ISSN NO: 2347-4890
International Journal of software & Hardware Research in Engineering
Volume 1 Issue 2, October 2013

An Efficiency of Eliminating the Fan out Bottleneck in Parallel Long BCH Encoders using VHDL

P.Sandhaya,M.TECH student,Department of ECE,MLECollege,singarayakonda,Prakasam(DT),India, sandhya.pandilla@gmail.com
S.Madhava Rao,Assoc.professor,Department of ECE,MLECollege,singarayakonda,Prakasam(DT),India, smadhav.r2@gmail.com

Abstract—Long BCH codes can achieve about 0.6-dB additional coding gain over Reed–Solomon codes with similar code rate in long-haul optical communication systems. BCH encoders are conventionally implemented by a linear feedback shift register architecture. Encoders of long BCH codes may suffer from the effect of large fanout, which may reduce the achievable clock speed. The data rate requirements of optical applications require parallel implementations of the BCH encoders. In this paper, a novel scheme based on look-ahead computation have been designed and implemented for (15,k) BCH Encoder on FPGA using VHDL to eliminate the effect of large fanout in parallel long BCH encoders. Also a comparative performance between original BCH encoder architecture and the modified architecture have been presented based synthesis and simulation on Xilinx Spartan 3 FPGA using VHDL.

I. INTRODUCTION

BCH codes are among the most widely used error-correcting codes. Compared to Reed–Solomon codes, BCH codes can achieve around additional 0.6 dB coding gain over the additive white Guassian noise (AWGN) channel with similar rate and codeword length. High-rate Reed–Solomon codes of length longer or equal to 255 have wide applications, such as in long-haul optical communication systems used in International Telecommunication Union-Telecommunication Standardization Sector (ITU-T) G.975, magnetic recording systems and digital communications. Hence, long BCH codes are of great interest. The encoders of BCH codes are conventionally implemented by a linear feedback shift register (LFSR) architecture. While such an architecture is simple and can run at very high frequency, it suffers from the serial-in and serial-out limitation.

In optical communication systems, where throughput of over 1 Gbps is usually desired, the clock frequency of such LFSR-based encoders cannot keep up with the data transmission rate, and thus parallel processing must be employed. Meanwhile, long BCH encoders face another problem. Due to the large number of nonzero coefficients in the generator polynomial, the effect of large fanout on the gate delay is no longer negligible. The delay of a single XOR gate grows linearly with the fanout. When the BCH code has a generator polynomial with a large number of nonzero coefficients, there will be some XOR gates with a large fanout in the LFSR architecture, which can slow down the encoder significantly. In this paper, a novel scheme based on look-ahead computation is proposed to eliminate the effect of large fanout in the parallel BCH encoders. Unfolding technique [4] is applied to achieve parallel processing.

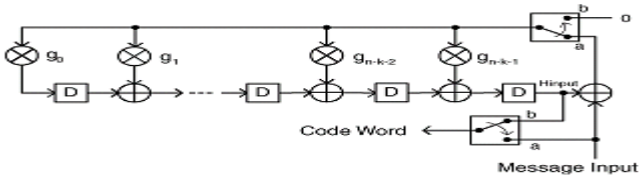
In recent years there has been an increasing demand for digital transmission and storage system and it has been accelerated by the rapid development and availability of VLSI technology and digital processing. Programmable Logic Device (PLD) and Field Programmable Gate Arrays (FPGAs) [1], [2] has revolutionised hardware design and its implementation advantages provides various solution like FPGA is fully reprogrammable and reconfigurable. Here implementation of encoder for (15, k) BCH code organized by LFSR using VHDL [6], [7] on FPGA is presented and also performance compared based on synthesis and simulation result to understand the device utilization and timing simulation by targeting on Xilinx Spartan 3S 1000 FPGA and XSA 3S1000 Board of Xess Corporation [3]. For simulation and synthesis Xilinx ISE 10.1 is used.

