A Parametrical Architecture for Reed-Solomon Decoders

Mariana-Eugenia Petre
University "Politehnica" of Bucharest
Dept. of Electronics and Telecommunications
Bucharest, ROMANIA
mariana@hera.gef.pub.ro

Guido Masera
Politecnico di Torino
Electronics Department
Torino, ITALY
masera@polito.it

Abstract

Reed-Solomon decoders are digital decoders that use RS detecting and correcting of errors codes. RS codes are widely diffused in the transmission and storage of digital information and they are often used in concatenated encoding schemes to obtain great correction capabilities and good robustness to burst errors. In this study, a parametrical approach was chosen for decoder implementation at gate-level, based on the Berlekamp algorithm. This means that the decoder structure depends on two parameters: the bit number used for the symbol representation (m), and the error correction capability (t). The obtained architecture is suitable for a large number of different application (including high definition digital TV) and can be quickly synthesised using Synopsys for any required values of m and t.

1 Introduction

A (n, k) RS code is a block code [2] whose code words are blocks of n symbols, including k symbols of information and $n-k$ parity-check symbols. Each symbol is m-bit-long. Hence, the calculations are performed in a Galois field of 2^m elements noted $GF(2^m)$, defined by a primitive monic polynomial $f(x)$ over $GF(2)$ of degree m. The length of the block code is $n = 2^m - 1$. The error correcting capability of the code is defined by $2t + \rho \leq n - k$, where t is the maximum number of erroneous symbols that can be corrected, and ρ is the number of erased symbols, which are errors with known locations.

Due their remarkable capability of combatting combinations of random as well as burst errors, RS codes have a lot of applications. The digital audio discs and compact discs use RS codes for the error correction and error concealment. Other applications include mobile data transmission systems and high-readability communications systems. RS codes were also used in NASA and ESA planetary exploration missions, for the deep space transmission.

Sections 2 and 3 describe the encoding and decoding algorithms for RS codes used for our implementation while section 4 presents the architecture obtained and section 5 presents the estimates in terms of the RS decoder complexity.

2 Reed-Solomon Encoding Algorithms

Let the sequence of k data symbols in $GF(2^m)$ be $d = [d_0, d_1, \ldots d_{k-1}]$. The data vector can be represented by a polynomial as follows: $d(x) = d_0 + d_1 x + \ldots + d_{k-1}x^{k-1}$. RS encoding consists in adding $n-k$ extra-symbols to the k information symbols, obtaining in this way a systematic code word. This is done by adding the remainder of $\frac{x^{n-k}d(x)}{g(x)} = p(x)$ to $d(x)$, where $g(x) = \sum_{i-0}^{n-k-1} \prod(x + \alpha^{i+1})$ is the generator polynomial of degree $n-k$, α is the root of the primitive polynomial $f(x)$ and $d(x)$ is the data polynomial. The resulting code polynomial is $c(x) = d(x) + p(x) \cdot x^{n-k}$. It is divisible by $g(x)$, and $g(\alpha^i) = 0$ for $i = 1, 2, \ldots, 2t$. There are also other methods for encoding the message [6]. The code word polynomials could be formed as $d(x) \cdot g(x)$, or an RS code can be generated in the so-called frequency domain. But these codes are not systematic because the k data symbols are not explicitly present in the code word. Hence, one extra step is needed in the decoding process to extract the information from the corrected code word.

3 Decoding Algorithms

The task of the decoder is to generate the syndrome equations, depending on the additive noise, and to solve them in order to find the locations and the values of errors and erasures. The following main steps are required.

Syndromes Generator The received code word is evaluated at the zero's of the generator polynomial. In this way we compute a set $\{S_{2t}\}$ of non-linear equations called syndromes: $S_j = v(\alpha^j) = \sum_{i=0}^{n-1} v_j \alpha^{ij}, j = 1, 2, \ldots, 2t$.

Calculation of Erasure Locator Polynomial The era-

0-8186-7502-0/96 $5.00

sures are detected outside the decoder. Therefore the positions of the erased symbols are obtained at the same time the data is being input. The resulting polynomial is $\Gamma(x) = \prod_{i-1}^{\rho}(1 - x\alpha^{j_i})$, where j_i, $i = 1, 2, \ldots, \rho$ are locations of ρ erasures.

