International Journal of Sensor and Its Applications for Control Systems
Vol.1, No.1 (2013), pp.1-12
http://dx.doi.org/10.14257/ijsacs.2013.1.1.01

Implementation of Parallel BCH Encoder Employing Tree-Type Systolic Array Architecture

Je-Hoon Lee and Sharad Shakya

Dept. of Electonic Enginerring, Samcheock Campus, Kangwon National University
jehoon.lee@kangwon.ac.kr

Abstract

A BCH (Bose-Chaudhuri and Hochquenghem) code is one of the most widely used error correcting codes for the detection and correction of random errors. This paper presents an error correction coding with a suitable BCH code for MLC (multi-level cell) flash memory. The conventional bit serial BCH code cannot adequate with the recent high speed circuit. Therefore, parallel encoding and decoding algorithms are always a necessity. We introduced a new systolic array type BCH parallel encoder and syndrome generator for decoder. The area and speed of the encoder is compared with several parallel factors. Furthermore, to prove the efficiency of the proposed algorithms using tree-type structures, the throughputs and the area overhead of the encoder and the syndrome generator were compared with their counterparts also. The proposed algorithm has a great flexibility in parallelization and the speed with tree-type structure was increased by 29% and 33% respectively. The synthesis and simulation results were implemented on FPGA using VHDL.

Keywords: *BCH, Error correction code, flash memory, multi-level cell, parallel processing*

1. Introduction

In recent years, there has been an increasing demand for efficient and reliable digital data transmission and storage systems. This demand has been accelerated by the rapid development and availability of VLSI technology, high speed data networks, and storage of digital information. It is obvious that a digital system must be fully reliable, as a single error may shutdown the whole system, or cause unacceptable corruption of data. In such situations, this error control must be employed so that an error may be detected and afterwards corrected. For this reason, the use of for error control code has become an integral part in the design of modern communication and digital storage systems.

Flash memories are widely used storage elements in embedded systems and mobile applications. Low-power consumption, non-volatile and high density storage are the properties of flash memories. There are two commonly used flash memories: NAND and NOR flash memory. Basically NAND type is used as high density data storage, whereas NOR type is used for code storage and direct execution. Some major differences between NAND and NOR type are shown in Table 1. SLC *(single-level cell)* stores a single bit in a cell, whereas MLC *(multi-level cell)* stores multiple bits in a cell. There is a high chances of occurring error during the read and write cycles due to the reduction of the programming voltage in MLC for each of the data levels [1-2]. The organization of MLC flash memories is shown in Figure 1. Each plane consists of 2KB page register, with additional 64 spare bytes to store the ECC parity bits. Each block consists of several pages. The page registers are divided into four 512byte sectors. Page registers are organized in two ways as shown in Figure 2(a) and Figure 2(b).

Table 1. Differences between NAND and NOR type Flash Memory

	NAND type	NOR Type
Cost per bit	Low	High
File storage	Easy	Hard
Code execution	Hard	Easy
Capacity	High	Low
Write Speed	High	Low
Read Speed	Medium	High
Addressing of data	By page number: commonly 512B	Memory mapped addresses

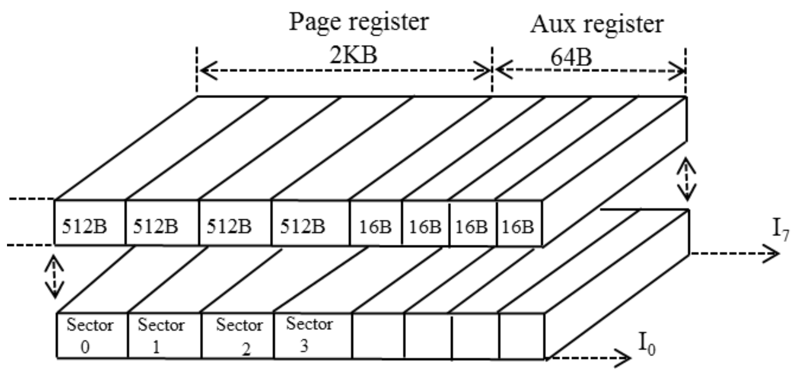
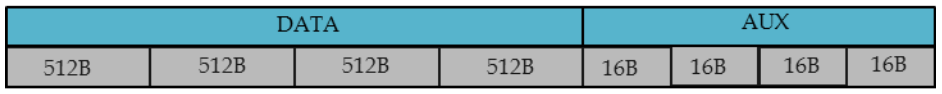
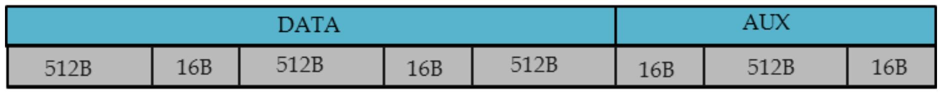


