MAKARA, SAINS, VOL. 10, NO. 2, NOVEMBER 2006: 58-62

GENERIC REED SOLOMON ENCODER

Petrus Mursanto

Fakultas Ilmu Komputer, Universitas Indonesia, Depok 16424, Indonesia

Email: santo@cs.ui.ac.id

Abstract

Reed Solomon (RS) codes is a mechanism to detect and correct burst of errors in data transmission and storage systems. It provides a solid introduction to foundation mathematical concept of Galois Field algebra and its application. With the development of digital hardware technology, the RS concepts were brought into reality, i.e. the implementation of RS codec chips. This paper presents the development steps of a generic RS encoder using VHDL. The encoder is able to handle generic width of data, variable length of information, number of error as well as variable form of primitive polynomial and generator polynomial used in the system. The design has been implemented for FPGA chip Xilinx XC3S200-5FT256 and has a better performance than commercially available equivalent encoder.

Keywords: reed solomon, error correction codes, galois field, VHDL

1. Introduction

Reed Solomon (RS) codes are block-based error-correcting mechanism in a wide range of applications in digital transmission and storage. The RS encoder takes a block of data and adds extra redundant bits. A block of RS codes described in Figure 1 consists of *K* information symbols added by *2t* parity symbols to make an *N* symbol codeword. In this scheme, the RS decoder can correct up to *t* symbols that contain errors in the codeword, and specified as *RS(N,K)*. Given a symbol size *m*-bit, the maximum codeword length is $N=2^m-1$. These variables are referred frequently throughout the discussion in this paper.

While most published RS codec implementation have specific size and features [1-3], this paper presents the development of a generic Reed-Solomon encoder, whose characteristics can be altered by modifying the parameters. They are: length of information symbol (*K*), length of parity symbol (*2t*), size of symbol (*m*), primitive polynomial *p(x)* and generator polynomial *P(x)*. The motivation of designing a generic encoder is to increase modularity and reusability, whereas design

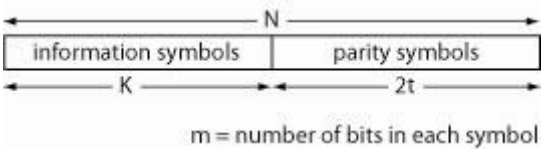

Figure 1. *RS(N, K)*

reuse is one of best practices in increasing productivity and efficiency.

It is assumed that the readers are familiar with fundamental aspects of finite fields discussed in many textbooks, some of them are [4,5].

2. RS Encoding Process

This section gives an overview of RS encoding process with a short data as an example, i.e. RS (7,3). It means that the RS codes have seven codeword symbols, three of which are the original information symbols and the rest are parity symbols. Let each is 3-bit symbol. The information that will be transmitted is divided into three 3-bit blocks, each of which is added by four 3-bit symbols to make 7 codewords symbols. The field elements for $m = 3$ using primitive polynomial $x^3 + x + 1$ are presented in Table 1.

Table 1. Elements of GF(2^3)

Elemen	Polynomial	α_2	α_1	α_0
0	0	0	0	0
α^0	1	0	0	1
α^1	α	0	1	0
α^2	α^2	1	0	0
α^3	$\alpha+1$	0	1	1
α^4	$\alpha(\alpha+1)= \alpha^2+\alpha$	1	1	0
α^5	$\alpha^3+\alpha^2=\alpha^2+\alpha+1$	1	1	1
α^6	α^2+1	1	0	1

Let a block of information is 100011110. By dividing it into three 3-bit symbols (100 011 110), we have the information symbols: $\alpha^2\ \alpha^3\ \alpha^4$.

A Reed Solomon codeword is constructed using a special polynomial called *generator polynomial*. All valid codewords are divisible by the generator poly-nomial. The general form of a generator polynomial is:

$$g(x) = (x + \alpha^i)(x + \alpha^{i+1})...(x + \alpha^{i+2t-2})(x + \alpha^{i+2t-1}) \quad (1)$$

and the codeword is constructed using:

$$c(x) = g(x).i(x) \quad (2)$$

where $g(x)$ is the generator polynomial, $i(x)$ is the information block, $c(x)$ is a valid codeword.

