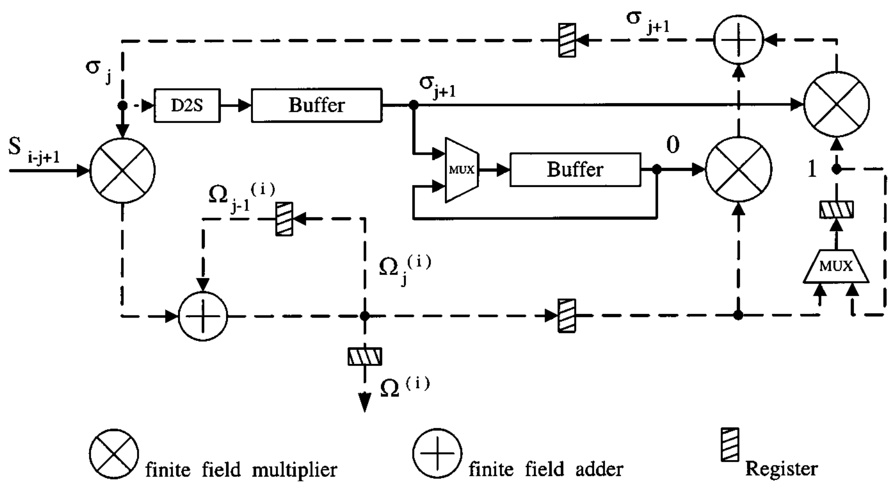


Fig. 7. Three-FFM architecture reconfigured to compute $\Omega(x)$. Note the labels in this figure are different from those in Fig. 6.

$\Omega_j^{(i)}$'s are the partial results in computing $\Omega^{(i)}$ and we could derive it as follows:

$$\Omega_j^{(i)} = \begin{cases} S_{i+1}\sigma_0, & \text{for } j = 0 \\ \Omega_{j-1}^{(i)} + S_{i-j+1}\sigma_j, & \text{for } 1 \le j \le i. \end{cases} \tag{7}$$

At the last cycle of the ith iteration in (7), $\Omega_i^{(i)} = \Omega_{i-1}^{(i)} + S_1\sigma_i = \cdots = \Omega^{(i)}$. In Fig. 7, we show how the same three-FFM architecture can be reconfigured to compute $\Omega(x)$.

3) Application Conditions for Errors and Erasures: For decoding errors and erasures, the key equation is modified to $T(x)\sigma(x) = \hat{\Omega}(x) \bmod x^{N-K}$, where $\hat{\Omega}(x)$ is the errata evaluator polynomial, $T(x) = S(x)\Lambda(x) \bmod x^{N-K}$ is the Forney syndrome polynomial, and $\Lambda(x) = \prod_{\alpha^i\in\Lambda}(1+\alpha^i x)$ and Λ is the erasure set [1]. Furthermore, we could rewrite the inversionless Berlekamp–Massey algorithm as follows:

```
Initial condition:
    $D^{(\rho-1)} = 0, \quad \hat{\delta} = 1,$
    $\hat{\sigma}^{(\rho-1)}(x) = \hat{\tau}^{(\rho-1)}(x) = \Lambda(x),$
    $\hat{\Delta}^{(\rho)} = S_{\rho+1}\hat{\sigma}_0^{(\rho-1)} + S_\rho\hat{\sigma}_1^{(\rho-1)} + \cdots + S_1\hat{\sigma}_\rho^{(\rho-1)}$
    $= S_{\rho+1}\Lambda_0 + S_\rho\Lambda_1 + \cdots + S_1\Lambda_\rho$
for $i = \rho$ to $N-K-1$
    $\hat{\sigma}^{(i)}(x) = \hat{\delta}\cdot\hat{\sigma}^{(i-1)}(x) - \hat{\Delta}^{(i)}x\hat{\tau}^{(i-1)}(x),$
    $\hat{\Delta}^{(i+1)} = S_{i+2}\hat{\sigma}_0^{(i)} + S_{i+1}\hat{\sigma}_1^{(i)} + \cdots + S_{i-\nu_i+2}\hat{\sigma}_{\nu_i}^{(i)}$
    If $\hat{\Delta}^{(i)} = 0$ or $2D^{(i-1)} \ge i+1$
        $D^{(i)} = D^{(i-1)}, \quad \hat{\tau}^{(i)}(x) = x\hat{\tau}^{(i-1)}(x);$
    else
        $D^{(i)} = i+1-D^{(i-1)}, \quad \hat{\delta} = \hat{\Delta}^{(i)}, \quad \hat{\tau}^{(i)}(x) = \hat{\sigma}^{(i-1)}(x)$
```

