FPGA BASED DESIGN OF PARALLEL CRC GENERATION FOR HIGH SPEED APPLICATION

Sri N.V.N.Prasanna Kumar[1], S.Bhagya Jyothi[2],G.K.S.Tejaswi[3]
[1]prasannakumar429@gmail.com, [2]sjyothi567@gmail.com, [3]tejaswikakatiya@gmail.com
[1] (Assistant professor, Department of ECE, Sri Vasavi Engineering College, JNTU KAKINADA)
[2] (B.Tech student, Department of ECE, Sri Vasavi Engineering College, JNTU KAKINADA)
[3] (B.Tech student, Department of ECE, Sri Vasavi Engineering College, JNTU KAKINADA)

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. This project primarily focuses on error detection in the Ethernet applications. Serial CRC implemented by linear feed -back shift registers using flip-flops and XOR Gates. This means that "k+m" clock cycles will be required to calculate the CRC values for an k-bit data stream, m is the order of generator polynomial. This project presents parallel CRC architecture based on F-matrix with order of generator polynomial is 32 and 64,to design high speed parallel circuits, which requires less number of clock cycles compared to serial CRC. The methodology will be implemented by using VHDL,Xilinx ISE for simulation.

KEYWORDS: *Cyclic Redundancy Check (CRC), parallel pipe lining, LFSR, VHDL code.*

1. 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. Possible solution is to send more bits in parallely variants of CRC are used in applications like CRC-16 BISYNC protocols, crc-32 in Ethernet frame for error detection. CRC-8 bit is used in ATM, CRC-CCITT used in X-25 protocol and disc storage, SDLC, and XMODEM, CRC-64 in ISO standards used in HDLC, Swiss-port etc., 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 etc. There are many CRC algorithms, each of which as a predetermined generator polynomial G(x) that is utilized to generate the CRC code. For 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)=x32+x26+x23+x22+x16+x12+x11+x10+x8+x7+x5+x4+x2+x1+x0

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=p32;p31;:: :p1;p0.

G = [10000010011000001000111011011011;

Another algorithm used in HDLC protocols with ISO standardization is CRC 64 ISO which is having the generator polynomial as shown below:

G(x) = x64+x4+x3+x1+x0;

Where m=64 is the higher order are called the degree of generator polynomial and also the length of the CRC code. We can extract the coefficient of G(x) and represent in the binary form as:

G=[1000000000000000000000000000000000000000000000000000000000000000000011011];

Which has m+1= 65 bits. The most significant bit of P, P64, corresponds to the coefficient of x64, the highest order of G(x). Similarly, p63 corresponds to the

coefficient x63, 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 2m+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, transmission error must have occurred.

2. A SERIAL IMPLENENTATION 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. 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. As shown in figure is serial data input, X is presents state (generated CRC), X' is next state and p is generator polynomial, Working of basic LFSR architecture is expressed in terms of following equations.

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

$X_0' = (P_0 \otimes X_{m-1}) \oplus d$

$X_1' = (P_0 \otimes X_{m-1}) \oplus X_{i-1}$

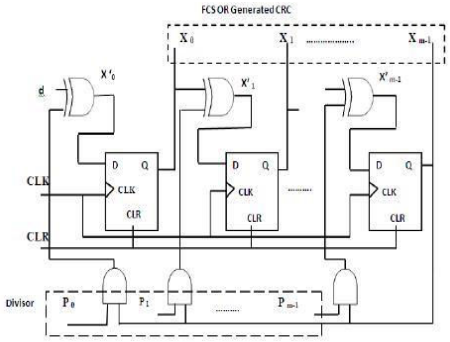

Figure1. Basic LFSR Architecture

3. 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. Unfolding, Retiming and pipelining Algorithm
4. F matrix based parallel CRC generation

The pipelined CRC architecture consists of lookup tables, d-flipflops, xor gates

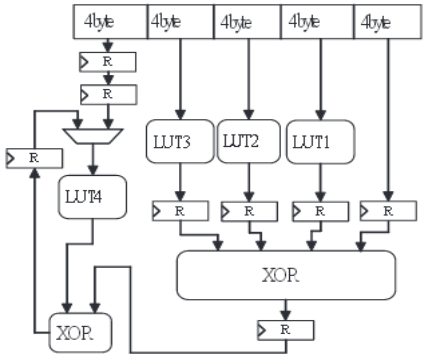

Figure 2: Pipelined CRC Architecture

The pipelined architecture in Figure2 has five blocks as input; four of them are used to read four new blocks from the message in each iteration. 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. Drawback is table based architecture required pre-calculated LUT, so, it will not be used for generalized CRC. 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. Drawback is Fast CRC update technique required buffer toster the old CRC and data.