Figure 1. MLC Flash Memory Block Diagram



(a)



(b)

Figure 2. Organizations of Page Register of a NAND Type

The most common way is to store data in lower part of the page along with error correction bits in the upper part as in Figure 2(a). However, some implementation prefers different organization to increase performance as in Figure 2(b). For a write operation, the input data are encoded before writing to the page register. For a read operation, the data from the registers are decoded. The decoding failure reported is the number of errors exceeded the designed capability of the ECC circuits. In a flash memory, the read and write operations of data are conducted in bytes at each clock cycle. Thus, byte-wise parallel encoding and decoding shall be desired for high-speed flash memories.

The theory of error detecting and correcting codes deals with the reliable storage of data. Information media is not completely reliable in practice since the noise frequently causes data to be distorted. In particular, BCH code for multiple error correction is widely used in MLC flash memory. Generally, BCH encoder can be implemented either by hardware or software methods. Since software implementation of BCH code cannot reach the needed limited speed, the hardware design is preferred for high-speed applications. In conventional BCH design, a simple shift register, that is, LFSR (linear feedback shift register) is associated with specific XOR gates to computes one bit message data per cycle. Owing to the ever increasing demands for high speed communication appliances, the conventional bit-serial BCH encoder should be replaced to the new parallel BCH encoders that have been presented until now.

In a parallel BCH encoder circuit, *p*-bits of data are processed at a time, where the speed is *p* times faster with a relatively small increase in hardware. Several parallel processing methods have been introduced earlier such as matrix multiplication, unfolding method, CRT based encoding [3-5]. The high speed circuit is always in required but the area and the flexibility should be also considered. In this paper, we present a systolic array BCH encoder employing tree-type structure. The proposed BCH encoder has a great flexibility in parallelization without any complexity with high throughput. It can improve the performance compared to its counterparts without any significant area increase.

2. Serial and Parallel BCH Encoder

For any positive integers $m \leq 3$ and $t < 2^m-1$, there are the possibilities of binary BCH (n, k) code with following parameters:

Block Length: $n = 2^m-1$

Parity Check Bits: $n - k \leq mt$

Minimum distance: $d \geq 2t + 1$

This *BCH (n, k)* code is capable of correcting t errors in a block of $n = 2^m-1$, where, *n* is the length of the codeword, *k* is the length of the message bits, n-k is the length of the parity bits and t is the number of errors to be corrected. The BCH *(4122, 4096)* is a shortened code of BCH *(8191,8165)* over Galois Field GF(2^{13}). This code has an ability to correct two-errors. A shortened BCH code is used when the block length is smaller than 2^m-1, and can be considered as a standard BCH code. This code form a subset of BCH code with good error correction and detection capabilities as the original long BCH code. The encoding and decoding of the shortened BCH code is almost same as the long BCH code.

In a binary BCH *(n,k)* code, a *k*-bit message is encoded in *n*-bit codeword. It consists of *k*-bit message and *n-k* parity bits. The *n*-bit codeword is defined as $(C_{n-1}, C_{n-2}, \ldots, C_0)$, where as $C_i \in GF(2), (0 \leq i \leq n-1)$ as the coefficient of a degree *n-1* of polynomial *C(x)* and *k*-bit message is defined as $(m_{k-1}, m_{k-2}, \ldots, m_0)$, where as $m_i \in GF(2), (0 \leq i \leq k-1)$ as the coefficient of a degree *k-1* of polynomial *m(x)*. The encoding of a BCH code can be expressed as $c(x) = m(x)g(x)$, where *g(x)* is the generator polynomial of a degree *n-k*.

Consistently, BCH encoding is constructed with three steps. Multiplying the message *m(x)* by x^{n-k} and dividing it by generator polynomial *g(x)*, where the remainder $Rem(m(x)x^{n-k})_{g(x)}$ is obtained. The remainder is now added to the message to form a codeword as shown in Eq. 1. In this paper, the proposed BCH encoder is implemented with *(31, 16, 3)* BCH code [6]. Thus, we got the generator polynomial, *g(x)* as shown in Eq. 2.