The structure of this paper is as follows. Section II contains a brief description of the LFSR-based BCH encoder architecture. In Section III, a novel parallel encoder architecture, which eliminates the effect of large fanout, is explained in detail. The performance analysis of an example based on simulation results is described in Section IV. Section V provides conclusions.

II. BCH ENCODER ARCHITECTURE

An (n, k) binary BCH code encodes a k-bit message block into an n-bit codeword. Considering a k-bit message $(m_{k-1}, m_{k-2}, \ldots, m_0)$ $(m_i \in GF(2), 0 \le i \le k-1)$ as the coefficients of a degree $k-1$ message polynomial $m(x)$, and the corresponding n-bit codeword $(c_{n-1}, c_{n-2}, \ldots, c_0)$ $(c_i \in GF(2), 0 \le i \le n-1)$ as the coefficients of a degree $n-1$ codeword polynomial $c(x)$, the encoding of BCH codes can be performed as $c(x) = m(x)g(x)$, where the degree $n-k$ polynomial $g(x)$ is the generator polynomial of the (n, k) BCH code. However, systematic encoding is usually desired, since in this case the message bits can be read from the codeword directly. The systematic encoding can be implemented by:

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



ISSN NO: 2347-4890
International Journal of software & Hardware Research in Engineering
Volume 1 Issue 2, October 2013

Fig. 1. Serial BCH encoder Architecture

where $Rem(a(x))_{b(x)}$ denotes the remainder polynomial of dividing $a(x)$ by $b(x)$. Assuming $g(x)$ can be expressed as $g(x) = g_{n-k}x^{n-k} + g_{n-k-1}x^{n-k-1} + \ldots + g_0$ ($g_i \in GF(2)$, $0 \le i \le n-k$ and $g_0 = g_{n-k} = 1$), a systematic binary (n, k) BCH encoder can be implemented by the architecture illustrated in Fig. 1. For binary BCH codes, each multiplier in Fig. 1 can be replaced by connection or no connection when gi ($0 \le i \le n-k$) is '1' or '0', respectively. Using the encoder architecture in Fig. 1, the whole encoding process takes n clock cycles. During the first k clock cycles, the two switches are connected to the 'a' port, and the k message bits are input to the LFSR serially with most significant bit (MSB) first. Meanwhile, the message bits are also sent to the codeword output to form the systematic part of the codeword. After the k^{th} clock cycle, the $n-k$ delay elements contain $Rem(m(x)\cdot x^{n-k})_{g(x)}$. At this time, the switches are moved to the 'b' port, and the remainder bits are shifted out of the delay elements to the codeword output bit by bit to form the remaining bits of the systematic codeword. As can be observed from Fig. 1, the critical path of this architecture consists of two XOR gates and the output of the right-most XOR gate is the input to all the other XOR gates. In the case of long BCH codes, this architecture may suffer from the large fanout effect on the right-most XOR gate, which can slow down the encoder significantly. In addition, parallel processing needs to be employed when the encoder cannot run as fast as the application requirements. Fanout bottleneck also exists in parallel long BCH encoders.

III. PARALLEL BCH ENCODER WITH ELIMINATED FANOUT BOTTLENECK

Retiming can always be applied to eliminate the effect of large fanout in serial long BCH encoders. To make notations simple, we refer to the input to the right-most XOR gate in Fig. 1, which is the delayed output of the second XOR gate from the right, as 'Hinput'[8]. Since there is at least one delay element at the Hinput of the right-most XOR gate, and delay elements can be added to the message input, retiming can always be performed

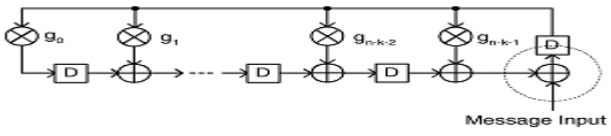


Fig 2. (a) An LFSR example

along the dotted cutset as shown in Fig. 1 by removing one delay element from each input of the right-most XOR gate and adding one to the output.

