ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 6, June 2015

Design of Parallel CRC Generation for High Speed Application

Chaitali Tohgaonkar[1], Prof. Sanjay B. Tembhurne[2], Prof. Vipin S. Bhure[3]

Electronics and Communication Engineering, GHREAT, RTMN University, Nagpur, India[1]

Professor, Electronics and Communication Engineering, GHREAT, RTMN University, Nagpur, India[2,3]

Abstract: A Cyclic Redundancy Check (CRC) is the remainder or residue of binary division of a potentially long message, by a CRC polynomial. This technique is ubiquitously employed in communication and storage applications due to its effectiveness at detecting errors and malicious tampering. The hardware implementation of a bit-wise CRC is a simple linear feedback shift register. This means that 'n' clock cycles will be required to calculate the CRC values for an n-bit data stream. This project primarily focuses on error detection in the Ethernet applications. This paper presents implementation of parallel Cyclic Redundancy Check (CRC) based upon DSP algorithms of pipelining, retiming and unfolding. The architectures are first pipelined to reduce the iteration bond by using novel look-ahead techniques and then unfolded and retimed to design high speed parallel circuits. The methodology to be employed with VHDL, Xilinx ISE for simulation and test-bench verification.

Keywords: Cyclic Redundancy Check (CRC), Parallel Pipelining, LFSR, VHDL code.

I. INTRODUCTION

CRCs are based on the theory of cyclic error-correcting codes. The use of systematic cyclic codes, which encode messages by adding a fixed-length check value, for the purpose of error detection in communication networks was first proposed by W. Wesley Peterson in 1961. A CRC (Cyclic Redundancy Check) is a popular error-detecting code computed through binary polynomial division. To generate a CRC, the sender treats binary data as a binary polynomial and performs the modulo-2 division of the polynomial by a standard generator (e.g., CRC-32). The remainder of this division becomes the CRC of the data, and it is attached to the original data and transmitted to the receiver. Receiving the data and CRC, the receiver also performs the modulo-2 division with the received data and the same generator polynomial. Errors are detected by comparing the computed CRC with the received one as shown in fig. below.

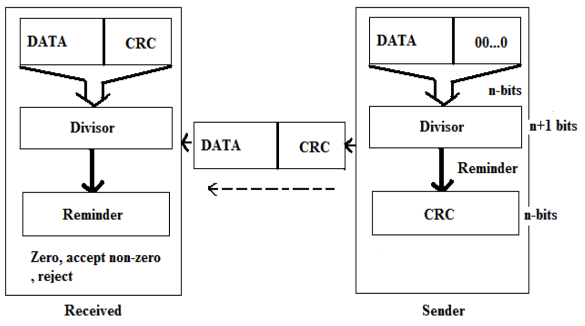


Fig.1. Operation of CRC performed at sender and receiver side.

The CRC algorithm only adds a small number of bits (32 bits in the case of CRC-32) to the message regardless of the length of the original data, and shows good performance in detecting a single error as well as an error burst.

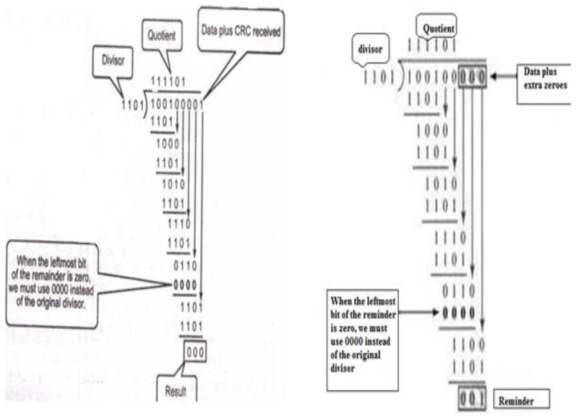

Divisor
Quotient
Data plus CRC received
When the leftmost bit of the remainder is zero, we must use 0000 instead of the original divisor.
Result
divisor
Quotient
Data plus extra zeroes
When the leftmost bit of the reminder is zero, we must use 0000 instead of the original divisor
Reminder

Fig.2. Example of CRC generation.

