Journal of Circuits, Systems, and Computers
Vol. 12, No. 2 (2003) 159–170

World Scientific
www.worldscientific.com

A LOW-POWER DESIGN FOR REED–SOLOMON DECODERS

HSIE-CHIA CHANG and CHEN-YI LEE

Department of Electronics Engineering, National Chiao Tung University, Hsinchu, Taiwan, 300, Republic of China

Received 15 April 2002
Accepted 15 July 2002

In this paper, a low-power design for the Reed–Solomon (RS) decoder is presented. Our approach includes a novel *two-stage* syndrome calculator that reduces the syndrome computations by one-half, a modified Berlekamp–Massey algorithm in the key equation solver and a terminated mechanism in the Chien search circuit. The test chip for $(255, 239)$ and $(208, 192)$ RS decoders are implemented by 0.25 μm CMOS 1P5M and 0.35 μm CMOS SPQM standard cells, respectively. Simulation results show our approach can work successfully and achieved large reduction of power consumption on the average.

Keywords: Low-power design; Reed–Soloman codes; two-stage syndrome calculator; Berlekamp–Massey algorithm; Euclidean algorithm.

1. Introduction

Among the most well-known error-correcting codes, the Reed–Solomon (RS) codes are undoubtedly the most widely used block codes in communications and storage systems to enhance the immunity to burst errors. An (N, K) RS code contains N coded symbols with K message symbols in each codeword, and is capable of correcting up to $t = \lfloor (N - K)/2 \rfloor$ symbol errors, where each symbol belongs to the finite field.[1] Due to the increasing demand for high capacity communication systems and portable wireless applications, low-power implementations of RS decoders are desirable to meet higher data rates for system-level integration.

The most popular RS decoder architecture 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*,[2] which is

$$\Omega(x) = S(x)\sigma(x) \bmod x^{2t}\,, \tag{1}$$

where $S(x)$ represents the syndrome polynomial, $\sigma(x)$ is the error locator polynomial, and $\Omega(x)$ is the error evaluator polynomial. As a consequence, existing RS decoders usually contain a syndrome calculator, a key equation solver, a Chien search,

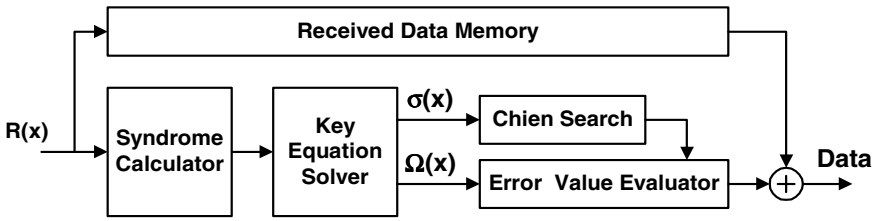


Fig. 1. Reed–Solomon decoding flowchart.

and an error value evaluator, which are illustrated in Fig. 1. The syndrome calculator generates a set of syndromes from the received codeword polynomial $R(x)$. From the syndromes, the key equation solver produces the error locator polynomial $\sigma(x)$ and the error evaluator polynomial $\Omega(x)$, which can be used by the Chien search and the error value evaluator to determine the error locations and error values, respectively. The received data memory is used to store the received symbols. In accordance with the error value and its location, the output of the finite-field adder shown in Fig. 1 is the corresponding corrected symbol.

While implemented for portable storage systems or optical communications with higher data rates, all existing RS decoders cause relatively large power consumption, leading to difficulty in system-level integration. As a result, we propose a low-power design for RS decoders using a novel *two-stage* syndrome calculator, which is addressed in Sec. 2, to detect whether the received codeword carrying errors. If there is no error occurring, the power consumption can be reduced significantly by terminating the follow-up decoding procedure. Section 3 illustrates a modified Berlekamp–Massey algorithm to reduce many *unnecessary* calculation counts and Sec. 4 describes the Chien search with the terminated mechanism and an area-efficient architecture for the error value evaluator. The $(255, 239)$ as well as $(208, 192)$ RS decoder are implemented as the design examples and simulation results are shown in Sec. 5. Finally, the conclusion is given in Sec. 6.

