ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

Design and Implementation of Parallel CRC Generator for 64-Data Bit

Dasari Mahesh
M.Tech Student,
Digital Electronics & Communication Systems,
Department Of Electronics and Communication Engineering,
ACE Engineering College.

B. Giriraju
Professor,
Department Of Electronics and Communication Engineering,
ACE Engineering College.

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.

I.INTRODUCTION:

A cyclic redundancy check (CRC) is an error-detecting code commonly used in digital networks and storage devices to detect accidental changes to raw data. Blocks of data entering these systems get a short check value attached, based on the remainder of a polynomial division of their contents; on retrieval the calculation is repeated, and corrective action can be taken against presumed data corruption if the check values do not match.

CRCs are so called because the check (data verification) value is a redundancy (it expands the message without adding information) and the algorithm is based on cyclic codes.

CRCs are popular because they are simple to implement in binary hardware, easy to analyze mathematically, and particularly good at detecting common errors caused by noise in transmission channels. Because the check value has a fixed length, the function that generates it is occasionally used as a hash function.

Background:

There are several techniques for generating check bits that can be added to a message. Perhaps the simplest is to append a single bit, called the "parity bit," which makes the total number of 1-bits in the code vector (message with parity bit appended) even (or odd). If a single bit gets altered in transmission, this will change the parity from even to odd (or the reverse).

The sender generates the parity bit by simply summing the message bits modulo 2—that is, by exclusive or'ing them together. It then appends the parity bit (or its complement) to the message. The receiver can check the message by summing all the message bits modulo 2 and checking that the sum agrees with the parity bit. Equivalently, the receiver can sum all the bits (message and parity) and check that the result is 0 (if even parity is being used).

For bit serial sending and receiving, the hardware to generate and check a single parity bit is very simple. It consists of a single exclusive or gate together with some control circuitry. For bit parallel transmission, an exclusive or tree may be used, as illustrated in Figure 14–1. Efficient ways to compute the parity bit in software.

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Monthly Peer Reviewed Open Access International e-Journal

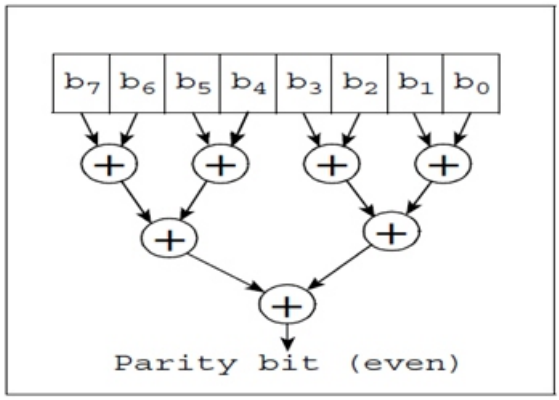

Figure 1: XOR Tree

Theory:

The CRC is based on polynomial arithmetic, in particular, on computing the remainder of dividing one polynomial in GF (Galois field with two elements) by another. It is a little like treating the message as a very large binary number, and computing the remainder on dividing it by a fairly large prime such as intuitively, one would expect this to give a reliable checksum.

Common Name	r	Generator	
		Polynomial	Hex
CRC-12	12	$x^{12}+x^{11}+x^3+x^2+x+1$	80F
CRC-16	16	$x^{16}+x^{15}+x^2+1$	8005
CRC-CCITT	16	$x^{16}+x^{12}+x^5+1$	1021
CRC-32	32	$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$	04C11DB7

Table 1: generator polynomials of some CRC codes

II.CYCLIC REDUDENCY CHECK:

Cyclic Redundancy Check (CRC) is an error-checking code that is widely used in data communication systems and other serial data transmission systems. CRC is based on polynomial manipulations using modulo arithmetic. Some of the common Cyclic Redundancy Check standards are CRC-8, CRC-12, CRC-16, CRC-32, and CRC-CCIT. The bits of data to be transmitted are the coefficients of the polynomial. As an example, the bit stream 1101011011 has 10-bits, representing a 10-term polynomial:

$$M(x) = 1\cdot x^9+1\cdot x^8+0\cdot x^7+1\cdot x^6+0\cdot x^5+1\cdot x^4+1\cdot x^3+0\cdot x^2+1\cdot x^1+1\cdot x^0$$
$$M(x) = x^9+x^8+x^6+x^4+x^3+x^1+1$$

To compute the CRC of a message, another polynomial called the generator polynomial G(x) is chosen. G(x) should have a degree greater than zero and less than that of the polynomial M(x).

Advantages:

- Little overhead.
- Extreme error detection capabilities (it is virtually impossible for a random change in a block of data to still generate the same checksum.)
- Ease of implementation.

Applications:

- Storage devices (including tape, Compact Disk, DVD, etc)
- Wireless or mobile communications (including cellular telephones, microwave links, etc)
- Satellite communications
- Computer networking and communication.

