www.ijecs.in

International Journal Of Engineering And Computer Science ISSN:2319-7242

Volume 3 Issue 12 December 2014, Page No. 9761-9765

An Efficient Technique for Parallel CRC Generation

CH. Janakiram, K.N.H.Srinivas,

P.G.Scholar ,Dept. of Electronics and Communication Engineering,

JNTU KAKINADA, Sri Vasavi Engineering College, Pedatadepalli, Tadepalligudem, West Godavari, A.P.

chjanakiram.ece@gmail.com

Professor, Dept. of Electronics and Communication Engineering,

JNTU KAKINADA, Sri Vasavi Engineering College, Pedatadepalli, Tadepalligudem, West Godavari, A.P.

knh.tridents@gmail.com

Abstract : ***Cyclic Redundancy Check is playing a vital role in the networking environment to detect the errors. With challenging speed of transmitting data and to synchronize with speed, it's necessary to increase speed of CRC generation. This paper presents 64 bits parallel CRC architecture based on F-matrix with order of generator polynomial is 32. Implemented design is hardware efficient and requires 50% less cycles to generate CRC with same order of generator polynomial. CRC32 bit is used in Ethernet frame for error detection. The whole design is functionally developed and verified using Xilinx ISE 12.3i Simulator.***

Keywords: Cyclic Redundancy Check(CRC), Parallel CRC calculation, Linear Feedback Shift Register (LFSR), F matrix.

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 parallelly Variants of CRCs are used in applications like CRC-16 BISYNC protocols, crc32 in Ethernet frame for error detection. CRC8 bit is used in ATM[5][8], CRC-CCITT used in X-25 protocol and disc storage, SDLC, and XMODEM[2]. 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[1][3][6]. 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[8], the most frequently utilized CRC algorithm is the CRC-32 algorithm employed by Ethernet[5], 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 = [100000100110000010001110110110111];

Which has m + 1 = 33 bits. The most significant bit of P, P32, corresponds to the coefficient of x32, the highest order of G(x).

Similarly, p31 corresponds to the coefficient of x31, 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.

2. A Serial Implementation of CRC

In hardware implementations, the CRC calculation (modulo 2 divisions)[1] can be easily performed by logical combinations of shift registers and XOR gates. The Linear Feedback Shift Register (LFSR)[2] 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.

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.

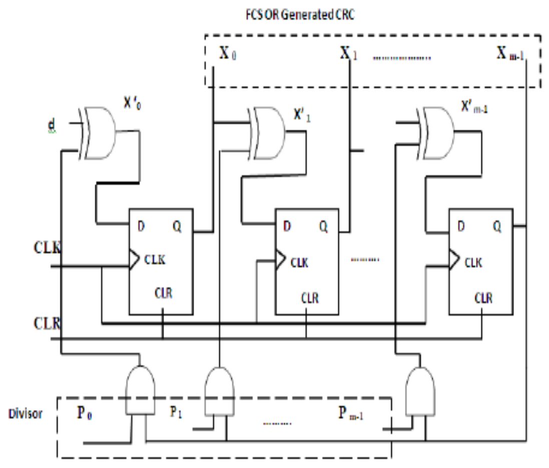

Figure 1. Basic LFSR Architecture.

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

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.

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.

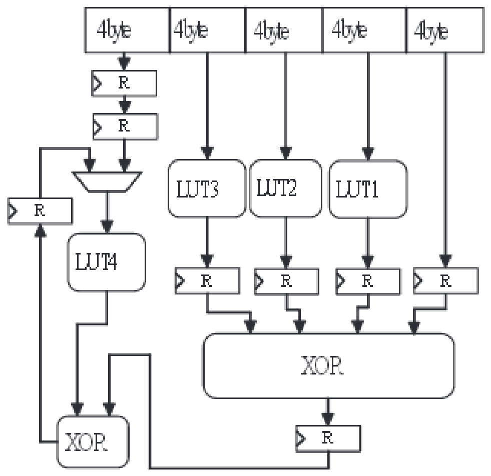

Figure 2 : Pipelined CRC architecture

The pipelined architecture [7] in Fig.2 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[7]. 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 used for generalized CRC.