$$c(x) = m(x).x^{n-k} + Rem(m(x).x^{n-k})_{g(x)} \quad (1)$$

$$g(x) = 1+x+x^2+x^3+x^5+x^7+x^8+x^9+x^{10}+x^{11}+x^{15} \quad (2)$$

For the conventional bit-serial encoding, the *k* message bits are input to the LFSR with bit-serial manner. At the *k-th* cycle, the registers contain $Rem(m(x).x^{n-k})_{g(x)}$, which is also called the parity bits. Figure 3 illustrates the circuit connection of a conventional serial BCH encoder. The critical path of this bit-serial architecture consists of two XOR gates as shown in Figure 3. This architecture is quite straightforward, but it cannot run in a high speed as the application requirement. The three steps of the BCH encoding can be done with a simple LFSR architecture. The conventional serial encoder for BCH *(4122, 4096)* is shown in the Figure 3. The feedback terms are determined by the generator polynomial g(x) in the Eq. (2). Modulo-2 addition is processed with XOR gate and multiplication is with the finite field multiplier. There are total number of 26 (n-k) registers to store the remainder $Rem(m(x)x^{n-k})_{g(x)}$. For binary BCH code, the multiply operation is simplified to a connection or a disconnection when g_i is $(0 \le i \le n\text{-}k)$ equals to ‘1’ or ‘0’.

The parallelization of the circuit is the method of sending the *p* number of message bits at a time *t*. When *p* message bits arrive in each clock cycle, only *k/p* clock cycles are required to compute the remainder in the registers. Unfolding is a method of parallelization of a circuit and has a high throughput [4]. In the *J*-unfolded architecture, there are *J* copies of each node with the same function as in the original architecture. It is assumed that there is a path from node *U* to node *V* in the original architecture with *W* delay elements. Therefore, node U_i is connected to *V(i+w) percent J* with *[(i+w)/J]* delay elements, where, U_i, $V_j (0 \le i, j \le J)$ are the copies of nodes *U* and *V* respectively. Fan out problem also exist in the unfolding method, but retiming is not accessible when the *J* factor is larger than the degree difference between the highest and the second highest order of *g(x)*.

$$g(x) = 1 + g_1x + g_2x^2 + . . . + g_{n-k-1} x^{n-k-1} + x^{n-k} \qquad (3)$$

In the case, if a *J* unfolded BCH encoder is acquired, the generator polynomial needs to be modified and the remainder $Rem(m(x)x^{n-k})_{g(x)}$ in the BCH encoding can be implemented by the steps illustrated in Figure 4. Additional hardware will increase dramatically when large unfold factor *J* is used [5].

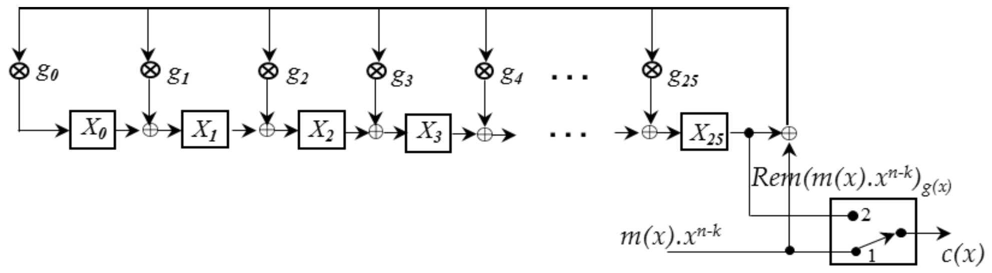


Figure 3. Serial BCH Encoder Architecture

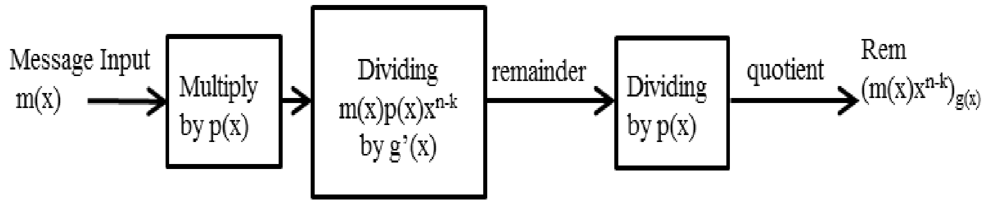


Figure 4. Block Diagram of BCH Decoder

3. Proposed Tree-type Systolic Array BCH Encoder

The major drawback of the unfolding method is the modification of the generator polynomial, when two highest degree orders are smaller than *J*, to perform retiming. This problem can be easily solved in the systolic array BCH encoder without any modification in the generator polynomial and additional hardware. Before implementing a *p*-parallel encoder, we need to know the state of the LFSR at time *t+1* from the state t. Let $X(t) = [X_0, X_1,\ldots,X_{n-1}]$ denotes the state of the registers at time *t* and *z(t+1)* denotes the input bit to be entered at time *t+ 1*. Then the state of the encoder at time *t+1* can be represented as:

$$X(t+1) = \begin{bmatrix} g_{n-1} & 1 & 0 & . & . & . & . & 0 \\ g_{n-2} & 0 & 1 & . & . & . & . & 0 \\ & 0 & 0 & . & . & . & . & 0 \\ . & & . & & & . & & \\ . & & . & & & . & & \\ . & & . & & & . & & \\ g_1 & 0 & 0 & & & & & 1 \end{bmatrix} \times \begin{bmatrix} x_{n-1}(t) + z(t+1) \\ x_{n-2}(t) \\ x_{n-3}(t) \\ . \\ . \\ . \\ x_0(t) \end{bmatrix} \tag{4}$$