In this example, the maximum error can be corrected is $t = 2$. Let we use $i = 1$ for equation (1), hence the generator polynomial is

$$\begin{aligned} g(x) &= (x + \alpha)(x + \alpha^2)(x + \alpha^3)(x + \alpha^4) \\ &= (x^2 + \alpha^2 x + \alpha x + \alpha^3)(x^2 + \alpha^4 x + \alpha^3 x + \alpha^7) \\ &= (x^2 + (\alpha + \alpha^2)x + \alpha^3)(x^2 + (\alpha^4 + \alpha^3)x + \alpha^7) \\ &= (x^2 + \alpha^4 x + \alpha^3)(x^2 + \alpha^6 x + 1) \\ &= x^4 + \alpha^3 x^3 + x^2 + \alpha x + \alpha^3 \\ &= (1\ \alpha^3\ 1\ \alpha\ \alpha^3) \end{aligned} \quad (3)$$

The generator polynomial $g(x)$ is used to divide the information symbols multiplied by $x^{(N-K)}$. The remainder must be added to the information symbols to make it divisible by $g(x)$. Hence, the remainder of the division is the parity symbols P_i.

$$p(x) = i(x).x^{N-K}/g(x) \quad (4)$$

Therefore, the dividend symbols plus parity symbols are the codewords to be transmitted through a communi-cation channel.

$$c(x) = i(x).x^{N-K} + p(x) \quad (5)$$

Manual calculation can be done by applying GF algebra in a long division of the symbols.

$$\begin{array}{l} \qquad\qquad\qquad\quad \alpha^2\ \alpha^2\ \alpha^6 \\ \alpha^0\ \alpha^3\ \alpha^0\ \alpha^1\ \alpha^3 \quad \alpha^2\ \alpha^3\ \alpha^4\ 0\ \ 0\ \ 0\ \ 0 \\ \qquad\qquad\quad\ \alpha^2\ \alpha^5\ \alpha^2\ \alpha^3\ \alpha^5 \\ \qquad\qquad\qquad\ \alpha^2\ \alpha^1\ \alpha^3\ \alpha^5\ 0 \\ \qquad\qquad\qquad\ \alpha^2\ \alpha^5\ \alpha^2\ \alpha^3\ \alpha^5 \\ \qquad\qquad\qquad\qquad\ \alpha^6\ \alpha^5\ \alpha^2\ \alpha^5\ 0 \\ \qquad\qquad\qquad\qquad\ \alpha^6\ \alpha^2\ \alpha^6\ \alpha^0\ \alpha^2 \\ \qquad\qquad\qquad\qquad\qquad\ \alpha^3\ \alpha^0\ \alpha^4\ \alpha^2 \end{array}$$

The remainder symbols $p(x) = \alpha^3\ \alpha^0\ \alpha^4\ \alpha^2$ are added to the information symbols that have been shifted 4 'spaces' to the left.

$$c(x) = i(x).x^{N-K} + p(x) = \alpha^2\ \alpha^3\ \alpha^4\ \alpha^3\ \alpha^0\ \alpha^4\ \alpha^2 \quad (6)$$

Now we have the whole codeword to be transmitted, i.e. the binary representation of all symbols: 100 011 110 011 001 110 100.

Thus, a Reed Solomon encoding process consists of the following steps:

1. Multiply the information symbols with $X^{(N-K)}$. This can be done by shifting the information symbols to the left to allow space for *2t* parity symbols.
2. Divide the result of step 1 with the generator polynomial using GF algebra
3. Add the result of step 2 (remainder of division) to the result of step 1

3. Implementation Methods

Adapted from [4], the schematic design of RS Encoder is shown in Figure 2. *g(x)* is the generator polynomial used to generate parity symbols *P(x)*. The number of registers used is equal to *N-K*. Parity symbols are generated by serial entry of the information symbols into *i(X)*. Right after *N-K* pulses of clock, *P* holds the generated parity symbols. In our sample codes, the sequence of parity symbols constructed in *P(x)* is described in Table 2.