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.
The commonly used generator polynomial for Ethernet is given by:

x32 + x26 + x23 + x22 + x16 + x12 + x11 + x10 + x8 + x7 + x5 + x4 + x2 + x + 1
We will use this polynomial for all further calculations involved.

II. SERIAL CRC

Traditional method for generating serial CRC is based on linear feedback shift registers (LFSR). The main operation of LFSR for CRC calculations is nothing more than the binary divisions. Binary divisions generally can be performed by a sequence of shifts and subtractions. Here CRC checking is done serially. The data input will be single (binary) and every clock pulse the data input will be one.
The input message 256 bit is transmitted serially and it passes through CRC-32 (generator polynomial, single execution unit) and CRC is calculated.

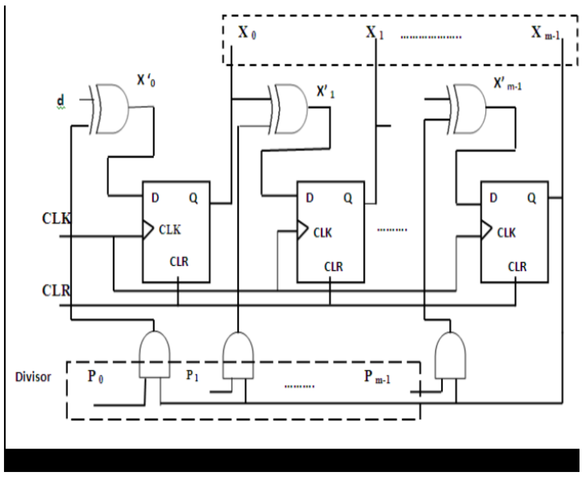


Fig.3. Basic LFSR architecture.

It will take more time to calculate CRC. Therefore it has high latency and low throughput. So, to overcome this problem, we are using multiple parallel execution units.

III. 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

Parallel processing used to increasing the throughput by producing the no. of output same time. Retiming used to increasing clock rate of circuit by reducing the computation time of critical path. In fast CRC update technique, it is not required to calculate CRC each time for all the data bits, instead of that calculating CRC for only those bits that are change. There are different approaches to generate the parallel CRC having advantages and disadvantages for each technique. Table based architecture required pre-calculated LUT, so, it will not used for generalized CRC, fast CRC update technique required buffer to store the old CRC and data. In unfolding architecture increases the no. of iteration bound.
The F matrix based architecture are more simple and low complex.

A. *Design Procedure*

In this implementation, the data input will be 4 bits at one clock pulse if the data input size will be increase so clock pulse will be decreased. It will complete the result in lesser clock pulse so it will be used for high performance.
The data 64 bit distributed into four blocks. Each consists of 16 bit. Here, four execution units are used. They are parallel pipelined with each other and four 32-bit remainders are obtained. They are ex-or with each other and got final CRC which is of 32-bit.

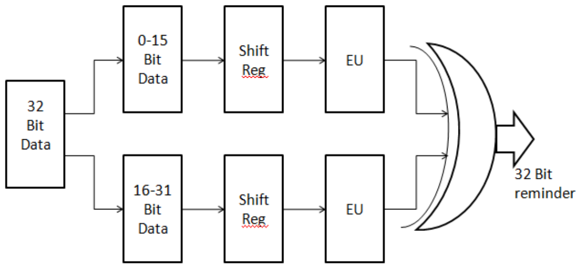


Fig.4. Shows CRC calculation using four parallel execution units.

B. *Verification and Simulation*

In this part the test bench and simulation result will be shown. In this the signals will be generated when the data will be given to the input and the output will be shown. And finally the simulation result will be displayed. The simulation result is given below.