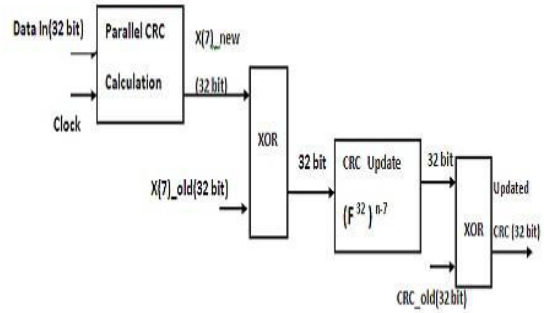

FIGURE 3: Fast CRC Update

Figure 3 consists of calculation block, xor gates, CRC update. In Unfolding, retiming and pipelining algorithm. Iteration bound is defined as the maximum of all the loop bounds. Loop bound defined as t/w, where "T" is the computation time of the loop and "w" is the no of delay elements in the loop. The largest iteration bound of a general serial CRC architecture is also 2TXOR. Drawback is unfolding architecture increases the no of iteration bound. Algorithm and parallel architecture for CRC generation based on F matrix. Parallel data input and each element off 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.

4. PARALLEL CRC GENERATION

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

F-matrix follows the algorithm as :

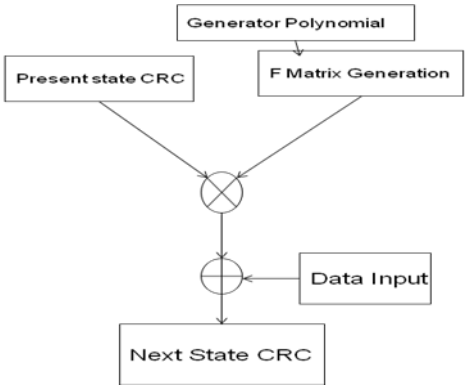

FIGURE 4: F-matrix Algorithm

4.1. Generation of F-matrix

F-matrix generation[1] from the generated polynomial, matrix form can be represented as:,

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

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. Here w=m=4, for that F4 matrix calculated as follow G(x) = x^4+x+1

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

Here w=m=4, for that F4 matrix calculated as follow

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

4.2 Parallel Architecture

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

$Xi = (P0 \otimes X_{m-1}) \oplus d$

Where, X(i) represents the ith state of the registers, X(i+1) denotes the (i+1)th state of the registers, d denotes the one bit shift-in serial input. F is an m x n matrix and G is a 1 x m matrix.

G = [0 0 -------0 1] T

This can be represented in the matrix form as:

$$\begin{pmatrix} X'm\text{-}1 \\ X'm\text{-}2 \\ \dots \\ X'0 \end{pmatrix} = \begin{pmatrix} 1\ 1\ 0\ 0 \\ 0\ 0\ 1\ 0 \\ 0\ 0\ 0\ 1 \\ 1\ 0\ 0\ 0 \end{pmatrix} \otimes \begin{pmatrix} Xm\text{-}1 \\ Xm\text{-}2 \\ \dots \\ X0 \end{pmatrix} \oplus \begin{pmatrix} 0 \\ 0 \\ \dots \\ 1 \end{pmatrix} .d$$

Finally it can be written as

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

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

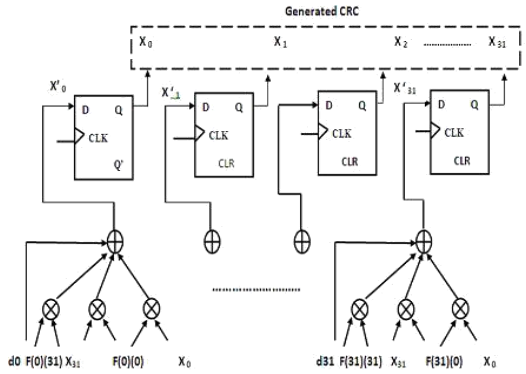

Figure 5: 32-bit parallel CRC calculation for CRC 32-bit

4.3 Existing Architecture

In the existing architecture w=64 bits are parallel processed and order of the generator polynomial is m=32, as shown in the figure6, If 32 bits are parallel processed as shown in figure 5 then CRC-32 will be generated after (k+m)/w cycles. If we increase number of bits to be processed parallel number of cycles required to calculate CRC can be reduced. Existing architecture is realized by below equation:

$X_{temp} = F^W \otimes D\ (0 \text{ to } 31) \oplus D(32 \text{ to } 63)$

$X' = F^W \otimes X \oplus X_{temo}$

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

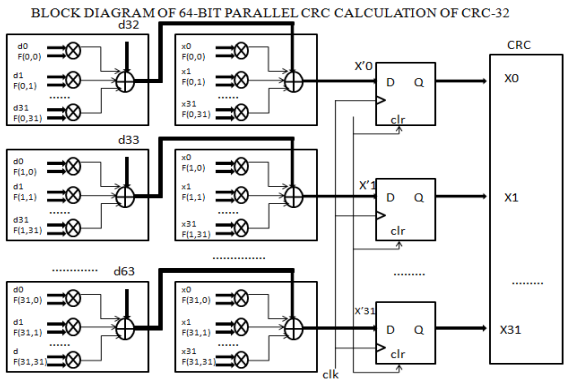

Figure6:64-Bit parallel CRC calculation for CRC- 32 Bit

In existing architecture d_i is the parallel input and F(i)(j), is the element of F^{32} matrix located at i^{th} row and j^{th} column. As shown in fig 3 input data bit d0,d1,…….,d31 anded with each row of F^w matrix and result will be xored individually d32,d33,……,d63 .Then each xor result is then xored with the X'(i) term of CRC32.Finally X will be the CRC generated after (k+m)/w cycles, where w=64.

4.4 Proposed Method

In the proposed method the CRC is generated using the CRC-64 ISO generator polynomial as given by

$$G(X) = X^{64}+X^4+X^3+X+1$$

In this proposed technique totally 128 bits are parallel processed.
w=128 (number of bits parallel processed)
m = 64 (order of generator polynomial)
The 128 bit parallel CRC calculation for CRC-64 ISO given by the below equation

$$X_{temp} = F^w \otimes D(0 \text{ to } 63) \oplus D(64 \text{ to } 127)$$

$$X' = F^w \otimes X \oplus X_{temp}$$

Where

D (0 to 63) = First 64 bits of parallel data input
D (64 to 127) = Next 64 bits of parallel data input
X' = next state of CRC
X = present state of CRC

If w bits are parallel processed then CRC 128-bit will be generated after (k+m)/w cycles. Where "k" is the length of data and "m" is the order of generator polynomial.