The benefit of a generic or parameterized design is the ease of system's characteristics modification by changing a set of parameters. Specifically in hardware description language (HDL), a single parameterized design can be compiled for any size of data. Hardware space and time consumptions can be quickly evaluated

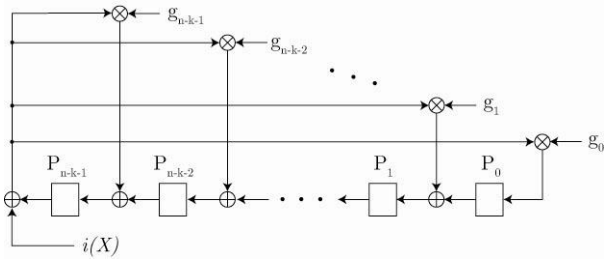

Figure 2. Schematic of Reed Solomon Encoder

Table 2. Parity generation for sample RS(7,3)

Input	Feedback	P_3	P_2	P_1	P_0
		0	*0*	*0*	*0*
α^2	α^2	α^5	α^2	α^3	α^5
α^3	α^2	α^3	α^5	α^2	α^5
α^4	α^6	α^3	α^0	α^4	α^2

by modifying the values of parameters and recompilation. The following sections describe major steps in developing a parameterized RS Encoder using Very High Speed Integrated Circuit HDL (VHDL). VHDL is a hardware description language used to describe the behavior and structure of digital systems. It allows a digital system to be designed and debugged at a higher level before converted to the gate and flip-flop levels. More on VHDL can be learnt from [6] and the RS VHDL design will be described in section 3.3.

Apparently from Figure 2, GF multipliers and adders are required for the system. Adders can easily be implemented by *m*-bit XOR gates. However, we need to implement a multiplier module, whose process depends on the primitive polynomial used for generating field elements.

To hold variable and temporary values, we use Shift Register with Parallel Load (SRwPL) as described in [7] for all registers used in the circuit. The SRwPL has generic width input and output, 2-bit operation mode (00=hold, 01=load, 10=shift left, 11=shift right), and 2-bit input (1-bit Left Input and 1-bit Right Input).

3.1. Generic Bit-Serial Multiplication

In building a parameterized encoder, we need to develop a generic serial multiplicator. A parallel multiplicator with fixed combinatorial logics cannot be used here, because its characteristic is tied to a certain size of data and polynomial type. The original design of standard-shift-register (SSR) was introduced by Peterson [8] and republished by Scott et.al. [9]. Multiplication holds the following equation:

$$C(x) = A(x)B(x) \bmod P(x) \quad (7)$$

for $A(x) = a_0 + a_1x + ... + a_{m-1}x^{m-1}$ and $B(x) = b_0 + b_1x + ... + b_{m-1}x^{m-1}$ and the product module $P(x)$ is $C(x) = c_0 + c_1x + ... + c_{m-1}x^{m-1}$. $P(x) = p_0 + p_1x + ... + p_{m-1}x^{m-1} + x^m$ is the polynomial used to generate field elements of $GF(2^m)$.

According to Mastrovito [10], $C(x)$ can be calculated as follows:

$$C(x) = [b_0A(x) + b_1xA(x) + ... + b_{m-1}x^{m-1}A(x)] \bmod P(x) \quad (8)$$

where each term $b_ix^iA(x)$ is calculated recursively:

$$b_ix^iA(x) \bmod P(x) = (...((b_ixA(x) \bmod P(x))x \bmod P(x)) ...)x \bmod P(x) \quad (9)$$

with x appears i times in the equation. Based on equation (8) and (9), we obtain the sequential multiplier design in Figure 3.