The first column in matrix *F* is the coefficients of the generator polynomial *g(x)* and Eq. (4) can be written as:

$$X(t+1) = F\ x\ [X(t) + z(t+1)] \tag{5}$$

Let *X(t+ p)* is the state of the register at time *(t+ p)*, from the equation (5), a recursive equation for *X(t+ p)* can be derived as:

$$X(t+ p) = F^p\ x\ (X(t) + Z_p) \tag{6}$$

Where, $Z_p = [z_0, z_1, \ldots\ldots\ldots z_{p-1} \mid 0\ldots\ldots 0]^T$, the matrix F^p can be constructed recursively when *i* changes from *2* to *p*. Thus when a BCH encoder processes *p*-bit message in each cycle, it takes *k/p* cycles to save the parity bits in the registers [7].

$$F^i = \left[\begin{array}{c|c} F^{(i-p)}\ \mathrm{x} \begin{bmatrix} g_{n-1} \\ g_{n-2} \\ g_{n-3} \\ . \\ . \\ . \\ g_1 \\ 1 \end{bmatrix} & \begin{array}{c} \text{First (r-1)} \\ \text{columns} \\ \text{of} \\ F^{i-1} \end{array} \end{array} \right]$$

Figure 4 illustrates the eight-parallel systolic array BCH encoder for the generator polynomial given in equation 2, which serves as our example. Let $X_0(t+8)$ to $X_{14}(t+8)$ represents the value of the registers at *(t+8)*. $z_0(t)$ to $z_7(t)$ represents the eight parallel input data at *(t+1)*.From the equation (6), we can get the value of the registers processed in the systolic array BCH encoder as follows.

$X_{14}(t+8) = [z_4(t+1) + X_{10}(t)] + [z_3(t+1) + X_{11}(t)] + [z_2(t+1) + X_{12}(t)] + [z_1(t+1) + X_{13}(t)] + X_6(t)$

$X_{13}(t+8) = [z_5(t+1) + X_9(t)] + [z_4(t+1) + X_{10}(t)] + [z_3(t+1) + X_{11}(t)] + [z_2(t+1) + X_{12}(t+1)] + [z_1(t+1) + X_{13}(t)] + X_5(t)$

$X_{12}(t+8) = [z_6(t+1) + X_8(t)] + [z_5(t+1) + X_9(t)] + [z_4(t+1) + X_{10}(t)] + [z_3(t+1) + X_{11}(t)] + [z_0(t+1) + X_{14}(t)] + X_4(t)$

$X_8(t+8) = [z_7(t+1) + X_7(t)] + [z_6(t+1) + X_8(t)] + [z_4(t+1) + X_{10}(t)] + [z_3(t+1) + X_{11}(t)] + [z_2(t+1) + X_{12}(t)] + [z_1(t+1) + X_{13}(t)] + X_0(t)$

$X_7(t+8) = [z_7(t+1) + X_7(t)] + [z_5(t+1) + X_9(t)] + [z_3(t+1) + X_{11}(t)] + [z_2(t+1) + X_{12}(t)] + [z_1(t+1) + X_{13}(t)]$
$X_1(t+8) = [z_7(t+1) + X_7(t)] + [z_6(t+1) + X_8(t)] + [z_3(t+1) + X_{11}(t)]$

$X_0(t+8) = [z_7(t+1) + X_7(t)] + [z_3(t+1) + X_{11}(t)] + [z_2(t+1) + X_{12}(t)] + [z_1(t+1) + X_{13}(t)] + [z_0(t) + X_{14}(t)]$

The systolic array BCH encoder is almost the same with the conventional serial BCH encoder. In the serial BCH encoder the output of the rightmost XOR gate is the input to the rest of the XOR gates as well as the first register. The stages in the Figure 5 are the number of parallel factor *p*. The position of the XOR gates in each of the stages is a replica of the first stage but the input to every stage is from the previous stage. This process can be concluded as a shift operation. After the XOR operation, the output of the *p-1* stage is the input to the *0th* stage, and is repeated consecutively.