2. The Novel Syndrome Calculator

By definition, the syndrome polynomial $S(x)$ is denoted as $S_1 + S_2x + \cdots + S_{2t}x^{2t-1}$, where $S_i = R(\alpha^i)$ with the received polynomial $R(x) = R_0 + R_1x + \cdots + R_{N-1}x^{N-1}$ in (N, K) RS codes. If the received codeword contains no errors, it can be shown that all syndromes, S_1 through S_{2t}, will all equal zeros. From the relations between syndromes and the coefficients of the error locator polynomial $\sigma(x)$ given by[3]:

$$\begin{bmatrix} S_1 & S_2 & \cdots & S_\nu \\ S_2 & S_3 & \cdots & S_{\nu+1} \\ \vdots & & & \vdots \\ S_\nu & S_{\nu+1} & \cdots & S_{2\nu-1} \end{bmatrix} \begin{bmatrix} \sigma_\nu \\ \sigma_{\nu-1} \\ \vdots \\ \sigma_1 \end{bmatrix} = \begin{bmatrix} -S_{\nu+1} \\ -S_{\nu+2} \\ \vdots \\ -S_{2\nu} \end{bmatrix},$$

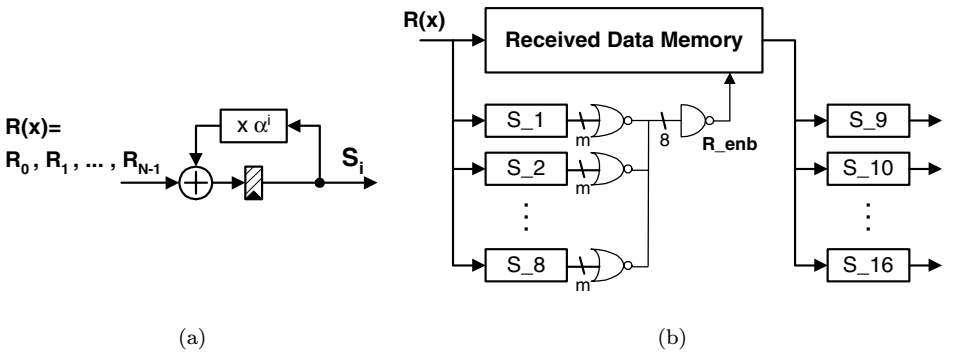


Fig. 2. (a) Syndrome calculator cell *S_i*. (b) Proposed two-staged structure of the syndrome calculator over $GF(2^m)$ for $t = 8$.

$S_{\nu+1}, S_{\nu+2}, \ldots,$ and $S_{2\nu}$ will be all zeros if both $\nu \leq t$ and $S_1 = S_2 = \cdots = S_\nu = 0$, where ν represents the number of actual errors and t is the number of correctable errors. Therefore, the first half of the syndromes can be seen as an error detector. Once there are t continuous syndromes equaling zeros, all $2t$ syndromes will equal zeros. Then the follow-up decoding procedure can be terminated and all error values are set to zeros directly whether no-error ($\nu = 0$) codeword or out-of-correction codeword ($\nu > t$) received. The novel syndromes calculating procedure can be shown as follows:

Let *continue* = *FALSE*;
For ($i = 1$ **to** t)
{ *Calculate* S_i;
If ($S_i \neq 0$) *continue* = *TRUE*; }
IF (continue = TRUE)
{ **For**($i = t + 1$ **to** $2t$) *Calculate* S_i; }
Else *Finish*;

The syndrome calculator cell *S_i* is shown in Fig. 2(a), where the partial syndrome is multiplied with α^i and accumulated with the received symbol at each cycle. After all received symbols from R_{N-1} to R_0 are processed, the accumulated result is the ith syndrome, S_i. In Fig. 2(b), the proposed two-staged structure of the syndrome calculator is illustrated for $t = 8$. After N cycles, the syndromes S_1 through S_8 are obtained and then the signal *R_enb* is used to control the access of the data memory. The syndrome calculator cells *S_9* through *S_16* will remain idle except the controlling signal *R_enb* goes to high.

Moreover, the look-forward architecture[4] can be used to improve the throughput rate. For example, the process of calculating the ith syndrome in $(255, 239)$ RS codes

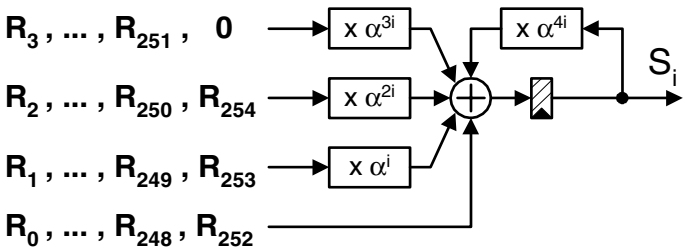


Fig. 3. Syndrome calculator cell of *S_i* using the look-forward architecture. Note that $R_{255} = 0$ in the $(255, 239)$ RS code.

can be derived as

$$S_i = \sum_{j=0}^{63} (R_{4j} + R_{4j+1}x + R_{4j+2}x^2 + R_{4j+3}x^3)x^{4j}\Big|_{x=\alpha^i}. \tag{2}$$

The syndrome calculator cell *S_i* using the look-forward architecture to process four symbols per cycle is illustrated in Fig. 3. At each cycle, the partial syndrome is multiplied with α^{4i}, accumulated with the received symbols and their multiplying results of α^i to α^{3i} in parallel. After all the received symbols are processed, the accumulated result is S_i. Thus, our proposed two-stage syndrome calculator can be also applied to applications of higher data rate.