Parallel BCH encoder architectures can be derived by unfolding[4]. However, if unfolding is applied directly to Fig. 1, retiming may not be applied in an obvious way to eliminate the large fanout effect in the parallel architecture. The original architecture can be expressed by data flow graphs (DFG), where each of the gates are expressed as nodes, and are connected by paths with delay elements. The J-unfolded architecture makes J copies of each node in the original architecture. However, the total number of delay elements does not change. Assuming node U is connected to node V by a path with w delay elements in the original architecture, in the J-unfolded architecture, the node U_i is connected to $V_{(i+w)\%J}$ by a path with $\lfloor (i+w)/J \rfloor$ delay elements, where U_i, V_j ($0 \le i, j < J$) are copies of the node U and V, respectively, and $(i+w)\%J$ is the remainder of dividing $i+w$ by J [4]. It can be derived that if the number of delay elements, w, in a path is less than the unfolding factor J, there will be w corresponding paths with one delay element and the rest $J-w$ paths without any delay elements in the J-unfolded architecture. Assuming the generator polynomial $g(x)$ of a BCH code can be rewritten as:

$$g(x) = x^{t0} + x^{t1} + \cdots + x^{ts-2} + 1, \quad (2)$$

where $t_0, t_1, \cdots, t_{s-2}$ are positive integers with $t_0 > t_1 > \cdots t_{s-2}$ and s is the total number of non-zero coefficients in $g(x)$, then there are $t_0 - t_1$ consecutive delay elements at the Hinput of the right-most XOR gate in Fig. 1. In the case of $t_0 - t_1 < J$, there will be at least one input path to the copies of the right-most XOR gate without any

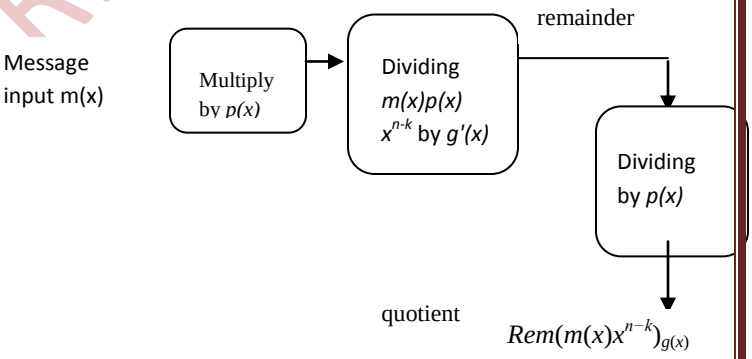


Fig 3. Block diagram of modified BCH encoding

.delay elements in the J-unfolded architecture. As a result, retiming cannot be applied directly to eliminate the large fanout effect. For example, Fig. 2(a) shows the LFSR for a serial (15, 11) BCH encoder with generator polynomial $g(x) = x^4 + x + 1$. In this example, there are three delay elements at the Hinput of the right-most XOR gate. Applying 4-unfolding, the architecture in Fig. 2(b) can be derived. As can be observed, there are 4 copies of the right-most XOR gate, and 3 of them have one delay element at the Hinput while the other 4-3=1 does not have any delay elements at the Hinput. Accordingly, retiming cannot be applied around the last XOR gate to eliminate the fanout bottleneck.

In the case of $t_0 - t_1 < J$, the scheme proposed in [5] can be employed to modify the generator polynomial to enable retiming of the right-most XOR gates in the J-unfolded LFSR architecture. $m(x)x^{n-k}$ can be written as:

ISSN NO: 2347-4890
International Journal of software & Hardware Research in Engineering
Volume 1 Issue 2, October 2013

$$m(x)x^{n-k} = q(x)g(x) + r(x), \quad (3)$$

where $q(x)$ and $r(x)$ are the quotient polynomial and the remainder polynomial, respectively. Multiplying a polynomial $p(x)$ to both sides of (3), we can get:

$$m(x)p(x)x^{n-k} = q(x)(g(x)p(x)) + r(x)p(x). \quad (4) \qquad 1$$

Since $r(x)$ is the remainder polynomial of the division by $g(x)$, the degree of $r(x)$ is less than that of $g(x)$. It follows that $deg(r(x)p(x)) < deg(g(x)p(x))$, where $deg(f(x))$ denotes the degree of $f(x)$. Hence, $r(x)p(x)$ can be considered as the remainder of dividing $m(x)p(x)x^{n-k}$ by $g(x)p(x)$. According to (4), the proposed architecture in [5] computes $r(x) = Rem(m(x)x^{n-k})_{g(x)}$ by a three-step process:

1) multiply $m(x)$ by $p(x)$;

2) compute the remainder of dividing $m(x)p(x)x^{n-k}$ by $g(x)p(x)$;

3) compute the quotient of dividing the remainder from step 2 by $p(x)$.

The quotient computed from the third step is $r(x)$. The first step can be implemented by an architecture similar to that of an FIR filter, while the other two steps can be implemented by LFSR architectures. The modified generator polynomial, $g'(x) = p(x)g(x)$, can be written as:

$$g'(x) = x^{t'0} + x^{t'1} + \cdots + x^{t's-2} + 1, \quad (5)$$

where $t'_0, t'_1, \cdots, t'_{s-2}$ are positive integers with $t'_0 > t'_1 > \cdots > t'_{s-2}$ and s is the total number of non-zero coefficients in $g'(x)$. In the case of $t_0 - t_1 < J$ in (2), a $p(x)$ can be found to ensure $t'_0 - t'_1 \geq J$. Accordingly, retiming can be applied to eliminate the effect of large fanout in the J unfolded LFSR implementing the division by $g'(x)$. Such a $p(x)$ can be computed by clustered look-ahead computation Algorithm [4]. In Algorithm A, E is set to the desired $t'_0 - t'_1$ in $g'(x)$, and $g'(x)$ has a-b ≥ E consecutive zero coefficients after the highest power term at the end of the algorithm.

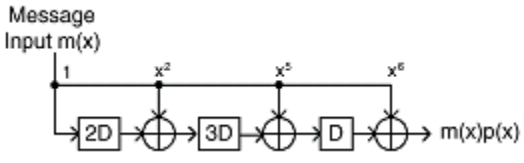


Fig 4.Step 1 of modified BCH encoding.

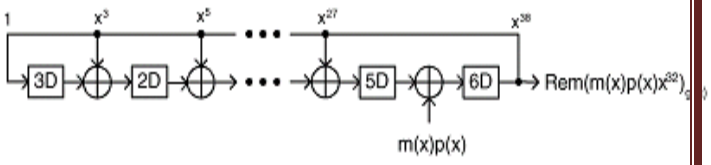


Fig 5.Step 2 of modified BCH encoding.

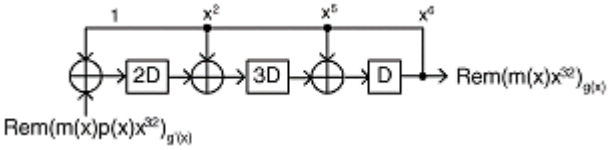


Fig 6. Step 3 of modified BCH encoding.

Algorithm A:
Set a = t_0; b = t_1;
Set $\tilde{p}(x) = 1$; $\tilde{g}(x) = g(x)$;
loop: while a-b < E
{
$\tilde{g}(x) = \tilde{g}(x) + g(x).x^{b-a}$;
$\tilde{p}(x) = \tilde{p}(x) + x^{b-a}$;
num=a-b;
b = highest power in ($\tilde{g}(x) - x^a$);
}
final step:
$p(x) = \tilde{p}(x).x^{num}$;
$g'(x) = \tilde{g}(x).x^{num}$;

Algorithm A is applied to transform the encoder computation of a BCH (15,11) code in the example below.

