Efficient Reconfigurable Vandermonde Matrix Inverter for Erasure-Correcting Generalized Integrated Interleaved Decoding

Yok Jye Tang
Dept. of Electrical & Computer Engineering
The Ohio State University
Columbus, OH 43210 U.S.A.
tang.1121@osu.edu

Xinmiao Zhang
Dept. of Electrical & Computer Engineering
The Ohio State University
Columbus, OH 43210 U.S.A.
zhang.8952@osu.edu

***Abstract*—Generalized integrated interleaved (GII) codes constructed using Reed-Solomon (RS) codes enable local erasure correction with low complexity and are essential to shorten the failure recovery latency in hyper-scale distributed storage. Erasure corrections for storage systems are usually done by multiplying the inverse of a Vandermonde matrix to the syndrome vector. Previous Vandermonde matrix inversion architectures suffer from long latency. Besides, the GII decoding rounds have increasing erasure-correcting capability. Using a Vandermonde inverter dedicated for worst-case erasure correction leads to low hardware efficiency. This paper first proposes a low-latency Vandermonde matrix inverter architecture. By exploiting the regularity of our proposed architecture, an efficient re-configurable inverter supporting matrices of variable sizes is also developed to increase the efficiency of GII-RS erasure-correcting decoding. For 8×8 matrix inversion over $GF(2^8)$, our proposed architecture reduces the latency by 77% with similar complexity compared to the previous design. Our reconfigurable inverter for an example GII code achieves 64% and 32% reductions on latency and area, respectively, compared to the best alternative design.**

***Index Terms*—Erasure-correcting decoding, generalized integrated interleaved codes, Reed-Solomon codes, Vandermonde matrix inversion.**

I. Introduction

Reed-Solomon (RS) codes are traditionally used for failure recovery in distributed storage. However, an (n, k) RS code needs to access k symbols to recover from any failure. k grows as the storage system scales. Accessing a large number of symbols for failure recovery causes long latency and large penalty on the network bandwidth. To enable the continued scaling of distributed storage, erasure codes that read much fewer symbols for recovery are essential. Generalized integrated interleaved (GII)-RS codes [1], [2] are among the best locally recoverable erasure codes that achieve this goal. Most of the time, the number of erasures is small and they can be corrected by localized decoding over sub-codewords with short latency. Besides, the sub-codewords are nested to produce codewords of stronger RS codes that can correct more erasures. The decoding locality is further improved by optimizing the nesting scheme [3], [4].

GII-RS decoding consists of multiple rounds of RS decoding with increasing correction capability. Although error-correcting GII-RS decoding was explored in [5]–[8], erasure-correcting GII-RS decoding has not been investigated previously. For storage systems, the number of erasures is not large and their locations are known. Hence erasure-correcting RS decoding is typically implemented by multiplying the syndrome vector with the inverse of a Vandermonde matrix that is decided by the erasure locations. Such decoding in software has been extensively studied [9]–[14]. However, hardware implementation of Vandermonde matrix inversion that is needed to achieve high speed has only been investigated in [6]. The erasures can be also computed by the simplified formula in [15] when there are at most 4 of them. Nevertheless, such formulas become much more complicated for more erasures.

The Vandermonde matrix inverter in [6] suffers from long latency, which also increases significantly with the matrix size. Besides, although the correction capability increases over the GII decoding rounds, the later rounds are activated with low probability. If the architecture in [6] configured for the highest correction capability is used, the area requirement is large and the majority of the hardware units idle most of the time. Hence, an efficient Vandermonde matrix inverter supporting different matrix dimensions is needed for GII decoder.

In this paper, by reformulating the Vandermonde matrix inversion formulas, an inverter architecture is first developed to reduce the latency. Our proposed inverter architecture also has the advantage that it consists of identical copies of substructures. By mapping the computations to a substructure in a time-multiplexed manner, an efficient reconfigurable inverter architecture supporting variable matrix size is also developed for GII decoding. For 8×8 matrix inversion over $GF(2^8)$, our proposed architecture reduces the latency by 77% with similar complexity compared to that in [6]. Our reconfigurable architecture supporting both 4×4 and 8×8 inversions achieves

This material is based upon work supported by the National Science Foundation under Award No. 2011785.