where ρ is the number of erasures, $\hat{\sigma}^{(i)}(x)$ is the ith step errata locator polynomial with degree ν_i, and $\hat{\sigma}_j^{(i)}$'s are the coefficients of $\hat{\sigma}^{(i)}(x)$.

Let us now calculate the total number of cycles required to compute $\hat{\sigma}(x)$ and $\hat{\Omega}(x)$ using our decomposed architecture. It is clear that the degree of $\hat{\sigma}^{(i)}(x)$ at most increases by one during each iteration. Therefore, we use $\nu_i = \nu_{i-1}+1$ to set the upper bound of ν_i.

Because both errors and erasures are corrected, we need $\rho+1$ cycles to compute the initial $\hat{\Delta}^{(\rho)}$ and we have

$$\begin{cases} \nu_i \le \rho + i, & \text{for } 0 \le i < t \\ \nu_i \le \rho + t, & \text{for } t \le i < 2t. \end{cases} \tag{8}$$

TABLE III
NUMBER OF CYCLES REQUIRED TO IMPLEMENT THE INVERSIONLESS BERLEKAMP–MASSEY ALGORITHM FOR AN (N, K) RS CODE USING OUR DECOMPOSED ALGORITHM

$N-K$	t	ρ	cycles	t	ρ	cycles
4	2	-	13	1	2	16
6	3	-	25	1	4	31
8	4	-	41	2	4	51
10	5	-	61	2	6	76
12	6	-	85	3	6	106
14	7	-	113	3	8	141
16	8	-	145	4	8	181

The number of cycles to compute $\hat{\sigma}(x)$ is

$$\sum_{i=0}^{2t-1}(\nu_i+1) \le \sum_{i=0}^{t-1}(\rho+i+1)+\sum_{i=t}^{2t-1}(\rho+t+l) = \frac{3}{2}t^2+\left(2\rho+\frac{3}{2}\right)t. \quad (9)$$

The number of cycles required to compute $\hat{\Omega}(x)$ is

$$\sum_{i=0}^{t+\rho-1}(i+1) = \frac{1}{2}t^2+\left(\rho+\frac{1}{2}\right)t+\frac{1}{2}\rho(\rho+1). \quad (10)$$

Hence the total number of cycles is less than $2t^2+(3\rho+2)t+(1/2)\rho(\rho+1)+\rho+1$.

Table III shows the maximum number of cycles for different (N, K) RS codes with $N-K$ ranging from 4 to 16. If N is larger than the number of cycles required, then our area-efficient architecture can be applied to reduce the hardware complexity while maintaining the overall decoding speed.

C. Chien Search

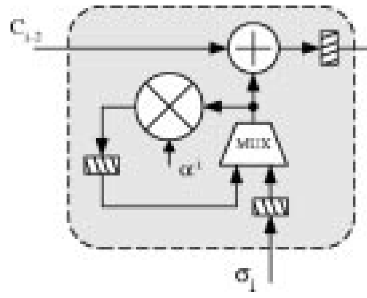
In an RS decoding algorithm, a Chien search is used to check whether the error locator polynomial $\sigma(x)$ equals zero or not while $x=\alpha^{-n}$, $n=0, 1, \ldots, N-1$. If $\sigma(\alpha^{-n})=0$, it means there is an error at R_n, where the received polynomial is defined as $R(x)=R_0+R_1x+R_2x^2+\cdots+R_{N-1}x^{N-1}$ and R_{N-1} is the first received symbol. Fig. 8(a) shows the circuit of the ith Chien search cell. The upper side of the Chien search cell accumulates the result of this and the previous cell, and sends the sum to the next cell.

