A Novel Approach for Parallel CRC generation for high speed application

Hitesh H. Mathukiya  
Electronics and communication Department,  
Sarvajnanik College of Engineering and Technology,  
Surat, India  
hitesh.mathukiya@gmail.com

Naresh M. Patel  
Electronics and communication Department,  
Sarvajnanik College of Engineering and Technology,  
Surat, India  
naresh.patel@scet.ac.in

Abstract— High speed data transmission is the current scenario in networking environment. Cyclic redundancy check (CRC) is essential method for detecting error when the data is transmitted. With challenging the speed of transmitting data, to synchronize with speed, it’s necessary to increase speed of CRC generation. Starting from the serial architecture identified a recursive formula from which parallel design is derived. This paper presents 64 bits parallel CRC architecture based on F matrix with order of generator polynomial is 32. Proposed design is hardware efficient and required 50% less cycles to generate CRC with same order of generator polynomial. The whole design is functionally verified using Xilinx ISE Simulator.

Keywords— Cyclic Redundancy Check, Parallel CRC calculation, Linear Feedback Shift Register, LFSR, F matrix

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 [6]. 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 [1]. 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.

Albertengo and Sisto [2], has proposed z transform based architecture for parallel CRC, for which it’s not possible to write synthesizable VHDL code. Braun et al [4] presented an approach suitable for FPGA implementation, which has very complex analytical proof. Another approach based on Galois field has been proposed by Shieh et al. [3]. Campobello [1], has presented pre-calculated F matrix based 32 bit parallel processing, which doesn’t work if polynomial change. In this paper, the proposed architecture deal with 64bit parallel processing based on built in F matrix generation; this gives CRC with half number of cycles.

This paper starts with the introduction of serial CRC generation based on LFSR. F matrix based parallel architecture for 32 bits and 64 bits are described in section 3 and 4. Finally, simulation results are shown in section 5 and concluded in section 6.

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. In modulo 2 arithmetic the addition and subtraction are equivalent to bitwise XORs (denoted by “⊕” in this paper) and multiplication is equivalent to AND (denoted by “⊗” in this paper). Figure 1 illustrates the basic architecture of LFSRs for serial CRC calculation.

![Figure 1. Basic LFSR Architecture [2]](image)

As shown in fig.1 d 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.

\[ X_{i}^{'} = (P_{0} \otimes X_{m-1}) \oplus d \]

\[ X_{i} = (P_{0} \otimes X_{m-1}) \otimes X_{i-1} \]  

(1)

The generator polynomial for CRC-32 is as follows

\[ G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{3} + x^{2} + x^{1} + x^{0} \]

We can extract the coefficients of \( G(x) \) and represent it in binary form as

\[ P = \{p_{32}, p_{11}, \ldots, p_{0}\} \]

\[ P = \{10000010011000001000111011011011\} \]
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.

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

LUT base architecture provides lower memory LUT and by the high pipelining Table base architecture has input, LUT3, LUT2, and LUT1. LUT3 contains CRC values for the input followed by 12 bytes of zeros, LUT2 8 bytes, and LUT4 4 bytes. Basically this algorithm it can be obtain higher throughput. The main problem it with pre-calculating CRC and store it in LUT so, every time required to change LUT when changing the polynomial. Pipelining algorithm used to reducing critical path by adding the delay element.

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 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 more simple and low complex. Below algorithm and its implementation is given.

![Fast CRC update architecture](image)

**A. Algorithm for F matrix based architecture.**

Algorithm and Parallel architecture for CRC generation based on F matrix is discussed in this section. As shown in fig. 2 it is basic algorithm for F matrix based parallel CRC generation.

![Algorithms for F matrix based architecture](image)

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.