978-1-6654-8524-1/22/$31.00 ©2022 IEEE

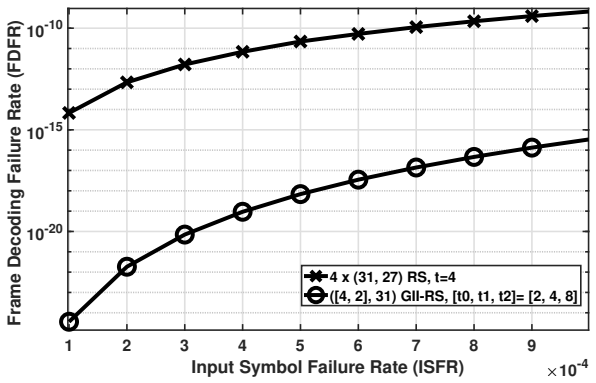

Fig. 1. Frame decoding failure rates (FDFRs) of GII-RS and un-nested RS codes for erasure correction.

32% area reduction and reduces the latency by around 64% compared to the best alternative reconfigured inverter design.

This paper is organized as follows. Section II introduces GII-RS erasure-correcting decoding and prior Vandermonde matrix inverter architectures. The two proposed inverter architectures are detailed in Section III. Complexity comparisons are done in Section IV and conclusions follow in Section V.

II. Backgrounds

A $([m, v], n)$ GII code [1], [2] can be defined by using $v+1$ RS codes $\mathcal{C}_v(n, k_v) \subseteq \cdots \subseteq \mathcal{C}_1(n, k_1) \subset \mathcal{C}_0(n, k_0)$ over $GF(2^q)$ of length n and dimensions $k_v \leq \cdots \leq k_1 < k_0$. A codeword of GII code consists of m sub-codewords $c_0, c_1, \cdots, c_{m-1} \in \mathcal{C}_0(n, k_0)$. Their nesting produces v codewords of $\mathcal{C}_1(n, k_1), \cdots, \mathcal{C}_v(n, k_v)$ as follows

$$\mathcal{C} \triangleq \left\{ c=[c_0,\cdots,c_{m-1}] : c_i \in \mathcal{C}_0, \tilde{c}_j = \sum_{i=0}^{m-1} \alpha^{ij} c_i \in \mathcal{C}_{v-j}, 0 \leq j < v \right\},$$

where α is a primitive element of $GF(2^q)$. GII decoding consists of two stages. The first is the traditional RS decoding over individual sub-codewords that can correct up to $t_0 = n - k_0$ erasures. If any sub-codeword has more than t_0 erasures, the second-stage nested decoding that has up to v rounds is activated. In the η-th $(1 \leq \eta \leq v)$ round, higher-order syndromes for $v+1-\eta$ sub-codewords are calculated by utilizing the nested codewords. Then RS decoding is carried out to correct $t_\eta = n - k_\eta$ erasures in each of those sub-codewords. Most of the time, the number of failures in distributed storage is small and they are corrected by decoding individual short sub-codewords. Compared to decoding a RS code whose length is mn, such localized decoding has much shorter latency. On the other hand, more erasures are correctable through the nesting. Fig. 1 shows the frame decoding failure rate (FDFR) of a $([4, 2], 31)$ GII-RS code over $GF(2^8)$ with $[t_0, t_1, t_2] = [2, 4, 8]$ for a range of input symbol failure rates (ISFRs). It is much lower than that of four traditional un-nested $(31, 27)$ RS code that has the same redundancy and similar decoding latency as that of the sub-codewords.

