

ERROR CORRECTION FOR MULTI-LEVEL NAND FLASH MEMORY USING REED-SOLOMON CODES

Bainan Chen and Xinmiao Zhang

Case Western Reserve University
10900 Euclid Ave., Cleveland, OH 44106-7071

Zhongfeng Wang

Broadcom Corporation
5300 California Avenue, Irvine, CA 92617

ABSTRACT

Prior research efforts have been focusing on using BCH codes for error correction in multi-level cell (MLC) NAND flash memory. However, BCH codes often require highly parallel implementations to meet the throughput requirement. As a result, large area is needed. In this paper, we propose to use Reed-Solomon (RS) codes for error correction in MLC flash memory. A (828, 820) RS code has almost the same rate and length in terms of bits as a BCH (8248, 8192) code. Moreover, it has at least the same error-correcting performance in flash memory applications. Nevertheless, with 70% of the area, the RS decoder can achieve a throughput that is 121% higher than the BCH decoder. A novel bit mapping scheme using Gray code is also proposed in this paper. Compared to direct bit mapping, our proposed scheme can achieve 0.02 dB and 0.2dB additional gains by using RS and BCH codes, respectively, without any overhead.

Index Terms— Reed-Solomon codes, BCH codes, VLSI architecture, flash memory

1. INTRODUCTION

The increasing demand for non-volatile multi-media and mass data storage has created the need for low-cost and high-capacity flash memories. In recent years, multi-level cell (MLC) flash memories [1, 2] have been developed as an effective solution for increasing the storage density and reducing the cost of flash memories. In MLC flash memories, multiple bits are stored per memory cell by programming each cell with multiple threshold levels. The reduced spacing between adjacent threshold levels significantly degrades the reliability of flash memories. To ensure data integrity, error-correcting codes (ECC) need to be employed in these memories. Traditionally, single-bit error-correcting codes, such as Hamming codes, are used for single-bit cell (SBC) flash memories. For MLC flash memories, especially for NAND flash memories which have a large page size, e.g. 8K bits, single-bit error-correcting codes are no longer sufficient.

Several works [2, 3, 4] have proposed to use strong Bose-Chaudhuri-Hocquenghem (BCH) codes for MLC NAND flash

memories. BCH codes provide a wide range of error-correcting capability and support flexible code length through the shortening process. Efficient decoding algorithms, such as the Berlekamp-Massey algorithm (BMA) [5] and Euclid algorithms, exist for decoding BCH codes. However, the large page size of NAND flash memories demands codes of long length. Hence, the binary BCH codes to be used in NAND flash memories are constructed over finite fields of very large order. Consequently, highly parallel implementations are necessary in order to meet the decoding speed requirement. As a result, large area is required for the decoder implementation.

In this paper, we propose to use Reed-Solomon (RS) codes for error correction in NAND flash memories. RS codes are good at correcting burst errors since they are non-binary codes constructed over extension fields. For a RS code constructed over $GF(2^m)$, if a symbol error can be corrected, then all the m bit errors in this symbol are corrected. From simulations, it was found that the errors in flash memory applications are random errors, *i.e.* errors happen one bit at a time, if proper bit mapping is used. Although the burst error-correcting capability of RS codes is wasted by having a t -symbol error-correcting code to correct t random bit errors, the RS decoder has much lower complexity than the decoder of a t -bit error-correcting BCH code with about the same code rate and length in terms of bits. The major reason is that in order to construct a code of the same bit length, the required order of the finite field for RS codes is much smaller than that required for BCH codes. Hence, the syndrome computation in RS decoding involves much less number of coefficients and Chien search needs to try much smaller number of finite field elements. Accordingly, much smaller parallel processing factor is required by the RS decoder to achieve a given throughput. Smaller parallel processing factor translates to smaller number of computational units. As a result, RS decoders require much less area than BCH decoders, despite the extra step of error magnitude computation. As an example, a (828, 820) RS code constructed over $GF(2^{10})$ and a binary (8248, 8192) BCH code constructed over $GF(2^{14})$ are selected to correct 4 bit errors in an 8K data block for flash memories. It can be computed that with 70% of the area, the RS decoder can achieve 121% higher throughput than the BCH decoder.

