SSRG International Journal of VLSI and Signal Processing
ISSN: 2394 – 2584 / https://doi.org/10.14445/23942584/IJVSP-V9I2P101

Volume 9 Issue 2, 1-4, May-Aug 2022
© 2022 Seventh Sense Research Group®

Original Article

Randomized Verification of Ethernet

Dhyan V[1], Venu S[2], Syed Shabaz[3], Shaik Muinuddin[4], Madhura R[5]

[1,2,3,4,5]*ECE, Dayananda Sagar College of Engineering, Karnataka, India.*

Received: 11 June 2022 Revised: 22 July 2022 Accepted: 28 July 2022 Published: 06 August 2022

Abstract - *CRC stands for cyclic redundancy check, a well-known error detection algorithm found in Ethernet, PCIe, etc. The Cyclic redundancy check (CRC) code is a simple but effective method for detecting errors during digital data transmission and storage. CRC implementation can use any hardware or software method. This application report introduces different software algorithms, comparing themselves according to memory and speed utilized. Various standard CRC codes will be used. Correction codes are a way of finding and correcting errors introduced by a transmission channel. Block and convolution codes are two important parts of a code. Both eliminate unwanted or redundant data by adding to message data the rating symbols. Even though correction techniques are not used here, they are post-response of CRC. Cyclic redundancy check (CRC) codes come under cyclic codes, which in turn come under linear block codes. Hardware and software program techniques can be used in CRC implementation; within the conventional hardware implementation, an easy shift signs up circuit plays the computations by dealing with the facts one bit at a time. Managing records as bytes or phrases in software program implementations becomes extra handy and faster. Verification of CRC is challenging; hence, it's been done using the system Verilog based on a standard verification methodology.*

Keywords - *CRC, FPGA, PCIe, HWICAP.*

1. Introduction

In the networking context, CRC plays a crucial function in detecting errors. It is vital to raise the speed of CRC creation to keep up with the challenges of data transmission speed. A lot of engineers know about the cyclic redundancy check (CRC). Many know it is used to detect bit errors in communication protocols and is mainly a reminder of the modulo-2 long division operation. The linear feedback shift registers (LFSRs), which take care of data serially, are commonly used in the hardware implementation of CRC computations as a critical way of dealing with data errors. The CRC codes' serial calculation cannot achieve high throughput. The throughput of CRC computations can be considerably increased by using constant concurrent CRC calculations.

The Cyclic redundancy check (CRC) code provides an easy but powerful way to find errors that erupted during the transfer and storage of digital data. CRC implementation can use any hardware or software method. This application report introduces different software algorithms, comparing themselves according to memory and speed utilized. Various standard CRC codes will be used. Correction codes are a way to find and correct errors occurring because of the transmission channel. Two important parts of code exist block codes and convolutional codes. Both eliminate unwanted or redundant data by adding rating symbols to message data. Cyclic redundancy check (CRC) codes are a subset of cyclic codes, which are also a subset of linear block codes. CRC implementation can use hardware and software program techniques; within the conventional hardware implementation, an easy shift signs up circuit plays the computations by dealing with the facts one bit at a time. Managing records as bytes or phrases in software program implementations becomes extra handy and faster.

2. Background Survey

Algorithms based on tables and matrices were schematically displayed on paper [1]. The matrix-driven technique was also studied using other implementations, including single-byte, two-byte, and four-byte implementations. The graphical results of a supercomputer cluster experiment to gauge the implementation performance of CRC32 software were presented. It is shown that a high-speed four-byte matrix-driven algorithm should be used in embedded systems and industrial data transmission systems.

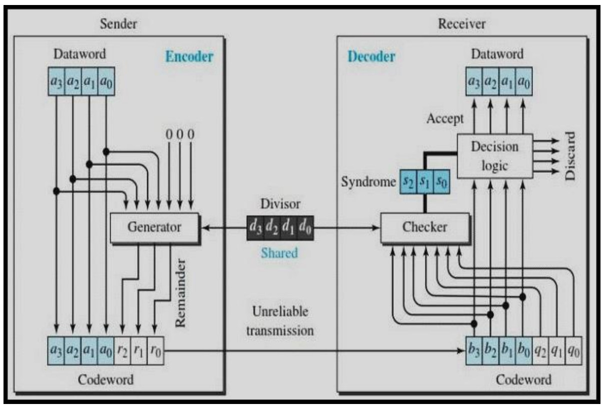
The findings of the first thorough analysis of the 32-bit CRC design space are provided in the paper [2]. For data word sizes of 12112 bits, the performance of the entire set of 1,073,774,592 distinct polynomials has been evaluated. An exhaustive search has led to a definitive list of polynomial classes that can and cannot deliver HD better than the 802.3 CRC for MTU-sized messages. This list has discovered a class of polynomials that exhibits excellent performance for MTU-sized messages and good performance for longer messages.