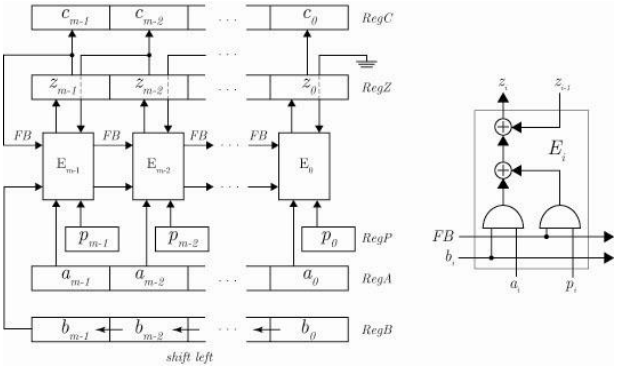

Figure 3. Serial Polynomial-based Multiplier

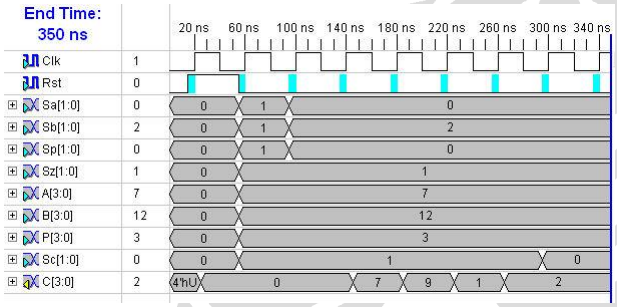

Figure 4. Simulation result of serial multiplication

The internal process of multiplication can be described as follows:

For example we want to compute $\alpha^{10} \times \alpha^{6}$ in $GF(2^4)$ with primitive polynomial $P(x)=x^4+x+1$. Referring to Figure 3, initially all registers are reset at the first clock. With $P(x) = x^4 + x + 1$, P is set to 0011 or 3_{10}. Input $A(x)=\alpha^{10}=0111=7_{10}$ and $B(x)=\alpha^{6} = 1100 = 12_{10}$.

During the cycle, the polynomial $b_{m-1}A(x)$ is entered into Z register. Shifting Z register is equivalent to multiplication by x, and since this could result in a term of degree greater than $m-1$, the polynomial $P(x)$ provides for a reduction of such a term by the equation $x^m = p_{m-1}x^{m-1} + p_{m-2}x^{m-2} + \ldots + p_0$. The result is $b_{m-1}A(x)$ mod $P(x)$. $b_{m-2}A(x)$ is added to this polynomial to produce the result $[b_{m-1}xA(x) + b_{m-2}A(x)]$ mod $P(x)$. Upon the application of next clock cycles, the sequence of operations repeats itself to produce the polynomial $\{[b_{m-1}xA(x) + b_{m-2}A(x)]x + b_{m-3}A(x)\}$ mod $P(x)$ in the Z register. The product $C(x)$ is obtained after n clock cycles, where n is the degree of polynomial $P(x)$.

As we can see from the simulation in Figure 4, multiplication module has four main registers, i.e. two registers for the operands (A and B), one register for primitive polynomial (P), and one register for the result (C). Each register has 2-bit selector mode Sa, Sb, Sp and Sc. As described in Figure 2, this multiplicator module is duplicated N-K times and concurrently driven by clock to perform synchronous multiplications.

3.2. Controller

We have learnt from Figure 4 that a certain number of states are required for the multiplication. To produce parity symbols in RS codes, the multiplication process is performed N-K times. A controller is required to perform this series of multiplication. In addition, it also drives the behavior of register $P(x)$ and feeds the $i(x)$ input with appropriate symbols serially. Algorithm of the controller behaves as follows:

1. Reset all registers
2. Set primitive polynomial and generator polynomial used

3. Initialize all registers (all Sx = "1" to load new values)
4. Loop *m* times for multiplication process: Sa = Sp = "0" (hold) and Sb = "2" (shift left)
5. Store the product for next iteration
   Repeat step 2 - 5 *(N-K)* times.
6. Deliver the final result (parity codes)

The final result is the parity codes added to the information codes to form the RS codeword. Figure 5 shows controller states for *N-K=3* and *m=4*.

3.3. Main VHDL File

The main VHDL codes has the following tasks:
1. generates *N-K* registers, multiplicators and adders, each is of m-bit wide.
2. connects internal signals within the encoder
3. connects controller outputs to the corresponding encoder's signals.