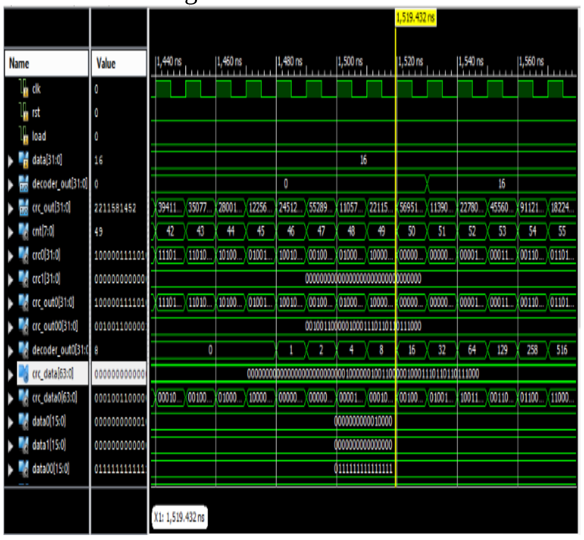
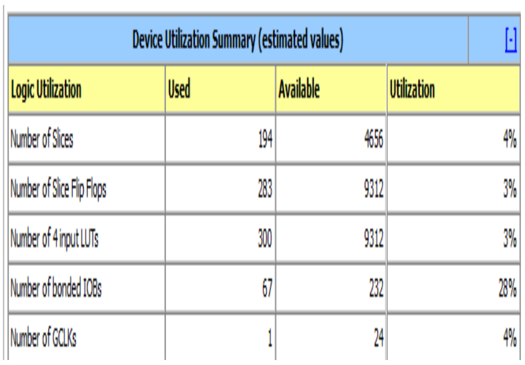


Fig.5. Simulation result of decoder.

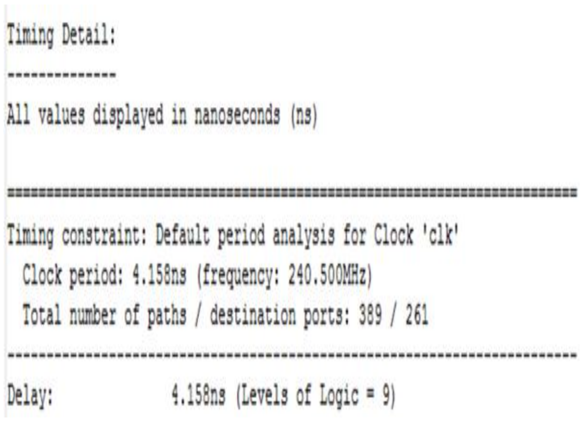
C. *Synthesis Report*

Synthesis process will be done by using RTL Compiler tool. The process will generate area report, power report, timing report, clock generating report. The generated output will be the net list file.
Both Serial and Parallel CRC Checker for Ethernet has been designed and synthesized. Now the comparison will be done between these two designs. The comparison will be done for the simulation results, area and timing.



Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slices	194	4656	4%
Number of Slice Flip Flops	283	9312	3%
Number of 4 input LUTs	300	9312	3%
Number of bonded IOBs	67	232	28%
Number of GCLKs	1	24	4%

Fig.6. Device utilization summary.



```
Timing Detail:
--------------
All values displayed in nanoseconds (ns)

=========================================================================
Timing constraint: Default period analysis for Clock 'clk'
  Clock period: 4.158ns (frequency: 240.500MHz)
  Total number of paths / destination ports: 389 / 261
-------------------------------------------------------------------------
Delay:               4.158ns (Levels of Logic = 9)
```

Fig.7. Timing summary.

IV. LITERATURE REVIEW

[1] Hitesh H. Mathukiya; Naresh M. Patel; "A Novel Approach for Parallel CRC generation for high speed application". In this paper, the parallel CRC generation deal with 64bit parallel processing based on built in F matrix with order of generator polynomial is 32. This gives CRC with half number of cycles. It drastically reduces computation time to 50% and same time increases the throughput. The no. of LUT get increased, so area also get increase.

[2] Yan Sun; Min Sik Kim; "A Pipelined CRC Calculation Using Lookup Tables". In this paper, they present a fast cyclic redundancy check (CRC) algorithm that performs CRC computation for an arbitrary length of message. This paper proposes a table-based hardware architecture for calculating CRC by taking advantage of CRC's properties and pipelining the feedback loop. It achieves considerably higher throughput than existing serial or byte-wise lookup CRC algorithms. With delay increase in the critical path.