This work was supported by NSF under grant 0802159.

Another contribution of this paper is that, for the first time, a Gray code-based mapping scheme is proposed to map the multi-level voltage value in a memory cell to binary numbers. Compared to the direct mapping scheme that was employed previously, our scheme can achieve 0.02dB and 0.2dB coding gains for the (828, 820) RS code and (8248, 8192) BCH code, respectively, without any overhead.

This paper is organized as follows. Our proposed bit mapping scheme is presented in Section 2. This section also describes the simulation setups. Section 3 presents the decoder design for RS codes and BCH codes. Complexity analyses and comparisons for the two decoders are provided in Section 4. Section 5 draws conclusions.

2. FLASH MEMORY MODELS AND GRAY MAPPING

2.1. Modeling of Cell Threshold Voltage in Multi-level Flash Memory

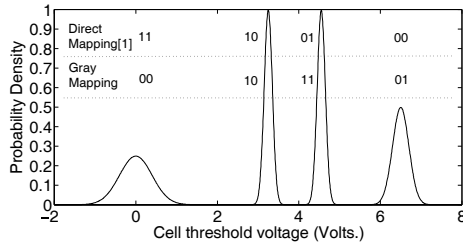


Fig. 1. Cell threshold voltage distribution model for 2-bits/cell MLC flash memories

Table 1. Mean and deviation for the threshold voltage levels in a 2-bit/cell MLC flash memory

Level	3	2	1	0
Direct Mapping [1]	11	10	01	00
Gray Mapping (This paper)	00	10	11	01
Mean (V)	0	3.25	4.55	6.5
Deviation (V)	4σ	σ	σ	2σ

The threshold voltage distribution measurement results for 2-bit/cell MLC flash memories are given by Intel in [1]. For simulation purpose, we use an approximate simulation model [3] shown in Fig. 1. Each of the four threshold levels follows a Gaussian distribution. The mean values and deviations of the four threshold levels are listed in Table 1.

2.2. Gray Mapping and Detection

For 2-bit/cell MLC flash memories, data are grouped into 2-bit pairs, and each pair is mapped to one of the four threshold levels. In prior works, a direct mapping scheme is used. In the direct mapping scheme, the bits mapped to a threshold

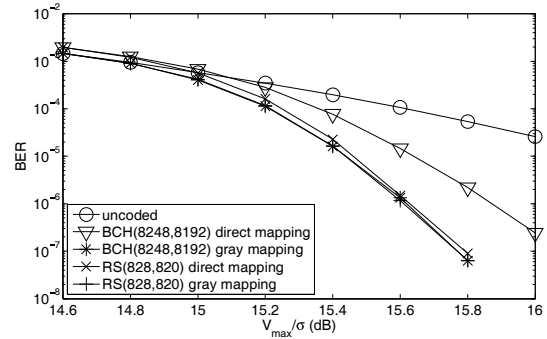


Fig. 2. BER performance for 2-bit/cell MLC flash memories

level are the binary representation of the threshold number, as shown in Table 1. From Fig. 1, when errors happen, it is more likely that a voltage level will be taken as one of its adjacent levels in the detector. In addition, the two voltage levels in the middle have both bits different. Hence two bit errors will happen if one of the middle voltage levels is taken as another by mistake. Alternatively, Gray codes can be used for the bit-mapping. As a result, adjacent voltage levels have only one different bit as shown in Fig. 1, and bit error rate (BER) can be reduced. Fig. 2 shows some simulation results of BERs for using BCH and RS codes in flash memory applications. In our simulations, the decision threshold for signal detection is set to a voltage, whose distances from the two adjacent means are proportional to the corresponding deviations. Accordingly, the three decision thresholds are set to 2.6V, 3.9V and 5.2V. As it can be observed from Fig. 2, around 0.2 dB coding gain can be achieved by the BCH code at BER of 10^{-7} if Gray code mapping is used. The coding gain can be achieved by the RS code using Gray mapping is only 0.02 dB. The coding gain for RS codes is trivial because RS codes are symbol-based. In most of the cases, the two bit errors resulted from taking one voltage level in the middle as another by mistake are located in one symbol and can be corrected. Despite the small coding gain for RS codes, the employment of Gray mapping does not require any overhead.