Thanks to VHDL's feature, we can duplicate module easily with 'generate' command and define variable values with 'generic' keyword. Both features are shown in the VHDL codes in Figure 6. For clarity, the codes show main signals only. The signal names in the codes refer to the implementation of RS Encoder illustrated in Figure 7. A set of generator polynomial and information symbols as well as the primitive polynomial are stored in a text file. During simulation, these symbols are sequentially read from the file and fed to the encoding process. The result of encoding process is written to an output file. To keep the encoder remains synthesizable, all read and write processes are done externally from the encoder to simulate on-the-fly encoding process during real data transmission.

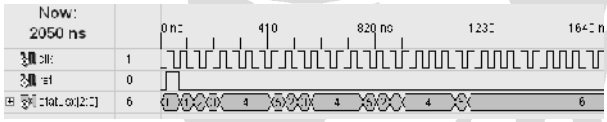

Figure 5. Controller states for *N-K = 3* and *m = 4*

```
entity Encoder is
   generic ( Nbit, NPar : positive );
   PORT ( ... );
end Encoder;

architecture Structural of Encoder is
component SRwPL generic ( Nbt : positive );
   PORT ( ... ); end component;

component SMulp generic ( Nbt : positive );
   PORT ( ... ); end component;

Regxs : for x in (NPar-1) downto 1 generate
    Rx: SRwPL generic map ( Nbt => Nbit )
       PORT map ( i => Pi(x), q => P(x), .... );
end generate;

RegLSB : SRwPL generic map ( Nbt => Nbit )
 -- special for LSB register
       PORT map ( i => F(0), q => P(0), .... );

Multx : for y in (NPar-1) downto 0 generate
   Mulx: SMulp generic map ( Nbt => Nbit)
       PORT map ( a=>FB, b=>g(y), p=>Poly, Sx=>Selc, c=>F(y),... );
end generate;

Addxs : for z in (NPar-1) downto 1 generate
   Addx : Pi(z) <= P(z-1) XOR F(z);
end generate;
```

```
FB <= P(NPar-1) XOR Infx;
end Structural;
```

Figure 6. RS Encoder VHDL Skeleton

4. Experimental Results and Discussion

The generic RS Encoder has been exercised for a great variety of parameters using Xilinx ISE 8.1i Simulator. For short data and small number of bits, the output written to files was compared with the result of manual calculation. For long information symbols, say $N>7$ and $m>3$, the results were examined using the Reed-Solomon Calculator available in [11]. Experiments with great variety of parameter combinations show that the RS Encoder works correctly.

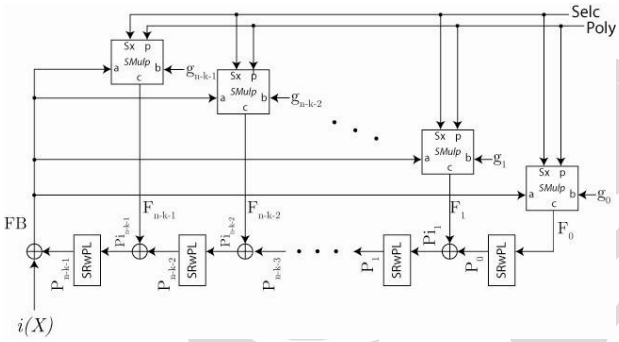

Figure 7. RS Encoder implementation

The VHDL codes has been synthesized for FPGA chip Xilinx XC3S200-5FT256. The device utilization shows that a 4-bit SRwPL requires four flip-flops and four 4-to-1 Multiplexers, whereas a 4-bit Serial Multiplicator requires 20 Flip-flops, 20 4-to-1 Multiplexers and four 1-bit XOR3. As shown in Figure 7, a pair of SRwPL and Serial Multiplicator is duplicated *(N-K)* times to build an *RS(N,K)* codes. Hence, device utilization for *m*-bit *RS(N,K)* is *6m(N-K)* Flip-flops, *6m(N-K)* 4-to-1 Multiplexers and *N-K* XOR3. The parity symbols are generated in *(N-K)(4+m)+2* clocks with the maximum frequency of 241 MHz. Hence, for RS(255,223) 8-bit (NASA standard) [12], the encoder requires 388 clocks per block. In other words, 255 8-bit symbols are ready for transmission every 1.6 µs. Thus, the throughput is 1.275Gbps or 159Mbytes/sec. This is much faster than commercially available equivalent encoder in [13], whose throughput is 750Mbps and maximum frequency is 100MHz. The performance of [13] is measured based on the implementation on Xilinx Spartan III 50-4, which is the same family with XC3S200-5FT256. The result is also much better than [14] whose maximum throughput is 108Mbytes/sec implemented on Spartan II XC2S100.

