Implementation of (15, 9) Reed Solomon Minimal Instruction Set Computing on FPGA using Handel-C

Jia Jan Ong, L.-M. Ang and K. P. Seng
Department of Electrical and Electronic
The University of Nottingham Malaysia
Semenyih, Malaysia
e-mail: keyx9ojj@nottingham.edu.my, kezklma@nottingham.edu.my, kezkps@nottingham.edu.my

***Abstract*— Reed Solomon coding has an important role to play as to sustain reliability of data communication. However, the encoder consumes significant amount of power that affects the cost of producing the hardware. Besides, the complicated encoder circuit also does affect the cost of implemented hardware. There is a need to build a simple encoder which does the similar data encoding function. By amalgamate One Instruction Set Computer (OISC) and the Galois Field arithmetic, a Reed Solomon Minimal Instruction Computer (MISC) processor is developed. This processor has simpler circuit that still has the same encoded codeword produced.**

***Keywords* — Reed Solomon; Reduced Instruction Set Computing.**

I. INTRODUCTION

In data communication, high-integrity data transmission is an important issue to consider. Error correction plays an important role for retaining the integrity of data [13]. Besides, error correction can improve the performance of data communication. This includes allowing more data to be transmitted across a limited bandwidth, reduces the power consumption in transmitting data and also produces less costly equipments [12]. Therefore, data communication systems are built in with error correcting capabilities. However, the built in error correction encoder needs to operate in low power. In order to realize such system, a simple Minimal Instruction Set Computer (MISC) processor with Reed Solomon encoder capabilities can be developed. By programming the processor, it can encode the data input and produce the required codeword as output. Therefore, the encoder would have simple architecture and circuit to implement onto hardware. The proposed MISC processor can be further developed to be integrated in the transmitter for communication system.

II. GALOIS FIELD

Finite field arithmetic operations play an important role in cryptography systems [6,8] and error-correction coding like Reed Solomon [9,10]. For Reed Solomon, the finite field arithmetic operations involve addition and multiplication of two 4-bit data.

The addition of two field elements GF(2) involves addition of respective bit in the data. However, the addition result is in modulo 2, meaning that $1 + 1 = 0$. The same principle also applies to the extension field, $GF(2^4)$ [16].

Multiplication in $GF(2^2)$ involves the product of multiplication of two field elements, a(x) and b(x) with modulo primitive polynomials, $p(x) = x^2+x+1$. Polynomial p(x) is a polynomial over GF(2) since its coefficients x^2 and x are members of GF(2) [16]. Fig. 1 shows the hardware implementations of $GF(2^2)$ multiplier, which can be later used in constructing $GF(2^4)$ multiplier. The symbol represents XOR of two bits.

As for the $GF(2^4)$ multiplier, the primitive polynomials used will be $p(x) = x^4+x+1$. The multiplier can also be constructed using subfield $GF(2^2)$ multiplier [5,6,7,8]. Fig. 2 shows the hardware implementation of $GF(2^4)$ multiplication block using subfield $GF(2^2)$. The $GF(2^2)$ subfield are represented by the box with an X in it. The $GF(2^4)$ multiplier used in the AES system [6,8] produces different result compare to the $GF(2^4)$ multiplier used for the Reed Solomon.

I. REED SOLOMON

In 1959, Reed Solomon code was introduced by Irving Reed and Gus Solomon [15]. This code has wide range of applications, from CD players to outer space communications with spacecraft. It has also become an integral part of telecommunication revolution. Reed Solomon codes use finite field arithmetic, also known as Galois Field (GF), for encoding and decoding the data.

The Reed Solomon encoder implemented will produce 15 symbols for a particular codeword. This includes 9 data symbols and 6 redundant symbols. With the redundant symbols, the receiver could detect 6 symbols error and correct 3 symbols error. This is known as the (15,9) Reed Solomon. Fig. 3 shows the arrangement of the generated redundant symbols and the information symbols (data) in a particular codeword [14,16,17].

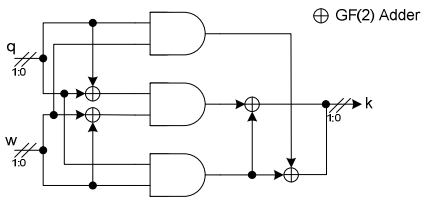

Figure 1. Hardware implementation $GF(2^2)$ multiplier.