Fig. 8(b) shows the structure of the Chien search module with eight Chien scarce cells. An XOR gate is used to check if the final sum is zero.

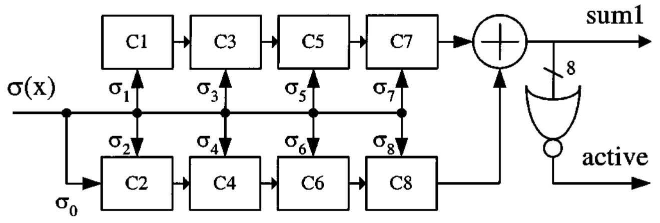
It is instructional to observe the similarity between the syndrome cell and the Chien search cell in our architecture. The only difference is that the location of the finite-field adder and the multiplexer is interchanged. In a fully custom layout, such similarity is very helpful to reduce chip area.

D. Error Value Evaluator

For calculating the error value, there are two popular methods, namely the transform decoding process in the frequency domain and the Forney algorithm in the time domain. Although the transform decoding process does not need any FFI and Chien search, it requires t variable–variable FFMs and N constant–variable FFMs. While N and t are large, the Forney algorithm is preferred because of its lower circuit complexity.



(a)



(b)

Fig. 8. (a) Chien search cell C_i. (b) Chien search structure for $t=8$.

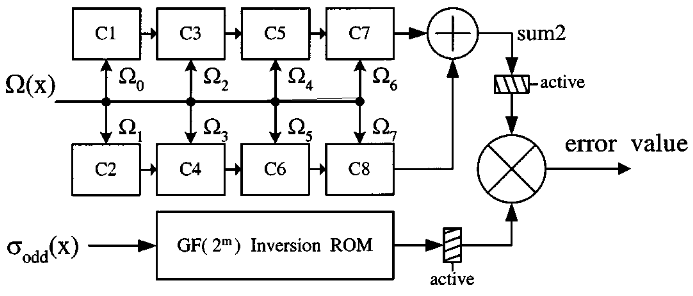


Fig. 9. Error value evaluator structure for $t=8$.

In the Forney algorithm, the error value becomes

$$e_l=\frac{\Omega(\beta_l{}^{-1})}{\sigma'(\beta_l{}^{-1})} \quad (11)$$

where β_l^{-1} indicates the root of $\sigma(x)$, for $l=1, \ldots, t$. Because of the fact that any $GF(2^m)$ element will be zero while multiplying an even constant value, and will be its original value while multiplying an odd constant, the first derivative of $\sigma(x)$ can be represented as

$$\begin{aligned}\sigma'(x) &= \left(\sum_{k=0}^{t}\sigma_k x^k\right)' \\ &= \sum_{k=1}^{t} k\cdot\sigma_k x^{k-1} \\ &= \sigma_1+\sigma_3x^2+\cdots+\sigma_{t_{\text{odd}}}x^{t_{\text{odd}}-1} \\ &\triangleq \frac{1}{x}\,\sigma_{\text{odd}}(x). \end{aligned} \quad (12)$$

Note that t_{odd} is the largest odd number less than or equal to t, and $\sigma_{\text{odd}}(x)=\sigma_1x+\sigma_3x^3+\cdots+\sigma_{\text{odd}}x^{t_{\text{odd}}}$.

