International Journal of Computer Trends and Technology (IJCTT) – volume 4 Issue 9– Sep 2013

An Optimization Technique for CRC Generation

Author
K.V.Krishna Reddy
Dept. of ECE, Sreenidhi institute of Science and Technology (SNIST) Hyderabad AP, India.

Abstract: ***In networking environments, the cyclic redundancy check (CRC) is widely utilized to determine whether errors have been introduced during transmissions over physical links. In this paper, we present a fast cyclic redundancy check (CRC) algorithm that performs CRC computation for an arbitrary length of message in parallel. This paper proposes 64 bits parallel CRC architecture based on F matrix with order of generator polynomial is 32 and showed CRC-64 is having less latency and high throughput compared to CRC-32 parallel architecture through Xilinx Simulator***.

Index Terms—Keywords: CRC, lookup table, Fast update

I.INTRODUCTION

Cyclic Redundancy Checking is one of the most frequently used techniques for detecting transmission errors. One of the CRC techniques utilized in networking is the CRC-32 algorithm employed by Ethernet. The Cyclic Redundancy Check (CRC) is an error detection technique that is widely utilized in digital data communication and other fields such as data storage, data compression, and etc. There are many CRC algorithms, each of which has a predetermined generator polynomial G(x) that is utilized to generate the CRC code. F or example, in TCP/IP protocol suite, the most frequently utilized CRC algorithm is the CRC-32 algorithm employed by Ethernet, which has the following generator polynomial:

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + x^0$$

where m = 32 is the highest order or called the degree of the generator polynomial and also the length of the CRC code. We can extract the coefficient of G(x) and represent it in binary form as: P = p_{32}; p_{31};::: ; p_1; p_0g = [10000010011000001000111010110111]; which has m + 1 = 33 bits. The most significant bit of P, P_{32}, corresponds to the coefficient of x^{32}, the highest order of G(x).

Similarly, p_{31} corresponds to the coefficient of x^{31}, which is 0 in this case, and the other bits follow the coefficients at their corresponding positions. P is called the **generator**, and uniquely coincides with the generator polynomial.CRC calculation can be performed in hardware and software. The general hardware solution for CRC calculation is linear feedback shift register (LFSR), in which simple serial bit architecture is used for encoding and decoding the message.

When CRC technique is applied, a CRC code is appended to the end of the data message during transmission. Assume that the data message is represented by D, which may have hundreds of bits and the CRC code is denoted by C with the length m, the degree of the generator polynomial. Accordingly, the transmitted data unit with CRC code can be denoted by T = fDCg = D 2^m + C.

The CRC code C is generated so that T is an exact multiple of generator P. Therefore, if T is transmitted and there is no error during transmission, the received message T must also be an exact multiple of the same P. Otherwise, a transmission error must have occurred.

A Serial Implementation of CRC

In hardware implementations, the CRC calculation (modulo 2 divisions) can be easily performed by logical combinations of shift registers and XOR gates. The Linear Feedback Shift Register (LFSR) is a common approach designed to accomplish the serial calculation of CRC in hardware.

Figure 1 illustrates the basic architecture of LFSR for serial calculation of CRC. The inputs- outputs in the figure are shift registers which store the remainder after every subtraction. The number of shift registers equals m, the degree of the generator polynomial

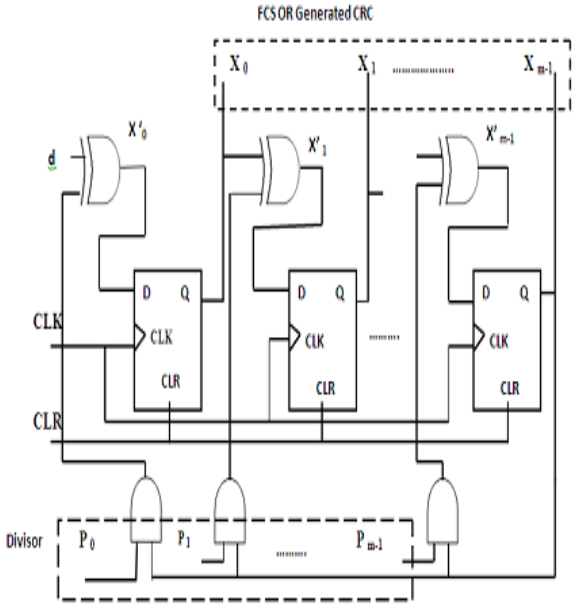


Figure 1. Basic LFSR Architecture

As shown in fig.1 is serial data input, X is present state (generated CRC), X' is next state and p is generator polynomial. Working of basic LFSR architecture is expressed in terms of following equations.

