International Journal For Technological Research In Engineering
Volume 2, Issue 1, September-2014 *ISSN (Online): 2347 - 4718*

IMPLEMENTATION OF PARALLEL CYCLIC REDUNDANCY CHECK FOR HIGH SPEED COMMUNICATION USING FPGA

P. Veena[1], Prof. S. Jagadeesh[2], S. Rekha[3]
[1]M. Tech Student, [2]Head of Department, [3]Assistant Professor
Department of Electronics and Communication Engineering, SSJ Engineering College,
Affiliated to JNTU-Hyderabad, India

ABSTRACT: A cyclic redundancy check (CRC) is an error-detecting code commonly used in digital networks and storage devices to detect accidental changes to raw data. Blocks of data entering these systems get a short check value attached, based on the remainder of a polynomial division of their contents; on retrieval the calculation is repeated, and corrective action can be taken against presumed data corruption if the check values do not match.

I. INTRODUCTION

Cyclic redundancy check is commonly used in data communication and other fields such as data storage, data compression, as a vital method for dealing with data errors. Usually, the hardware implementation of CRC computations is based on the linear feedback shift registers (LFSRs), which handle the data in a serial way. Though, the serial calculation of the CRC codes cannot achieve a high throughput. In contrast, parallel CRC calculation can significantly increase the throughput of CRC computations. For example, the throughput of the 32-bit parallel calculation of CRC-32 can achieve several gigabits per second However, that is still not enough for high speed application such as Ethernet networks. A possible solution is to process more bits in parallel; Variants of CRCs are used in applications like CRC-16 BISYNC protocols, CRC32 in Ethernet frame for error detection, CRC8 in ATM, CRC-CCITT in X-25 protocol, disc storage, SDLC, and XMODEM.

A. CRC's in high speed wireless LAN:

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 focus on the CRC calculation in WLAN where the packet size is huge and hence slow CRC calculation may become bottleneck for communication process. Based on this concept, paper present a novel implementation of the CRC implementation through multiple execution units that calculate CRC on different part of packet and then combine the result to get the final CRC. Consequently, the number of cycles utilized to recalculate the CRC codes is dramatically reduced. Furthermore, estimation on the maximum throughput is made based on synthesis results of our implementation and under that assumption, calculate actual CRC operation has been done. A performance study of the project is done by calculation of CRC with 2, 4, 8 execution units on same data block. There are several techniques for generating check bits that can be added to a message. Perhaps the simplest is to append a single bit, called the "parity bit," which makes the total number of 1-bits in the code vector (message with parity bit appended) even (or odd). If a single bit gets altered in transmission, this will change the parity from even to odd (or the reverse). The sender generates the parity bit by simply summing the message bits modulo 2—that is, by exclusive or'ing them together. It then appends the parity bit (or its complement) to the message. The receiver can check the message by summing all the message bits modulo 2 and checking that the sum agrees with the parity bit. Equivalently, the receiver can sum all the bits (message and parity) and check that the result is 0 (if even parity is being used).This simple parity technique is often said to detect 1-bit errors. Actually it detects errors in any odd number of bits (including the parity bit), but it is a small comfort to know you are detecting 3-bit errors if you are missing 2-bit errors.

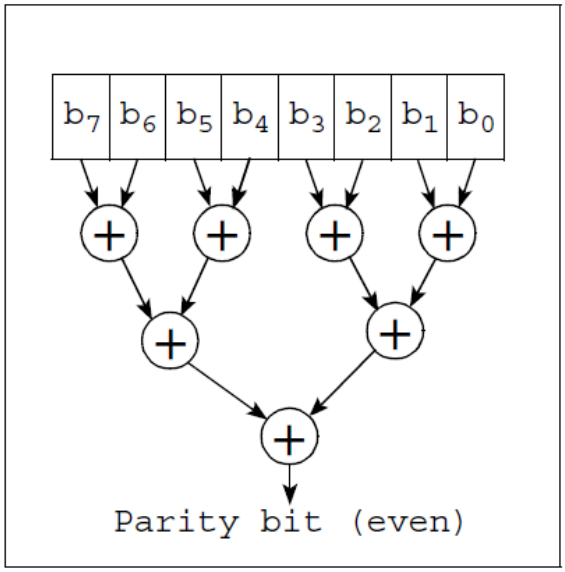


Fig 1: XOR Tree

Other techniques for computing a checksum are to form the exclusive or of all the bytes in the message, or to compute a sum with end-around carry of all the bytes. In the latter method the carry from each 8-bit sum is added into the least significant bit of the accumulator. It is believed that this is more likely to detect errors than the simple exclusive or, or the sum of the bytes with carry discarded. A technique that is believed to be quite good in terms of error detection, and