Note that the finite-field multipliers (FFMs) implemented in the syndrome calculator are all constant–variable FFMs, which have one input as a constant and the other input as a variable, indicating that the circuit complexity and power consumption are much lower than that of variable–variable FFMs, whose inputs are both variables. Since the transmission data in realistic systems are almost correct (i.e., $S_1 = S_2 = \cdots = S_t = 0$), our proposal reduces by almost half the syndrome computations, leading to a good effect on the power consumption of the entire RS decoder.

3. The Key Equation Solver

The techniques frequently used to solve the key equation include the Berlekamp–Massey (BM) algorithm[2,5] and the Euclidean algorithm.[6] The BM algorithm is generally considered to be the one with the least hardware complexity for solving the key equation. Another advantage is that the constant term of $\sigma(x)$ and $\Omega(x)$ always equals 1 and S_1, suggesting an efficient decoding procedure to eliminate redundant computations in the BM algorithm. However, the BM algorithm is an iterative procedure and after calculating the first iteration in advance, the modified BM algorithm with some differences in initial conditions can be shown as follows:

Initial condition:

$$\delta = \Delta^{(0)} = S_1\,, \qquad D^{(0)}(x) = 1\,,$$
$$\sigma^{(0)}(x) = 1 + S_1 x\,, \quad \tau^{(0)}(x) = \Omega^{(0)}(x) = \gamma^{(0)}(x) = 1\,,$$

For $(i = 1$ **to** $2t - 1)$

$$\Delta^{(i)}(x) = S_{i+1} + S_i\sigma_1^{(i-1)} + \cdots + S_{i-t+1}\sigma_t^{(i-1)}\,, \tag{3}$$

$$\sigma^{(i)}(x) = \sigma^{(i-1)}(x) + \frac{\Delta^{(i)}}{\delta}\, x\tau^{(i-1)}(x)\,, \tag{4}$$

$$\Omega^{(i)}(x) = \Omega^{(i-1)}(x) + \frac{\Delta^{(i)}}{\delta}\, x\gamma^{(i-1)}(x)\,, \tag{5}$$

If $(\Delta^{(i)} = 0$ **or** $2D^{(i-1)} \geq i + 1)$

$$D^{(i)} = D^{(i-1)}\,,$$
$$\tau^{(i)}(x) = x\tau^{(i-1)}(x)\,, \quad \gamma^{(i)}(x) = x\gamma^{(i-1)}(x)\,,$$

Else

$$D^{(i)} = i + 1 - D^{(i-1)}\,, \quad \delta = \Delta^{(i)}\,,$$
$$\tau^{(i)}(x) = \sigma^{(i-1)}(x)\,, \qquad \gamma^{(i)}(x) = \gamma^{(i-1)}(x)\,,$$

where $\sigma^{(i)}(x)$ is the ith error locator polynomial, $\Omega^{(i)}(x)$ is the ith error evaluator polynomial, and $\sigma_j^{(i)}$'s are the coefficients of $\sigma^{(i)}(x)$; $\Delta^{(i)}$ is the ith discrepancy and δ is the previous discrepancy; $D^{(i)}$ is an auxiliary degree variable in the ith iteration, and $\tau^{(i-1)}(x)$ and $\gamma^{(i-1)}(x)$ are auxiliary polynomials for calculating $\sigma^{(i)}(x)$ and $\Omega^{(i)}(x)$, respectively. Note that the ith error locator polynomial $\sigma^{(i)}(x)$ calculated by Eq. (4) will be equal to the previous polynomial $\sigma^{(i-1)}(x)$ if $\Delta^{(i)} = 0$. After $2t-1$ iterations, $\sigma^{(2t-1)}(x)$ and $\Omega^{(2t-1)}(x)$ are equivalent to the error locator polynomial $\sigma(x)$ and the error locator polynomial $\Omega(x)$, respectively.