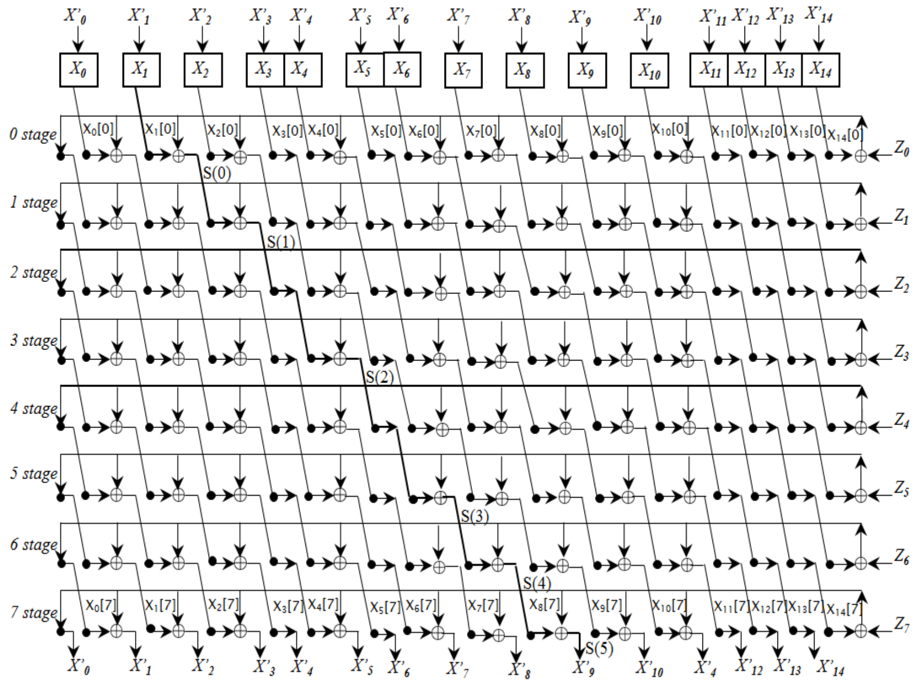

X'0
X'1
X'2
X'3
X'4
X'5
X'6
X'7
X'8
X'9
X'10
X'11
X'12
X'13
X'14
X0
X1
X2
X3
X4
X5
X6
X7
X8
X9
X10
X11
X12
X13
X14
0 stage
1 stage
2 stage
3 stage
4 stage
5 stage
6 stage
7 stage
X0[0] X1[0] X2[0] X3[0] X4[0] X5[0] X6[0] X7[0] X8[0] X9[0] X10[0] X11[0] X12[0] X13[0] X14[0]
X0[7] X1[7] X2[7] X3[7] X4[7] X5[7] X6[7] X7[7] X8[7] X9[7] X10[7] X11[7] X12[7] X13[7] X14[7]
S(0)
S(1)
S(2)
S(3)
S(4)
S(5)
Z0
Z1
Z2
Z3
Z4
Z5
Z6
Z7

Figure 5. Systolic Array Type BCH Encoder

Constructing a systolic array BCH encoder is simple and can be performed from the equation given in Eq. (7). The equation is to trace the path of the node from one register to another through different stages. The node can be mentioned by the degree of the register in a column and the position of the *p*-factor stages in a row. First, the conventional serial BCH encoder has to be drawn. The positions of the XOR gates are same as in first stage and are repeated till *p-1* stages.

$$X_a [b] = X_{a+r} [b+r] \tag{7}$$

Where, *a* is the degree of the register *b* is the parallel stage and *r* is the number of registers in between two XOR gates.

For example, to find a path from a node $X_2[3]$, $a = 2$, $b = 3$ and $r = 2$, we get a node $X_4[5]$ from the equation (4). The nodes of the registers are also shown in the Figure 5 as a dot. The first node without name, which is the feedback of the rightmost XOR gate, is always shifted by one to the next stage. The output of the rightmost XOR gate is input to all other XOR gates in a same manner of the serial BCH encoder. As the output of the *p-1* stages is the input to the registers, there is no need to trace the path. Instead of one message bit input in the serial BCH encoder, the encoder receives one byte of message bits in parallel in our example. This allows for multiple bits of encoding to be performed simultaneously and at greater speeds than using comparative models.

As the stages are the replicas of the first stage, it can be increased or decreased to any factor without any complexity in the circuit. The key reason of its flexibility is that the stages can be moved to any number of parallel factors. However, the value of the p-1 stage should be the input to the first stage.