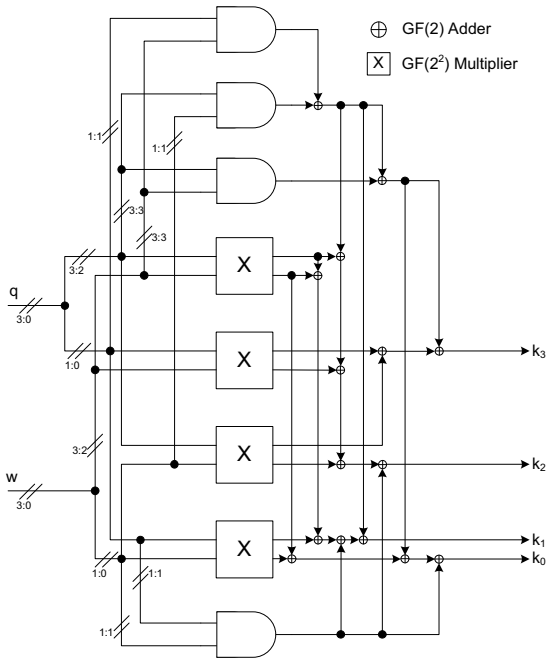

Figure 2. Hardware implementation GF(2^4) multiplier.

For (15,9) Reed Solomon encoder implemented, the Galois Field involved would be GF(2^4). Consider α to be the primitive element in GF(2^4). 3 error correction Reed Solomon code with symbols from GF(2^4) will have 0, α^0, α^1, … , α^{14} as roots. Then the generator polynomial [9,11] would be:

$$g(x) = (x-\alpha^1)(x-\alpha^2)(x-\alpha^3)(x-\alpha^4)(x-\alpha^5)(x-\alpha^6) \quad (1)$$

The result of the generator polynomial from equation (1) is $g(x) = x^6+7x^5+9x^4+3x^3+12x^2+10x+12$. As shown in Fig. 3, the Reed Solomon encoder circuit [11,16] can be determined with the generator polynomial. Each codeword output from the encoder will be in the arrangement as shown in Fig. 4.

II. Reduced Instruction Set Computer

Reduced Instruction Set Computer (RISC) is a processor that is made up of small number of simple, fixed-format and fixed-length of instructions [1]. There are three different models of one instruction set computer (OISC) architecture [2,3]. This includes Subtract and Branch if Negative (SBN) processor, Half Adder processor, and Move processor. The Reed Solomon MISC implemented is based on SBN processor.

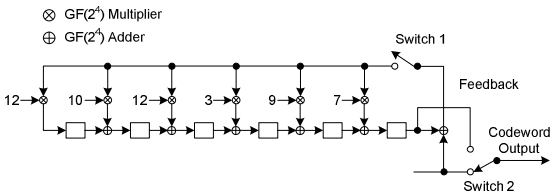

Figure 3. (15,9) Reed Solomon Encoder.

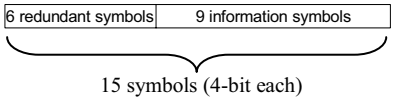

Figure 4. Codeword generated.

A. SBN Processor

The SBN processor was proposed by van der Poel [1]. It performs the operations of subtracting the second data, B (2nd operand) with first data, A (1st operand) from the memory. The result of the arithmetic operation is stored back to the memory location of data B [1]. The arithmetic operation is B = B – A. However, the instruction does have one condition, which is jump to a particular address once negative result is obtained from the arithmetic operation. Fig. 5 shows the format of the SBN instruction for the SBN processor.

B. Reed Solomon MISC Processor

The (15,9) Reed Solomon MISC processor proposed consist of an 8-bit adder block, a 4-bit XOR block and a GF(2^4) multiplier block. The adder block does the conditional branching of program (SBN), determines the next program address to be read, moving of the data from one location to another in the memory. As for the XOR block, its function is to perform the GF(2^4) arithmetic additions of two finite elements. The GF(2^4) multiplier block would produce the result of multiplying two finite elements in modulus of the primitive polynomial.