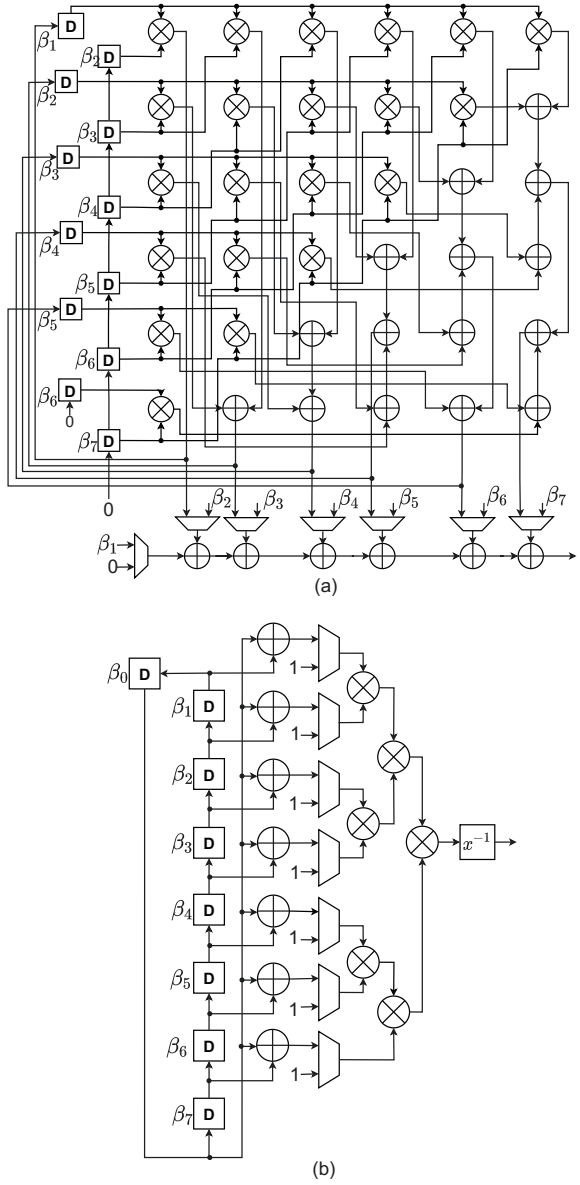

Fig. 2. Inverter architecture for 8×8 Vandermonde matrix [6]: (a) numerator computation unit; (b) denominator computation unit.

When t is a small number as needed for most storage systems, t-erasure correction can be more efficiently done by multiplying the syndrome vector with the inverse of a $t \times t$ Vandermonde matrix, V, that is specified by the t erasure locations $\beta_0, \beta_1, \cdots, \beta_{t-1}$. The entries of V^{-1} for $0 \leq i < t$ and $0 \leq j < t-1$ can be computed as

$$V^{-1}_{i,j} = \frac{\sum_{0 \leq i_1 < \cdots < i_{t-1-j} < t; i_1, \cdots, i_{t-1-j} \neq i} \beta_{i_1} \cdots \beta_{i_{t-1-j}}}{\Pi_{0 \leq l < t, l \neq i} (\beta_i - \beta_l)}, \quad (1)$$

and $V^{-1}_{i,t-1} = 1/\Pi_{0 \leq l < t, l \neq i}(\beta_i - \beta_l)$.

For an example 8×8 matrix inversion, the architecture shown in Fig. 2(a) [6] calculates the numerators of $V^{-1}_{0,j}$ according to (1) for $j = 6, 5, \cdots, 0$ in clock cycles $0, 1, \cdots, 6$, respectively. Then different β's are loaded into the registers

$$N_{0,6} = \beta_1 + \beta_2 + \beta_3 + \cdots + \beta_7$$

$$N_{0,5} = \beta_1(\beta_2 + \cdots + \beta_7) + \beta_2(\beta_3 + \cdots + \beta_7) + \beta_3(\beta_4 + \cdots + \beta_7) + \cdots + \beta_6\beta_7$$

$$N_{0,4} = \beta_1(\beta_2(\beta_3 + \cdots + \beta_7) + \beta_3(\beta_4 + \cdots + \beta_7) + \cdots + \beta_6\beta_7) + \beta_2(\beta_3(\beta_4 + \cdots + \beta_7) + \cdots + \beta_6\beta_7) + \cdots + \beta_5\beta_6\beta_7$$

$$\vdots$$

$$N_{0,0} = \beta_1\beta_2\beta_3\beta_4\beta_5\beta_6\beta_7$$

Fig. 3. Reformulated numerator computations for the inverse of 8×8 Vandermonde matrix.

to compute the numerators in the other rows of V^{-1}. In total, it takes $t(t-1) = 56$ clock cycles to compute all the numerators. The t entries in the same row of V^{-1} have the same denominator and the architecture in Fig. 2(b) computes one denominator in each clock cycle by cyclically shifting the β's stored in the registers. The architectures in Fig. 2 can be also used to invert smaller matrices. To invert a $t' \times t'$ matrix with $t' < 8$, the last $8 - t'$ of each of the two columns of register in Fig. 2(a) are initialized to zero, and the $\beta_{t'}$ through β_{t-1} inputs to the multipexers are also set to zero. In Fig. 2(b), the registers for $\beta_{t'}$ through β_{t-1} hold invalid values. The corresponding inputs sent to the multipliers are replaced by '1' to take out the effects of these invalid entries. Overall, the latency for inverting a $t \times t$ matrix is $t(t-1)$ and it increases significantly for larger t. Additionally, for GII decoding, t_v is usually much larger than t_{v-1} as in the example code whose FDFR is shown in Fig. 1. Besides, later decoding rounds are activated with lower probability. If an inverter for t_v-erasure correction is employed, it requires large area and most of its hardware units would be idling during the earlier decoding rounds.