**B. F Matrix Generation**

F matrix is generated from generator polynomial as per (2).

$$F = \begin{bmatrix}
p_m & 1 & 0 & 0 & 0 \\
p_{m-1} & 0 & 1 & 0 & 0 \\
p_{m-2} & 0 & 0 & 1 & 0 \\
p_{m-3} & 0 & 0 & 0 & 1 \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
p_0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

(2)

Where, \(\{p_0, \ldots, p_{m-1}\}\) is generator polynomial. For example, the generator polynomial for CRC4 is \(\{1, 0, 0, 1, 1\}\) and w bits are parallely processed.
Given the context of the document, the mathematical expressions and diagrams are provided to explain the process of generating a CRC (Cyclic Redundancy Check) code. The key points include:

- **Equation (8)**: A recursive equation for generating the next state \(X'\) based on the current state \(X\) and the input \(d\).

- **Equation (9)**: Equations for calculating \(X_1\) and \(X_2\) using the current state and input.

- **Figure 5**: Illustrates a parallel calculation of CRC-32 for 32-bit data, showing how the process is divided into multiple stages to efficiently calculate the CRC code.

- **Property of the \(F^w\) matrix**: It is mentioned that the \(F^w\) matrix can be easily derived from \(F^{32}\) since the length of the message is 32 bits. For messages of different lengths, additional cycles are needed to complete the calculation.

Overall, the document explains the process of generating a CRC code for parallel data processing, using matrices and recursive equations to efficiently handle different message lengths.
for calculating the extra bits if the data message length is not the multiple of w, the number of parallel processing bits. It is worth to notice that in CRC-32 algorithm, the initial state of the shift registers is preset to all `1’s. Therefore, X(0) = 0xFFFF. However, the initial state X (0) does not affect the correctness of the design. In order for better understanding, the initial state X (0) is still set to 0x0000 when the circuit is implemented.

IV. PROPOSED PARALLEL ARCHITECTURE

In proposed architecture w= 64 bits are parallely processed and order of generator polynomial is m= 32 as shown in fig. 3. As discussed in section 3, 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. Proposed architecture can be realized by below equation.

\[
X_{\text{temp}} = F^w \otimes D(0\text{to}31) \oplus D(32\text{to}63) \\
X' = F^w \otimes X \oplus X_{\text{temp}}
\]

(11)

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

From the table it is observe that, the architecture proposed by [1][8] require 17 clock cycle to generate CRC as per equation (k+ m)/w and for proposed architecture it required only 9 cycle, CPD for proposed architecture is less than the architecture [1][5][6]. The proposed architecture is synthesized in Xilinx 9.2i ISE Simulator, which required half cycle then the previous 32-bit design[1][5]. In our programming in VHDL by specifying only generator polynomial, it directly gives F matrix useful for parallel CRC generation that is not available in previous methods [1][5][6].

V. RESULT AND ANALYSIS

The proposed architecture is synthesized in Xilinx-9.2i and simulated in Xilinx ISE Simulator, which required half cycle then the previous 32-bit design[1][5]. In our programming in VHDL by specifying only generator polynomial, it directly gives F matrix useful for parallel CRC generation that is not available in previous methods [1][5][6]. Hardware utilization is compared in table I for different approaches for different parameter like LUT, CPD and cycle.

<table>
<thead>
<tr>
<th>CRC w parallel bits</th>
<th>Clock cycles</th>
<th>LUTs</th>
<th>CPD</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRC32 w=32bit[1]</td>
<td>17</td>
<td>162</td>
<td>7.3ns</td>
</tr>
<tr>
<td>CRC32 w=32bit[8]</td>
<td>17</td>
<td>220</td>
<td>30.5ns</td>
</tr>
<tr>
<td>CRC32 w=64bit (Proposed )</td>
<td>9</td>
<td>370</td>
<td>5.7ns</td>
</tr>
</tbody>
</table>

The proposed CRC-32 architecture with 64bit parallel bit simulated in Xilinx 9.2i ISE simulator. Input data bit to to system is FFFFFFFF (64 bit). The final result obtain after (k + m)/w cycle for 32-bit residual will be 7DC225CD (hexadecimal form).
VI. CONCLUSION

32bit parallel architecture required 17 \((k + m)/w\) clock cycles for 64 byte data[1][5]. Proposed 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. Pre-calculation of F matrix is not required in proposed architecture. Hence, this is compact and easy method for fast CRC generation.

REFERENCES