$$\left.\begin{aligned} X_0' &= (P_0 \otimes X_{m-1}) \oplus d \\ X_i' &= (P_0 \otimes X_{m-1}) \oplus X_{i-1} \end{aligned}\right\}$$

Frame Check sequence (FCS) will be generated after (k+m) cycle, where k indicates number of data bit and m indicates the order of generator polynomial. For 32 bits serial CRC if order of generator polynomial is 32 then serial CRC will be generated after 64 cycles.

B. Parallel CRC

There are different techniques for parallel CRC generation given as follow.

1. A Table-Based Algorithm for Pipelined CRC Calculation.
2. Fast CRC Update
3. F matrix based parallel CRC generation.
4. Unfolding, Retiming and pipelining Algorithm

The pipelined architecture in Fig.2 has five blocks as input; four of them are used to read four new blocks from the message in each iteration.

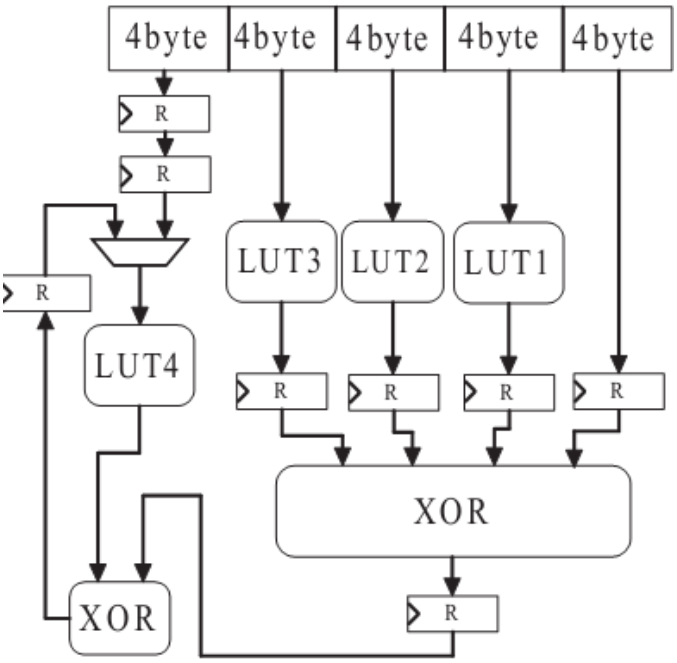


Fig. 2 Pipelined CRC architecture

They are converted into CRC using lookup tables: LUT3, LUT2, and LUT1.LUT3 contain CRC values for the input followed by 12 bytes of zeros, LUT2 8 bytes, and LUT1 4 bytes. Note that the rightmost block does not need any lookup table. It is because this architecture assumes CRC-32, the most popular CRC, and 4-byte blocks. If the length of a binary string is smaller than the degree of the CRC generator, its CRC value is the string itself. Since the rightmost block corresponds to A4, it does not have any following zero and thus its CRC is the block itself. The results are combined using XOR, and then it is combined with the output of LUT4, the CRC of the value from the previous iteration with 16 bytes of zeros concatenated. In order to shorten the critical path, we introduce another stage called the pre-XOR stage right before the four-input XOR gate.

In fast CRC update technique we don't required to calculate CRC each time for all the data bits, instead of that calculating CRC for only those bits that are change.