[3] Weidong Lu and Stephan Wong, "A Fast CRC Update Implementation", IEEE Workshop on High Performance Switching and Routing Oct. 2003.In this paper, they presented a novel method to update the CRC code when packets are passing through interconnecting devices. And focus on the CRC calculation that is performed during the routing of the Ethernet packets be encapsulating the packets into Ethernet frames, adding a frame header and adding a frame trailer. It calculates the intermediate results of the changed fields based on the parallel CRC calculation and performs a single step update afterwards. And the number of cycles is dramatically reduced. The fast CRC update only calculates the changed portion of a frame.

[4] Campobello, G.; Patane, G.; Russo, M.; "Parallel CRC realization". This paper presents a theoretical result in the context of realizing high speed hardware for parallel CRC checksums. The number of bits processed in parallel can be different from the degree of the polynomial generator. Presented Pre-calculated F-matrix based 32 bit parallel processing. Which is faster and more compact and is independent of the technology used in its realization. But it doesn't work if polynomial change.

V. CONCLUSION

From all the analysis of both Serial and Parallel implementation of CRC Checker, it has been concluded that Parallel CRC Checker for Ethernet is of high speed and high performance Checker. Pipelining has decreased the iteration bound of the architecture effectively. This paper shows the use of Parallel CRC for high speed application such as Ethernet. Proposed design (32 bits) reduces the computation time and also reduces the no. of slices used. So applying pipelining to the CRC has increased the throughput to achieve high speed design.

ACKNOWLEDGEMENT

We would like to extend my gratitude & sincere thanks to my supervisors Professor, coordinator and HOD of department of Electronics and communication Engineering, for there constant motivation and support during the course of our work.

REFERENCES

[1] Hitesh H. Mathukiya and Naresh M. Patel; "A Novel Approach for Parallel CRC generation for high speed application," 2012 IEEE DOI.
[2] Y. Sun and M. S. Kim; "A table-based algorithm for pipelined CRC calculation," in Proceedings of IEEE International Conference on Communications, May 2010.'
[3] G. Campobello, G. Patane, and M. Russo; "Parallel CRC realization," IEEE Transactions on Computers, Oct. 2003.
[4] C. Cheng and K. K. Parhi; "High-speed parallel CRC implementation based on unfolding, pipelining, and retiming," IEEE Transactions on Circuits and Systems, Oct.2006.
[5] Weidong Lu and Stephan Wong, "A Fast CRC Update Implementation", IEEE Workshop on High Performance Switching and Routing, Oct. 2003.
[6] W. Jiang and V. K. Prasanna "A memory balanced linear pipeline architecture for trie based IP lookup". 2007.
[7] Sonali Selokar and P. H. Rangaree; "Design and implementation of CRC code generator based on parallel execution method for high speed wireless LAN".
[8] Deepti Rani Mahankuda and M. Suresh; "A high performance CRC checker for Ethernet application".
[9] J. S. Chitode "Data communication" coding technique.
[10] Indu I and Manu T. S. "Cyclic redundancy check generation using multiple lookup table algorithms".
[11] Martin Grymel and Steve B. Furber, "A Novel Programmable Parallel CRC Circuits" IEEE transactions, 2011.
[12] Chao Cheng and K. Parhi, "High Speed Parallel CRC implementation based on Unfolding, Pipelining and retiming", IEEE transaction October 2006.
[13] Rameshwar Murade, MD Manan Mujahid, M.A.M. Sabir, "The design and implementation of a Programmable CRC computation circuit architecture using FPGA" 2013.
[14] Adrian Simionescu, "CRC Tool Computing CRC in Parallel for Ethernet".
[15] Sprachmann, M. , "Automatic generation of parallel CRC circuits," *Design & Test of Computers, IEEE,* May 2001.
[16] Tenkasi V. Ramabadran and Sunil S. Gaitonde "Tutorial on CRC Generation" Iowa State University, IEEE Micro.