Indian Journal of Science and Technology Vol.2 No 4 (Mar. 2009) ISSN: 0974- 6846

FPGA based design of Reed Solomon codes

Shivani Pasricha and Sanjay Sharma[1]
ECE, Amity School of Engg.& Technol.Bijwasan, New Delhi-110 061; [1]Thapar University, Patiala, India.
Shivani.pasricha@gmail.com; sanjay_eced@rediffmail.com[1]

Abstract: This paper presents a novel architecture design for forward error correction technique based on RS coding scheme for wireless applications. The design was created using System Generator for DSP tool from Xilinx and was simulated on Matlab/Simulink environment. The hardware description language source code for different blocks was generated and the design was subjected to severe functional and timing constraints using Xilinx Foundation series and ModelSim tools. Synplify Pro tool was finally used to synthesize the complete design. The overall architecture for RS Coder-Decoder was implemented on Xilinx Virtex-II XC2v250 device and consumed slices 1429 for the CODEC at a clock frequency of 90 MHz. The architecture design power consumption analysis was done using the Xilinx Xpower tool and it came out to be about 783 mW.

Keywords: FPGA, Reed-Solomon coding.

Introduction

Reed-Solomon codes are block-based error correcting codes with a wide range of applications in digital communications and storage (Martin 1998]. Encoding data using an RS code is relatively straightforward, but decoding is time-consuming. Despite major efficiency improvements made by Berlekamp and others during the 1960's (Berlekamp 1968) Reed-Solomon codes are used to correct errors in many systems including Wireless or mobile communications (including cellular telephones, microwave links, etc), Satellite communications, High-speed modems, Storage devices (including tape, Compact Disk, DVD, barcodes, etc). A typical system is shown here in Fig.1

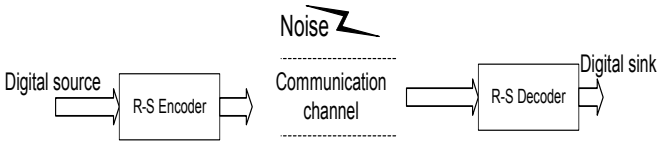

Fig.1. General block diagram of communication system

The Reed-Solomon encoder takes a block of digital data and adds extra "redundant" bits. Errors occur during transmission or storage for a number of reasons (for example noise or interference, scratches on a CD, etc). The Reed-Solomon decoder processes each block and attempts to correct errors and recover the original data. The number and type of errors that can be corrected depends on the characteristics of the Reed-Solomon code. An RS code differs from a Hamming code in that it encodes groups of bits instead of one bit at a time. We will call these groups “digits” (or “symbols” or “coefficients”). A digit is error free if and only if all of its bits are error-free. For instance, if a digit is an 8-bit character, and three bits of the same single character are in error, we will count that as one corrupted digit. There are two corrupted digits (but more than two corrupted bits) in the following example.

Original: 10110001 11011111 01001011 01011100
Received: 10110101 11011111 01110001 01011100
Corrupted: yes no yes no

If we want to send a k digit plaintext message, RS will send n = k + 2s digits, and guarantee that the correct message can be reconstructed at the other end if there are fewer than s corrupted digits. An example of commonly used parameters: k = 223, s = 16, n = k+2s = 255, giving the ability to correct 16 corrupted digits out of every 255-digit transmission. In general, the number of bits in a digit and the parameters n and s are tuned to optimize for your application.

Constructing the field

We introduce here the basic algebraic definitions which will be used in this section.

Groups

A group is a tuple (G; *; ε), where G is a set of elements, * is a binary operator on G, and e ε G is a designated identity element G must have the following properties:

1. * is closed. For all a; b ε G, a * b ε G.
2. * is associative. For all a; b; c ε G,(a * b)* c = a * (b * c).
3. a ε G, a * ε = ε * a = a.
4. For each a ε G there is an element a^{-1} ε G such that a^{-1} * a a^{-1} = a^{-1} * a = ε

Fields

A field is a tuple (F, +, *, 0, 1), where F is the set of elements, + is the addition operator, * is the multiplication operator, 0 ε F is the additive identity, and 1 ε F is the multiplicative identity. F must have the following properties:

1. (F; +; 0) forms a group.
2. * is associative and distributes over +.
3. (F \ {0}; *; 1) forms a group.

Note: ``the field F '' means the field whose set is F, with the standard operators for that set. Fields that we commonly work with include the real, complex, and rational numbers. The set of integers is not a field. It satisfies the first two properties, but fails the third property because not all integers have multiplicative inverses. A set like the integers which satisfies the first two field properties is called a “ring”.