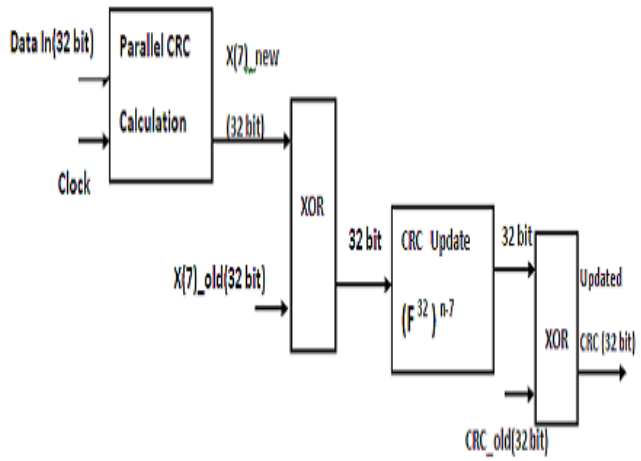


Figure .3 Fast CRC update architecture

Algorithm and Parallel architecture for CRC generation based on F matrix. Parallel data input and each element of F matrix, which is generated from given generator polynomial is anded, result of that will xoring with present state of CRC checksum. The final result generated after (k+ m) /w cycle.

C. Parallel architecture

When data are stored on or communicated through media that may introduce errors, some form of error detection or error detection and correction coding is usually employed. Mathematically, a CRC is computed for a fixed-length message by treating the message as a string of binary coefficients of a polynomial, which is divided by a generator polynomial, with the remainder of this division used as the CRC.

Fig.4.demonstrates an example of parallel CRC calculation with multiple input bits w = m = 4. The dividend is divided into three 4-bit fields, acting as the parallel input vectors D (0), D (1), D (2), respectively.

For our parallel CRC design, the serial computation demonstrated above should be rearranged into a parallel configuration.

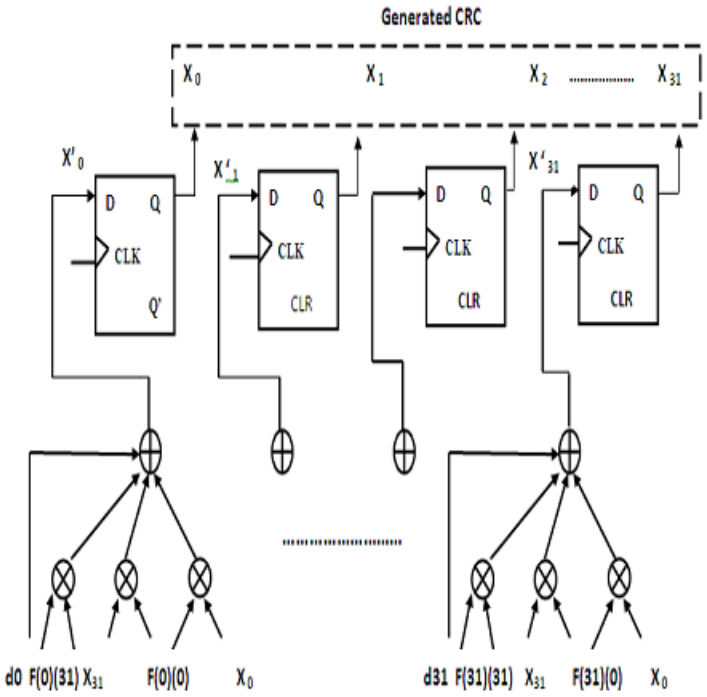


Figure .4 Parallel calculation of CRC-32 for 32bit

II. RELATED WORK

It is a simple circuit based on shift registers performs the CRC calculation by handling the message one bit at a time [1]. A typical serial CRC circuit using LFSRs is shown in Fig. 1. Fig. 1 illustrates one possible structure for CRC32.There are a total of 32 registers; the middle ones are left out for brevity. The combinational logic operation in the figure is the XOR operation. One data bit is shifted in at each clock pulse. This circuit operates in a fashion similar to manual long division. The XOR gates in Fig. 1 hold the coefficients of the divisor corresponding to the indicated powers of x. Although the shift register approach to computing CRCs is usually implemented in hardware, this algorithm can also be used in software when bit-by-bit processing is adequate.