In fast CRC update technique [5] 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 to store the old CRC and data.

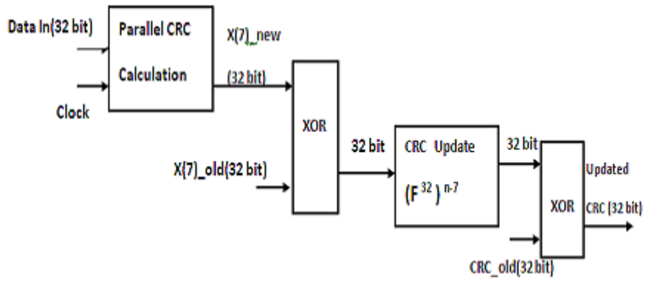

Figure 3: Fast CRC update architecture.

In Unfolding, retiming and pipelining algorithm [2] Iteration bound is defined as the maximum of all the loop bounds. Loop bound is 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 $2T_{XOR}$. 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 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.

4. F-Matrix Parallel CRC Generation

Parallely data 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[1]. F-matrix follows the algorithm as:

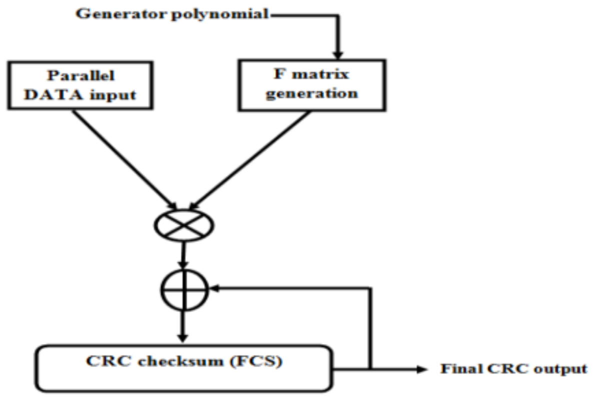

Figure 4: Algorithm for F-matrix based architecture

4.1 Generation of F-Matrix

F-matrix generation[1] 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 $_{\text{parallely}}$ processed.

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

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

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

4.2 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.

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

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 m matrix and G is a 1 x m matrix.

$$G = [0\ 0\ \text{--------}0\ 1]^T$$

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.

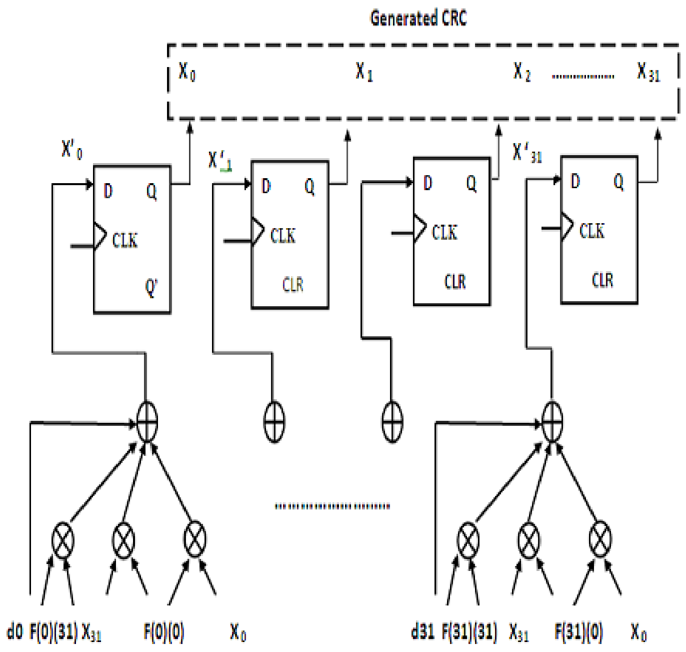

Figure 5 : CRC-32 Parallel calculation for 32bit.