3. ARCHITECTURES FOR RS AND BCH DECODERS

In [2, 3, 4], binary BCH codes are employed for error correction in MLC flash memories. To protect a page size of 8K bits, a (8248, 8192) BCH code is used. This code is a shortened 4-bit error correcting (16383, 16327) BCH code constructed over $GF(2^{14})$. To shorten an (n, k) systematic code, s message bits/symbols can be deleted. The resulted shortened code is an $(n - s, k - s)$ code. RS codes can be also employed for error correction. From Fig. 2, it can be observed that the (828, 820) RS code can achieve similar or higher coding gain than the (8248, 8192) BCH code. The

(828, 820) RS code is a shortened 4-symbol error correcting (1023, 1015) RS code constructed over $GF(2^{10})$. In addition, compared to the (8248, 8192) BCH code, the (828, 820) RS code has about the same rate and code length in terms of bits.

No prior work has compared the complexity of RS and BCH decoders. In the remaining of this section, the decoder complexity for the BCH and RS codes are compared. The algebraic decodings of RS and BCH codes share three major steps: (a) syndrome computation; (b) key equation solver; (c) error location computation using the Chien search. In addition, since RS codes are non-binary, an extra step is required to determine the error magnitudes using Forney's algorithm. The overall decoder architecture is illustrated in Fig. 3.

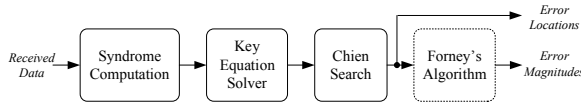


Fig. 3. RS/BCH decoder architecture

The decoding of a shortened code can be accomplished by the same circuit used for the unshortened code with minor modifications. Hereafter, we consider the decoder architecture for primitive t -error correcting (n, k) code constructed over $GF(2^m)$, where $n = 2^m - 1$. The architecture modifications for the shortened codes will be described when necessary. Throughout this paper, the parameters n , k , t and s denote bit numbers and symbol numbers in the context of BCH codes and RS codes, respectively.

3.1. Syndrome Computation

The received bits/symbols can be considered as the coefficients of a degree $n - 1$ polynomial $r(x) = \sum_{i=0}^{n-1} r_i x^i$. The first step of the decoding is to compute the $2t$ syndrome values, i.e. evaluate $r(x)$ over $2t$ consecutive finite field elements starting from α^b :

$$S_i = r(\alpha^{b+i}) = \sum_{j=0}^{n-1} r_j (\alpha^{b+i})^j, 0 \leq i < 2t$$

where α is a primitive element of the finite field and b is an integer. These $2t$ consecutive elements are roots of the generator polynomial of the BCH or RS code.

Applying the Horner's rule, each syndrome can be computed by using a recursive architecture as shown in Fig. 4a, which includes a constant finite field multiplier and an adder. In the RS decoder, $2t$ such units are required to compute $2t$ syndromes in parallel. In this case, $2t$ constant finite field multipliers and $2t$ adders are required. One clock cycle is required to process each received symbol. In shortened RS codes, the deleted message symbols can be considered as zero. Hence only $n - s$ clock cycles are required to compute the syndromes.