The conventional way to compute the error evaluator polynomial $\Omega(x)$, shown as above, is to do it in parallel with the computation of $\sigma(x)$. From the key equation and Newton's identity, the computation of $\Omega(x)$ can be shown as follows:

$$\begin{aligned} \Omega(x) &= S(x)\sigma(x) \bmod x^{2t} \\ &\triangleq \Omega_0 + \Omega_1 x + \cdots + \Omega_{(t-1)}x^{t-1}\,, \\ \Rightarrow \quad \Omega_0 &= S_1\,, \\ \Omega_1 &= S_2 + S_1\sigma_1\,, \\ &\vdots \\ \Omega_{t-1} &= S_t + S_{t-1}\sigma_1 + \cdots + S_1\sigma_{t-1}\,, \end{aligned} \tag{6}$$

where Ω_j's are the coefficients of the error evaluator polynomial $\Omega(x)$. Note that the proposed *direct* computation of $\Omega(x)$ after $\sigma(x)$ is computed requires fewer multiplications and additions than the original BM algorithm. Table 1 compares the average calculation counts between the original Euclidean and BM algorithm with the modified BM algorithm after many random test patterns are simulated.

In addition, the computation of the ith coefficient Ω_i is similar to that of the ith discrepancy $\Delta^{(i)}$. Therefore, the same hardware used to compute $\Delta^{(i)}$ can be *reconfigured* to compute the coefficient Ω_i. Depending on the implementation, there are

Table 1. Comparison of calculation counts in different architectures.

Type	Euclidean[6]	BM[2,5]	Modified BM
Division	8.92	4.44	4.44
Addition	156.07	113.43	94.55
Multiplication	155.15	121.30	94.54

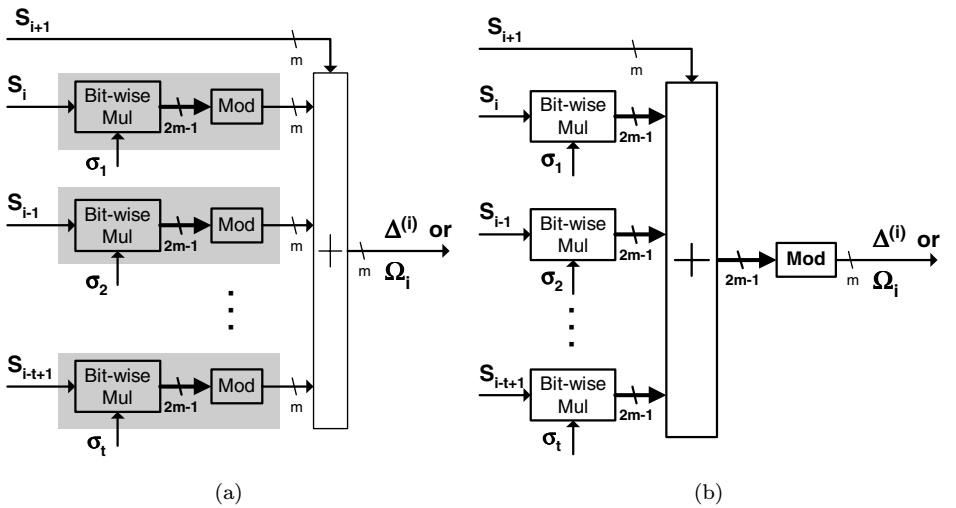
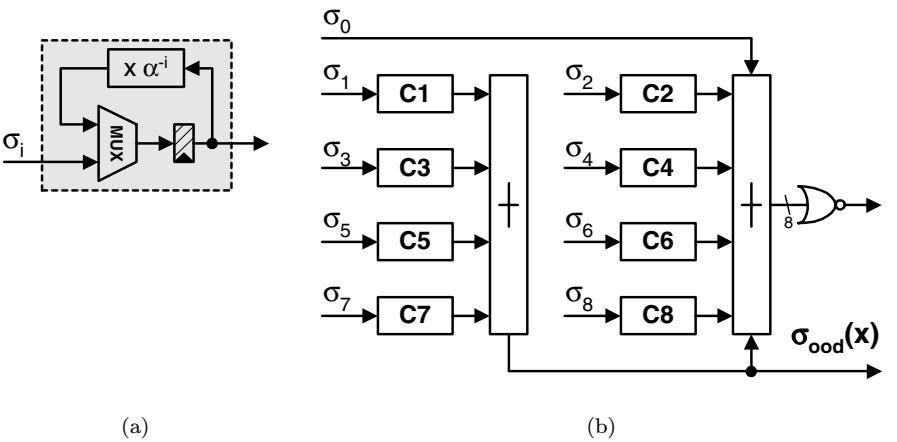


Fig. 4. (a) Original approach for calculating $\Delta^{(i)}$ or Ω_i. (b) Separated approach for calculating $\Delta^{(i)}$ or Ω_i.

two different approaches illustrated in Fig. 4 to compute $\Delta^{(i)}$ or Ω_i over $GF(2^m)$.[7] From the finite-field arithmetic, the multiplication of two operands can be split into a bit-wise multiplying operation and a modular operation. In Fig. 4(a), the original approach indicates that each multiplier requires both the bit-wise multiplying operation and the modular operation. However, the *separated* approach, shown as Fig. 4(b), reduces $t-1$ modular operations and only requires an extra $m-1$ XOR gates of t-input. Simulation results show the separated approach can achieve approximately both a 30% reduction of power consumption and a 15% reduction of circuit complexity as compared with the original approach for calculating $\Delta^{(i)}$ or Ω_i within $(t, m) = (8, 8)$.