4.3 Implemented Architecture

In Implemented architecture w= 64 bits are parallely processed and order of generator polynomial is m= 32 as shown in fig. 5, 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. Implemented architecture can be realized by below equation.

$$X\ temp = F_W \otimes D(0to31) \oplus D(32to63)$$

$$X' = F_W \otimes X \oplus X\ temp$$

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

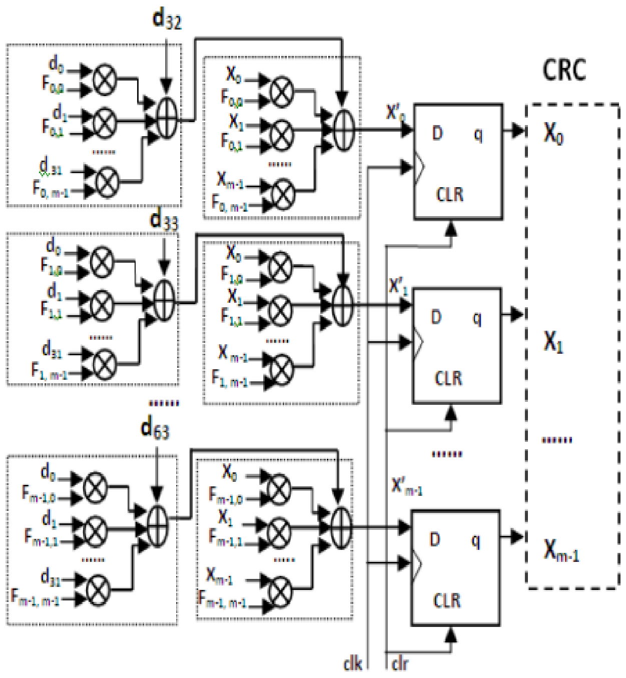

Figure 6: 64-bit parallel calculation for CRC-32 bit.

In Implemented architecture d_i is the parallel input and F(i)(j) is the element of F_{32} matrix located at ith row and jth column. As shown in figure 3 input data bits $d_0 \dots d_{31}$ anded with each row of F_W matrix and result will be xored individually with d_{32}, d_{33} ....d_{63}. 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.

5. Result and Analysis

The Implemented architecture is synthesized in Xilinx-12.3 and simulated in Xilinx ISE Simulator, which required half cycle then the previous 32bit design[2][5]. In our programming in Verilog module, Generated waveforms for CRC32 with w=64,w=32 as shown in below figures.

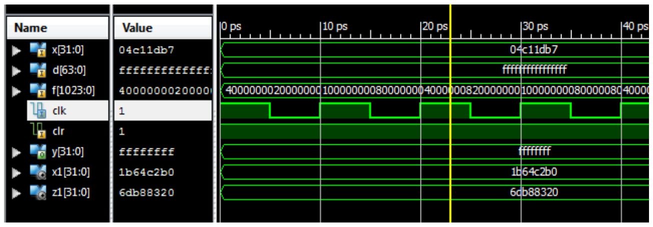
```
Name          Value
x[31:0]       04c11db7
d[63:0]       fffffffffffff:
f[1023:0]     40000000200000(
clk           1
clr           1
y[31:0]       ffffffff
x1[31:0]      1b64c2b0
z1[31:0]      6db88320
```

Figure 7: Generated waveform for CRC-32 (w=64, clr=1)

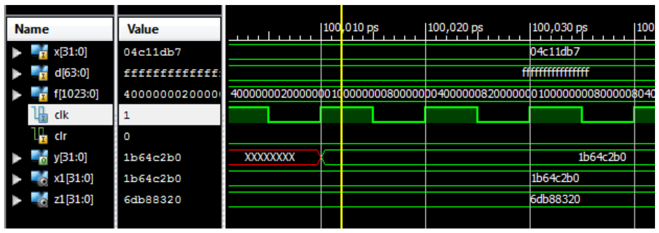

Figure 8: Generated waveform for CRC-32 (w=64, clr=0)

The Implemented CRC-32 architecture with 64bit parallel bit simulated in Xilinx 12.3 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 1B64C2B0 (hexadecimal form).