The creation of a simulation model for the performance study of configurable CRC-polynomials across Binary Symmetric Channels (BSCs) is detailed in the paper [3]. It presents a novel model for the analysis of different CRC polynomials codes with n parity bits ranging from 1 to 64. On Altera's FPGA Stratix II GX device, "EP2SGX90FF1508C3," which supports 10/100/1000 Mbps Ethernet over 1000Base-LX physical media, it also discusses the hardware implementation for CRC-32 "IEEE-802" and proposes an indirect methodology of CRC-performance using the Packet Error Rate (PER) parameter. It is suggested that top-notch CRC codes of 32, 40, and 64 bits be discovered and that the Ethernet protocol's performance for the maximum payload is investigated.

The paper [4] aims to show that the 32-bit CRC used in Ethernet (CRC-32) can be computed at a speed of 10 Gb/s using existing process technologies. We analyze two potential designs and discuss their performance based on our simulations. Because we lack access to cutting-edge process technology, we base our results on the extrapolation of device features.

3. Methodology

This proposed method reduces the zero-crossing problems in the existing cyclic redundancy check and introduces the shifting and XOR-based technique with the polynomial security codes. It will increase security and critical path delay in all distorted signal processing, thus transmission application. Thus, the proposed cyclic redundancy implementation proves the higher resource utilization in multiple data widths of 8-bit, 32-bit, 64-bit, and 128-bit data widths. Finally, this work was developed in Verilog HDL and synthesized in Xilinx vertex-5 FPGA and proved all the performance in area delay and power.



CRC Block Diagram

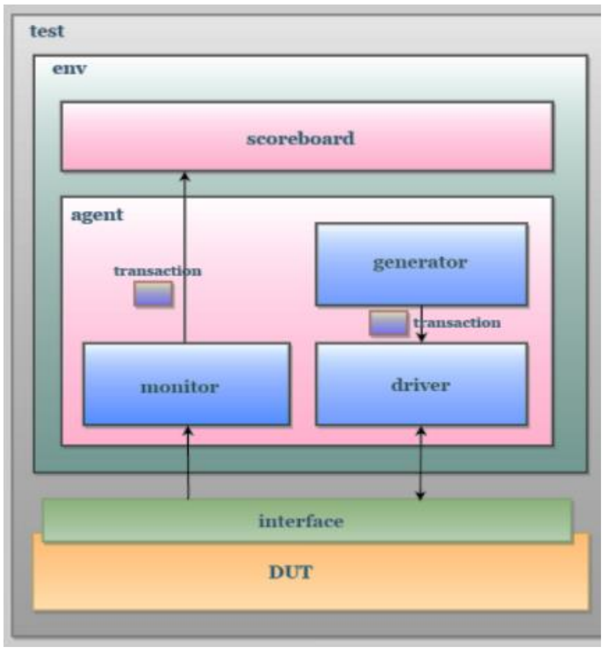
Consider a transmitter T, sends a sequence, S1 of k bits, S1= {b0, b1…, bk-1} to a receiver R. At the same time, T generates another sequence, of m bits, S2= {b0', b1'…,bm-1'}, to permit the receiver to recognize possible errors. The sequence S2 is commonly known as a Frame Check Sequence (FCS). It is generated because the complete sequence, S = S1 U S2 which is obtained by the concatenation of sequences S1 and S2, has the property that it is divisible (following arithmetic) by some predetermined sequence, P = {p0, p1…, pm} of m +1 bits.

After T sends S to R, R divides S, the message, and the FCS by P, using the same particular arithmetic after receiving the message. If there is no remainder, R assumesthere was no error. A modulo 2 arithmetic is used in the digital realization of the above concepts.

S is the sequence for error detecting, P is the divisor, and Q is the quotient. S1 is the original sequence of k bits to transmit. Finally, S2 is the FCS of m bits.

While bitwise XOR operators perform the sum and the subtraction, the product operator is accomplished by a bitwise AND. In this situation, a modulo - 2 division-performing CRC circuit can be built as a unique shift register known as an LFSR. Both the transmitter and the receiver can use it. The dividend in the instance of the transmitter is the sequence S1 joined to a series of m zeros to the right. P is the divisor. The received sequence serves as both the dividend and the divisor in the more straightforward scenario of a receiver.