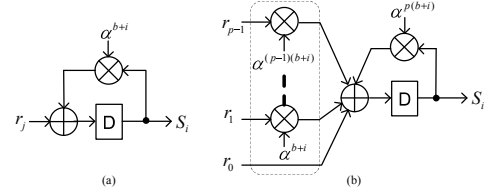


Fig. 4. Syndrome computation: a) serial architecture b) p -parallel architecture

For binary BCH codes, the syndrome computation can be simplified by using the property that $S_{2i} = S_i^2$. Taking $b = 1$, only t syndromes need to be computed explicitly and the rest can be derived through simple squaring computations [5]. However, the number of coefficients in $r(x)$ for the (8248, 8192) BCH decoding is much larger than that in the (828, 820) RS decoding. Hence, higher level of parallel processing needs to be employed in the BCH syndrome computation in order to achieve the same throughput as the RS decoder. A p -parallel processing syndrome computation unit [6] is shown in Fig. 4b, where p received bits are input at each clock cycle and weighted before the recursive summation. If p -parallel processing is employed, then the syndrome computation for BCH decoding requires pt constant multipliers, pt 2-input adders and t squarers. In addition, since each input r_j is binary, the constant multipliers in the dashed block in Fig. 4b can be replaced by directly wiring the inputs to the adder according to the nonzero bits in the $\alpha^{b+i}, \dots, \alpha^{(p-1)(b+i)}$ coefficients. Further area reduction can be achieved by applying substructure sharing [7] among all constant multipliers in the syndrome computation using the iterative matching algorithm (IMA)[8]. This optimization depends on the finite field order and the irreducible polynomials used for field construction.

3.2. Key Equation Solver

Assume ν errors occurred in the received data. The key equation solver step finds the error locations $i_1, i_2, \dots, i_\nu$. This is achieved by making use of an error locator polynomial defined as

$$\Lambda(x) = \prod_{l=1}^{\nu} (1 - \alpha^{i_l} x) = 1 + \lambda_1 x + \dots + \lambda_\nu x^\nu,$$

where α is a primitive element. It can be derived that

$$S_j - \sum_{i=1}^{\nu} \lambda_i S_{j-i} = 0 \quad (1)$$

for $1 \leq j < 2t$. In addition, for RS codes, an error magnitude polynomial $\Omega(x)$ is calculated in this step. Define the syndrome polynomial as $S(x) = S_0 x + S_1 x^2 + \dots + S_{2t-1} x^{2t}$, $\Omega(x)$ is the polynomial that satisfies the following condition:

$$(1 + S(x))\Lambda(x) = \Omega(x) \bmod x^{2t+1}.$$

Efficient algorithms, such as BMA, exist to compute $\Lambda(x)$ and $\Omega(x)$. Starting from $j = 1$, the BMA constructs the $\Lambda(x)$ iteratively to satisfy one more equation described by (1) each time. In each iteration, a discrepancy value $S_j - \sum_{i=1}^j \lambda_i S_{j-i}$ is first computed. If it is not zero, then the $\Lambda(x)$ is modified by making use of a correction polynomial to make the discrepancy zero. It can be observed that the computation of the discrepancy involves polynomial convolutions. In addition, the updating of $\Lambda(x)$ depends on the discrepancy. Therefore, long latency is required in each iteration loop. To increase the speed, a reformulated inversion-less BMA (RiBM) [9] has been proposed. The RiBM algorithm introduces a discrepancy polynomial to calculate the discrepancy value recursively. Moreover, it breaks the dependency between the error locator polynomial and the discrepancy. As a result, the coefficients of both polynomials can be updated simultaneously in each iteration. The RiBM algorithm leads to a simple control scheme and a systolic architecture, which has $3t + 1$ processing elements (PEs) with one multiplier and one adder in the critical path. To reduce the area, the RiBM architecture can be folded by a factor of $3t + 1$ and use only one PE [10]. The folded architecture is shown in Fig. 5.