III. Efficient re-configurable Vandermonde matrix inverter for GII-RS erasure decoder

In this section, a Vandermonde matrix inverter architecture is first developed to substantially reduce the latency with similar complexity compared to the design in [6] by reformulating the formula in (1). By exploiting the regularities of our first design, a re-configurable inverter architecture that supports variable matrix sizes is also proposed to reduce the area of GII-RS erasure decoders without sacrificing the latencies of earlier nested decoding rounds.

Let $N_{i,j}$ be the numerator in (1). For the example case of $t = 8$, the computations of $N_{0,j}$ for $0 \leq j < t-1$ can be reformulated as shown in Fig. 3. It can be observed that $N_{0,j}$ consists of $j+1$ terms and the sum of the last l terms multiplied by β_{j+1-l} happens to be the $(j+1-l)$-th term in the $N_{0,j-1}$ formula. For example, the last $l = 1$ term in $N_{0,6}$ is β_7. $\beta_7\beta_{j+1-l} = \beta_7\beta_{6+1-1} = \beta_7\beta_6$ is the $j+1-l = 6$-th term in the $N_{0,5}$ formula. The sum of the last $l = 6$ terms in $N_{0,6}$ is $\beta_2 + \beta_3 + \cdots + \beta_7$. Its product with $\beta_{j+1-l} = \beta_1$ is the first term of $N_{0,5}$. Similarly, the sums of the last 1, 4, and 5 terms in $N_{0,5}$ multiplied by β_5, β_2, and β_1 equal to the fifth, second, and first terms, respectively, in the $N_{0,4}$ formula as showed by the circles in Fig. 3. Similar reformulations for $N_{i,j}$ with $i = 1, 2, \cdots, 7$ can be also derived. Accordingly, intermediate results can be shared and an efficient architecture that computes the $N_{i,j}$ in the same row of V^{-1} simultaneously is developed as shown in Fig. 4. Pipelining is applied according to the cutsets shown by the dashed lines to achieve one multiplier and one adder in the critical path. Taking into account the $t-3$ clock cycles of pipelining latency, our proposed architecture only takes $2t-3$ instead of $t(t-1)$ clock cycles as needed by the design in Fig. 2(a) to compute all $N_{i,j}$. The architecture for a larger matrix can be easily derived by adding more columns of adders and multipliers to that in Fig. 4.

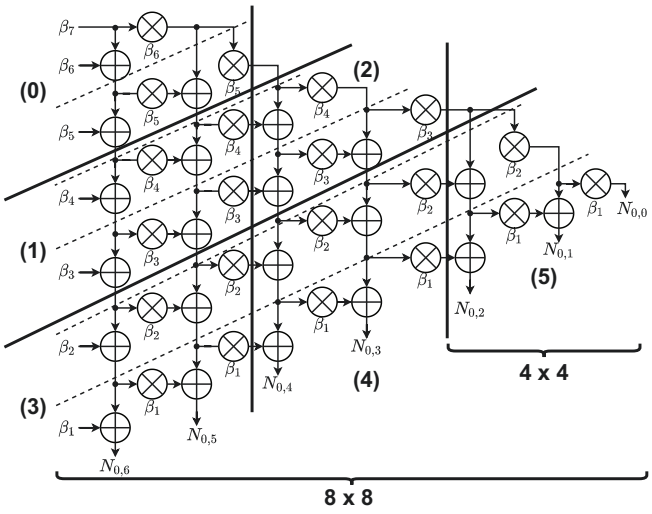

Fig. 4. Proposed numerator computation architecture for 8×8 Vandermonde matrix inversion.

For general cases, the complexity of the proposed numerator computation architecture and that of the design in [6] are listed in Table. I. To reduce the critical path, pipelining is applied to separate the multiplier-adder array from the bottom multiplexers in Fig. 2(a). The extra one clock cycle of pipelining latency and the complexities of the pipelining registers are included in Table. I. Since the multiplier-adder array has feedback loops, the critical path can not be further reduced by the pipelining. From this table, it can be observed that the proposed architecture has very similar complexity as

TABLE I
COMPLEXITIES OF VANDERMONDE MATRIX INVERSION NUMERATOR COMPUTATION ARCHITECTURES

	[6]	proposed
Mult.	$\sum_{i=1}^{t-2} i$	$\sum_{i=1}^{t-2} i$
Add.	$\sum_{i=1}^{t-2} i$	$\sum_{i=1}^{t-2} i$
Reg.	$3t-6$	$\sum_{i=2}^{t-2} i$
Mux.	$t-1$	$t-1$
Critical path	1 mult. +$\lceil \log_2(t-2) \rceil$ add.	1 mult. + 1 add.
Latency (# clks)	$t(t-1)+1$	$2t-3$

the design in [6]. Nevertheless, the proposed design has much shorter latency, especially when t is larger.