4. Chien Search and the Error Value Evaluator

In the (N, K) RS decoding algorithm, the 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. In Ref. 8, McEliece proposed three conditions to determine whether the received codeword can be corrected or not.



(a) (b)

Fig. 5. (a) Chien search cell C_i. (b) Chien search structure for $t = 8$.

The corresponding hardware is to compare the degree of $\sigma(x)$ with the number of roots found by the Chien search. While the out-of-correction received codeword is detected, we stop the follow-up decoding procedure of error value evaluation. Figure 5(a) shows the circuit of the Chien search cell C_i and the structure of the Chien search with t cells is illustrated in Fig. 5(b).

At the ηth cycle after initialization, the finite-field adder (FFA) in the right hand side of Fig. 5(b) calculates the value of $\sigma(\alpha^{-\eta})$ and the NOR gate is used to check whether the final sum equals zero or not. Note that $\sigma_{\text{odd}}(x) = \sigma_1 x + \sigma_3 x^3 + \cdots + \sigma_{t_{\text{odd}}} x^{t_{\text{odd}}}$ is prepared to calculate the error value. For $t = 8$, $t_{\text{odd}} = 7$ represents the largest odd number less than or equal to t. However, Fig. 5 checks one root at one cycle, whereas Fig. 6 illustrates another structure of the Chien search to check eight roots at each cycle for applications of higher data rate. Note that all input ports to the shadowed cells in Fig. 6 are identical. In the jth iteration after initialization, the outputs of the FFA in shadowed cells are $\sigma^{(\alpha^{-8j})}$, $\sigma^{(\alpha^{-8j-1})}, \ldots, \sigma^{(\alpha^{-8j-7})}$, respectively.

For calculating the error value, the Forney algorithm proposed in Ref. 9 is utilized and can be expressed as:

$$e_l = \frac{\Omega({\chi_l}^{-1})}{\sigma'(\chi_l^{-1})} = \frac{\Omega(\chi_l^{-1}) \cdot \chi_l^{-1}}{\sigma_{\text{odd}}(\chi_l^{-1})}, \tag{7}$$

where χ_l^{-1} indicates the root of $\sigma(x)$, for $l = 1, \ldots, t$. Figure 7 shows an area-efficient architecture for the error value evaluator, which requires only one variable–variable FFM. Note that the *Buffer-1* and *Buffer-2* store all the roots χ_l^{-1} found by the Chien search and the corresponding values of $\sigma_{\text{odd}}(\chi_l^{-1})$ respectively. The $\Omega(x)$-*buffer* is composed of one "*0*" and the coefficients of the error evaluator polynomial $\Omega(x)$, Ω_0 through $\Omega_{\nu-1}$, where ν represents the number of the actual errors. At the first ν cycles after initialization, the output of the multiplexer in Fig. 7 is χ_l^{-1},

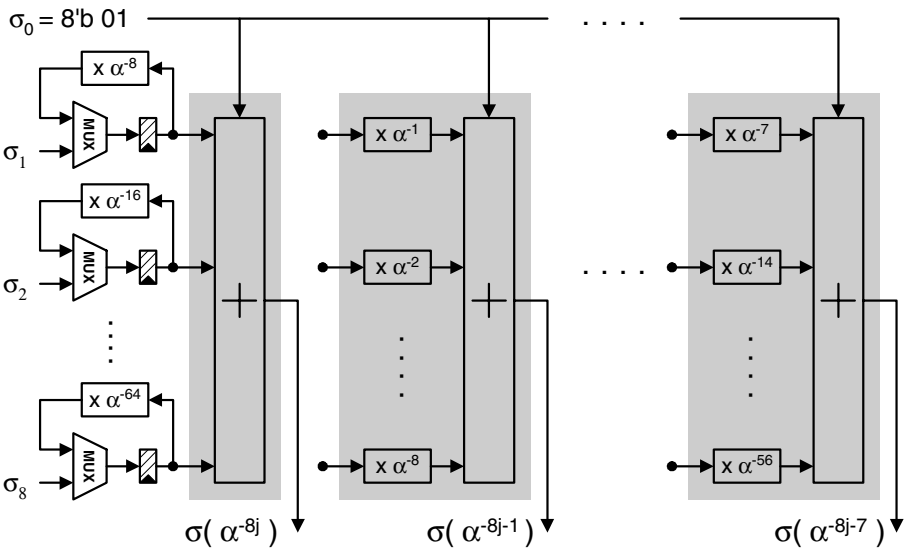


Fig. 6. Chien search structure to check eight roots per cycle for $t = 8$.