Example I: Given an BCH (15, 11) code using generator polynomial $g(x)=x^4+x+1$, we want to find $p(x)$ such that $t'_0 - t'_1 \geq 9$ in $g'(x)$ .In this example, E should be set to 9, and a = 4, b=1 at the beginning of Algorithm A. The intermediate values after each iteration in Algorithm A are given below.
After iteration I:
$\tilde{p}(x) = 1+x^{-3}$;
$\tilde{g}(x) = 1+x^4+x^{-2}+x^{-3}$;
num=3; b=0; a-b=4<9; continue.
After iteration II:
$\tilde{p}(x) = 1+x^{-3}+x^{-4}$;
$\tilde{g}(x) = x^4+x^{-2}+x^{-4}$;
num=4; b=-2; a-b=6<9; continue.
After iteration III:
$\tilde{p}(x) = 1+x^{-3}+x^{-4}+x^{-6}$;
$\tilde{g}(x) = x^4+x^{-4}+x^{-5}+x^{-6}$;

ISSN NO: 2347-4890
International Journal of software & Hardware Research in Engineering
Volume 1 Issue 2, October 2013

num=6; b=-4; a-b=8<9; continue.
After iteration IV:
$p\tilde{}(x) = 1+x^{-3}+x^{-4}+x^{-6}+x^{-8}$;
$g\tilde{}(x) = x^{4}+x^{-5}+x^{-6}+x^{-7}+x^{-8}$;
num=8; b=-5; a-b= 9 ; Stop.
Final step:
$p(x) = p\tilde{}(x).x^{num} = p\tilde{}(x).x^{8} = x^{8}+x^{5}+x^{4}+x^{2}+1$
$g'(x) = g\tilde{}(x).x^{num} = g\tilde{}(x).x^{8} = x^{12}+x^{3}+x^{2}+x+1.$

According to (4), the modified method of finding *Rem* $(m(x)p(x)x^{n-k})_{g'(x)}$ in the BCH encoding can be implemented by the steps illustrated in Fig. 3. Each step is explained using the *p(x), g(x)* and *g'(x)* derived in Example I in the remainder of this section. The first step in Fig. 3 is to multiply the message input polynomial by *p(x)*. This can be implemented by adding delayed message inputs according to the coefficient of *p(x)*. For example, using the *p(x)* derived in Example I, this step can be implemented by the diagram in Fig. 4. The four taps correspond to 1, x^2,x^4, x^5 and x^8 respectively, as shown in Fig. 4.

After *m(x)p(x)* is computed, it is fed into the second block to compute $Rem(m(x)p(x)x^{n-k})_{g'(x)}$ by using similar LFSR architectures as that in Fig. 1. However, since $deg(g'(x))>n-k$, the product of *p(x)* and *m(x)* should be added to the output of the $(n-k)^{th}$ register from left, instead of being added to the output of the right-most register. The addition of *m(x)p(x)* can break the consecutive a-b registers at the H input of the right-most XOR gate in the LFSR. The implementation of the second step using the BCH code in Example 1 is illustrated in Fig. 5.

As could be observed in Fig. 5, there are 12-3=9 consecutive registers at the H input of the right-most XOR gate according to *g'(x)* . However, after adding *m(x)p(x)* to the output of the 4^{th} register, only 8 consecutive registers are left. Therefore, at most 8-unfolding can be applied to Fig. 5 without suffering from large fan out problem. At the end of algorithm A, $deg(g'(x))=num+deg(g(x))=n-k+num$. Hence, only *num* consecutive registers left after adding *m(x)p(x)* , where $num \le E-1$ at the end of Algorithm A. Therefore, E is usually set to larger than the desired unfolding factor J to ensure $num \ge J$ at the end of Algorithm A. Alternatively, at the expense of a slight increase in the critical path and latency, the delays at the input of the last XOR gate can be retimed and moved to the output of this XOR gate.