Two additional instructions are available to program the MISC processor is the XOR and GF functions. The XOR instruction will take two data (4 LSB) from the memory and produce the result of XORing these data. For the GF instruction, the result is produced from multiplying two data (4 LSB) in Galois Field, GF(2^4). The results produced from both block are stored back to 1st operand memory location. Fig. 6 shows the two additional instruction formats for the Reed Solomon RISC processor. For these instructions, the 'Jump' Target Address is set to 0 such that following instruction in the program memory will be executed.

III. Implemented Minimal Instruction Set Computing

The implemented (15,9) Reed Solomon MISC processor is developed by enhancing the available Ultimate Reduced Instruction Set Computer (URISC) [1]. It is based on the Subtract and Branch if Negative (SBN) OISC processor [2]. The MISC processor is implemented onto a Celoxica RC10 board using Handel-C code. The Celoxica RC10 board consists of a Xilinx Spartan 3S1500L-4 FPGA, two 7-Segments, eight green LEDs and etc. With the used of Handel-C code, the architecture of the processor could be described onto the FPGA.

SBN	1st Operand's Address	2nd Operand's Address	'Jump' Target Address

Figure 5. SBN instruction format.

xor	1st Operand's Address	2nd Operand's Address	'Jump' Target Address
gf	1st Operand's Address	2nd Operand's Address	'Jump' Target Address

Figure 6. XOR, GF instruction format.

A. MISC Architecture

Fig. 7 shows the architecture of the (15,9) Reed Solomon MISC processor that was implemented. The architecture shows a number of control inputs that control the registers, multiplexers (MUX) and memory. During each clock cycle, different control signals are generated and fed into the architecture.

From the MISC architecture, there are 8 registers, 7 multiplexers (MUX), 8-bit adder block, 4-bit XOR block and GF(2^4) multiplier block. The adder block is to execute SBN instruction. For the 4-bit XOR block, it will perform the XORing on two 4 least significant bit (LSB) data. The GF(2^4) multiplier block will carry out Galois Field multiplication onto two 4 LSB data.

For the program counter (PC) register, it holds the memory address of the next program code, which determines the program code to be read from memory. There are two registers, OPCODE0 and OPCODE1, which store the opcode read from the most significant bit (MSB) of the program code. The opcode will determine which instruction that the processor needs to execute. Depending on the opcode, the data read from memory and R register will be fed to particular block. The Memory Address Register (MAR) stores and provides memory address for writing or reading from the memory. The Memory Data Register (MDR) stores the result produced from the XOR block, GF(2^4) block and arithmetic subtraction block. The data in MDR will then be written back to the memory, replaces the value of the second read data.

B. Memory

The memory used in this MISC architecture is designed based on the Von Neumann Architecture. The amount of memory available is 288 bytes (256 x 9-bit). From Fig. 8, the data memory takes up 10.9% of memory and the program memory takes up 50.0% of memory. However, the actual program written to the memory only takes up 43.4%.

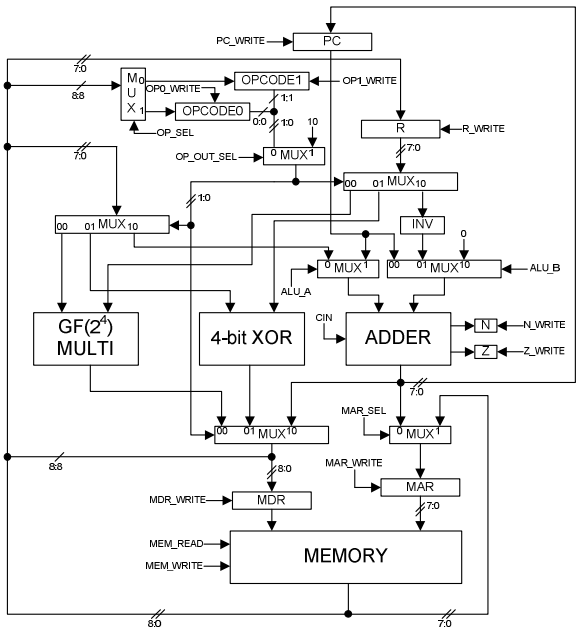

Figure 7. Reed Solomon MISC architecture.