Calculation of Error Locator Polynomial The coefficients are performed using Berlekamp-Massey algorithm, makes use of the following set of recursive equations:

$$\Delta_i = \sum_{j=0}^{i-1} \Lambda_j^{(i-1)} S_{i-j},$$

$$L_i = \delta(i - L_{i-1} - \rho) + (i - \delta)L_{i-1},$$

$$\begin{bmatrix} \Lambda^{(i)}(x) \\ B^{(i)}(x) \end{bmatrix} = \begin{bmatrix} 1 & -\Delta_i x \\ \Delta_i^{-1}\delta & (1-\delta)x \end{bmatrix} \begin{bmatrix} \Lambda^{(i-1)}(x) \\ B^{(i-1)}(x) \end{bmatrix}$$

for $i = 1, 2, \ldots, n - k$. The initial conditions are $\Lambda^{(\rho)}(x) = \Gamma(x), B^{(\rho)}(x) = \Gamma(x), L_0 = 0$, and $\delta = 1$ if both $\Delta_i \neq 0$ and $2L_{i-1} \leq i - 1 + \rho$, otherwise $\delta = 0$. Then $\Lambda^{(n-k)}(x)$ is the resulting polynomial. Similar equations give the error evaluator polynomial $\Omega(x)$.

Chien Search Algorithm This algorithm consists in finding the roots of the error locator polynomial and then the errors positions [2, 1].

Forney Algorithm, used to compute the values to add to correct the code word [2, 1].

4 The Parametrical Architecture of the Decoder

The global decoder architecture is shown in Fig. 1. It mainly consists on five processing blocks eachone requiring computation in $GF(2^m)$, which can be performed in either a bit-serial or bit-parallel fashion [1, 4]. In the following sections the gate-level implementation of main blocks is described.

4.1 Syndromes Generator

The syndromes computation block can be built with $2t = n - k$ identical cells. The received code word enters in bit-serial fashion and after a delay we obtain at the output all the syndromes in bit-parallel fashion. For implementing the T-cell, the triangular basis multiplication is used [6]; then the rows of the Hankel matrix are generated, the matrix-vector multiplication is performed and an inverse transformation is made to obtain the coordinates in the canonical basis. In Fig. 2 the T-cell implementation for $GF(2^4)$ is shown. At any clock time we obtain at the output of the flip-flop register $SREG3$ a row of the Hankel matrix. Next we

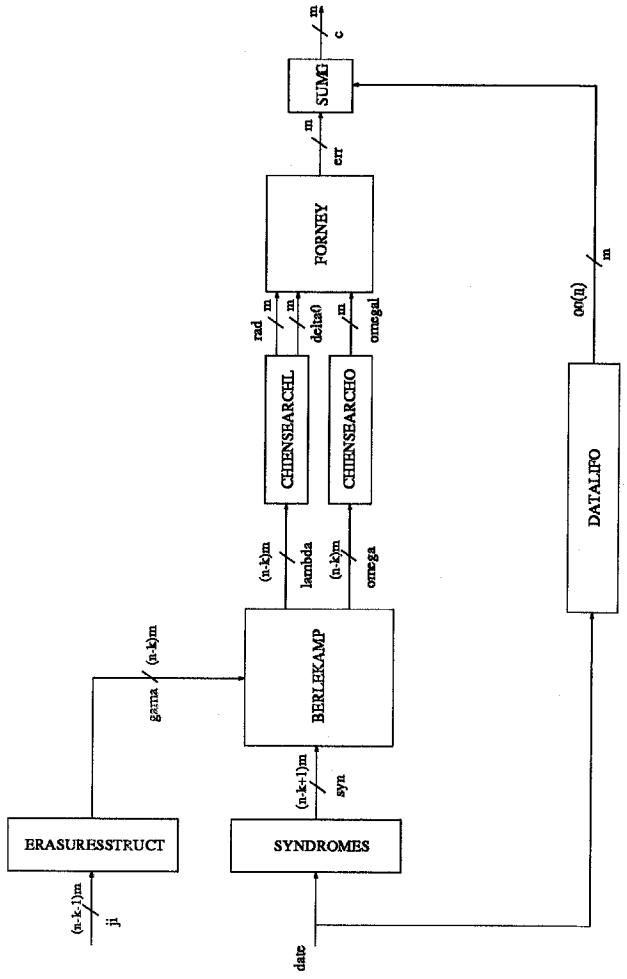
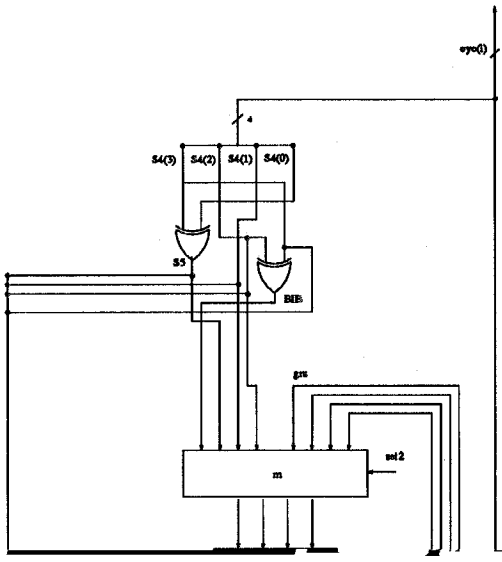