So we rewrite the Forney's algorithm as

$$e_l=\frac{\Omega(\beta_l{}^{-1})}{\sigma'(\beta_l{}^{-1})}=\frac{\Omega(\beta_l{}^{-1})\cdot\beta_l^{-1}}{\sigma_{\text{odd}}(\beta_l{}^{-1})}. \quad (13)$$

In Fig. 9, we calculate error values in parallel with the computation of the Chien search. Note that cells C1 $\sim$ C8 in Fig. 9 are all the same as the Chien search cells in Fig. 8(a). The only

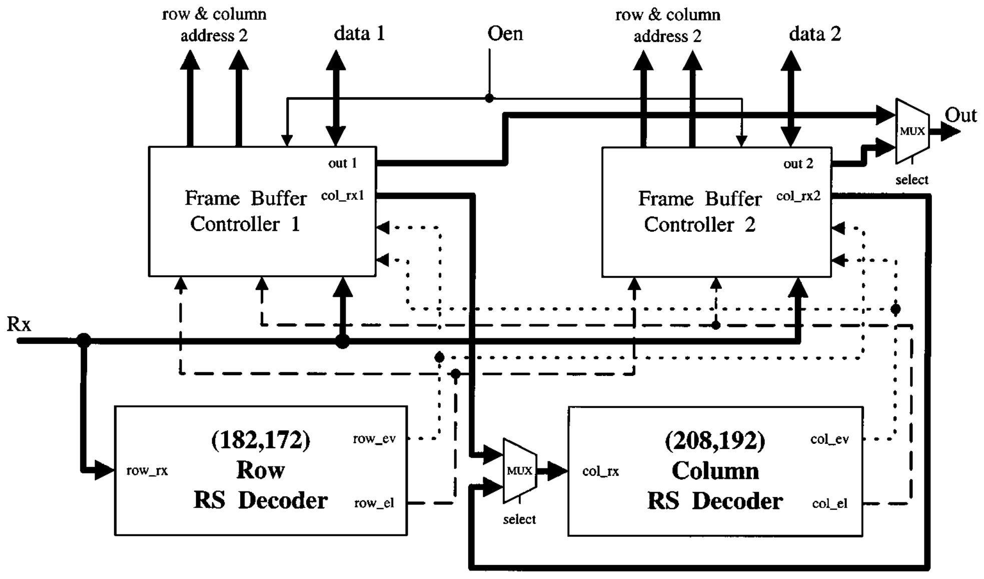


Fig. 10. RS-PC decoder chip architecture.

difference is the loaded coefficients are $\Omega_0 \sim \Omega_8$ instead of $\sigma_1 \sim \sigma_8$. While the computation of the Chien search is into nth iteration, the value of $\mathrm{sum1}$ in Fig. 8(b) is $\sigma(\alpha^{-n})$, and the value of $\mathrm{sum2}$ in Fig. 9 is $\Omega_0\alpha^{-n}+\Omega_1\alpha^{-2n}+\cdots+\Omega_7\alpha^{-8n} = \Omega(\alpha^{-n})\cdot\alpha^{-n}$. In other words, $\sigma(\alpha^{-n}) = 0$, the *active* signal goes to high, and the output of FFM in Fig. 9 is the error value of the received codeword R_n.

III. Frame Buffer Controller

The DVD RS-PC decoder can be implemented by a pair of *dedicated* row ($t = 5$) and column ($t = 8$) RS decoders, or by a *programmable* ($t = 8$) RS decoder. Through some added control logic, a programmable RS decoder with $t = 8$ can support both row and column RS decoding. The main drawback of using one RS decoder is that the throughput rate is reduced. In our work, we used two dedicated row and column RS decoders to maximize the throughput rate. Taking advantage of the area-efficient architecture mentioned in the previous section, our two-RS-decoder architecture is both feasible in complexity and fast in speed.