As aforementioned, increasingly larger Vandermonde matrices need to be inverted over GII decoding rounds. Besides, although the correction capability of the last round, t_v, is typically much larger than those of previous rounds, later decoding rounds are activated with much lower probability. Efficient GII decoding requires a reconfigurable Vandermonde matrix inverter that has similar area and latency as an inverter dedicated for a smaller matrix and can also implement the inversions of larger matrices with extra clock cycles.

Consider the example $([4,2],31)$ GII-RS code with $[t_0,t_1,t_2]=[2,4,8]$. The two nested decoding rounds require 4 and 8-erasure corrections. In the case of 4×4 matrix inversion, only the right tip of the architecture in Fig. 4 is needed. Besides, this architecture is very regular. It can be decomposed into six parts as separated by the thicker lines in Fig. 4 and each part shares similar structure as that for 4×4 inversion. Accordingly, a single substructure with 4 multipliers and 4 adders can implement the numerator computation for 4×4 matrix inversion and all the calculations for 8×8 matrix inversion can be implemented by the substructure in a time-multiplexed manner. Following the data flow, the calculations can be carried out according to the order listed in the parentheses in Fig. 4.

Fig. 5(a) shows the proposed reconfigured numerator computation architecture for both 4 and 8-erasure corrections. The pipelining applied to the architecture in Fig. 4 is preserved and hence the two gray registers are inserted in Fig. 5(a). In this case, the computation for each part in Fig. 4 takes two clock cycles. The other registers and multiplexers in Fig. 5(a) are used to store intermediate results and route them to proper units for the later computations needed for 8×8 matrix inversion according to the data flow in Fig. 4. Take the computation of part 0 as an example. β_7 and β_6 are sent to the inputs of the upper-left adder in Fig. 5(a) in clock cycle 0 and β_5 is connected to the horizontal input of the bottom-left adder in clock cycle 1. At the end of clock cycle 1, the three outputs of part 0 are available. The first two outputs are also the inputs to the top two adders of part 1. Hence, they are delayed by one register each and fed back to the two adders on the top of Fig. 5(a). The third output of part 0 is an input for part 2 calculation carried out in clock cycle $2\times2+0=4$. Hence,

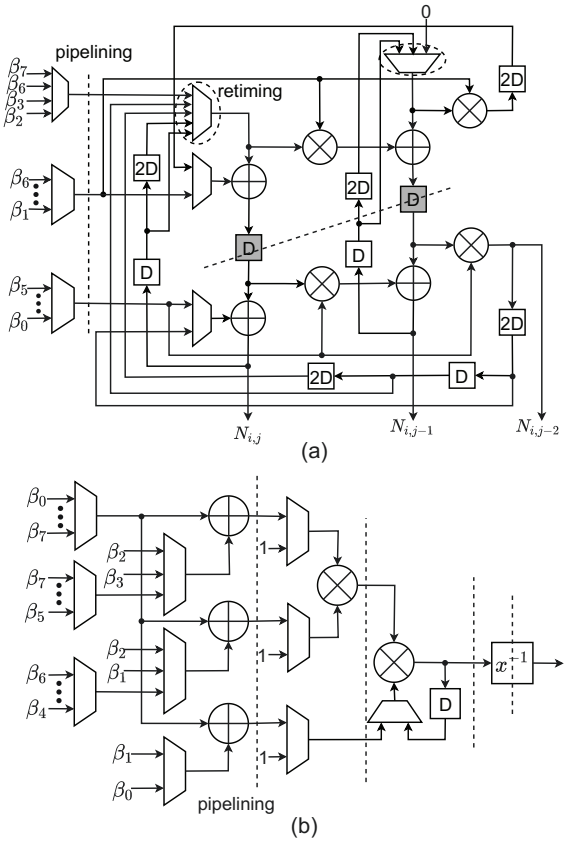

Fig. 5. Proposed reconfigurable inverter for both 4 and 8-erasure corrections: (a) numerator computation unit; (b) denominator computation unit.

this output is delayed by three registers in total before it is routed to the top left adder and multiplier. By following the data flow, registers and multiplexers can be added in a similar way to implement the computations of the other parts. Since the architecture in Fig. 4 is symmetric, assigning computation orders to the parts along the vertical direction would lead to similar numbers of registers and multiplexers as in Fig. 5(a).