which is easy to implement in hardware, is the cyclic redundancy check. This is another way to compute a checksum, usually eight, 16, or 32 bits in length, that is appended to the message. We will briefly review the theory and then give some algorithms for computing in software a commonly used 32-bit CRC checksum. The CRC is based on polynomial arithmetic, in particular, on computing the remainder of dividing one polynomial in GF(2) (Galois field with two elements) by another. It is a little like treating the message as a very large binary number, and computing the remainder on dividing it by a fairly large prime such as Intuitively, one would expect this to give a reliable checksum. A polynomial in GF(2) is a polynomial in a single variable x whose coefficients are 0 or 1. Addition and subtraction are done modulo 2—that is, they are both the same as the exclusive or operator. For example, the sum of the polynomials
x3 + x + 1 and
x4 + x3 + x2 + x
is as is their difference. These polynomials are not usually written with minus signs, but they could be, because a coefficient of –1 is equivalent to a coefficient of 1. Multiplication of such polynomials is straightforward. The product of one coefficient by another is the same as their combination by the logical and operator, and the partial products are summed using exclusive or. Multiplication is not needed to compute the CRC checksum. Division of polynomials over GF(2) can be done in much the same way as long division of polynomials over the integers. Below is an example.

$$
\begin{array}{r}
x^4 + x^3 + 1 \\
x^3+x+1 \overline{)\, x^7 + x^6 + x^5 + \qquad\quad x^2 + x} \\
\underline{x^7 + \quad x^5 + x^4} \\
\quad x^6 + \quad x^4 \\
\quad \underline{x^6 + \quad x^4 + x^3} \\
\qquad\qquad x^3 + x^2 + x \\
\qquad\quad \underline{x^3 + \quad\; x + 1} \\
\qquad\qquad\; x^2 + \quad 1
\end{array}
$$

The reader might like to verify that the quotient of multiplied by the divisor of x3 + x + 1 plus the remainder of equals the dividend. The CRC method treats the message as a polynomial in GF(2). For example, the message 11001001, where the order of transmission is from left to right (110…) is treated as a representation of the polynomial x7 + x6 + x3 + 1. The sender and receiver agree on a certain fixed polynomial called the generator polynomial. For example, for a 16-bit CRC the CCITT (Comité Consultatif Internationale Télégraphique et Téléphonique)1 has chosen the polynomial x16 + x12 + x5 + 1 which is now widely used for a 16-bit CRC checksum. To compute an r-bit CRC checksum, the generator polynomial must be of degree r. The sender appends r 0-bits to the m-bit message and divides the resulting polynomial of degree m + r – 1 by the generator polynomial. This produces a remainder polynomial of degree r – 1 (or less). The remainder polynomial has r coefficients, which are the checksum. The quotient polynomial is discarded. The data transmitted (the code vector) is the original m-bit message followed by the rbit checksum.

Hardware feedback shift register:

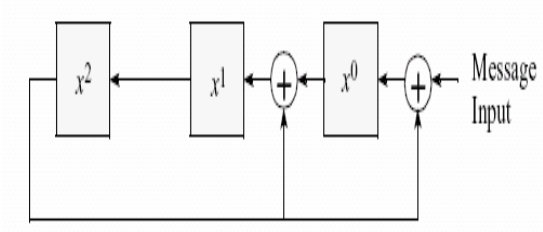


Fig 2: Feedback Shift Register

Initialize the CRC register to all 0-bits. Get first/next message bit m. If the high-order bit of CRC is 1, Shift CRC and m together left 1 position, and XOR the result with the low-order r bits of G. Otherwise, Just shift CRC and m left 1 position. If there are more message bits, go back to get the next one. CRC is playing a main role in the networking environment to detect the errors. With challenging the speed of transmitting data, to synchronize with speed, it's necessary to increase speed of CRC generation. Most electrical and computer engineers are familiar with the cyclic redundancy check (CRC). Many know that it's used in communication protocols to detect bit errors, and that its essentially a remainder of the modulo-2long division operation. Some have had closer encounters with the CRC and know that it's implemented as a linear feedback shift register (LFSR) using flip-flops and XOR gates. They likely used an online tool or an existing example to generate parallel CRC code for a design .