Galois fields

We now turn to the question of constructing the field F from which the coefficients of m(x) are drawn. A basic result from number theory is that if p is prime, then the set of integers modulo p (denoted Z_p) is a field. So if there are a prime number p of possible digits, we can use Z_p as our field (Azaleah 2003). However, that is true if and only if p is prime. Unfortunately, in computer applications we are

Indian Journal of Science and Technology Vol.2 No 4 (Mar. 2009) ISSN: 0974- 6846

likely to want digits that we can encode with r > 1 bits; for instance, 8-bit characters. This means we have a non-prime number 2^r of possible digits. Fortunately, it can be proven that for any prime p and any natural number r there exists a finite field with p_r numbers. (In fact the reverse is also true---every finite field has p_r numbers, where p is prime). There is a way to generate such a field. It is called a Galois field, and it can be shown that any finite field of size p_r is isomorphic to a Galois field. Galois fields are constructed with the help of Z_p [x], the set of polynomials with coefficients in Z_p .We're used to dealing with polynomials with real coefficients (polynomials in R[x]), so the arithmetic of polynomials in Z_p[x] may seem counterintuitive. Take F = Z_2, for instance.

A refresher on how arithmetic modulo 2 works:

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 0

Now for example, let a; b ε Z_2 [x],
$a(x) = x^2 + x$, $b(x) = x$.
Then we can do the addition
$a(x) = x^2 + 1x$
$b(x) = 0x^2 + 1x$
$a(x) + b(x) = 1x^2 + 0x = x^2$

which is of course a different result than we would have gotten if a(x); b(x) ε R[x]; in that case we would have had $(x^2 + x) + (x) = x^2 + 2x$. Now to define modulo arithmetic for polynomials: a(x) modulo g(x) is another polynomial r(x) with degree strictly less than g(x), and satisfying a(x) = g(x) q(x)+r(x) for some polynomial q(x). Exercise: Prove that the polynomial r(x) with those properties exists, is unique, and is the same as the remainder of polynomial long division when dividing a(x) by g(x). Consider the set Z_p [x] of polynomials over the field Z_p and an irreducible polynomial g(x) of degree r. An irreducible polynomial is one that cannot be factored into a product of lower-degree polynomials over Z_p. Now let F be the set of all polynomials in Z_p[x] with degree at most r - 1, with the operations of polynomial additional and polynomial multiplication modulo g(x). Obviously there are p_r such polynomials (each of r coefficients can take one of possible values). We will denote the field GF (p_r) (recall that r is the degree of polynomial g(x)). Thus we now have a method of constructing finite fields with p_r elements. Note that we don't need g(x) to enumerate the elements of GF (p_r). The elements are all the polynomials with degree at most r - 1. However, we need g(x) to define operations in this field (i.e., the multiplication of the fields' elements will be taken modulo g(x)) 1. Unfortunately, there is no simple method of obtaining an irreducible polynomial (there are complicated probabilistic algorithms that can do that,but we won't describe them here). However, irreducible polynomials for most common finite fields have been found and published (Christian 1989). Also, it can be proven that an irreducible polynomial of degree r over Zp exists for every positive integer r. Let us consider an example of the Galois field GF(ε 3). As stated above we can construct such a field by enumerating the possible polynomial residues modulo some irreducible polynomial of degree 3 over Z ε . It can be shown that 1 + x + x 3 is irreducible over Z ε . Therefore the field we seek is {0,1, x, 1+ x, x^2 ; 1 + x^2; x+ x^2 ; 1 +x+ x^2 g. That is a simple example. The most commonly used Galois field is GF(2^8), since we are usually interested in bytes as information units. Finite fields (recall that any finite field of size p_r is isomorphic to a Galois field) have some nice properties (Arshad et. Al 2004). One of them is that the multiplicative group of a finite field F is cyclic. It means that a group contains an element α such that every a ε G equals α^i for some integer i. α is called a generator(or a primitive element) of the group. The finite order of α is the smallest integer k > 0 such that $\alpha^k = 1$ (where 1 is the identity of the group). It is easy to show that the finite order of generator of GF(p^r) is p^r-1.

Reed-Solomon Encoder

Reed Solomon encoding is a block-encoding scheme. The system implemented in this study was a (219,201) system, in which (*n, k*) denotes an output codeword length of *n* and an input word of length *k*, as shown in *Fig. 2.1.* It has a symbol size**,** *s*, and equal to 8. The decoder has a correcting capability of *t* symbol errors in the code word, with $n - k = 2t$, in this case, $t=9$.