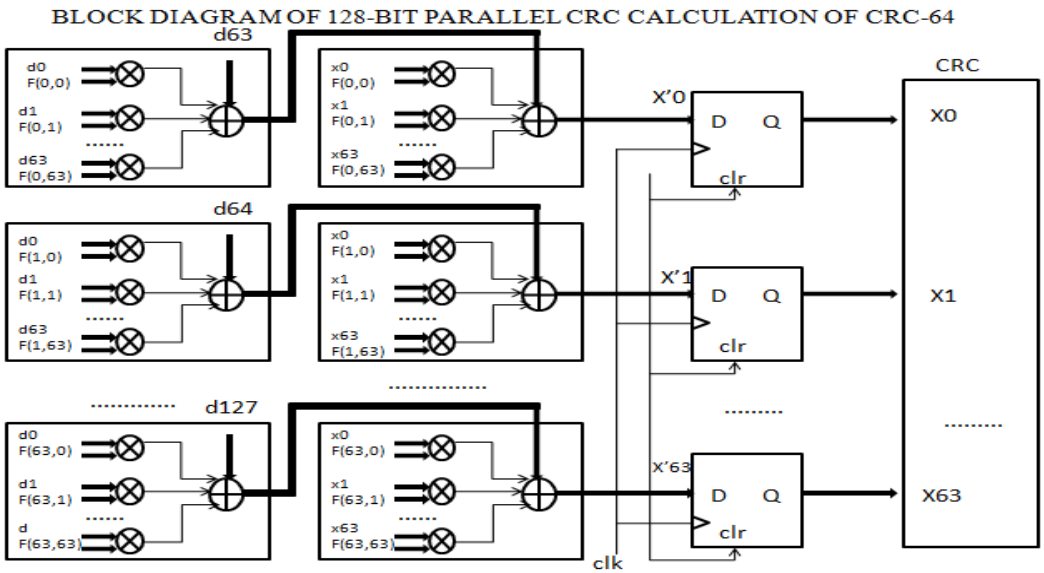

Figure7: 128 bit parallel CRC calculation for CRC64 ISO generator polynomial

5. Simulation Results

The simulation result for 32-Bit parallel processing of CRC-32 bit generator polynomial is shown in figure 8:

Figure: 8 Simulation results for CRC-32 bit generator polynomial

The simulation result for 64-Bit parallel processing of CRC-32 bit generator polynomial is shown in figure 9:

Figure: 9 Simulation results for CRC-64 bit generator polynomial

The simulation result for proposed 128-Bit parallel processing of CRC-64 bit generator polynomial is shown in figure 10:

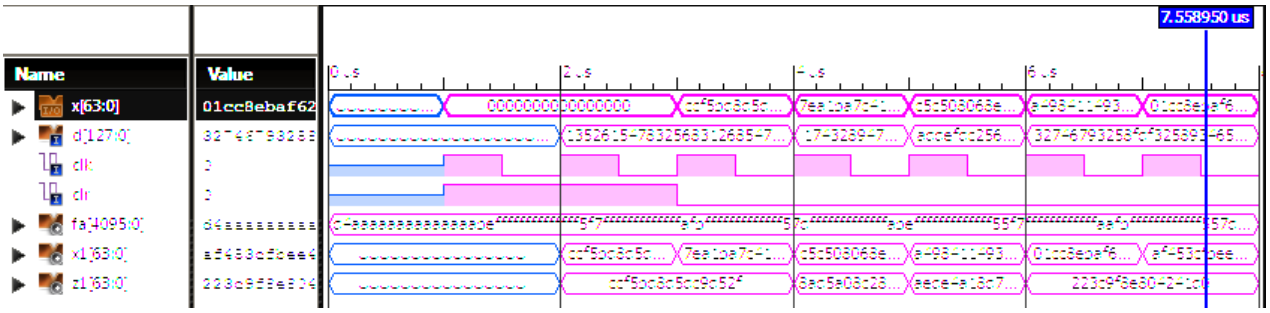

Figure 10: Simulation results for proposed 128-Bits parallel processing of CRC-64 generator polynomial

6. Conclusion

In this project firstly parallel CRC is calculated for CRC-32 bit generator polynomial. In the CRC-32 bit generator polynomial, there are two cases, one is 32-bit are parallel processed and the other is 64 bits are parallel processed .Consider data length 64 byte For 32-Bit parallel processing,17 clock cycles are required for generation of CRC. For 64 bit parallel processing ,9 clock cycles are required for the generation of CRC. In

the proposed method generator polynomial is of higher order,ie.,CRC64 IS .Using this generator polynomial 128 bit parallel CRC is implemented .For 128 bit parallel processing ,5 clock cycles are required for the CRC. So it reduces the computation time to 50% and at the same time increases the throughput. Hence it is easy method for fast CRC generation.

Order of the generator Polynomial	Number of parallel processed Bits	Number of clock cycles
32	32	17
32	64	9
64	128	5

References

1. D. Venkanna Babu, M.CH.P.V.L. Kumar, M. M. Venkatesh; "Generalized Parallel CRC Computations" IJMTST, volume: 2, Issue 04, April 2016. ISSN: 2455-3778
2. Chaitali Tohgaonkar, Prof. Sanjay B. Tembhurne, Prof. Vipin S. Bhure Albertengo: "Design of Parallel CRC generation for High Speed Applications," IJARCC, Issue 6, June 2015
3. CH. Janaki Ram, K.N.H. Srinivas, “An Efficient Technique for Parallel CRC Generation,” ISSN, Volume3, Issue12, December2014, Page no.9761-9765
4. Weidong Lu and Stephen Wong, A Fast CRC UpdateImplementation{wlu,Stephan}@dutepp0.tudelft,nl,http://ce.et.tudelft.nl .