Figure 1. A general scheme

have to multiply the resulting Hankel matrix with α^i vector. This is done in the *Inner_product* block which has at each clock as inputs the four elements of a Hankel matrix row and the four elements of α^i vector. This block contains four AND gates, a XOR gate and a shift register. After four clocks the result is transferred to the SUM block.

4.2 *Erasurestruct* Block

This block computes the coefficients of the erasure-locator polynomial and has as inputs the positions of the ρ erasures. It includes three main subblocks: a ROM having as inputs the locations of the erasures and giving as outputs the inverses of the $\Gamma(x)$ roots; a block giving at any clock the inverse of a root; a circuit for computing the coefficients of the polynomial $\Gamma(x)$ after any m clocks.



$\Lambda(x)$, $\Omega(x)$ and $A(x)$ are loaded with the coefficients of the polynomial $\Gamma(x)$; simultaneously, register $S(x)$ is loaded with syndromes values. During one iteration, register Λ is shifted twice, first to compute *delta*, then to be updated.
The error locator as well as the error evaluator polynomials can also be obtained using the Euclidean algorithm, a method for finding the greatest common divisor of two polynomials. But, using shift registers, the realization of the Berlekamp algorithm is straightforward and it does not involve polynomial multiplication and division.

4.4 *Chiensearch* and *Forney2* Blocks

Using the Chien search method, $\Lambda(1), \Lambda(\alpha^{-1}), \Lambda(\alpha^{-2}), \ldots, \Lambda(\alpha^{-(2^m-2)})$ are computed

shows the chosen realization. The *err* value obtained is summed with the input data delayed by a LIFO. The

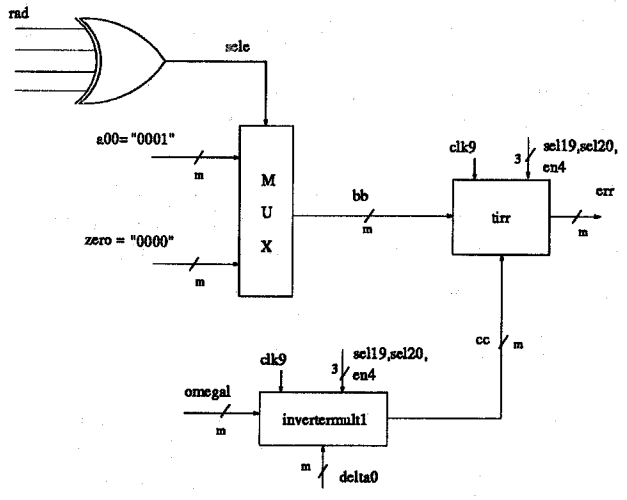


Figure 4. Realization of the Forney algorithm

summation is made by the *SUMG* block.