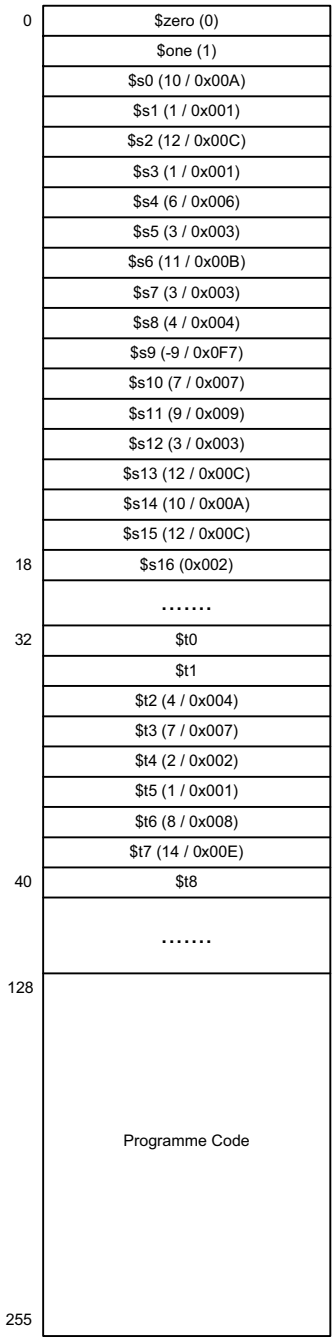

Figure 8. Data & program memory arrangement.

The data takes the memory location from 0 to 18, which are the fixed data and input data. As for the data situated at memory location from 32 to 40, they are temporary data which will be overwritten during execution of the program. At the end of program, encoded redundant symbols will also be written onto these memory locations. The corresponding first to last produced redundant symbols are written to $t2 to $t7 respectively. Whereas the program code, it takes the memory location from 128 to 239.

C. Control Signals

The Reed Solomon RISC processor architecture shown in Fig. 7 requires control signals to operate properly. Each program instruction requires a total of 9 clock cycles. During each clock cycles, different sequences of control signals are needed to be fed into the architecture. The

control signals generated are based on the logic equation (2) till (15).

Combinational logic circuits are implemented, with the use of a 4-bit counter, to generate the control signals. The counter will count from 0 to 8 and restart to 0 once it reaches 8. Each counter value will change each individual control signals accordingly to the equations. Fig. 9 shows the combinational logic circuits implemented using the equation (2) to (15).

$$ALU\_A = \overline{C_3}\overline{C_2} + \overline{C_3}\overline{C_0} + \overline{C_2}\overline{C_1}\overline{C_0} \quad (2)$$

$$ALU\_B_1 = \overline{C_3}\overline{C_2} + \overline{C_3}\overline{C_0} + \overline{C_2}\overline{C_1}\overline{C_0} \quad (3)$$

$$ALU\_B_0 = \overline{C_3}C_2\overline{C_1}C_0 \quad (4)$$

$$CIN = \overline{C_3}C_2\overline{C_1} + \overline{C_3}\overline{C_1}C_0 + C_3\overline{C_2}\overline{C_1}\overline{C_0} \quad (5)$$

$$MAR\_SEL = \overline{C_3}C_2\overline{C_1} + \overline{C_3}\overline{C_1}C_0 \quad (6)$$

$$PC\_WRITE = \overline{C_3}C_2C_1C_0N + \overline{C_3}\overline{C_2}\overline{C_1}C_0 + \overline{C_3}C_2\overline{C_1}\overline{C_0} + C_3\overline{C_2}\overline{C_1}\overline{C_0} \quad (7)$$

$$R\_WRITE = \overline{C_3}\overline{C_2}C_1\overline{C_0} \quad (8)$$

$$Z\_WRITE = \overline{C_3}\overline{C_2}\overline{C_1}\overline{C_0} \quad (9)$$

$$N\_WRITE = \overline{C_3}C_2\overline{C_1}C_0 \quad (10)$$

$$MAR\_WRITE = \overline{C_3}\overline{C_2}C_0 + \overline{C_3}C_2\overline{C_0} + \overline{C_3}C_1\overline{C_0} \quad (11)$$

$$MDR\_WRITE = \overline{C_3}C_2\overline{C_1}C_0 \quad (12)$$

$$MEM\_READ = \overline{C_3}C_2\overline{C_1} + \overline{C_3}C_1C_0 + \overline{C_3}\overline{C_1}C_0 + \overline{C_3}\overline{C_2}C_1\overline{C_0} \quad (13)$$

$$MEM\_WRITE = \overline{C_3}C_2C_1\overline{C_0} \quad (14)$$

$$OP\_OUT\_SEL = \overline{C_3}\overline{C_1}\overline{C_0} + \overline{C_3}\overline{C_2} \quad (15)$$

D. Reed Solomon Encoder Program

With the Galois Field arithmetic functional block, program code is required to program the processor to perform the Reed Solomon encoding. For the processor to operate as Reed Solomon encoder, a total 37 lines of instructions need to be executed.