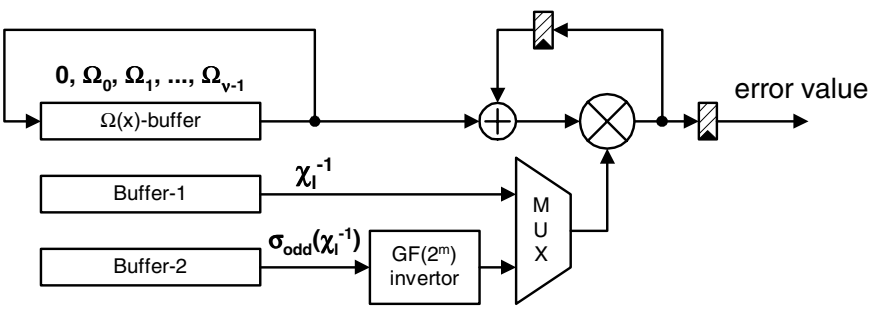


Fig. 7. The error value evaluator structure.

and the detailed computation of $(\Omega(x) \cdot x)|_{x=\chi_l^{-1}}$ can be shown as

$$\begin{aligned}
&(\Omega_0 + \cdots + (\Omega_{\nu-2} + \Omega_{\nu-1}\chi_l^{-1})\chi_l^{-1} \cdots)\chi_l^{-1} \\
&\quad = \Omega_0\chi_l^{-1} + \Omega_1(\chi_l^{-1})^2 + \cdots + \Omega_{\nu-1}(\chi_l^{-1})^\nu \\
&\quad \equiv \Omega(\chi_l^{-1}) \cdot \chi_l^{-1}\,.
\end{aligned}$$

At the $(\nu+1)$th cycle, the output of the $\Omega(x)$-*buffer* is 0 and the output of the multiplexer in Fig. 7 should be altered to $1/[\sigma_{\mathrm{odd}}(\chi_l^{-1})]$, then the output of the FFM, which will be latched by the register shown in the right hand side of Fig. 7, is the error value corresponding to the root χ_l^{-1}.

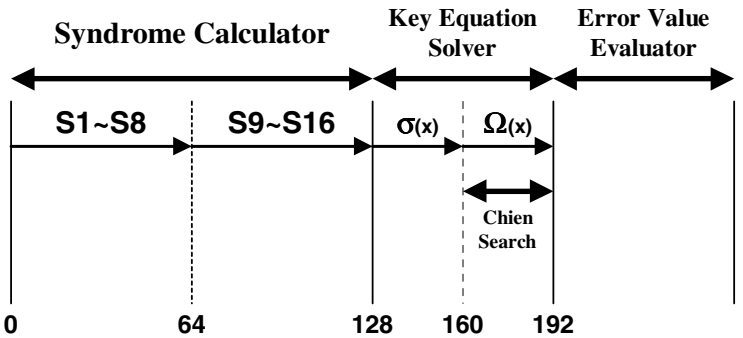


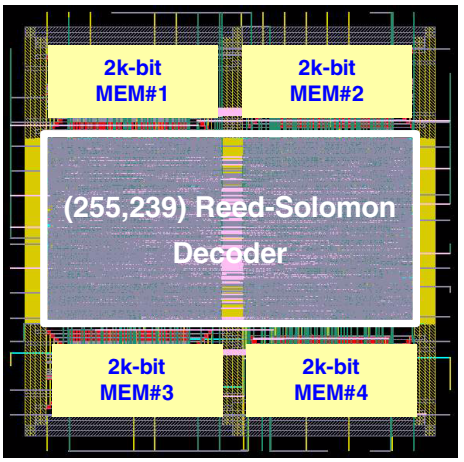
Fig. 8. Pipelining diagram of our proposed $(255, 239)$ RS decoder with $t = 8$.