In the paper, we focus on the critical path delay of the systolic array BCH encoder in the Figure 5. Since the circuit functions as a loop, it faces a critical path delay. In the eight parallel systolic array BCH encoder, the longest critical path lies from X_0 to X_8, X_1 to X_9, X_2 to X_{10}, X_3 to X_{11} and X_4 to X_{12}. All of them have a critical path of 7T, where T is the delay of an XOR gate. For example, the route from the register X_1 to X_9 is shown with a bold line.

During the XOR operation, the signal s(0) is obtained from the value of the register X_1 and the value from Z_0 and X_{14} , which has a critical path of 2T. Similarly the signal s(0) is transferred to the stage 1 to calculate the value from Z_1 and X_{13} and creates a new signal s(1). The critical path is added to 3T. The overall data delay path for the longest critical path for the signal s(5) is 7T . The critical delay path from the registers X_1 to X_9 is shown in Figure 6 in detail. In order to acquire a low delay, tree-type structure has been optimized for the Figure 6, which is not shown due to complexity. The critical path of the proposed eight parallel systolic array BCH encoder has been reduced to 5T from 24T as shown in Figure 7. Using tree type structure, the delay caused by the cascaded XOR gates is much less than that caused by large fan-out [4]. Compared to the encoder in the Figure 4, the proposed tree-type encoder is speed up by 40%.

$$\underset{2T}{s(0)} = \underbrace{\underbrace{X_1}_{1T} \oplus \underbrace{[Z_0 \oplus X_{14}]}_{1T}}_{2T}$$

$$\underset{3T}{s(1)} = \underbrace{\underbrace{s(0)}_{2T} \oplus \underbrace{[Z_1 \oplus X_{13}]}_{1T}}_{3T}$$

$$\underset{4T}{s(2)} = \underbrace{\underbrace{s(1)}_{3T} \oplus \underbrace{[Z_3 \oplus X_{11}]}_{1T}}_{4T}$$

$$\underset{5T}{s(3)} = \underbrace{\underbrace{s(2)}_{4T} \oplus \underbrace{\{\underbrace{[Z_5 \oplus X_9]}_{1T} \oplus \underbrace{\underbrace{[Z_1 \oplus X_{13}]}_{1T} \oplus \underbrace{[Z_0 \oplus X_{14}]}_{1T}}_{2T}\}}_{3T}}_{5T}$$

$$\underset{6T}{s(4)} = \underbrace{\underbrace{s(3)}_{5T} \oplus \underbrace{\{\underbrace{\underbrace{[Z_6 \oplus X_8]}_{1T} \oplus \underbrace{[Z_2 \oplus X_{12}]}_{1T}}_{2T} \oplus \underbrace{\underbrace{[Z_1 \oplus X_{13}]}_{1T} \oplus \underbrace{[Z_0 \oplus X_{14}]}_{1T}}_{2T}\}}_{4T}}_{6T}$$

$$\underset{7T}{s(5)} = \underbrace{\underbrace{s(4)}_{6T} \oplus \underbrace{\{\underbrace{[Z_7 \oplus X_7]}_{1T} \oplus \underbrace{\underbrace{\underbrace{[Z_3 \oplus X_{11}]}_{1T} \oplus \underbrace{[Z_2 \oplus X_{12}]}_{1T}}_{2T} \oplus \underbrace{\underbrace{[Z_1 \oplus X_{13}]}_{1T} \oplus \underbrace{[Z_0 \oplus X_{14}]}_{1T}}_{2T}}_{4T}\}}_{5T}}_{7T}$$

Figure 6. Data Delay Path from Register X_1 to X_9

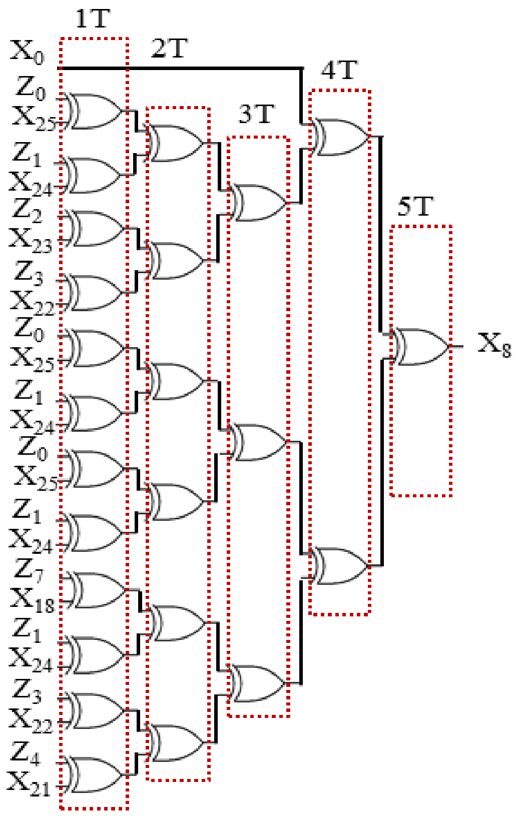


Figure 7. Optimization with Tree-Type Structure

4. Performance Results

The BCH (31, 16, 3) code encoder is synthesized on the Xilinx ISE 10.1, Vertex II PRO with a device XC2VP30. These device families provide logic array blocks that can accommodate a bunch of basic logic elements with fast interconnections. The simulation result for the encoder is implemented with the generator polynomial in Eq. (3) with degree 26 and 13 non-zero terms for the encoder.