In RS-PC decoding, each symbol is subject to a row RS decoding and a column RS decoding. There are two possible frame-buffer architectures—*single* and *dual*. In the single-frame-buffer architecture, the ith incoming row of frame K is stored at the location of the ith outgoing column of frame $K - 1$. In other words, each adjacent frame is stored in a *transposed* fashion. The frame-buffer size in the single-frame-buffer architecture for DVD RS-PC is 208×208 [max (row, column)]. The drawback of the single-frame-buffer architecture is that the RS-PC decoder output sequence is different from the input sequence. The former is column-wise while the latter is row-wise. This effect is similar to passing the input data through an *interleaver*. To deinterleave the data, a frame buffer is also required. Therefore, unless the downstream processing (e.g., MPEG decoding) can be done using the interleaved data directly, the single-frame-buffer RS-PC decoder architecture is not preferred because it simply *transfers* the storage requirement to downstream processing.

In our design, we use a dual-frame-buffer architecture. Each frame buffer is controlled by a frame-buffer controller. The RS-PC decoder architecture is illustrated in Fig. 10 which contains two frame-buffer controllers that interface with two off-chip frame buffers, a (182, 172) row RS decoder and a (208, 192) column RS decoder. At any time, one (primary) frame buffer is serving the incoming data, the outgoing data, and the (182, 172) row RS decoder, and the other (secondary) frame buffer is serving the (208, 192) column RS decoder. The error locations and error values computed by the RS decoders are sent to the frame-buffer controllers to update the frame-buffer content accordingly. This parallel architecture minimizes the amount of frame-buffer access and timing constraint on the RS decoders. The architecture also allows the frame buffers to be incorporated as on-chip embedded SRAMs or DRAMs, which are not yet realized in the current chip.

Since we only need to correct the user data part of the frame, for each input row, the last ten parity checking bytes are used only by the row RS decoder and not stored in the primary frame buffer. The size of the frame buffer is therefore 172×208. The remaining memory bandwidth is used by the row RS decoder for error correction. Likewise, in the column RS decoding, only 172 columns are processed.

The frame-buffer controller consists of an address plane and a data plane, as shown in Fig. 11. The address plane consists of a row address generator and a column address generator, each selecting one out of three possible addresses: counter, counter -2, and the error location address. The data plane provides a great number of different data routes: input to buffer, buffer to output, buffer to column RS decoder input, and error correction.

The detailed symbol and memory interface timing for the row and column decoders is illustrated in Figs. 11 and 12, respectively. During each DVD symbol time, each frame buffer undergoes one read and one write operation, both at the same address. For the row decoder and the primary frame buffer, as shown

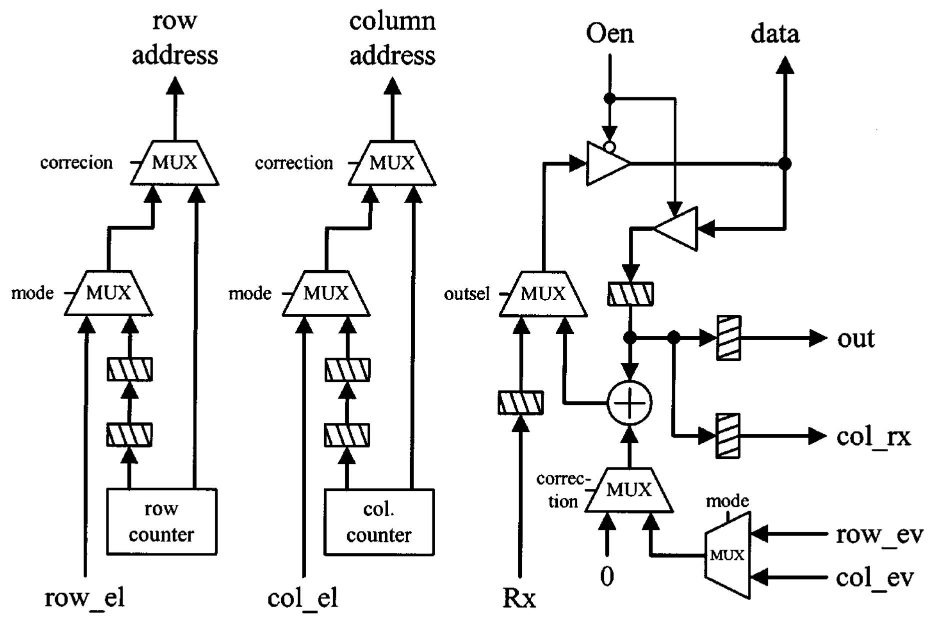


Fig. 11. Frame-buffer controller diagram.