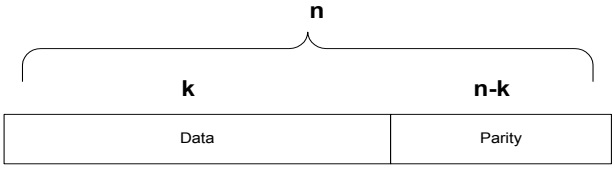

Fig.2.1. Typical RS codeword

The encoder forms a code word $x^{n-k}m(x) + r(x)$ by means of the following equation:

$$\frac{x^{n-k}m(x)}{g(x)} = q(x) + \frac{r(x)}{g(x)} \qquad (1)$$

(From here on, the term "polynomial" shall pertain to polynomials with GF element coefficients.) The divisor, *g(x)* is known as the generator polynomial. It is a polynomial of degree *(n-k)* and which is a factor of (x^n+1). To maximize the minimum distance between codes, the roots of this polynomial should all be consecutive. This is a direct consequence of the BCH bound, which states that the minimum distance is always larger than the number of consecutive factors of *g(x)*. The system Used adapted a generator polynomial with roots from α^1 to α^{32}.*Eq. 1* also implies that the generator polynomial *g(x)* is a factor of all possible code words. In digital hardware, the encoder is an LFSR with internal feedback connections corresponding to *g(x) ,* as seen in *Figure 2*.2. The operations involved are GF addition and multiplication, as previously defined in this text.

Indian Journal of Science and Technology Vol.2 No 4 (Mar. 2009) ISSN: 0974- 6846

The computation of the remainder is implemented on digital hardware using a linear feedback shift register configuration as shown in Fig. 2.2. Note that this setup resembles the iterative method of polynomial division. The final contents of the shift registers will contain the remainder.

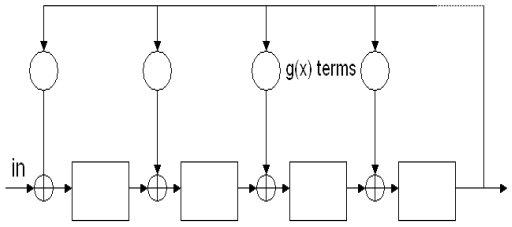

Fig. 2.2. Encoder LFSR

Since the input and output data for the encoder are serial streams, appropriate serial-parallel converters were implemented for both the encoder and decoder. A systematic encoding algorithm for an (n, k) cyclic code is (Parhi 1998)

Step 1. Multiply the message polynomial $m(x)$ by x_{n-k}.

Step 2. Divide the result of Step 1 by the generator polynomial g(x). Let d(x) is the remainder.

Step 3. Set $c(x) = x_{n-k}m(x) - d(x)$.

Summarizing above, the following encoding procedure for RS codes is established.

The RS encoding procedure

To construct a t-error correcting q-ary RS code of length n ($n = q - 1$):

1. Find a primitive nth root of unity α in GF(q)
2. Select $2t$ consecutive powers of α, starting with α_b (If b = 1, the code is narrow-sense RS). Obtain the generator polynomial $g(x)$ as in (1.1)
3. Follow the 3 steps shown above for systematic encoding of a message given by $m(x)$ RS codes are spectrally efficient in the sense that they introduce less amount of redundancy, i.e., $r = n - k$, than usual BCH codes. The redundancy introduced by RS is determined by the degree of $g(x)$, i.e., $r = 2t$. BCH codes, however, usually have cyclotomic cosets with cardinality larger than one. To provide the same t-error correcting capability, $2t$ consecutive powers of α and all their conjugates are selected and result in a generator polynomial of degree at least $2t$. A lower rate BCH code might have a higher minimum distance and thus provide higher error-correcting capability (Sharma et. al 2003).

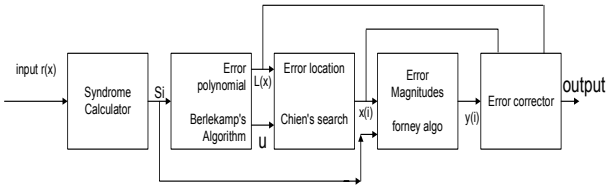

Fig.3.1. Decoder architecture

Unfortunately it is beyond the predictability of 'design distance' and we are unable to take advantage of it. RS codes are classified as maximum-distance separable (MDS) codes, as an (n, k) RS code always has minimum distance exactly equal to ($n - k + 1$).