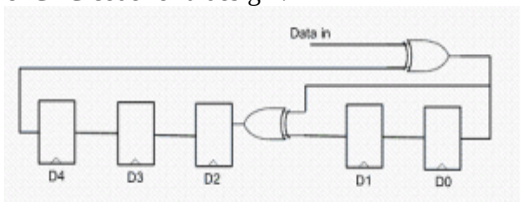


Fig3: Hardware CRC5 Implementation

Fast CRC:
Our fast CRC update method is extended from the parallel CRC calculation and can adapt to a number of bits processed in parallel. The method can also reduce the data traffic and power consumption of the CRC calculation unit.

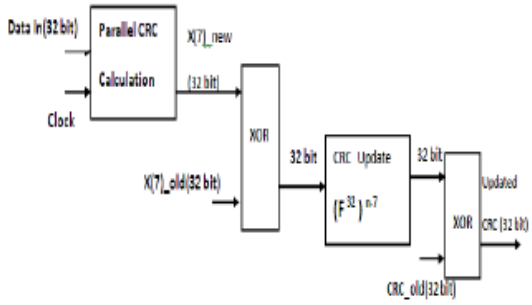


Fig4: Fast CRC Update Architecture

F-matrix parallel CRC generation:
F-matrix follows the algorithm as:

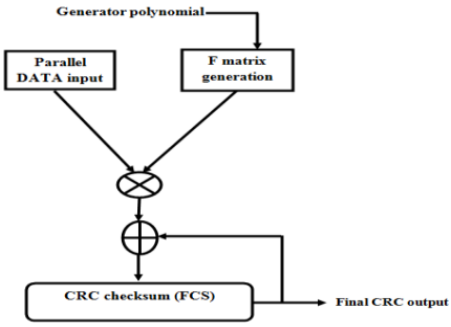


Fig 5:F-matrix algorithm

Parallely data input is processed; it is ANDed with the F-matrix generation from the generated polynomial. Result of that will XORed with present state CRC checksum. The final result will obtained after (k+m)/w cycles. Generation of F-matrix:

F-matrix generated from the generated polynomial, matrix form can be represented as:

$$F = \begin{bmatrix} P_{m-1} & 1 & 0 & 0 & 0 \\ P_{m-2} & 0 & 1 & 0 & 0 \\ P_{m-3} & 0 & 0 & 1 & 0 \\ P_{m-4} & 0 & 0 & 0 & 1 \\ .. & .. & .. & .. & .. \\ P_0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Where{p0……pm-1}is generator polynomial. For example, the generator polynomial for CRC4 is {1, 0, 0, 1, 1} and w-bits are parallely processed.

$$F = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

$$F^4 = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Here w=m=4, for that matrix calculated as follow. Parallel architecture: Parallel architecture based on F- matrix"d" is data that is parallel processed (i.e. 32bit), 'X is next state, X is current state (generated CRC), F(i)(j) is the ith row and jth column of FW matrix. If X = [xm1 …..x1 x0]T is utilized to denote the state of the shift registers, in linear system theory, the state equation for LFSRs can be expressed in modular 2 arithmetic as follow.

Xi´= (P0⊗ Xm-1)⊕d

Where, X(i) represents the state of the registers, X(i + 1) denotes the state of the registers, d denotes the one bit shift-in serial input. F is an m x m matrix and G is a 1 x m matrix. G = [0 0 --------0 1]T This can be represented in the matrix form as

$$\begin{bmatrix} X'_{m-1} \\ X'_{m-2} \\ . \\ X'_0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} X_{m-1} \\ X_{m-2} \\ . \\ X_0 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 0 \\ . \\ 1 \end{bmatrix} .d$$