Today's applications need faster processing speed and there has been much work on parallelizing CRC calculation. Cheng and Parhi discussed unfolding the serial implementation and combined it with pipelining and retiming algorithms to increase the speed [2]. The parallel long Bose-Chaudhuri-Hocquenghen (BCH) encoders are based on the multiplication and division operations on the

generator polynomial and they are efficient to speed up the parallel linear feedback shift register (LFSR) calculation [3], [4]. Unfortunately, the implementation cost is rather high because of the complexity of multiplication and division operations. Another approach to unroll the serial implementation was proposed by Campobello et al. [5] using linear systems theory. This algorithm is, however, based on the assumption that the packet size is a multiple of the CRC input size. Satran and Sheinwald proposed an incremental

algorithm in which CRC is computed on out-of-order fragments of a message [6].
In their algorithm, a CRC is computed incrementally and each arriving segment contributes its share to the message's CRC upon arrival, independently of other segments' arrivals, and can thus proceed immediately to the upper layer protocol.
A number of software-based algorithms have also been proposed [7]–[9], as well as FPGA-based approaches [10]–[11]. Simionescu proposed a scheme to calculate CRC in parallel [14] and the idea has been widely used. Walma designed a hardware-based approach [12], where he compared the area and throughput of pipelined and non-pipelined CRC designs and proposed a pipelined CRC calculation to increase the throughput.

III. PROPOSED SYSTEM:

To solve the problem of the parallel calculation of CRC, we first try to simulate the behaviour of a serial feedback shift register in Fig. 1 with a parallel finite state machine shown in Fig. 5
At every clock pulse, the parallel machine receives n input bits at a time and must return the updated CRC value. The length of the data may not be known in advance. In Fig.5 X represents the next n input bits of the message, and Y is the current CRC value, which is calculated with the data bits preceding X.

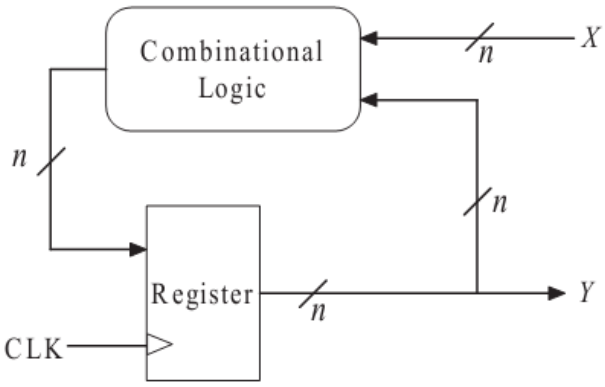


Fig.5. Parallel CRC structure

The combinational logic must produce the CRC value for the all data bits up to X, inclusively, from the new input bits X and the current CRC value Y. In other words, the output of the combinational logic is a function of X and Y only. If Y is the current content of the serial feedback shift register in Fig. 1 and X is the next n input bits, the output of the combinational logic must be the same as the content of the shift register after n clocks.
In proposed architecture w= 64 bits are parallely processed and order of generator polynomial is m= 32 as shown in fig. 3. As discussed previously, if 32 bits are processed parallely then CRC-32 will be generated after (k+m)/w cycles. If we increase number of bits to be processed parallely, number of cycles required to calculate CRC can be reduced. Proposed architecture can be realized by below equation.

$$\left.\begin{aligned} Xtemp &= F^{W} \otimes D(0to31) \oplus D(32to63) \\ X' &= F^{W} \otimes X \oplus Xtemp \end{aligned}\right\}$$

Where,
D (0 to 31) =first 32 bits of parallel data input D (0 to 63) = next 32 bits of parallel data input
X'=next state
X=present state