To reduce the critical path of the numerator computation architecture to one multiplier and one adder, one register is inserted after each of the left-most multiplexers through pipelining and then retiming is applied to the cutsets circling the multiplexers in Fig. 5(a). For 4×4 matrix inversion, it takes 2+3+1=6 clock cycles to compute all the numerators considering the pipelining latency. For 8×8 matrix inversion, since there are 6 parts in Fig. 4, it takes $6\times2=12$ clock cycles to compute the numerators for one row in the matrix. However, in the last clock cycle, the computations for the next row can start. Hence, $12+11\times7+1=90$ clock cycles are required to compute all numerators for an 8×8 matrix inversion.

A low-complexity reconfigurable denominator computation architecture is also proposed as shown in Fig. 5(b). The adders compute $\beta_i-\beta_l$, and their product is calculated by the multipliers according to (1). The horizontal and vertical inputs of each adder correspond to β_i and β_l, respectively. Similarly, pipelining can be applied to the cutsets denoted by

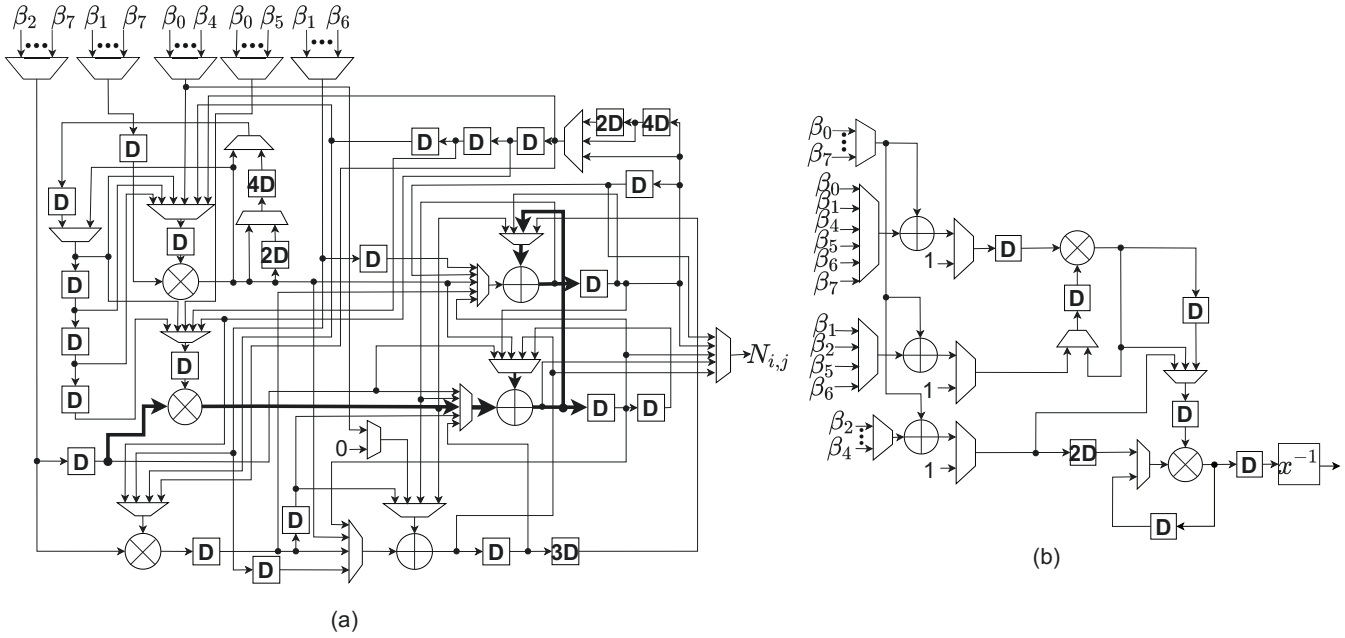

Fig. 6. Reconfigurable Vandermonde matrix inverter architecture for both 4 and 8-erasure corrections by applying optimized folding to the design in Fig. 2: (a) numerator computation unit; (b) denominator computation unit.

TABLE II
COMPLEXITIES OF 8×8 VANDERMONDE MATRIX INVERSION NUMERATOR COMPUTATION ARCHITECTURES OVER $GF(2^8)$

	Mult.	Add.	Reg.	Mux.	Total # XORs	Crit. path # gates	Latency # clks
[6]	21	21	18	7	2714	9	57
Proposed	21	21	20	7	2762	7	13

the dotted lines in Fig. 5(b) such that the critical path consists of one multiplier and one multiplexer. For 4×4 matrices, one denominator is available at the output of the rightmost multiplier in each clock cycle. Hence, $4 + 4 = 8$ clock cycles are needed to compute all the denominators considering the pipelining latency. To invert an 8×8 matrix, the register in the feedback loop holds the partially computed product of $\beta_i - \beta_l$, which is multiplied to two additional $\beta_i - \beta_l$ in each of the following clock cycles. Hence, three clock cycles are required to compute one denominator for 8×8 matrix inversion and it takes $3 \times 8 + 4 = 28$ clock cycles to compute the 8 denominators taking into account the pipelining latency.