5. Chip Implementation and Simulation Results

Here we propose two design examples to verify our low-power architecture. One is the $(255, 239)$ RS decoder, which is recommended in ITU-T $G.975$ to resist burst errors for optical fiber submarine cable systems on the STM-16 basis. Note that the STM-16 format has a data rate of 2.5 Gbps. For meeting a higher data rate with a lower clock rate, the look-forward architecture is utilized in our $(255, 239)$ RS decoder to process four bytes at each cycle. Figure 8 shows the pipelining diagram with the latency of 192 cycles. In the first pipelining stage, the front half syndromes S_1 through S_8 are calculated and used to detect whether the received codeword carrying errors. The received codeword without errors indicates that the calculations of the later half syndrome, S_9 through S_{16}, are needless in the second stage. However, the Chien search takes only half of the third stage, checking eight roots at each cycle. The error value is evaluated and simultaneously added with the corresponding received symbol for correcting errors in the fourth stage.

After being implemented by the Verilog and Synopsys design tool with the Artisan 0.25 μm CMOS 1P5M standard cells, the $(255, 239)$ RS decoder contains four 2K bit embedded single-port synchronous memory and has gate counts of 32.9 K. The total size is 2.23 mm $\times$ 2.23 mm with the RS core of 2.01 mm $\times$ 1.01 mm. The layout view and chip summary are shown in Fig. 9. While simulated at 1.8 V of the supply voltage by EPIC PowerMill, our proposed RS decoder can work successfully with the 2.5 Gbps data rate and consumes 14.8 mW core power and 53.7 mW memory power under the bit-error rate (BER) of 10^{-4}, indicating an approximately 20% error probability of the received codewords. For the BER less than 10^{-5}, only the syndrome calculator operates to detect errors and the other three parts — key equation solver, Chien search, and error value evaluator — almost remain idle.

The other design example is the $(208, 192)$ RS decoders utilized for DVD applications. Since the data rate of $1 \times$ DVD is below 4 MBps, the overall decoding speed can be maintained by the two-stag approach of calculating syndromes, each



Chip Summary

Technology	.25μm 1P5M CMOS
Chip Size	2.23mm x 2.23mm
RS Core Size	2.01mm x 1.01mm
Gate Counts	32.9 K
Embedded Memory	8K-bit SRAM

Power dissipation (83.68MHz, 1.8Volt)

	RS core	Memory
BER < 10^{-5}	13.5 mW	48.2 mW
BER = 10^{-4}	14.8 mW	53.7 mW
BER = 10^{-3}	22.5 mW	70.2 mW
BER > 10^{-2}	27.9 mW	72.9 mW

Fig. 9. Layout view of the $(255, 239)$ RS decoder chip.

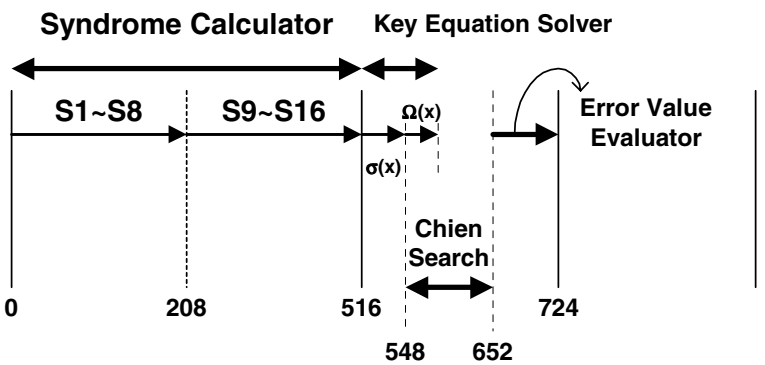
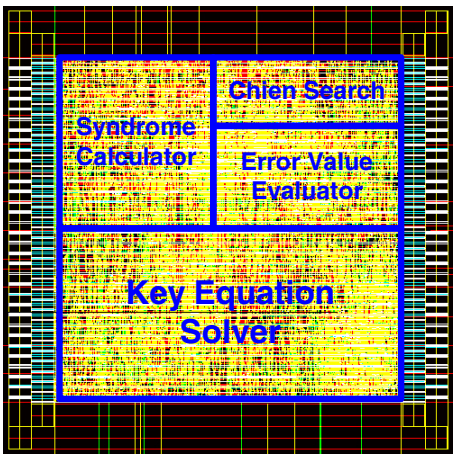


Fig. 10. The pipelining diagram of our proposed $(208, 192)$ RS decoder with $t = 8$.

taking 208 cycles to finish. Figure 10 illustrates the pipelining diagram with the latency of 724 cycles. From our proposal, the Chien search takes only half of the third stage, checking two roots at each cycle and therefore, 104 cycles are required totally. The error value is evaluated after the Chien search and 72 cycles are needed in the worst case of eight errors occurring.