Reed Solomon decoder

r(x) codeword	Received
Si	Syndromes
L(x) polynomial	Error locator
Xi locations	Error
Yi magnitudes	Error
c(x) code word	Recovered
v errors	Number of

The received codeword r(x) is the original (transmitted) codeword c(x) plus errors: r(x) = c(x) + e(x)

A Reed-Solomon decoder attempts to identify the position and magnitude of up to t errors (or 2t erasures) and to correct the errors or erasures [Sklar 2001; Amina 2003]. The purpose of the decoder is to process the received code word to compute an estimate of the original message symbols. There are three main blocks to the decoder: the Syndrome Generator, Euclid's Algorithm, and the Chien/Forney block. The output of the Chien/Forney block is an estimate of the error vector. This error vector is then added to the received codeword to form the final codeword estimate. A top-level block diagram of the decoder is shown in Figure 3.1. Note that the error value vector Y comes out of the Chien/Forney block in reverse order, and it must pass through a last-in, first-out (LIFO) block before it is added to the received codeword R(x).

Syndrome generator

The first step in the decoder is to compute the syndrome. The syndrome consists of $n - k$ symbols and the values are computed from the received code word. The syndrome depends only on the error vector, and is independent of the transmitted code word. That is, each error vector has a unique syndrome vector. However, many different received code words will have the same syndrome if their error pattern is the same. The purpose of computing the syndrome first is that it narrows the search for the actual error vector. Originally, a total of 2^n possible error vectors would have to be searched. However, by finding the syndrome first, this search is narrowed down to looking at just 2^{n-k} possibilities. The syndrome can be computed mathematically by dividing the received code word by the generator polynomial using GF algebra. The remainder of this division is called the syndrome polynomial $s(x)$. The actual syndrome vector $S(x)$ is computed by evaluating $s(x)$ at a through α^{n-k}. However, this method is not efficient from a hardware

Indian Journal of Science and Technology Vol.2 No 4 (Mar. 2009) ISSN: 0974- 6846

standpoint. The alternative method typically used in hardware is to directly evaluate the received code word $R(x)$ at a through α^{n-k}. The Syndrome Generator module computes the syndrome S by evaluating the received code word $R(x)$ at through α^{n-k}. That is, $R(\alpha)$ through $R(\alpha^{n-k})$. In the RS code $n - k = 2t$, and thus there are $2t$ syndrome values to compute: [$S1$ $S2$ $S3$...S ($2t$)]. These values are computed in parallel as shown in Figure 4. The first syndrome generator evaluates the received code word at to form $S1$; the next generator evaluates the

S. No	Name of Communication Block	Slices	Percentage utilization
1.	RS Encoder	112	7.3
2.	Convolutional Interleaver	210	13.7
3.	Convolutional Encoder	76	5.0
4.	Puncturing	28	1.8
5.	RS Decoder	676	44.0
6.	Convolutional De-Interleaver	210	13.7
7.	Viterbi Decoder:	86	5.6
8.	De-puncturing	31	2.1
	Total	1429	93.2

Table 1. FPGA Resource Utilization

received code word at a^2 to form $S2$, and so on. The Syndrome Generator module will also contain hardware that checks for an all-zero syndrome. If all of the syndrome values are zero, then there are no errors and the Euclid's algorithm block and the Chien/Forney block are disabled. The received code word then becomes the codeword estimate.

Euclid's algorithm

Euclid's algorithm [Christian 1989; Ahmed 2001] processes the syndrome S(x) to generate the error locator polynomial Λ(x) and the error magnitude polynomial Ω(x)[6]. That is, it solves the following equation that is referred to as the Key Equation:

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

The algorithm used in RS decoding is based on Euclid's

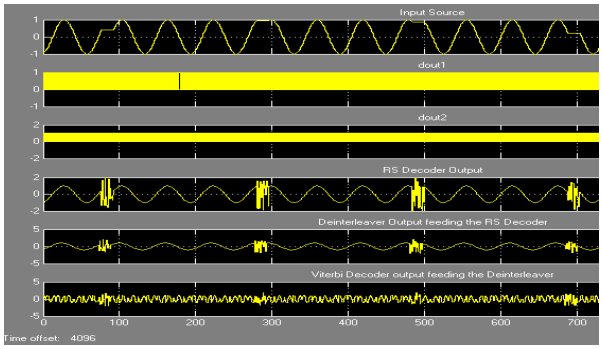

Fig.4.1.Simulation results

algorithm for finding the greatest common devisor (GCD) of two polynomials.

Berlekamp's algorithm: Finds the error locator polynomialL(x), and the number of errors. This involves solving s simultaneous equations with s unknowns.

Chien search: Given L(x) and v, finds the roots xi of L(x).