IV. HARDWARE COMPLEXITY COMPARISONS

This section analyzes the complexities of our proposed inverter architectures. They are also compared with prior designs for example 8×8 matrix inversion and reconfigurable inversion supporting the example $([4, 2], 31)$ GII-RS decoding.

For fixed-size matrix inversion, the same denominator architecture as in [6] can be used and hence only the numerator computation architectures are compared in Table II. Codes over $GF(2^8)$ are preferred for storage systems since each element is represented by a byte. Each adder over $GF(2^8)$ is implemented as bit-wise XOR and each 8-bit 2-to-1 multiplexer has similar complexity. An 8-bit register has around the same area as 24 XOR gates. A $GF(2^8)$ multiplier can be implemented using the area of 98 XOR gates and has a critical path of 6 gates [6]. These assumptions are used to estimate the complexity in Table II. Besides, one 2-to-1 multiplexer is needed for each different β input in Fig. 2(a) and Fig. 4 to compute numerators for different rows. The complexities of these multiplexers and pipelining registers are also included in Table II. Considering the pipelining latency, the design in [6] needs $8(8 - 1) + 1 = 57$ clock cycles. Compared to this design, our architecture reduces the latency in terms of clock cycle number by $1 - 13/57 = 77\%$ with similar complexity. Besides, our design has shorter critical path.

Reconfigurable Vandermonde matrix inverter architecture does not exist previously. An alternative possible design can be developed by applying folding [16] to the architectures in Fig. 2. To compute the numerators for the 4×4 matrix inversion, the multipliers in the $3, 4, 5, 6$-th anti-diagonals counting from the top left corner of Fig. 2(a) have zero inputs. Hence, increasing folding orders are assigned to the multipliers in anti-diagonal pattern starting from the one in the top left corner. In this case, those multiplications over zeros can be skipped to reduce the latency of inverting the 4×4 matrix in the folded architecture. The folding orders also affect the numbers of registers and multiplexers needed to store and route intermediate results in the folded architecture. The folded architectures shown in Fig. 6 are derived by using optimized folding orders that reduce registers and multiplexers. Also pipelining and retiming can be applied to reduce the critical path to 1 multiplier and 7 adders/multiplexers, which is highlighted by the thicker wire in Fig. 6(a). Since there are multiple feedback loops in Fig. 6(a), the critical path can not be further reduced.

TABLE III
COMPLEXITIES OF RECONFIGURABLE VANDERMONDE MATRIX INVERTERS FOR BOTH 4 AND 8-ERASURE CORRECTIONS OVER $GF(2^8)$

	Mult.	Add.	Reg.	Mux.	Inv.	Total # XORs	Critical path # gates	Latency # clks
reconfigurable numerator computation architecture from [6] folded	3	3	35	68	0	1702	13	257 (for 8×8 matrix inversion) 13 (for 4×4 matrix inversion)
reconfigurable denominator computation architecture from [6] folded	2	3	9	24	1	803	8	28 (for 8×8 matrix inversion) 8 (for 4×4 matrix inversion)
reconfigurable Vandermonde matrix inverter architecture from [6] folded	5	6	44	92	1	2505	13	257 (for 8×8 matrix inversion) 13 (for 4×4 matrix inversion)
proposed reconfigurable numerator computation architecture	4	4	16	21	0	976	7	90 (for 8×8 matrix inversion) 6 (for 4×4 matrix inversion)
proposed reconfigurable denominator computation architecture	2	3	7	20	1	723	7	28 (for 8×8 matrix inversion) 8 (for 4×4 matrix inversion)
proposed reconfigurable Vandermonde matrix inverter architecture	6	7	23	41	1	1699	7	90 (for 8×8 matrix inversion) 8 (for 4×4 matrix inversion)

Table III shows the complexity of our proposed reconfigurable Vandermonde matrix inverter architecture for both 4 and 8-erasure corrections and that of the alternative design derived by folding. Unlike our proposed design in Fig. 4, the architecture in Fig. 2 can not be divided into parts with similar structures. Hence, the architecture derived from folding requires a larger number of multiplexers to route proper signals to the multipliers and adders. Despite our optimization on the folding orders, a large number of registers are still needed to hold intermediate results. Assume that the area of an $GF(2^8)$ inverter, which is around the same as 175 XOR gates [6]. Our proposed reconfigured inverter architecture achieves 1-1699/2505=32% area reduction and reduces the latency in terms of clock cycle number for inverting either a 4×4 or 8×8 matrix by around 64%. Besides, our design has much shorter critical path.