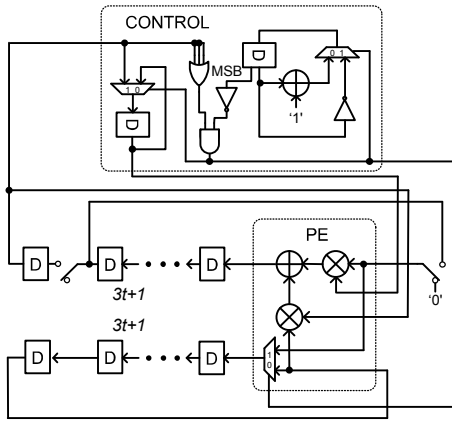


Fig. 5. Ultra folded RiBM architecture [10]

In the architecture shown in Fig. 5, the $2t$ syndromes are loaded into the first $2t$ registers in both rows at the beginning. The rest of the registers are initialized as '0' except that the rightmost registers in both rows are initialized as '1'. Since $2t$ iterations are required and each iteration takes $3t + 1$ clock cycles, the key equation solver step for RS decoding requires $2t \times (3t + 1)$ clock cycles. After $2t \times (3t + 1)$ clock cycles, the coefficients of $\Omega^{(h)}(x)$ and $\Lambda(x)$ are available in first t and the following $t + 1$ registers, respectively, in the top row. Here $\Omega^{(h)}(x)$ satisfies $x^{2t}\Omega^{(h)}(x) + \Omega(x) = 0|_{x=\alpha^{-i_l}}$. Although $\Omega^{(h)}(x)$ does not equal $\Omega(x)$, it has the same evaluation values on α^{-i_l} as $\Omega(x)$. These evaluation values are what actually needed to compute the error magnitudes in the

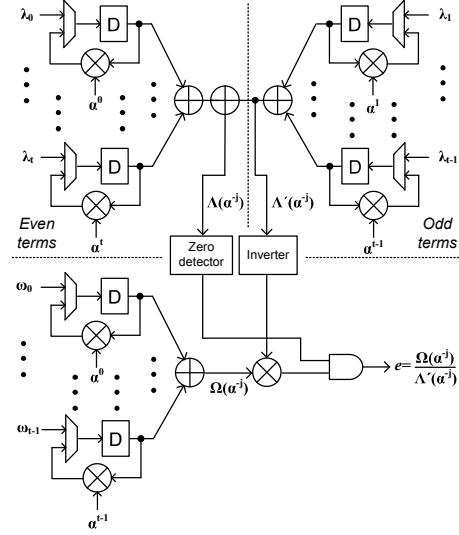


Fig. 6. Architecture for Chien search and error magnitude computation

Forney's algorithm. For the detailed description of this architecture, the interested reader is referred to [9, 10]. The critical path of this architecture has one multiplier, one adder and one multiplexor.

For binary BCH codes, the discrepancy is always zero in each odd iteration. Hence, all the odd iterations can be skipped and only t iterations are required [5]. In addition, since $\Omega^{(h)}(x)$ does not need to be computed, only t registers are required in each row in Fig. 5 and each iteration takes $2t$ clock cycles [4]. Therefore $t \times (2t) = 2t^2$ clock cycles are required for the key equation solver step in BCH decoding.

3.3. Chien Search and Forney's algorithm

After $\Lambda(x)$ is computed, the error locations can be found through computing the roots of $\Lambda(x)$. Root computations over finite fields can be carried out by Chien search, which is based on exhaustive search. In addition, RS decoding requires an additional step to determine the error magnitudes. The Forney's algorithm can be used to calculate the error magnitude for the error located at position i_l :

$$e_{i_l} = \frac{x^{i_l} \Omega(x)}{x \Lambda'(x)} \Big|_{x=\alpha^{-i_l}}. \quad (2)$$

In (2), $x \Lambda'(x) = \lambda_1 x + \lambda_3 x^3 + \dots + \lambda_{2t-1} x^{2t-1}$ is the sum of the odd degree terms of $\Lambda(x)$. It can be directly obtained during the computation of $\Lambda(x)$. Since the error magnitude polynomial $\Omega^{(h)}(x)$ from the RiBM algorithm satisfies $x^{2t}\Omega^{(h)}(x) + \Omega(x) = 0|_{x=\alpha^{-i_l}}$, $\Omega^{(h)}(x)$ is used instead of $\Omega(x)$ in (2), the term x^b can be eliminated by setting $b = -t$.