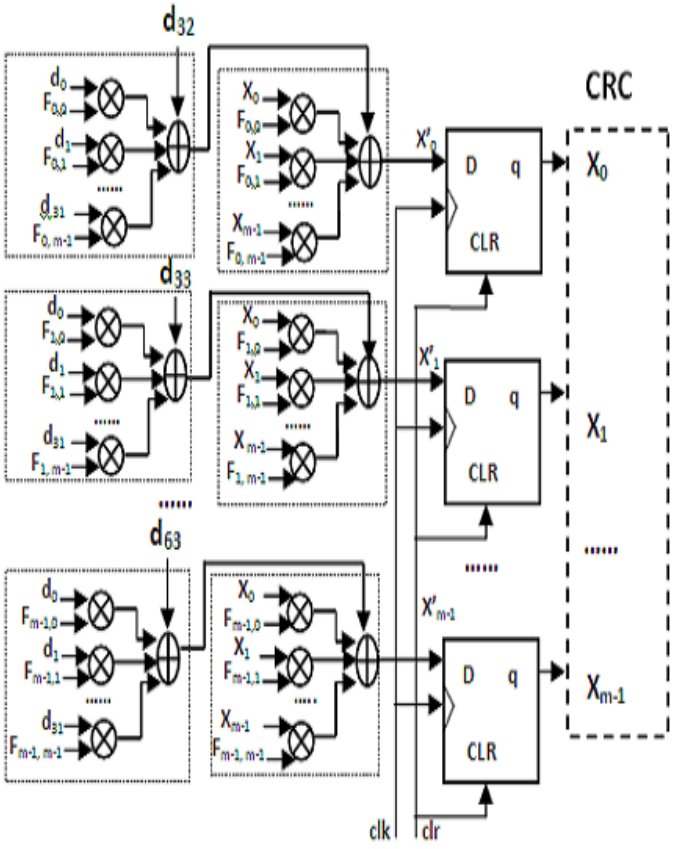
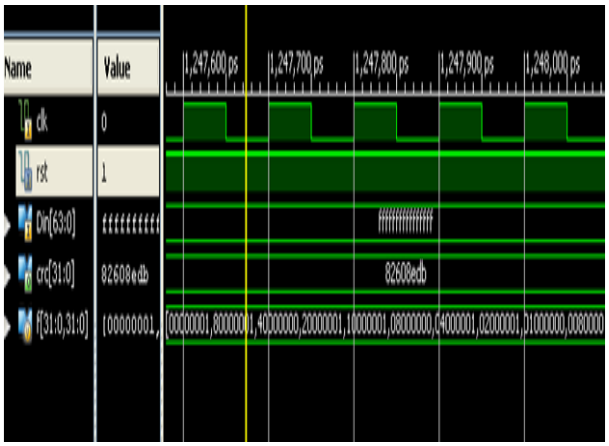


Figure 6. Block diagram of 64-bit parallel calculation of CRC-32.

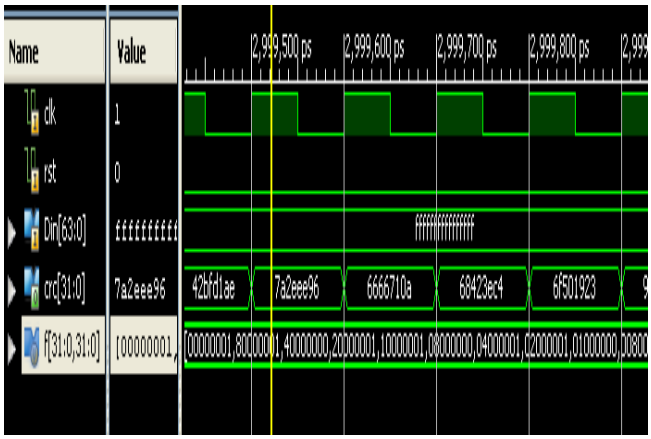
In proposed architecture di is the parallel input and F (i) (j) is the element of F32matrix located at ith row and jth column. As shown in figure 3 input data bits d0….d31anded with each row of FW matrix and result will be xored individually with d32, d33…….d63. Then each xored result is then xored with the X' (i) term of CRC32. Finally X will be the CRC generated after (k+m)/w cycle, where w=64.