Initially, the program will write the first information symbol memory address to the program code 0x92. Next, the first information symbol ($s0) is read from the memory. With the data read, it will perform GF(2^4) arithmetic addition to the 1st register (from right of Reed Solomon encoder) value ($t2) to give the feedback. Next, it will process the feedback by performing GF(2^4) multiplication with 7 ($s10). Then perform GF($2^4$) addition with the 2nd register value ($t3) and stored the result to first register $t2. These operations are shown in equation (16). The feedback value will be used again to perform multiplication and follow up with addition, which is shown in equation (17) to (21). This is done sequentially instead of parallel operations.

$$1^{st} \text{ register} = \text{feedback} \times 7 (\$s10) + 2^{nd} \text{ register} (\$t2) \quad (16)$$

$$2^{nd} \text{ register} = \text{feedback} \times 9 (\$s11) + 3^{rd} \text{ register} (\$t3) \quad (17)$$

$$3^{rd} \text{ register} = \text{feedback} \times 3 (\$s12) + 4^{th} \text{ register} (\$t4) \quad (18)$$

$$4^{th} \text{ register} = \text{feedback} \times 12 (\$s13) + 5^{th} \text{ register} (\$t5) \quad (19)$$

$$5^{th} \text{ register} = \text{feedback} \times 10 (\$s14) + 6^{th} \text{ register} (\$t6) \quad (20)$$

$$6^{th} \text{ register} = \text{feedback} \times 12 (\$s15) \quad (21)$$

From Fig. 10, there are 37 lines of instruction, where 31 instructions are repeated for 9 times. At program code 0xE9, it modifies the value of program code at 0x92 such that the next information symbol stored in the memory can be read. This does not follow the standard program memory, where the program code is not changed during execution. The changing of the program code during execution is to save a significant amount of instructions that need to be written. Without the modification made, it requires additional 247 lines of program instructions to substitute the repetitions. After 9 repeat instructions executed, the redundant symbols are produced and stored in location from $t2 to $t7.

I. RESULT

Each line of instructions executed by the processor would require 9 clock cycles. Since there are a total of 37 instructions, with 31 instructions are repeated for 9 times, there will be a total of 2565 clock cycles. The number of clock cycles required is calculated as shown in equation (22).