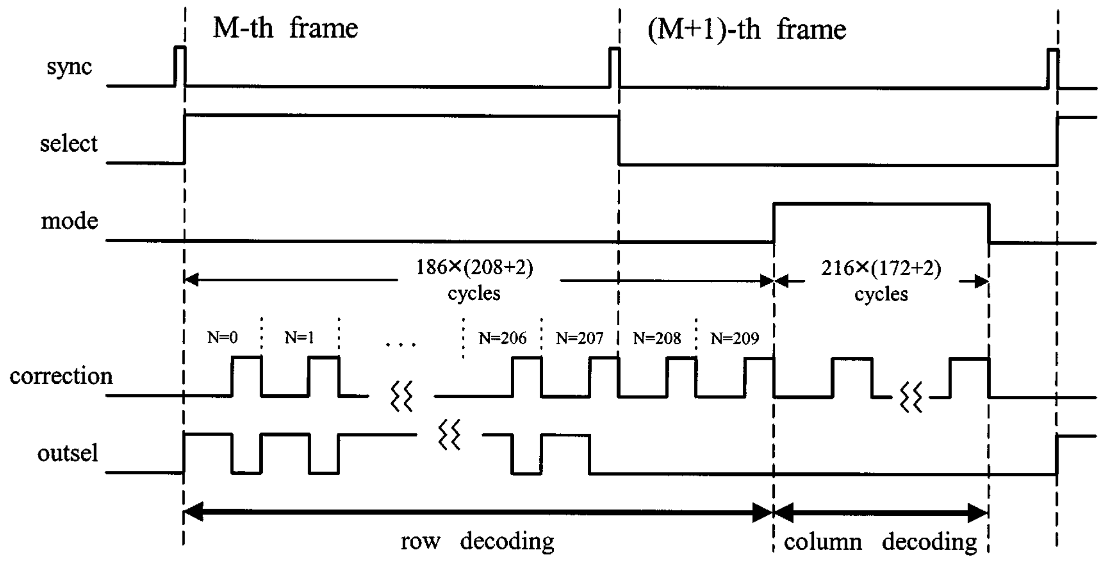


Fig. 12. Frame-buffer controller control signals.

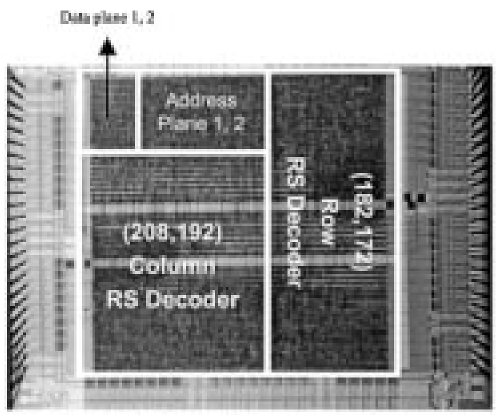
in Fig. 12, in the first 172 symbol times, the (decoded) output of frame $M-2$ is read out immediately before the incoming symbol is stored. The DVD timing specification demands two sync symbols every 91 data symbols per row. After the 172 data symbols, the memory bandwidth is used to perform error correction of the second previous row. Since $t=5$ for row RS decoder, it takes up to five symbol times to finish the error correction. For error correction, the frame-buffer content is read out, XOR-ed with the row or column error value and then written back in the same DVD symbol time.