IV.RESULT AND ANALYSIS

The proposed architecture is synthesized in Xilinx-9.2i and simulated in Xilinx ISE Simulator, which required half cycle then the previous 32bit design[1][5]. In our programming in Verilog by specifying only generator polynomial, it directly gives F matrix useful for parallel CRC generation that is not available in previous methods



CRC-64



CRC-64

The proposed CRC-32 architecture with 64bit parallel bit simulated in Xilinx 9.2i ISE simulator. Input data bit to system is FFFFFFFFFFFFFFFF (64 bit). The final result obtain after (k+ m)/w cycle for 32-bit residual will be 7A2EEE96 (hexadecimal form).

For fast CRC Calculation, Latency and Throughput plays a vital role to show the performance of system.

Latency: Latency is measure of time delay experienced in system. This Latency should be calculated and evaluated by

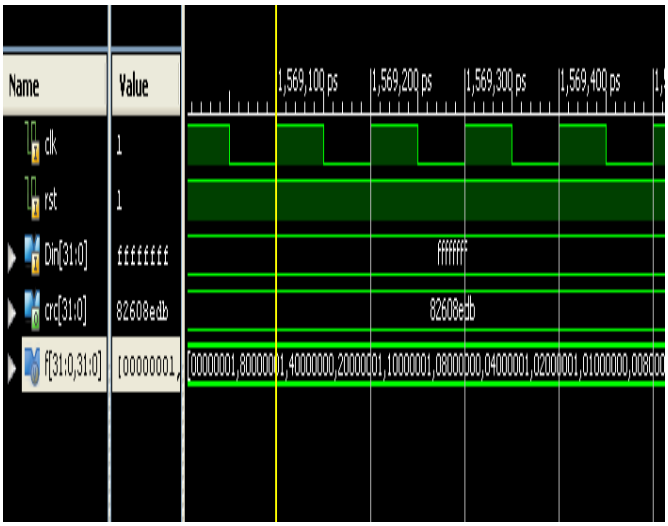
$$latency(s) \cong \frac{NC_{comp} + (M-1)C_{comb}}{F_{system}}$$

Where,
N = number of blocks/bytes.
M = number of divisions of the input data. Or
C comp = number of cycles to calculate the CRC.
C comb = number of cycles to calculate the XOR of all remainders.
F system = System clock frequency.

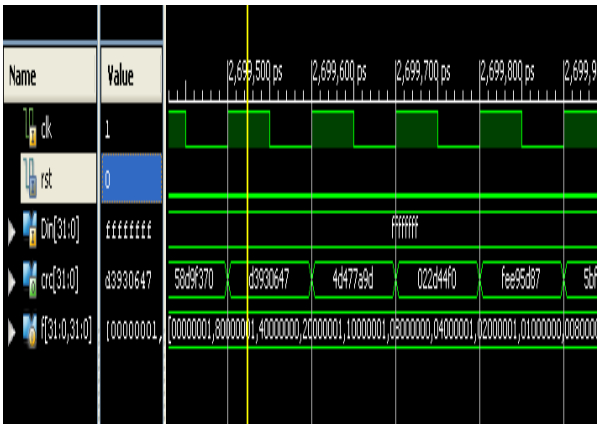
Throughput: Throughput is the average rate of successful message delivery over communication channel. Throughput is calculated using following equation

$$Throghput(bits/s) \cong F_{system} \frac{M+N+B}{NC_{comp} + (M-1)C_{comp}}$$

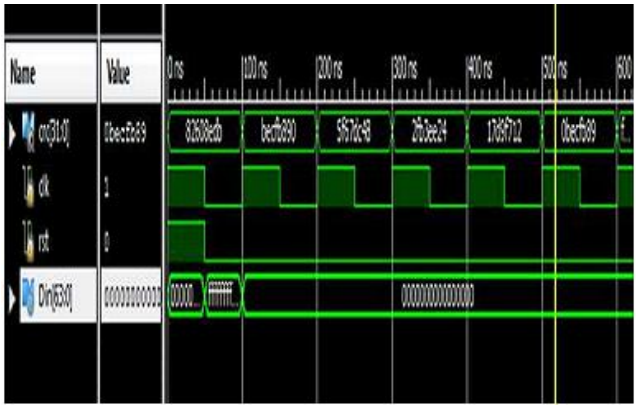
Latency and Throughput are calculated for CRC using CRC-32 and CRC-64 provides less latency and more throughput.