5 Complexity

The decoder complexity is now evaluated at the functional level. The gate count of basic cells used to write the block complexity are the following: 2 gates for $AND2$ and $OR2$ functions, 8 gates for $XOR8$ gates, 4 gates for $XOR2$ and $MUX2$, 9 gates for a D-flipflop. In addition, $m2^m$ bits are needed for the ROM storing the inverse values of the GF elements. From these figures, the following numbers of functional gates needed by each block have been obtained: $C_{syndr} = 78mt + 64t + 9m$, $C_{Chien} = 48mt + 72t - 8$, $C_{Berlek} = 2m2^m + 90t + 175m + 160$, $C_{Forney} = 66m + m2^m + 64$, $C_{Datalifo} = 30m2^m - 35m$, where $t = n - k$ is the correction capability of the decoder. Instead of the *Datalifo* block previously described, a RAM as a LIFO can be used: in this case the complexity of the block becomes $2m(2^m + 4t)$. The resulting RS decoder complexity is: $C_{dec} = 182mt + 292t + 215m + 5m2^m + 208$. The formula calculated for the decoder complexity should be multiplied by 1.3 to take into account the extra hardware needed by buffers, control and test. The graph on Fig. 5 gives the gate complexity of the decoder with parameter m varying from 4 to 8 and parameter t varying from 1 to 10.

6 Conclusions

The architecture of a parametrical VLSI architecture for the decoding of generic RS codes has been

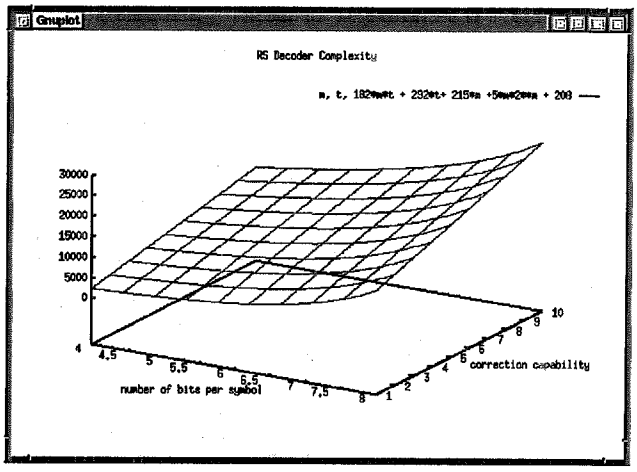


Figure 5. The Reed-Solomon Decoder Complexity

presented. The design and testing of the described decoder has been performed in the Synopsys environment for several codes. Future work includes the study of different approaches for the inverse elements computation.
The decoder behaviour was simulated using the VHDL language and the Synopsys Simulator under the UNIX environment. Our study also estimates the hardware complexity of the RS decoder.

References

[1] E. R. Berlekamp. Bit-serial Reed-Solomon encoder. *IEEE Transactions on Information Theory*, 28(6):869–874, November 1982.

[2] R. E. Blahut. *Theory and Practice of Error Control Codes*. Addison Wesley, Reading, Mass., 1984.

[3] C.C.Wang, T. K. Truong, H. M. Shao, L. J. Deutsch, J. Omura, and I. S. Reed. VLSI architecture for computing multiplications and inverses in GF(2^m). *IEEE Transactions on Computers*, 34(8):709–717, August 1985.

[4] M.A.Hasan and V.K.Bhargava. Division and bit-serial multiplication over $GF(q^m)$. *IEE Proceedings-E, Computers and Digital Techniques*, 139(3):230–236, May 1992.

[5] H. M.Shao, T. K. Truong, L. J. Deutsch, J. H. Yuen, and I. S. Reed. A VLSI design of a pipeline Reed-Solomon decoder. *IEEE Transactions on Computers*, 34(1):393–403, January 1985.

[6] S. B. Wicker and V. K. Bhargava. *Reed-Solomon codes and their applications*. IEEE PRESS, New York, 1994.