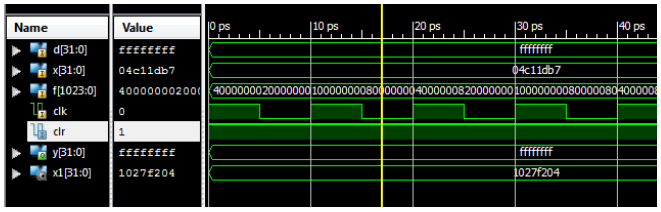

Figure 9: Generated waveform for CRC-32 (w=32, clr=1)

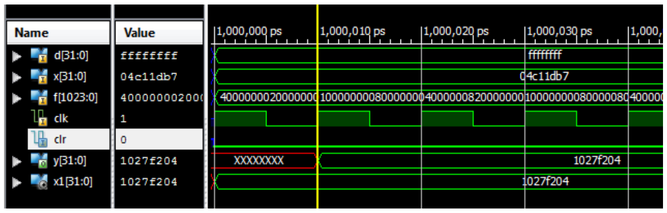

Figure 10: Generated waveform for CRC-32 (w=32, clr=0)

6. Conclusion

32bit parallel architecture required 17, ((k + m)/w) clock cycles for 64 byte data [1] [6]. Implemented design (64bit) required only 9 cycles to generate CRC with same order of generator polynomial. So, it drastically reduces computation time to 50% and same time increases the throughput. Hence, this is compact and easy method for fast CRC generation.

References

[1] Campobello, G.; Patane, G.; Russo, M.; "Parallel CRC realization," *Computers, IEEE Transactions on* , vol.52, no.10, pp. 1312- 1319, Oct.2003

[2] Albertengo, G.; Sisto, R.; , "Parallel CRC generation," *Micro,IEEE* , vol.10, no.5, pp.63-71,Oct1990

[3] M.D.Shieh et al., “A Systematic Approach for Parallel CRC Computations,” Journal of Information Science and Engineering, May 2001.

[4] Braun, F.; Waldvogel, M.; "Fast incremental CRC updates for IP over ATM networks," *High Performance Switching and Routing,2001 IEEE Workshop on* , vol., no., pp.48-52, 2001

[5] Weidong Lu and Stephan Wong, “A Fast CRC Update Implementation”, IEEE Workshop on High Performance Switching and Routing, pp. 113-120, Oct. 2003.

[6] S.R. Ruckmani, P. Anbalagan, “ High Speed cyclic Redundancy Check for USB” Reasearch Scholar, Department of Electrical Engineering, Coimbatore Institute of Technology, Coimbatore-641014, DSP Journal, Volume 6, Issue 1, September, 2006.

[7] Yan Sun; Min Sik Kim; , "A Pipelined CRC Calculation Using Lookup Tables," *Consumer Communications and Networking Conference (CCNC), 2010 7th IEEE* , vol., no., pp.1-2, 9-12 Jan. 2010

[8] Sprachmann, M.; , "Automatic generation of parallel CRC circuits," *Design & Test of Computers, IEEE* , vol.18, no.3, pp.108-114, May 2001.

Author’s Profile

CH.Janakiram presently pursuing M.Tech.(Digital Electronics and Communication Systems) in Department of Electronics and Communications in Sri Vasavi Engineering College (SVEC) Padatadepalli, Tadepalligudem, A.P., India.

K.N.H.Srinivas, Professor in Department of Electronics and Communications in Sri Vasavi Engineering College (SVEC) Padatadepalli, Tadepalligudem, A.P., India