Finally it can rewritten as

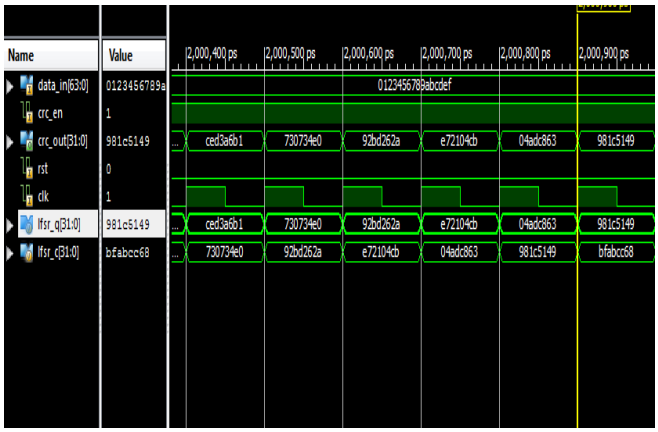
$$X' = F^W \otimes X \oplus d$$

If W-bits are parallel processed,the result of the CRC will generated after (k+m)/w cycles.

II. CONCLUSION

Generally when high-speed data transmission is required serial implementation is not preferred because of slow throughput. So parallel implementation is preferred which takes less time. CRC-32 requires 17 clock cycles to transmit 64bytes of data. But CRC-64 needs 9 clock cycles to transmit the same data.So, it drastically reduces computation time to 50% and same time increases the throughput.

Results:



REFERENCES

[1] Peterson, W. W. and Brown, D. T. (January 1961). "Cyclic Codes for Error Detection". Proceedings of the IRE 49 (1):228–235. doi: 10.1109/JRPROC. 1961. 287814.

[2] Ritter, Terry (February 1986). "The Great CRC Mystery". Dr. Dobb's Journal 11 (2): 26–34, 76–83. Retrieved 21 May 2009.

[3] Stigge, Martin; Plötz, Henryk; Müller, Wolf; Redlich, Jens-Peter (May 2006). Reversing CRC – Theory and Practice. Berlin: Humboldt University Berlin. p. 17. Retrieved 4 February 2011. "The presented methods offer a very easy and efficient way to modify your data so that it will compute to a CRC you want or at least know in advance."

[4] Cam-Winget, Nancy; Housley, Russ; Wagner, David; Walker, Jesse (May 2003). "Security Flaws in 802.11 Data Link Protocols". Communications of the ACM 46 (5): 35–39. doi : 10.1145/769800. 769823.

[5] Williams, Ross N. (24 September 1996). "A

Painless Guide to CRC Error Detection Algorithms V3.00". Retrieved 5 June 2010.

[6] WH; Teukolsky, SA; Vetterling, WT; Flannery, BP (2007). "Section 22.4 Cyclic Redundancy and Other Checksums". Numerical Recipes: The Art of Scientific Computing (3rd ed.). New York: Cambridge University Press.ISBN 978-0-521-88068-8.

[7] Koopman, Philip; Chakravarty, Tridib (June 2004)."Cyclic Redundancy Code (CRC) Polynomial Selection for Embedded Networks". The International Conference on Dependable Systems and Networks: 145 – 154. doi : 10.1109/DSN. 2004.1311885. ISBN 0-7695-2052-9. Retrieved 14 January 2011.

[8] Cook, Greg (6 July 2012). "Catalogue of parametrised CRC algorithms". Retrieved 7 July 2012.

[9] Castagnoli, G.; Bräuer, S.; Herrmann, M. (June 1993). "Optimization of Cyclic Redundancy-Check Codes with 24 and 32 Parity Bits". IEEE Transactions on Communications 41 (6): 883. doi:10.1109/26.231911.

[10] Koopman, Philip (July 2002). "32-Bit Cyclic Redundancy Codes for Internet Applications". The International Conference on Dependable Systems and Networks: 459–468. doi : 10.1109/DSN.2002. 1028931.ISBN 0-7695-1597-5. Retrieved 14 January 2011.

[11] Brayer, Kenneth (August 1975). Evaluation of 32 Degree Polynomials in Error Detection on the SATIN IV Autovon Error Patterns. National Technical Information Service. p. 74. Retrieved 3 February 2011.

[12] Hammond, Joseph L., Jr.; Brown, James E.; Liu, Shyan-Shiang (1975). "Development of a Transmission Error Model and an Error Control Model". Unknown (National Technical Information Service, published May 1975) 76: 74. Bibcode: 1975STIN...7615344H. Retrieved 7 July 2012.

[13] Brayer, Kenneth; Hammond, Joseph L., Jr. (December 1975)