For larger matrices, our proposed inverter architectures can achieve even more significant improvements over prior designs since the latency of our design is linear to t instead of t^2. Besides, our reconfigurable design can be easily extended to support more than 2 matrix sizes. This is difficult to achieve by folding the architecture in [6] since it can not be divided into similar substructures.

V. CONCLUSIONS

For the first time, this paper addresses Vandermonde matrix inverter for efficient erasure-correcting GII-RS decoding. By reformulating the Vandermonde matrix inversion, intermediate results can be shared. Accordingly, an architecture consisting of similar substructures is developed to compute the numerators with much shorter latency and similar complexity. Additionally, by exploiting the regularity, a reconfigurable architecture is proposed to implement the inversions of matrices of different sizes efficiently with much smaller area and shorter latency compared to the best alternative possible design.

REFERENCES

[1] X. Tang and R. Koetter, "A novel method for combining algebraic decoding and iterative processing," *Proc. of IEEE Int. Symp. Info. Theory*, pp. 474-478, Seattle, WA, USA, 2006.

[2] Y. Wu, "Generalized integrated interleaved codes," *IEEE Trans. on Info. Theory*, vol. 63, no. 2, pp. 1102-1119, Feb. 2017.

[3] X. Zhang, "Generalized three-layer integrated interleaved codes," *IEEE Commu. Letters*, vol. 22, no. 3, pp. 442-445, Mar. 2018.

[4] X. Zhang and Z. Xie, "Relaxing the constraints on locally recoverable erasure codes by finite field element variation," *IEEE Commun. Letters*, vol. 23, no. 10, pp. 1680-1683, Oct. 2019.

[5] W. Li, J. Lin and Z. Wang, "A 124-Gb/s decoder for generalized integrated interleaved codes," *IEEE Trans. on Circuits and Syst.-I*, vol. 66, no. 8, pp. 3174-3187, Aug. 2019.

[6] X. Zhang and Z. Xie, "Efficient architectures for generalized integrated interleaved decoder," *IEEE Trans. on Circuits and Syst.-I*, vol. 66, no. 10, pp. 4018-4031, Oct. 2019.

[7] Z. Xie and X. Zhang, "Scaled nested key equation solver for generalized integrated interleaved decoder," *IEEE Trans. on Circuits and Syst.-II*, vol. 67, no. 11, pp. 2457-2461, Nov. 2020.

[8] Z. Xie and X. Zhang, "Fast nested key equation solvers for generalized integrated interleaved decoder," *IEEE Trans. on Circuits and Syst-I*, vol. 68, no. 1, pp. 483-495, Jan. 2021.

[9] J. S. Plank and K. Greenan, "Jerasure: A library in C facilitating erasure coding for storage applications_version 2.0 technical report UT-EECS-14-721," University of Tennessee, 2014.

[10] P. Trifonov, "Low-complexity implementation of RAID based on Reed-Solomon codes," *ACM Trans. Storage*, vol. 11, no. 1, Feb. 2015.

[11] S. Lin, T. Y. Al-Naffouri and Y. S. Han, "FFT algorithm for binary extension finite fields and its application to Reed–Solomon codes," *IEEE Trans. on Info. Theory*, vol. 62, no. 10, pp. 5343-5358, Oct. 2016.

[12] S. J. Lin, "An encoding algorithm of triply extended Reed-Solomon codes with asymptotically optimal complexities," *IEEE Trans. on Commun.*, vol. 66, no. 8, pp. 3235-3244, Aug. 2018.

[13] L. Yu *et. al.*, "Fast encoding algorithms for Reed-Solomon codes with between four and seven parity symbols," *IEEE Trans. on Comp.*, vol. 69, no. 5, pp. 699-705, May 2020.

[14] Y. J. Tang and X. Zhang, "Fast en/decoding of Reed-Solomon codes for failure recovery," *IEEE Trans. on Comp.*, vol. 71, no. 3, pp. 724-735, Mar. 2022.

[15] X. Zhang, S. Sprouse and I. Ilani, "A flexible and low-complexity local erasure recovery scheme," *IEEE Commun. Letters*, vol. 20, no. 11, pp. 2129-2132, Nov. 2016.

[16] K. K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementations*, Wiley, 1999.