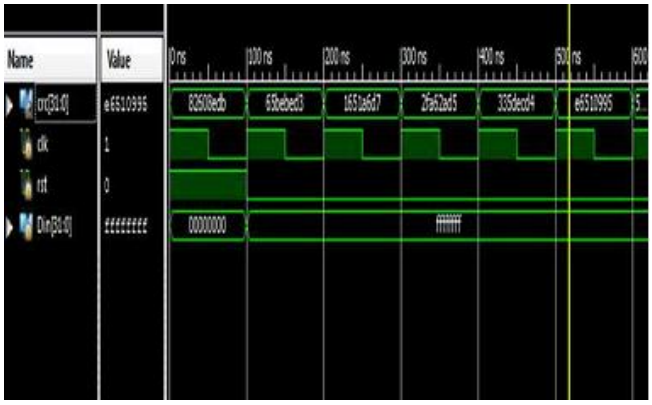
CRC-32



CRC-32



CRC-64



CRC-32

V.CONCLUSION

In this proposed paper 32bit parallel architecture required more clock cycles for 64 byte data. Proposed design (64bit) required only few cycles to generate CRC with same order of generator polynomial. So, it drastically reduces computation time to 30% and same time increases the throughput. Pre-calculation of F matrix is not required in proposed architecture. Hence, this is compact and easy method for fast CRC generation. CRC-64 provides less latency and more throughput.

VI. REFERENCES

[1] T. V. Ramabadran and S. S. Gaitonde, "A tutorial on CRC computations," IEEE Micro, vol. 8, no. 4, pp. 62–75, Aug. 1988.

[2] C. Cheng and K. K. Parhi, "High-speed parallel CRC implementation based on unfolding, pipelining, and retiming," IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 53, no. 10, pp. 1017–1021, Oct. 2006.

[3] X. Zhang and K. K. Parhi, "High-speed architectures for parallel long BCH encoders," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 13, no. 7, pp. 872–877, Jul. 2005.

[4] K. K. Parhi, "Eliminating the fanout bottleneck in parallel long BCH encoders," IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 51, no. 3, pp. 512–516, Jul. 2004.

[5] G. Campobello, G. Patane, and M. Russo, "Parallel CRC realization," IEEE Transactions on Computers, vol. 52, no. 10, pp. 1312–1319, Oct. 2003.

[6] J. Satran, D. Sheinwald, and I. Shimony, "Out of order incremental CRC computation," IEEE Transactions on Computers, vol. 54, no. 9, pp. 1178–1181, Sep. 2005.

[7] D. Feldmeier, "Fast software implementation of error detection codes," IEEE/ACM Transactions on Networking, vol. 3, no. 6, pp. 640–651, Dec. 1995.

[8] A. Simionescu, "CRC tool computing CRC in parallel for Ethernet," http://space.ednchina.com/upload/2008/8/27/ 300b83c-43ea-459b-ad5c-4dc377310024.pdf, 2001.

[9] M. E. Kounavis and F. L. Berry, "Novel table lookup-based algorithms for high-performance CRC generation," IEEE Transactions on Computers, vol. 57, no. 11, pp. 1550–1560, Nov. 2008.

[10] M. Braun, J. Friedrich, T. Grn, and J. Lembert, "Parallel CRC computation in FPGAs generation," Field-Programmable Logic Smart Applications, New Paradigms and Compilers, vol. 1142, pp. 156–165, 1996.

[11] R. Ahmad, O. Sidek, and S. Mohd, "Development of the CRC block for Zigbee standard on FPGA," in Proceedings of Interna-tional Conference for Technical Postgraduates, Dec. 2009.

[12] M. Walma, "Pipelined cyclic redundancy check (CRC) calculation," in Proceedings of the 16th International Conference on Computer Communications and Networks, Aug. 2007, pp. 365–370.

BIO DATA

K.Venkata Krishna Reddy presently pursuing M.Tech in Department of electronics and communications in Sreenidhi institute of science and technology (SNIST) Hyderabad AP, India