5. Conclusion

A generic Reed Solomon encoder has been developed with the following parameters: information symbol (*K*), parity symbol (*2t*), size of symbol (*m*), primitive polynomial *p(x)* and generator polynomial *P(x)*. Hence, its characteristics can be altered simply by modifying the parameters. The encoder has greater flexibility and reusability compared to the other similar implementa-tions such as in [1][2][3] whose characteristics are fixed for certain configuration and features. The encoder's codes in VHDL has been verified by simulation for any values of N, K and *m* to represent *m*-bit RS(*N,K*) using Xilinx ISE 8.1i. The codes can be compiled into bitstream for FPGA/CPLD chip implementation. It has been shown that the design's space and time complexity increases linearly by the number of parity symbols (*N-K*). The code implementation on FPGA chip Xilinx XC3S200-5FT256 results in an RS encoder with throughput 1.275 Gbps. It has a better performance than several commercially available encoders, such as [13], [14] and [15].

Acknowledgements

This material is based upon work supported by European Union Project VN/Asia-Link/001 (79754). The author would like to thank Prof. R.G. Spallek and Mr. Jörg Schneider for providing tools and facilities in the Professur für VLSI-Entwurfssysteme, Diagnostik und Architektur, Technische Universität Dresden, Germany.

References

[1] C.Paar and M.Rosner, *Comparison of Arithmetic Architectures for Reed-Solomon Decoders in Reconfigurable Hardware*, IEEE Symposium on FPGA-Based Custom Computing Machines (FCCM), pages 219-225, April 1997.

[2] Kamran Saleh, *Hardware Architectures for Reed-Solomon Decoders*, Technical Report, Amirkabir University of Technology, Iran, January 2003.

[3] Yousef R.Shayan, Tho Le-Ngoc, Vijay K.Bhargava, *Design of Reed-Solomon(16,12) Codec for North American Advanced Train Control System*, IEEE Transc. On Vehicular Technology 39(4):400-409, Nov 1990.

[4] Stephen B.Wicker, *Error Control Systems for Digital Communication and Storage*, Prentice Hall, New Jersey 07458, 1995.

[5] Robert H.Morelos and Zaragoza, *The Art of Error Correcting Codes*, John Wiley & Sons, Ltd., England, 2003

[6] Jr. Charles H.Roth, *Digital Systems Design using VHDL*, PWS Publishing Company, Boston, 1998.

[7] Daniel D.Gajski, *Principles of Digital Design*, Prentice Hall, US edition, 1007. Fig 7.5 – Page 272.

[8] W.W.Peterson, *Error Correcting Codes*, MIT Press, Cambridge, MA, 1961.

[9] P.A.Scott, S.E.Tavares, L.E.Peppard, *A Fast VLSI Multiplier for GF(2^m)*, IEEE Journal on Selected Areas in Communications, SAC-4(1):62-66, January 1986.

[10] E.F.Mastrovito, *VLSI Architectures for Computations in Galois Fields*, PhD Thesis, Dept. of Electrical Engineering, Linkping University, Sweden, 1991.

[11] Emina Soljanin, *Finite Field Calculator and Reed-Solomon Simulator*, http://cm.bell-labs. com/who/emina/applets/FFCalc.html. visited: Sept 1, 2006.

[12] Sklar, Bernard, "Digital Communications Fundamentals and Applications", Prentice Hall, Inc., New Jersey, 2nd ed., 2003.

[13] ASICSws, "Reed Solomon Encoder IP Core", http://www.asics.ws/doc/rse_brief.pdf, visited: Nov 8, 2006.

[14] Xilinx, Inc., *Reed-Solomon Solutions with Spartan-II FPGAs*, White Paper: Feb 10, 2000.

[15] VOCAL Technologies Ltd., "Reed Solomon Coding", 90A John Muir Drive, Buffalo - New York 14228, 2006.