4. Test Plan Components



Transaction – Defines the agent-generated pin level activity, which must either be noticed by the agent or driven to the DUT via the driver (Placeholder for the activity monitored by the monitor on DUT signals)

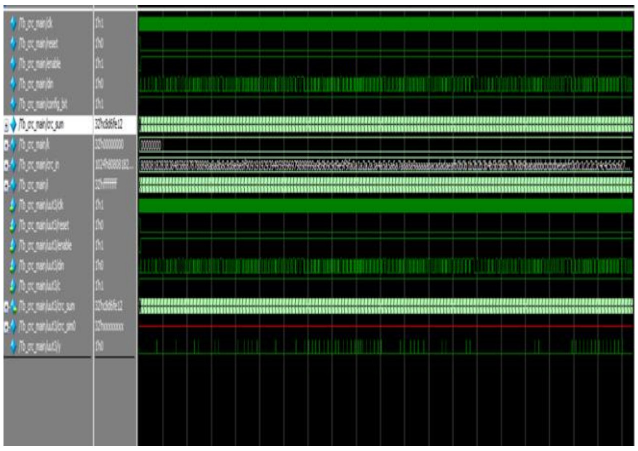
Generator – Generates the stimulus (create and randomize the transaction class) and sends it to the driver

Driver – Receives the stimulus (transaction) from a generator and transmits the transaction's packet-level data to pin level (to DUT)

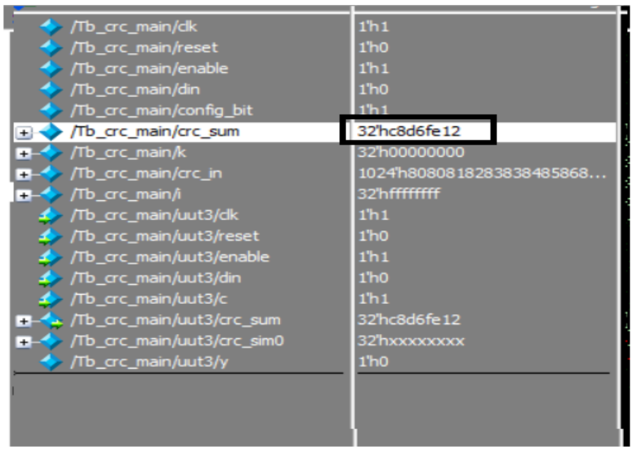
Monitor – Interface signals show pin-level activity, which is converted into packet-level activity and sent to components like the scoreboard.

Agent – An agent is a class that serves as a container for classes (such as generator, driver, and monitor) that are particular to an interface or protocol.

5. Result

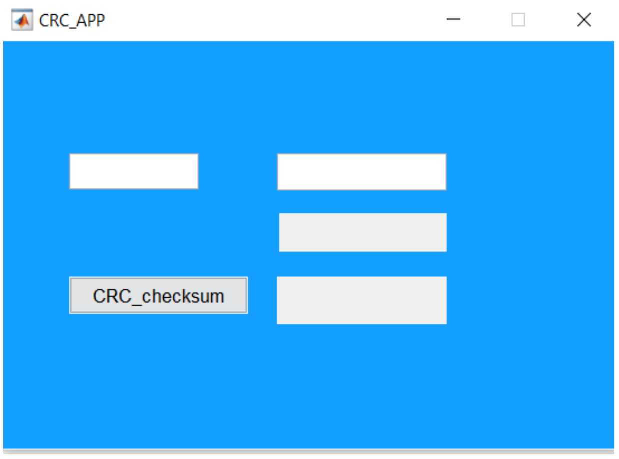


CRC-32 Output timing diagram



```
/Tb_crc_main/clk              1'h1
/Tb_crc_main/reset            1'h0
/Tb_crc_main/enable           1'h1
/Tb_crc_main/din              1'h0
/Tb_crc_main/config_bit       1'h1
/Tb_crc_main/crc_sum          32'hc8d6fe12
/Tb_crc_main/k                32'h00000000
/Tb_crc_main/crc_in           1024'h80808182838384858 68...
/Tb_crc_main/i                32'hffffffff
/Tb_crc_main/uut3/clk         1'h1
/Tb_crc_main/uut3/reset       1'h0
/Tb_crc_main/uut3/enable      1'h1
/Tb_crc_main/uut3/din         1'h0
/Tb_crc_main/uut3/c           1'h1
/Tb_crc_main/uut3/crc_sum     32'hc8d6fe12
/Tb_crc_main/uut3/crc_sim0    32'hxxxxxxxx
/Tb_crc_main/uut3/y           1'h0
```

CRC-32 Output timing diagram



MATLAB App Interface

The proposed architecture is developed, and it is important to validate whether the obtained Result is correct or wrong; hence for the verification of the obtained CRC, a MATLAB application is being developed, and the App is designed to return the CRC output for a given input, presently the application is designed and is working for generating 16 bit CRC. Therefore the results are validated using an online calculator for inputs greater than 16 bits.