Forney's algorithm: Given the syndromes and the roots of L(x), finds the symbol error values yi. Again, this involves solving s simultaneous equations with s unknowns.

Error corrector: Combines all of the pieces calculated above and reconstructs the original me

Simulation Results

All the wavwforms are simulated on the "Xilinx foundation series", ". Synplify Pro", "Model Sim"[20]. The waveforms explanation shown above in the simulation results are as follows

Input source: The input signal given to the encoder can be any input signal like sin wave, random wave or any signal from workspace. In our application we have applied the sin wave which will be applied to the encoder after some delay.

din1, din2: After encoding the input data will be interleaved, convolutionally encoded and punctured. After puncturing input data is divided into two streams din1, din2 which represents the encoded MSB and encoded LSB respectively. Encoded data after modulation (through AWGN Channel)is applied to the Viterbi decoder.

Viterbi output: At the receiving end Encoded signal is applied to de-puncture block so as to insert arbitrary symbol into your input data at the location specified by the de-puncture code and creates a new value. After that data is applied to Viterbi decoder. The output is shown above

De-interleaved output: The output from the Viterbi decoder is then applied to the de-Interleaver so as to remove the erasures

R-S decoder output: after de-Interleaving output is then applied to the R-S decoder so as to get the final output. As shown in the simulation results in figure 4.1 the output of the R-S decoder is same as that we have applied at the input.

Conclusions

In this paper, a novel architecture design for Forward Error Correction technique based on RS coding scheme for wireless applications is proposed. The RS code (126,112) has been selected, and the system has been simulated on Matlab/Simulink environment. The design was created using System Generator for DSP tool from Xilinx and was simulated on Matlab/Simulink environment. The hardware description language source code for different blocks was generated and the design was subjected to severe functional and timing constraints using Xilinx Foundation series and ModelSim tools. Synplify Pro tool was finally used to synthesize the complete design. The overall architecture for RS Coder-Decoder was implemented on Xilinx Virtex-II XC2v250 device and consumed slices 1429 out of 1536 for the CODEC at a clock frequency of 90 MHz. The architecture

design power consumption analysis was done using the Xilinx Xpower tool and it came out to be about 783 mW.

We have demonstrated a simplified and efficient implementation of a Reed Solomon COEDC with lower power consumption. The efficient implementation comes from tool "SYSTEM GENERATOR FOR DSP" and compact hardware management. The RS CODEC takes any inputs & generates Coding & Decoding signals. By manipulating input signal we can see the output for different signals. Also, by taking advantage of internal RS coder & decoder function, we can implement different code rate used for different applications

References

1. Martyn Riley and Iain Richardson "An introduction to Reed-Solomon codes: principles, architecture and implementation " IEEE Conference Proceedings, pp-122-126, New York , 1998
2. E. R. Berlekamp 'Algebric coding theory" McHill New york 1968
3. Barnaad & Saklar "Dgital communication" 2001
4. Azaleah Amina P. Chio, Jonathan A. Sahagun, and Delfin Jay M. Sabido IX "VLSI implementation of a (255, 223) Reed-Solomon error-correction codec ",SPIE Conference Proceeedings, pp. 256-261, Toronto, 2003
5. Christian SchulerGMD FOKUS Kaiserin-"Code generation tools for hardware implementation of FECcircuits" Berlin, Germany IEEE journal published, vol 22, pp. 1233-1241, August , 1989
6. Arshad Ahmed, Naresh R. Shanbhag, and Ralf Koetter "Systolic interpolation architectures for soft-decodingreed-solomon codes" IEEE journal vol 32, no4, pp. 541-547, 2004.
7. Tong Zhang and Keshab K. Parhi Reed "On the High-Speed VLSI Implementation of Errors-and-Erasures Correcting Reed-Solomon Decoders" IEEE journal of Communications, vol 40, pp 666-670, August 2005.
8. Tong Zhang and Keshab K. Parhi, IEEE Conference Proceedings, pp. 554-555, 1998, USA
9. Sanjay Sharma, Sanjay Attri, R, C, Chauhan, "Low-Power VLSI Synthesis of DSP Systems", Published in Elsevier Science B. V. INTEGRATION, the VLSI Journal, Issue 1-2, vol. 36, pp. 41-54, September 2003.
10. E. R Berlekamp , "Algebric coding theory" Mcgraw hill , NY 1968
11. E. R Berlekamp, "key papers in the development of coding theory" IEEE proceedigns, pp. 116-119, New York, 1994
12. R. E . Blahut theory and practice of error control codes . Addison-Wesley, MA , 1983