For the column RS decoder and the secondary frame buffer, as shown in Fig. 12, in the first 208 symbol times, the corresponding data symbols are read out in the memory read cycle. The memory write cycle in this period is idle. After the first 208 symbol times, the memory bandwidth is used to perform error correction of the second previous column. Since $t=8$ for column RS decoder, it takes up to eight symbol times to finish the error correction. The total time to process one column is therefore 216 symbol times. The total time to finish the column RS decoding is $216 \times (172+2)$ symbol times.

The two frame-buffer controllers change their roles by the control of a number of externally or internally generated control signals, illustrated in Fig. 12. The *select* signal selects the primary frame buffer, and is derived from the *sync* signal defined in DVD. Due to the pipeline latency, the secondary frame buffer does not start the column RS decoding until two row delays, indicated by the *mode* signal. The *correction* signal indicates the time period within which the error correction is performed.

IV. Chip Implementation and Testing

We implement the RS-PC decoder chip by Verilog and all of modules designed by gate-level description. The total Verilog



- **Gate Count : 26248**
  - 0.6 um Compass cell library
  - Package Type : 100 LD CQFP
- **Area**
  - chip : 4.22 x 3.64 mm^2
  - core : 2.90 x 2.88 mm^2
- **Power and Speed**
  - RS decoder
    - 102.6 mW at 3V, 33.3 MHz
    - 268.5 mW at 5V, 53.7 MHz
  - with off-chip memory
    - 18 MHz

Fig. 13. Chip die photo.

code takes about 3000 lines. In our design, the delay time between two registers is restricted to only permit one FFM and one finite-field adder for speed consideration. For complexity consideration, we choose constant–variable FFMs to implement the syndrome calculator, the Chien search, and the error value evaluator. Note that the constant–variable FFM only needs $3 \sim 24$ XOR gates while the variable–variable FFM requires 73 XOR gates and 64 AND gates [12].

The chip was designed using the Compass cell library in a 0.6-μm single-poly double-metal (SPDM) CMOS process. The chip size is $4.22\ \mathrm{mm} \times 3.64\ \mathrm{mm}$ with a core size of $2.90\ \mathrm{mm} \times 2.88\ \mathrm{mm}$. The chip die photo is shown in Fig. 13. The total gate count is about 26K, including 14K for the (208, 192) column RS decoder and 9K for the (182, 172) row RS decoder. The 99-pin chip is packaged in a 100 LD CQFP package, where 48 pins are for frame-buffer interface and can be eliminated with embedded frame buffers. In the test mode, the column RS decoder can operate on the input data directly and bypass the frame buffer (a connection not shown in Fig. 10). While operating at 3 V, the row and column RS decoders have been tested to work successfully at 33 MHz. The RS-PC decoder, however, is currently limited in speed by the off-chip frame buffer to about 18 MHz. The power dissipation of the chip is 102 mW at 33 MHz.

As the DVD symbol rate is less than 4 Mbytes/s, our RS-PC decoder can support a speed of $4\times$ DVD with off-chip frame buffers or $8\times$ DVD with embedded frame buffers. The improved speed performance is attributed to the parallel RS decoder architecture, which is made feasible by the proposed area-efficient key equation solver.

V. Conclusion

In this paper, the design and implementation of an area-efficient RS-PC decoder chip for DVD applications is presented. Based on a modified decomposed inversionless Berlekamp–Massey algorithm, more optimal hardware structure for the key solver equation can be achieved. Moreover, the derived structure can be applied to other functional blocks, leading to a very regular structure for the area-delay product. As a result, an area-efficient solution for RS-PC decoder chip can be obtained.

The proposed chip solution contains two frame-buffer controllers, a row RS decoder, and a column RS decoder. Implemented in a 0.6-μm CMOS SPDM standard cells, measurement results show that $8\times$ DVD speed can easily be achieved.

References

[1] E. Berlekamp, *Algebraic Coding Theory*. New York: McGraw-Hill, 1968.

[2] Y. Sugiyama, M. Kasahara, S. Hirasawa, and T. Namekawa, "A method for solving key equation for decoding Goppa codes," *Information and Control*, vol. 27, pp. 87–99, Jan. 1975.