Hardware feedback shift register.

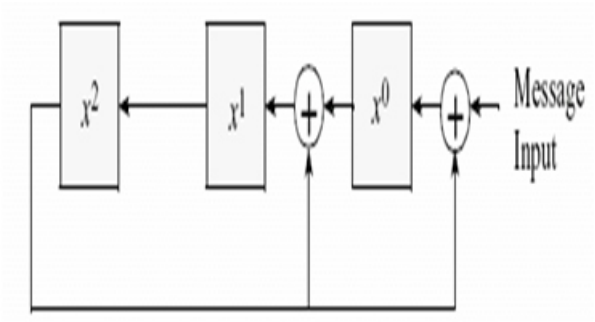

Figure 2: hardware feedback shift register

Initialize the CRC register to all o-bits. Get first/next message bit m. If the high-order bit of CRC is 1, Shift CRC and m together left 1 position, and XOR the result with the low-order r bits of G. Otherwise, Just shift CRC and m left 1 position. If there are more message bits, go back to get the next one.

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

Since the leftmost divisor bit zeroed every input bit it touched, when this process ends the only bits in the input row that can be nonzero are the n bits at the right-hand end of the row. These n bits are the remainder of the division step, and will also be the value of the CRC function (unless the chosen CRC specification calls for some post processing).

The validity of a received message can easily be verified by performing the above calculation again, this time with the check value added instead of zeroes. The remainder should equal zero if there are no detectable errors.

Implementation:

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 and multiplication is equivalent to AND. Figure 1 illustrates the basic architecture of LFSRs for serial CRC calculation.

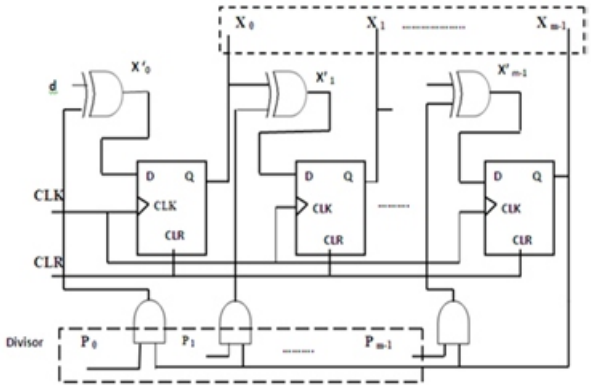

Figure 3: basic LFSR architecture

Serial data input, X is present state (generated CRC), X' is next state and p is generator polynomial.

There are different techniques for parallel CRC generation given as follow.

- A Table-Based Algorithm for Pipelined CRC Calculation.
- Fast CRC Update
- F matrix based parallel CRC generation.
- Unfolding, Retiming and pipelining Algorithm

III.PARALLEL CRC:

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.

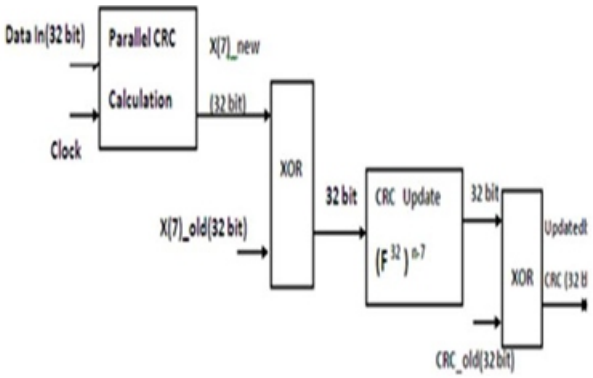

Figure 4: fast CRC update architecture

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

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Monthly Peer Reviewed Open Access International e-Journal

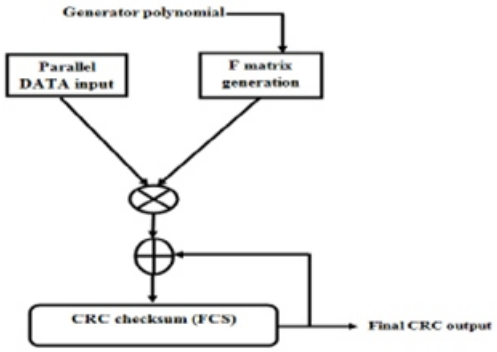

Figure 5: algorithm for f matrix based architecture

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. In proposed architecture w= 64 bits are parallel processed and order of generator polynomial is m= 32 as shown in fig. 3. As discussed in section 3, if 32 bits are processed parallel then CRC-32 will be generated after (k +m)/w cycles..

F Matrix Generation:

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

$$F = \begin{bmatrix} P_{m-1} & 1 & 0 & 0 & 0 \\ P_{m-2} & 0 & 1 & 0 & 0 \\ P_{m-3} & 0 & 0 & 1 & 0 \\ P_{m-4} & 0 & 0 & 0 & 1 \\ .. & .. & .. & .. & .. \\ P_0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (2)$$

Where, {po…… pm-1} is generator polynomial.