In the third step, $Rem(m(x)p(x)x^{n-k})_{g'(x)}$ needs to be divided by *p(x)* to get the final result. Similar architectures as that in Fig. 1 can also be used, except that the input data is added to the input of the left-most register, since the input polynomial does not need to be multiplied by any power of *x* . For example, the third step of the modified BCH encoding using the *p(x)* derived in Example I is illustrated in Fig. 6. Unfolding the modified BCH encoder in Fig. 3 by factor J, a parallel architecture capable of processing J message bits at a time is derived.

IV. Performance Analysis of an Example

In the J-unfolded block of computing *p(x)m(x)*, feedback loop does not exist. Thus, it can be pipelined to achieve desired clock frequency. In the second block, since the LFSR of the modified generator polynomial *g'(x)* has at least J registers at the H input of the right-most XOR gate, retiming can be applied to the J-unfolded architecture to eliminate the effect of large fanout after adding J registers to the output of *m(x)p(x).*

To observe the Fan-out and resource utilization, RTL is generated verified and synthesized. The proposed BCH encoder has been implemented on Spartan3 XC3S1000 target device by using Xilinx ISE 10.1

Design of Serial (15, 11) BCH Encoder

Encoder for (15, 11)BCH code is designed by organizing LFSR with generated polynomial $1+x+x^4$ and implemented on Spartan 3S1000 FPGA of Xilinx. The RTL Schematic is generated by synthesis with Xilinx ISE 10.1, shown in Fig 7. The timing simulation is shown in Fig. 8. Total of 15 clock cycle is taking to complete transmitting of 15 codeword, 11-bits are information bit and 4- bits are parity bit. 11 Information bits "01011001001" are transmitting as it is where as other 4 bits "1010" are transmitting as parity bit "1100".

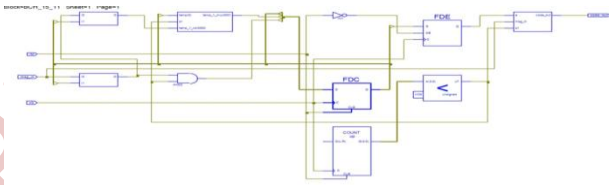


Fig.7 RTL Schematic for Serial (15,11) BCH encoder

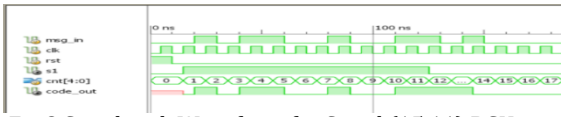
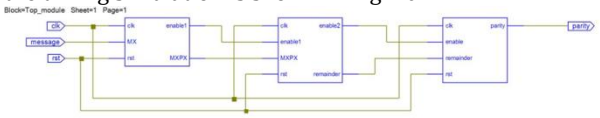


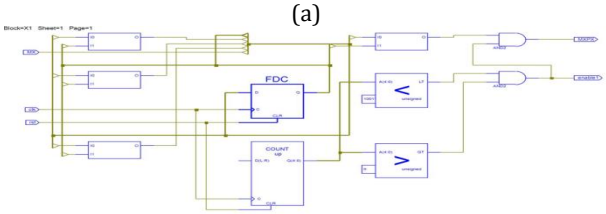
Fig.8 Simulated Waveform for Serial (15,11) BCH encoder

Design of parallel (15, 11) BCH Encoder

Encoder for parallel (15, 11)BCH code is designed by organizing LFSR with the modified BCH architecture and modified generated polynomial $1+x+x^2+x^3+x^{12}$ is implemented on Spartan 3S1000 FPGA of Xilinx. The RTL Schematic is generated by synthesis with Xilinx ISE 10.1, shown in Fig 9 and the timing simulation is shown in Fig. 10.