The architecture for the Chien search and error magnitude computation in RS decoding is shown in Fig. 6. In this architecture, $\omega_0, \omega_1, \dots, \omega_{t-1}$ are the coefficients of $\Omega^{(h)}(x)$. The upper part of this architecture carries out the Chien search and the lower part is for the error magnitude computation. The evaluations of the even terms and odd terms of $\Lambda(x)$ are done by the upper left and right parts, respectively. Then the outputs of these two parts are added up. If the sum is zero, it indicates that a root has been found. Since the evaluation values of the odd terms are the evaluation values of the denominator in (2), the output of the upper right part can be directly used in the error magnitude computation. From (2), the Forney's algorithm requires one finite field inversion. To reduce the area requirement of inversion, composite field arithmetic can be employed [11]. In addition, since the inversion is located in a feed-forward path, pipelining can be applied to reduce the length of the critical path. The hardware complexity for the Chien search and error magnitude computation block is $2t$ constant multipliers, $2t - 1$ 2-input adders, one multiplier and one inverter. The architecture of Fig. 6 requires the search to begin with the field element α or 1. Since the roots of $\Lambda(x)$ are the reciprocals of the error locations, the error correction would always begin from the α^{2^m-2} position. One field element is tried in each clock cycle. Hence, $2^m - 1$ clock cycles are required to find all the error locations.

In binary BCH decoding, there is no need to compute the error magnitudes. In addition, the same architecture in the upper part of Fig. 6 can be used for the Chien search to find the error locations. However, since the BCH code is constructed over a much larger finite field than the RS code, a much larger number of field elements need to be checked in the Chien search. Therefore, to achieve the same decoding throughput, a large parallel processing factor needs to be employed to the Chien search for BCH decoding. Assuming a parallel processing factor of p is used, the hardware complexity of the Chien search block is pt constant multipliers and $p(t - 1)$ adders. To further reduce the hardware, substructure sharing can be applied among all the constant multipliers. In addition, the Chien search can be carried out on $\tilde{\Lambda}(x) = \lambda_\nu + \lambda_{\nu-1}x + \dots + \lambda_1x^{\nu-1} + x^\nu$ in binary BCH decoding since the computation of $x\Lambda'(x)|_{x=\alpha^{-i_l}}$ is not required. In this case, the roots found for $\tilde{\Lambda}(x)$ are reciprocals of the roots of $\Lambda(x)$. Hence, the computed roots are already the error locations and the error correction can start from the α position. For shortened codes, there is no need to find the errors in the deleted message bits. Therefore, only $n - s$ field elements need to be checked in the Chien search. Employing p -parallel processing, the Chien search for the BCH decoding requires $\lceil \frac{n-s}{p} \rceil$ clock cycles.

3.4. Computation Scheduling

From the architectures discussed above, the latencies of both the syndrome computation and Chien search are proportional

to the code length. On the other hand, the latency for the key equation solver is proportional to t^2 . For the error correction in MLC flash memories, the error-correcting capability t is much smaller than the code length. Hence, the latencies of the syndrome computation and Chien search dominate. Accordingly, although pipelining can be employed to break the computation of all three steps, pipelining will be only applied to separate the Chien search step from the others in order to increase the hardware utilization efficiency. In this case, the clock cycle number required for decoding each block is the maximum of the clock cycle numbers required for syndrome computation and key equation solver and that of the Chien search (and error magnitude computation for RS decoding).