First, we compared the proposed systolic array type BCH encoder with several parallel factors. Figure 8 (a) and Figure 8 (b) below shows the comparison results in area overhead and throughputs of serial, 8-parallel, 16-parallel and 32-parallel of the systolic array BCH encoders in Figure 8 (a), the number of XOR gates for serial encoder is just 14 and increases linearly as the parallel factor *p* increases. The XOR gates for 8, 16 and 32 parallel systolic array BCH encoder are 112, 224 and 448 respectively. Similarly, the number of slices used in the FPGA device for the 16 parallel is increased by 5.2 times than the serial one, whereas, 32-parallel is increased by 1,8 times than the 16-parallel.

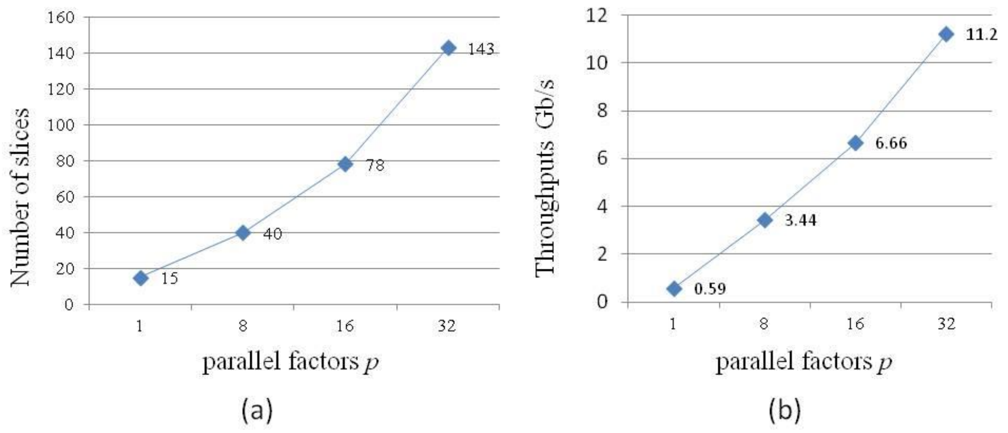


Figure 8. Performance Comparison Results according to Diverse Parallel Factor. (a) Comparison Results for Hardware Complexity, (b) Comparison Results for throughput

Figure 9 shows the simulation waveform of the *J* unfolding method and the systolic array type BCH encoder with the same parity bits in the register. Both of them are implemented with the generator polynomial in equation (2) with degree 15 and 9 non-zero terms.

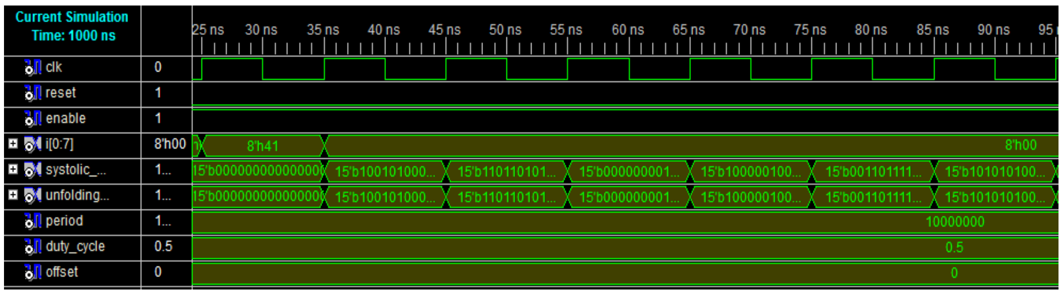

Current Simulation Time: 1000 ns
25 ns 30 ns 35 ns 40 ns 45 ns 50 ns 55 ns 60 ns 65 ns 70 ns 75 ns 80 ns 85 ns 90 ns 95
clk 0
reset 1
enable 1
i[0:7] 8'h00 8'h41 8'h00
systolic_... 1... 15'b000000000000000 15'b100101000... 15'b110110101... 15'b000000001... 15'b100000100... 15'b001101111... 15'b101010100...
unfolding... 1... 15'b000000000000000 15'b100101000... 15'b110110101... 15'b000000001... 15'b100000100... 15'b001101111... 15'b101010100...
period 1... 10000000
duty_cycle 0.5 0.5
offset 0 0