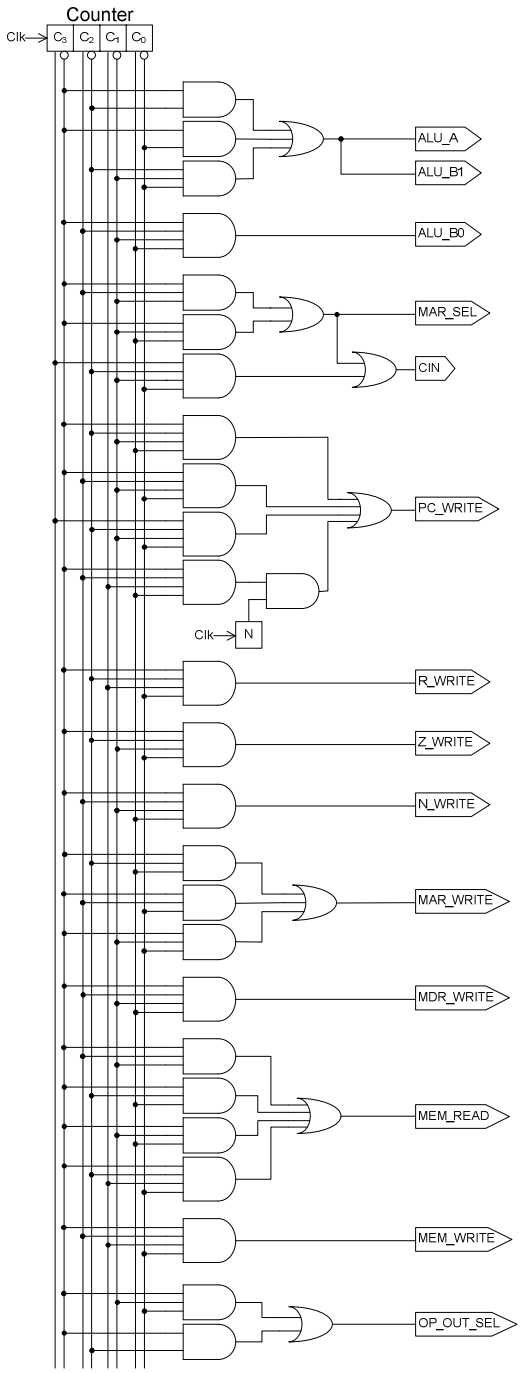

Figure 9. Combinational logic circuit to generate control signals.

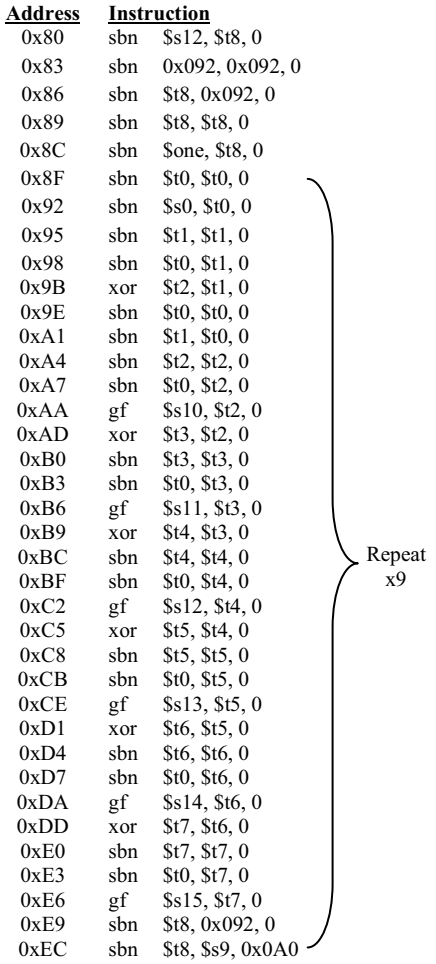

Figure 10. Reed Solomon encoder program code.

$$\begin{aligned} \text{Total clock cycles} &= 6 \times 9 + 9 \times (31 \times 9) \\ &= 54 + 2511 \\ &= 2565 \text{ clock cycles} \end{aligned} \quad (22)$$

A. *Changed Variables*

Once Reed Solomon MISC processor executes the program code, the redundant symbols produced can be verified by performing manual arithmetic on the information symbols. With the calculated redundant symbols, based on the encoder circuit, the symbols produced by the processor can be compared.

Using the two 7-segment LED displays available on the Celoxica RC10 board, the variables that stored in the memory can be displayed. The variables will be displayed after the processor has completed executing the program code. The value displayed will be from memory location 0x00 to 0xFF. This also includes the data and program memory. Table 1 shows the variables that are changed after the program is executed. The values in memory location from $t2 to $t7 are the redundant symbols produced from the encoding of information symbols. Besides, the table also includes the value program code (0x92) which is modified when the program is running.

TABLE I.
DATA AND PROGRAM MEMORY THAT CHANGES.

Memory Address	Variables	Value	
		Initial	Final
0x0B	$s9	0x0F7	0x000
0x20	$t0	0x000	0x0F6
0x21	$t1	0x000	0x006
0x22	$t2	0x000	0x004
0x23	$t3	0x000	0x007
0x24	$t4	0x000	0x002
0x25	$t5	0x000	0x001
0x26	$t6	0x000	0x008
0x27	$t7	0x000	0x00E
0x28	$t8	0x000	0x0FF
0x92	-	0x002	0x00B

TABLE II.
HARDWARE USAGE IN IMPLEMENTING REED SOLOMON MISC ENCODER.