4. HARDWARE COMPLEXITY AND THROUGHPUT COMPARISONS

Table 2 and 3 summarize the gate count and critical path of each building block except the control block in the (828, 820) RS decoder and (8246, 8192) BCH decoder, respectively. All gates in these tables refer to 2-input gates except the single input NOT gates, and the MUXes refer to 1-bit 2:1 multiplexers. In the (8246, 8192) BCH decoder, a parallel processing factor of 8 is employed in both the syndrome computation and Chien search blocks. Multiple irreducible polynomials can be used to construct $GF(2^{10})$ and $GF(2^{14})$. $p(x) = x^{10} + x^3 + 1$ and $p(x) = x^{14} + x^{10} + x^6 + x + 1$ are picked in our design in order to minimize the hardware complexity.

Table 2. Gate counts and critical paths for the building blocks in the (828, 820) RS decoder

-	Area	Critical path
Multiplier	101XOR+100AND	5XOR+1AND
Adder	10XOR	1XOR
Inverter	164XOR+160AND+36OR+5NOT	16XOR+5AND+3OR+1NOT
Syndrome Computation	116XOR	-
Key Equation Solver	212XOR+200AND+30MUX	6XOR+1AND+1MUX
Chien Search	50XOR+90R+90MUX	-
Forney algorithm	301XOR+270AND+36OR+5NOT	-
Total Gate Count	679XOR+470AND+45OR+5NOT+120MUX	6XOR+1AND+1MUX

Table 3. Gate counts and critical paths for the building blocks in the (8246, 8192) BCH decoder

-	Area	Critical path
Multiplier	221XOR+196AND	14XOR+1AND
Adder	14XOR	1XOR
Squarer	21XOR	3XOR
Syndrome Computation	339XOR	-
Key Equation Solver	456XOR+392AND+42MUX	15XOR+1AND+1MUX
Chien Search	727XOR+104OR+56MUX	-
Total Gate Count	1522XOR+392AND+104OR+98MUX	15XOR+1AND+1MUX

From Table 2 and 3, it can be observed that the multiplier in $GF(2^{10})$ requires less than 50% of the area of the multiplier in $GF(2^{14})$ and the critical path is also much shorter. The major reason is that a trinomial exists as an irreducible

polynomial for the construction of $GF(2^{10})$ but not $GF(2^{14})$. As it was mentioned before, the inverter required for the error magnitude computation in (2) is implemented by using composite field arithmetic [11]. In addition, the critical path of the composite field inverter is almost three times of that in the overall design. Hence, the inverter is pipelined into three stages.

As it was mentioned in the previous section, the constant multipliers for the syndrome computation in BCH decoding can be replaced by wiring the input bits, and substructure sharing can be applied among the multipliers. In addition, the number of syndromes needs to be computed explicitly is only t for a t -error-correcting BCH code. After taking these optimizations into account, the syndrome computation in the (8246, 8192) BCH decoding still requires more than twice of the area as that in the (828, 820) RS decoding due to the 8-parallel processing employed. From Table 2 and 3, it can be also observed that the key equation solver block in the RS decoder has much lower complexity than that in the BCH decoder. This is due to the simpler multipliers in $GF(2^{10})$. The Chien search in the RS decoder has a much smaller area than the one in BCH decoder. The major reason is that in the Chien search of BCH decoding, high parallel processing is employed to search a much larger number of finite field elements.

In summary, the overall hardware complexity for a (828, 820) RS decoder is 679 XOR gates, 470 AND gates, 45 OR gates, 5 NOT gates, 120 MUXes, and 534 registers, the implementation of the (8246, 8192) BCH decoder requires 1522 XOR gates, 392 AND gates, 104 OR gates, 98 MUXes, and 638 registers. Assume that each AND or OR gate occupies 3/4 of the area of an XOR, each NOT gate occupies half of the area of an XOR, a MUX has the same area as an XOR, and each register requires about 3 times of the area of an XOR. The (828, 820) RS decoder occupies 70% of the area of the (8246, 8192) BCH decoder, despite the extra step required to compute the error magnitudes using the Forney's algorithm.