Figure 9. Simulated Waveform of Two Kinds of BCH Encoder

Table 2. Area Summary of 8-parallel Encoders after Synthesis

	Conventional 8 unfolding method	Conventional 8 parallel systolic	Proposed 8 parallel tree type systolic
No. of Slices	27	27	28
No. of Slice Flip Flops	15	15	15
No. of 4 input LUTs :	50	50	52
No. of bounded IOBs:	24	26	26
No. of GCLK	1	1	1

Table 3. Performance of 8-parallel Encoders after Synthesis

	Conventional 8 unfolding method	Conventional 8 parallel systolic	Proposed 8 parallel tree type systolic
Critical path delay	2.078ns	2.097ns	1.497ns
Throughputs (Mb/s)	3670	3639	5096

The Tables 2 and Table 3 show the comparison results of the *J* unfolding method, systolic array and the proposed tree-type systolic array BCH encoders with their device utilization and the critical path delay respectively. The *J* unfolding method has been implemented without any change in the generator polynomial, *i.e.*, no retiming is applied. Some XOR gates in the proposed tree-type encoder are increased. The area is still efficiency (about 0% of the allocated area of the XC2VP30 device). In Table 2, the number of slices is slightly increased, which is negligible for the modern FPGA. The maximum frequency operation of the proposed tree-type encoder is around 657.9 MHz.

From Table 3, the critical path of the original systolic array BCH encoder is 2.097ns. Using tree-type systolic array encoder, the critical path is reduced to 1.49ns from VHDL simulation. The throughput in the proposed encoder has speeded up by 40 % than the original systolic array encoder.

4. Conclusions

The tree-type systolic array BCH encoder has been proposed. The proposed eight-parallel BCH encoder has been compared with the unfolding method. The result shows that the proposed one has a better result in its throughput than its counterpart with a minor increase in area. Considerably, the proposed BCH encoder has a great flexibility to any parallelization factor without any complexity. The fan-out effect has been disregarded in our experiment. Retiming can be applied in the proposed encoder without any modification in the generator polynomial and increasing its hardware. Future work can be directed toward reducing the fan-out effect in the proposed tree-type systolic array BCH encoder to amend its output.

References

[1] R. Martin, W. Jone and S. Das, “Fault detection and diagnosis for multi-level cell flash memories”, Proceedings of IMTC 2006, 1896-1901, Sorrento, Italia, **(2006)** April 24-27.
[2] M. Y. Rhee, “Error Correcting Coding Theory”, McGraw-Hill, **(1989)**.
[3] T. B. Pei and C. Zukowski, “High-speed parallel CRC circuits in VLSI”, IEEE Trans. on Communications, vol. 4, no. 40, **(1992)**, pp. 653-657.
[4] K. Parhi, “Eliminating the Fanout bottleneck in parallel long BCH encoders”, IEEE Trans. on Circuits and Systems I:Regular Papers, vol. 3, no. 51, **(2004)**, pp. 512-516.
[5] F. Liang and L. Pan, “A CRT-based BCH encoding and FPGA implementation”, Proc. of Int’l Conf. on Information Science and Application, Seoul, Korea, **(2010)** April 21-23.
[6] H. Wallace, “Error Detection and Correction using the BCH code”, UNDUH, **(2001)**.
[7] Z. Jun, W. Z. Gong, H. Q. Sheng and X. Jie, “Optimized design for high-speed parallel BCH encoder”, Proc. of IEEE Int’l Workshop, **(2005)**, pp. 97-100.

Authors

Je-Hoon Lee received the B.S. and M.S. degrees in Computer and Communication Engineering from Chungbuk National University, Cheongju, Korea in 1999 and 2001, respectively. He received Ph. D in Computer and Communication Engineering from Chungbuk National University in 2005. From 2005 to 2006, he was a visiting scholar at Univ. of Southern California, USA and from 2007 to 2008, he was a visiting scholar at Murdoch University, Australia. From 2006 to 2009, he was an assistant Professor in Chungbuk National University. He is currently an assistant Professor in Div. of Electronics, Information and Communication Engineering, Kangwon National University, Korea. His research interests in the high-speed low-power digital circuits and systems design.

Sharad Shakya received the B.S. and M.S. degrees in Electronic Engineering from Kangwon National University, Cheongju, Korea in 2010 and 2013, respectively. His research interests in the high-speed digital circuit design and embedded systems.