[3] L. Welch and R. Scholtz, "Continued fractions and Berlekamp's algorithm," *IEEE Trans. Inform. Theory*, vol. IT-25, pp. 19–27, Jan. 1979.

[4] T. Truong, W. Eastman, I. Reed, and I. Hsu, "Simplified procedure for correcting both errors and erasures of Reed–Solomon code using Euclidean algorithm," *Proc. IEE*, pt. E, vol. 135, no. 6, pp. 318–324, 1988.

[5] J. Massey, "Shift-register synthesis and BCH decoding," *IEEE Trans. Inform. Theory*, vol. IT-15, pp. 122–127, Jan. 1969.

[6] I. Reed, M. Shih, and T. Truong, "VLSI design of inverse-free Berlekamp–Massey algorithm," *Proc. IEE*, pt. E, vol. 138, pp. 295–298, Sept. 1991.

[7] E. Berlekamp, "Galois field computer," U.S. Patent 4 162 480, July 24, 1979.

[8] K. Liu, "Architecture for VLSI design of Reed–Solomon decoders," *IEEE Trans. Computers*, vol. C-33, pp. 178–189, Feb. 1984.

[9] Y. Oh and D. Kim, "Method and apparatus for computing error locator polynomial for use in a Reed–Solomon decoder," U.S. Patent 5 583 499, Dec. 10, 1996.

[10] H. Burton, "Inversionless decoding of binary BCH codes," *IEEE Trans. Inform. Theory*, vol. IT-17, pp. 464–466, July 1971.

[11] R. Blahut, *Theory and Practice of Error Control Codes*. Boston: Addison-Wesley, 1983.

[12] S. T. J. Fenn, M. Benaissa, and D. Taylor, "$GF(2^m)$ multiplication and division over the dual basis," *IEEE Trans. Comput.*, vol. 45, pp. 319–327, Mar. 1996.

Hsie-Chia Chang was born in Keelung City, Taiwan, in 1973. He received the B.S. and M.S. degrees in electronics engineering from National Chiao Tung University, Hsinchu, Taiwan, R.O.C., in 1991 and 1995, respectively. He is currently working toward the Ph.D. degree in electronics engineering at the same university.

His research interests include architectures and algorithms for communications and the signal processing, and the integrated circuit design for the Reed–Solomon decoder.

C. Bernard Shung (M'88) received the B.S. in electrical engineering from National Taiwan University, Taiwan, R.O.C., in 1981, and the M.S. and Ph.D. degrees in electrical engineering and computer science from the University of California, Berkeley, in 1985 and 1988, respectively.

He is currently a Design Manager at Allayer Communications Corporation, San Jose, CA. He was a Visiting Scientist at IBM Research Division, Almaden Research Center, San Jose, from 1988 to 1990, and a Research Staff Member from 1998 to 1999. He was a Faculty Member in the Department of Electronics Engineering, National Chiao Tung University (NCTU), Hsinchu, Taiwan, from 1990 to 1997. From 1994 to 1995, he was a Staff Engineer at Qualcomm Incorporated, San Diego, CA, while on leave from NCTU. His research interests include VLSI architectures and integrated circuits design for communications, signal processing, and networking. He has published more than 50 technical papers in various research areas.

Chen-Yi Lee received the B.S. degree from National Chiao Tung University, Hsinchu, Taiwan, R.O.C., in 1982, and the M.S. and Ph.D. degrees from Katholieke University Leuven (KUL), Belgium, in 1986 and 1990, respectively, all in electrical engineering.

From 1986 to 1990, he was with IMEC/VSDM, working in the area of architecture synthesis for DSP. In February 1991, he joined the faculty of the Electronics Engineering Department, National Chiao Tung University, where he is currently a Professor. His research interests mainly include VLSI algorithms and architectures for high-throughput DSP applications. He is also active in various aspects of high-speed networking, system-on-chip design technology, very low-bit-rate coding, and multimedia signal processing.