The minimum achievable clock period of both the RS and BCH decoders is limited by the feedback loop in the key equation solver block. It contains one multiplier, one adder and one MUX. Since there are no feedback loops involved in other parts of the decoder architecture, pipelining can be employed to have a critical path of one multiplier, one adder and one MUX. Hence, the critical path of the RS decoder is about 47% of that in the BCH decoder. The (828, 820) RS decoder requires 828, 104 and 1024 clock cycles for the syndrome computation, key equation solver, Chien search and error magnitude computation, respectively. The (8248, 8192) BCH decoder requires 1031, 32 and 1031 clock cycles for the three decoding steps, respectively. Applying the scheduling scheme mentioned in the previous section, it can be derived that decoding one block of (828, 820) RS code requires 1024 clock cycles, while 1063 clock cycles are required for the (8248, 8192) BCH code. Accordingly, the RS decoder

can achieve 121% higher throughput than the BCH decoder.

When codes constructed over other fields are considered, the area and throughput differences may become less if there is no trinomial that can be used to construct the finite field for RS codes. However, it is still expected that, with similar error-correcting performance, RS codes of similar code rate and length requires smaller area and can achieve higher efficiency in the decoder implementation.

5. CONCLUSION

In this paper, a novel bit-mapping scheme using the Gray code is proposed. The proposed scheme can achieve additional coding gain without any overhead. In addition, the complexity of RS codes and BCH codes with similar code rate and length are compared. It can be concluded that the RS codes can achieve similar error-correcting performance as the BCH codes in MLC flash memory applications with lower decoding complexity. Therefore, RS codes are more suitable for these applications. Future studies will be directed to develop more efficient decoders for flash memories applications.

6. REFERENCES

- [1] G. Atwood, A. Fazio, D. Mills and B. Reaves, "Intel StrataFlash memory technology overview," *Intel Technology Journal*, 4th Quarter pp. 1-8, 1997,
- [2] R. Micheloni, R. Ravasio, A. Marelli, *et. al.*, "A 4Gb 2b/cell NAND Flash Memory with Embedded 5b BCH ECC for 36MB/s System Read Throughput," *IEEE Inter. Solid-State Circuits Conf.*, Feb. 2006
- [3] F. Sun, K. Rose and T. Zhang, "On the Use of Strong BCH Codes for Improving Multilevel NAND Flash Memory Storage Capacity," *IEEE Workshop on Signal Processing Systems (SiPS)*, Oct. 2006
- [4] W. Liu, J. Rho and W. Sung, "Low-power high-throughput BCH error correction VLSI design for multi-level cell NAND flash memories," *IEEE Workshop on Signal Processing Systems (SiPS)*, pp. 248-253, 2006
- [5] S. Lin and D. J. Costello, Jr. *Error Control Coding: Fundamentals and Applications (2nd edition)*, Prentice Hall, 2004.
- [6] L. Song, M. Yu and M. S. Shaffer, "10- and 40-Gb/s Forward Error Correction Devices for Optical Communications," *IEEE Journal of Solid-State Circuits*, Vol. 37, No. 11, Nov. 2002
- [7] K. K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation*, Wiley, Jan. 1999
- [8] Y. Chen and K. K. Parhi, "Small area parallel Chien search architectures for long BCH codes," *IEEE Trans. on VLSI*, Vol 12, Issue 5, May 2004.
- [9] D. V. Sarwate and N. R. Shanbhag, "High-Speed Architectures for Reed-Solomon Decoders," *IEEE Trans. On VLSI*, vol 9, Oct. 2001.
- [10] K. Viswajith Seth, K.N. Srinivasan and S. Kamakoti, "Ultra folded high-speed architectures for Reed Solomon decoders," *VLSI Design'06*, pp.1063-9667, Jan. 2006.
- [11] C. Paar, "Efficient VLSI Architecture for Bit-Parallel Computations in Galois Field," Ph.D thesis, Institute for Experimental Mathematics, University of Essen, Germany, 1994.