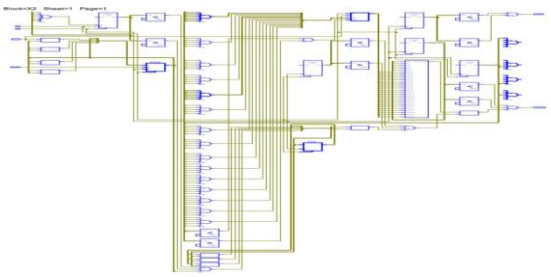


(a)

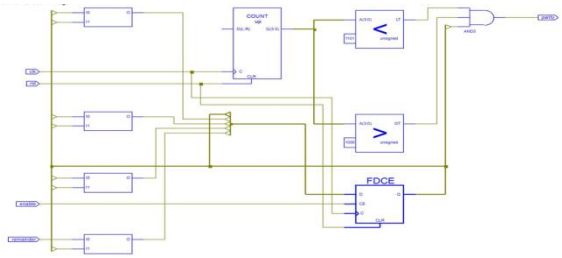


(b)

ISSN NO: 2347-4890
International Journal of software & Hardware Research in Engineering
Volume 1 Issue 2, October 2013



(c)



(d)

Fig 9. RTL Schematic at each step of Parallel BCH encoder (a) RTL Schematic of the modified BCH architecture (b) RTL Schematic of Step 1 (c) RTL Schematic of Step 2 (d) RTL Schematic step3.

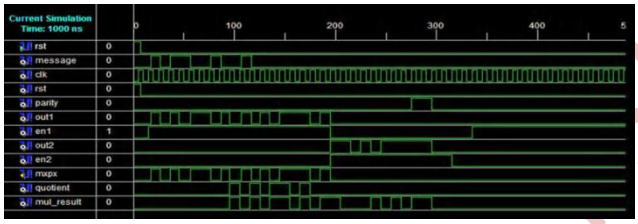


Fig 10 Simulated Waveform for parallel (15,11) BCH encoder

Table 1. Device Utilization and Timing Summary

Component Utilization/ time	Original BCH encoder	Modified BCH encoder
No of slices		
No of Slices FF		
4 input LTUs		
Fanout		

Synthesis report for the target device Xilinx Spartan FPGA

V. Conclusions

A novel parallel implementation of long BCH encoders has been proposed in this paper. The proposed encoder does not suffer from the effect of large fan out. This scheme can also be used in cyclic redundancy checking to reduce fan-out bottleneck. After the fan-out effect has been taken care of, further speedup can be achieved by reducing critical path. The critical path usually lies in the part of dividing by $p(x)$. Future work will be directed toward reducing the critical path of this part or make this part pipelinable after further algorithmic modifications. In addition, reducing the iteration bound of the LFSR implementing the division by $g'(x)$ is of great interest to achieve further speedup for parallel long BCH encoders.

References

[1] J J.Rose S.D. *Brown, R.J. Francis – "Field Programmable Gate Arrays*", Kluwer Academic Publishers, 1992

[2] Brown S., Vranesic Z "Fundamental of Digital Logic Design with VHDL" McGraw Hill, 2nd Edition. [3] Xess Corp.. XSA-3S1000 Board V1.1 User Manual. Available:http://xess.com/manuals/xsa-3S-manual-v1_1.pdf. Sept 2007.

[4] K. K. Parhi, "VLSI Digital Signal Processing Systems-Design and Implementation," John Wiley & Sons, 1999.

[5] K. K. Parhi, "Eliminating the Fanout Bottleneck in Parallel Long BCH Encoders," *IEEE. Trans. on Circuits and Systems-Part I: Regular Papers*, vol.51, No.3, Mar.2004.

[6] P. J. Ashenden, *The VHDL Cookbook*, 1st ed. Dept. Computer Science, University of Adelaide, South Australia: University of Adelaide, 1990.

[7] Bhasker J, "*A VHDL Primer*", P T R Prentice Hall, Pages 1-2, 4-13, 28-30.

[8]. Xinmiao Zhang and Keshab K. Parhi, " High-speed Architectures for Parallel Long BCH Encoders," *GLSVLSI'04,* April 26–28, 2004, Boston, Massachusetts, USA.

ISSN NO: 2347-4890
International Journal of software & Hardware Research in Engineering
Volume 1 Issue 2, October 2013