We implement the $(208, 192)$ RS decoder by the 0.35 μm CMOS SPQM standard cell library. The RS decoder has the core size of 1.05 mm $\times$ 1.07 mm and the gate counts of 16.3 K, including 3.2 K for the syndrome calculator, 8.0 K for the key equation solver, and 5.1 K for Chien search and the error value evaluator. Figure 11 shows the layout view and chip summary.

While simulated at the supply voltage of 3.3 V by EPIC PowerMill, our proposed RS decoder can work successfully with the clock rate of 100 MHz. Table 2 shows the circuit complexity and simulation results of power consumption between the



Chip Summary

Technology	.35μm 1P4M CMOS
RS Core Size	1.05mm x 1.07mm
Gate Counts	16.3 K

Power dissipation

BER < 10^{-5}	54.9 mW
BER = 10^{-4}	60.2 mW
BER = 10^{-3}	109.4 mW
BER > 10^{-2}	135.6 mW

Fig. 11. Layout view of the (208, 192) RS decoder chip. Note that the power dissipation is simulated at the clock rate of 100 MHz and the supply voltage of 3.3 V.

Table 2. Comparison of hardware complexity (gates) and power consumption (mW) with the previous architecture.

	Previous[10]			Proposed		
	Gates (K)	8 errors	0 error	Gates (K)	8 errors	0 error
Syndrome Calculator	3.1	88.1	87.9	3.2	83.1	54.9
Key Equation Solver	2.9	120.2	115.5	8.0	81.5	0
Chien Search	2.8	56.7	53.2	2.8	46.6	0
Error Value Evaluator	1.8	37.9	7.1	2.3	19.8	0
Total	10.6	302.9	263.7	16.3	231.0	54.9

previous architecture[10] and our proposal in this paper. Although the architecture proposed in Ref. 10 has less complexity, the proposed RS decoder consumes only 60% of the power dissipation of Ref. 10 and approximately no power consumption in the key equation solver, Chien search, and error value evaluator when no-error codewords are received. Note that the two different results in each architecture correspond to two extreme cases of the received codewords carrying eight errors and without errors. In realistic communication systems, the probability of no error is much larger than that of error. Our proposed two-stage syndrome calculator of reducing half syndrome calculations, and of terminated mechanisms in other parts can lead to a very power efficient solution for the RS decoder.

6. Conclusion

In this paper, the design and implementation of a low-power RS decoder is presented. The proposed architecture features the novel two-stage syndrome

calculator, the key equation solver using the modified BM algorithm, and the Chien search with the terminated mechanism. The test chip of the $(255, 239)$ RS decoder and the $(208, 192)$ RS decoder are implemented to verify our proposal. Since there are almost *correct* codewords in realistic communication systems, our derived structure can lead to a very power efficient solution for the RS decoder.

Acknowledgment

The work was supported by the National Science Council of Taiwan, R.O.C., under Grant *NSC* 90-2218-E-009-035.

References

1. R. Blahut, *Theory and Practice of Error Control Codes*, Addison-Wesley, 1983.
2. E. Berlekamp, *Algebraic Coding Theory*, McGraw-Hill, New York, 1968.
3. D. Gorenstein and N. Zierler, "A class of cyclic linear error-correcting codes in p^m symbols", *J. Soc. Ind. Appl. Math.* **9** (1961) 207–214.
4. J.-Y. Chang and C. B. Shung, "A high speed Reed–Solomon CODEC chip using look-forward architecture", *Proc. IEEE Asia-Pacific Conf. Circuits Syst.*, November 1994, pp. 212–217.
5. J. Massey, "Shift-register synthesis and BCH decoding", *IEEE Trans. Inform. Theor.* **IT-15** (1969) 122–127.
6. Y. Sugiyama, M. Kasahara, S. Hirasawa, and T. Namekawa, "A method for solving key equation for decoding Goppa codes", *Inform. Contr.* **27** (1975) 87–99.
7. W.-C. Tsai and S.-J. Wang, "Two systolic architectures for multiplication in $GF(2^m)$", *IEE Proc. Comput. Digit. Tech.* **147** (2000) 375–382.
8. R. J. McEliece, "The decoding of Reed–Solomon codes", *The Telecommunications and Data Acquisition Progress Report* **42-95** (1988) 153–167.
9. G. Forney, "On decoding BCH codes", *IEEE Trans. Inform. Theor.* **IT-11** (1965) 549–557.
10. H.-C. Chang, C. B. Shung, and C.-Y. Lee, "A Reed–Solomon Product-Code (RSPC) decoder chip for DVD applications", *IEEE J. Solid-State Circuits* **36** (2001) 229–238.