Component	Quantity	Total	Usage
Flip Flop	216 slice	13312	1%
4 input LUTs	372	26624	1%
- Logic	333	-	-
- Route-thru	37	-	-
- Dual Port Rams	0	-	-
- Shift registers	2	-	-
Bonded IOB	46	221	20%
Block RAMs	1	32	3%
BUFGMUXs	3	8	37%
DCMs	1	4	25%

B. *Hardware Usage*

The implementation of Reed Solomon MISC processor onto the RC10 board takes only a small amount of the hardware available. From Table 2, 1% of available flip-flop and LUTs in the FPGA is being used. The block usage is only 3%, which takes up 288 bytes of program and data memory.

II. CONCLUSION

Implementing Reed Solomon code to a MISC processor produces a simple error correction encoder. With only a small amount hardware needed, the cost of fabricating the encoder in a chip would be low. At the meantime, low power consumption could be achieved by this encoder, which is an advantage to be integrated into data communications. This integration is important, especially on communication device that has sacred amount of energy sources. With the Reed Solomon MISC processor, real-time applications could be realized by having data input to it and encoded codeword is produced.

REFERENCES

[1] Farhad Mavaddat, and Behrooz Parhami, "URISC: The Ultimate Reduced Instruction Set Computer", Int. J. Electrical Engineering Education, Vol. 25, No. 4, Manchester, pp. 327-334, October 1988.

[2] Phillip Laplante, and William Gilreath, "Single Instruction Set Architectures for Image Processing", Proceedings of SPIE Vol. 4868, pp. 20-28, 2002.

[3] William F. Gilreath, and Phillip Laplante, Computer Architecture: A Minimalist Perspective, Springer Science+Business Media, Inc., New York, 2003.

[4] Christof Paar, Peter Fleischmann, and Peter Roelse, "Efficient Multiplier Architectures for Galois Fields GF(24n)", IEEE Transactions on Computers, Vol. 47, No. 2, pp. 162-170, February 1998.

[5] Yong Suk Cho and Sang Kyu Park, “Design of GF(2m) multiplier using its subfields”, IEE Electronics Letters, Vol. 34, No. 7, pp 650-651, 2nd April 1998.

[6] Edwin NC Mui, “Practical Implementation of Rijndael S-Box Using Combinational Logic”, Texco Enterprise Ptd. Ltd., 2007.

[7] Christof Paar, “A New Architecture for a Parallel Finite Field Multiplier with Low Complexity Based on Composite Fields”, IEEE Transactions on Computers, Vol. 45, No. 7, pp. 856-861, July 1996.

[8] Atri Rudra, Pradeep K. Dubey, Charanjit S. Jutla, and et al, “Efficient Rijndael Encryption Implementation with Composite Field Arithmetic”, Cryptographic Hardware and Embedded Systems – CHES 2001, Lecture Notes in Computer Science 2162, pp. 171-184, 2001.

[9] David Banks, “Design of a Synthesisable Reed-Solomon ECC Core”, Publishing Systems and Solutions Laboratory, HP Laboratories Bristol, 2001.

[10] G.C. Cardarilli, S. Pontarelli, M.Re, A. Salsano, “Design of a Self Checking Reed Solomon Encoder”, Proceedings of the 11th IEEE International On-Line Testing Symposium, pp. 201-202, 2005.

[11] Shu Lin, and Daniel J. Costello, Error Control Coding Second Edition, Pearson Education, Inc., Upper Saddle River, 2004.

[12] “Primer: Reed-Solomon Error Correction Codes”, AHA Application Note.

[13] Elwyn R. Berlekamp, Robert E. Peile, and Stephen P. Pope, “ The Application of Error Control to Communications”, IEEE Communications Magazine Vol. 25, No. 4, pp. 44-57, April 1987.

[14] Joel Sylvester, “Reed Solomon Codes”, Elektrobit, January 2001.

[15] Stephen B. Wicker, and Vijay K. Bhargava, Reed-Solomon Codes and Their Applications, IEEE Press, New Jersey, 1994.

[16] Syed Shahzad Sha, Saqib Yaqub, and Faisal Suleman, “ Self-correcting codes conquer noise Part 2: Reed-Solomon codecs”, EDN, pp. 107-120, March 2001.

[17] Francisco J. Gacia-Ugalde, “Coding and Decoding Algorithms of Reed-Solomon Codes Executed on A M6800 Microprocessor”, Coding Theory and Applications, LNCS 311, Springer Berlin, pp. 183-196, 1988.