Conclusion

The proposed work revolves around using Cyclic Redundancy Check (CRC) theory and implementation. To detect errors in a message, CRC is used. Two implementations are shown, the implementation of the Stride-by- 5 algorithms and the implementation of the Pipeline-go-back algorithm using ethernet CRC-32 polynomial. CRC can be implemented in various formats, including CRC-32, CRC-CCITT, and other polynomials. The CRC algorithm frequently finds errors in transmitted messages or stored data. The CRC is a very effective method for obtaining data reliability that is also simple.

References

[1] Evgeniy Mytsko, Andrey Malchukov, Valeriy Kim, Alexander Osokin, Ivan Zoev, Svetlana Ryzova," Software Implementation Research of Crc Computation Algorithms Compatible With Pkzip, Winrar, Ethernet."

[2] Philip Koopman," 32-Bit Cyclic Redundancy Codes for Internet Applications"

[3] Vinaya R Gad, Rajendra S Gad, Gourish M Naik, "Configurable Crc Error Detection Model for Performance Analysis of Polynomial: Case Study for the 32-Bits Ethernet Protocol."

[4] Tomas Henriksson, Henrik Eriksson, Ulf Nordqvist, Per Larsson-Edefors, Dake Liu,“ Vlsi Implementation of Crc-32 for 10 Gigabit Ethernet.”

[5] Huan Liu, Zhiliang Qiu, Weitao Pan, Jun Li, Ling Zheng and Ya Gao "Low- Cost and Programmable Crc Implementation Based on Fpga" *Ieee Transactions on Circuits and Systems—Ii: Express Briefs*, vol. 68, no. 1, 2021.

[6] Cyclic Redundancy Check Computation By Patrick Geremia. (Online) Available: Https://Documents.Pub/Document/Cyclic-Redundancy-Check-Computation-An-Implementation-Redundancy-Check-Computation.Html

[7] M. E. Kounavis and F. L. Berry, "Novel Table Lookup-Based Algorithms for High-Performance Crc Generation," Ieee Trans. Comput., vol. 57, no. 11, pp. 1550–1560, 2008.

[8] A. Akagic and H. Amano, "High-Speed Fully-Adaptable Crc Accelerators,"*Ieice Trans. Inf. Syst*., vol. 96, no. 6, pp. 1299-1308, 2013.

[9] L. Kekely, J. Cabal, and J. Koˇrenek, "Effective Fpga Architecture for General Crc," *in Proc. Int. Conf. Archit. Comput. Syst.*, pp. 211–223, 2019.

[10] C. Toal, K. Mclaughlin, S. Sezer, and X. Yang, "Design and Implementation of a Field-Programmable Crc Circuit Architecture," *Ieee Trans. Very Large Scale Integr. (Vlsi) Syst*., vol. 17, no. 8, pp. 1142–1147, 2009.

[11] the P4 Language Specification, Version 1.0.5, P4 Lang. the Consortium, Stanford,Ca, Usa, 2018.

[12] M. Grymel and S. B. Furber, "A Novel Programmable Parallel Crc Circuit," *Ieee Trans. Very Large Scale Integr. (Vlsi) Syst*., vol. 19, no. 10, pp. 1898–1902, 2011.

[13] S. Gueron, "Speeding Up Crc32c Computations With Intel Crc32 Instruction," *Inf. Process. Lett.*, vol. 112, no. 5, pp. 179–185, 2012.

[14] G. Campobello, G. Patane, and M. Russo, "Parallel Crc Realization," *Ieee Trans. Comput.*, vol. 52, no. 10, pp. 1312–1319, 2003.

[15] H. Liu, Z. Qiu, W. Pan, J. Li, L. Zheng, and Y. Gao, "Low-Cost and Programmable Crc Implementation Based on Fpga," 2020.

[16] K. Vipin and S. A. Fahmy, "Fpga Dynamic and Partial Reconfiguration: A Survey of Architectures Methods and Applications," *Acm Comput. Surveys*, vol. 51, no. 4, pp. 1–39, 2018.

[17] P. Orosz, T. Tóthfalusi, and P. Varga, "Fpga-Assisted Dpi Systems: 100 Gbit/S and Beyond," *Ieee Commun. Surveys Tuts.*, vol. 21, no. 2, pp. 2015– 2040, 2019.

[18] M. Jubin and T. Nayak, "Reconfigurable Very High Throughput Low Latency Vlsi (Fpga) Design Architecture of Crc 32," *Integration*, vol. 56, pp. 1–14, 2017.