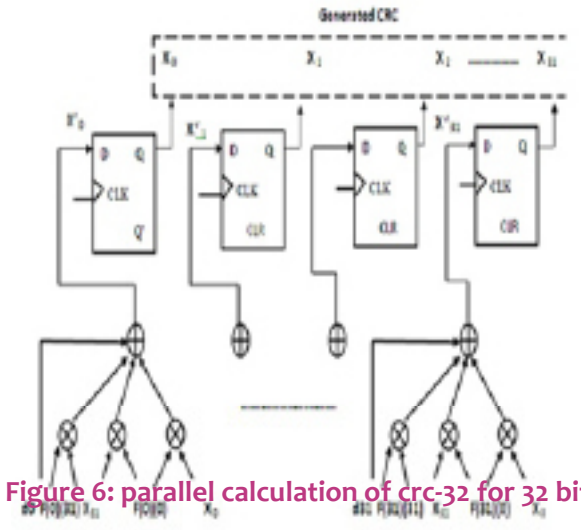

Figure 6: parallel calculation of crc-32 for 32 bit.

Property of the Fw matrix and the previously mentioned fact that Equation (8) can be regarded as a recursive calculation of the next state X' by matrix Fw, current state X and parallel input D, make the 32-bit parallel input vector suitable for any length of messages besides the multiple of 32 bits. Remember that the length of the message is byte based.

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

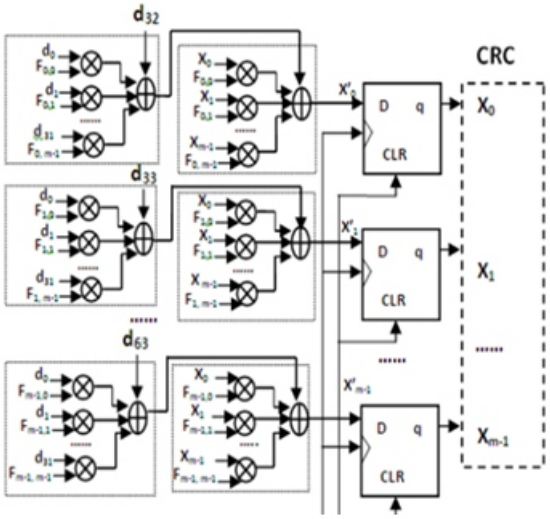

Figure 7: block diagram of 64-bit parallel calculation of crc-32.

In proposed architecture di is the parallel input and F(i)(j) is the element of F32 matrix located at ith row and jth column.

As shown in figure 3 input data bits do….d31 anded-With each row of FW matrix and result will be xored individually with d32, d33 …….d63. 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.

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

IV.RESULTS:

Simulation Results:

Figure 8: Simulation Result of Cyclic Redundancy Check

Timing Report:

```
Timing Summary:
---------------
Speed Grade: -3

   Minimum period: 1.562ns (Maximum Frequency: 640.266MHz)
   Minimum input arrival time before clock: 3.436ns
   Maximum output required time after clock: 3.597ns
   Maximum combinational path delay: No path found
```

Figure 9: Timing Summary of Cyclic Redundancy Check

RTL SCHEMATIC:

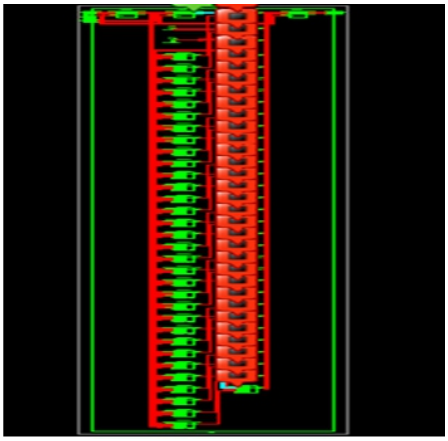

Figure 10: RTL Schematic of Cyclic Redundancy Check

Device Utilization Report:

```
Device utilization summary:
---------------------------

Selected Device : 6slx16csg324-3


Slice Logic Utilization:
 Number of Slice Registers:             86  out of  18224     0%
 Number of Slice LUTs:                  12  out of   9112     0%
    Number used as Logic:               12  out of   9112     0%

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:    86
   Number with an unused Flip Flop:      0  out of     86     0%
   Number with an unused LUT:           74  out of     86    86%
   Number of fully used LUT-FF pairs:   12  out of     86    13%
   Number of unique control sets:        2

IO Utilization:
 Number of IOs:                       162
 Number of bonded IOBs:                88  out of    232    37%

Specific Feature Utilization:
 Number of BUFG/BUFGCTRLs:              1  out of     16     6%
```

Figure 11: Device Utilization Summary of Cyclic Redundancy Check

V.CONCLUSION:

32bit parallel architecture required 17 ((k + m)/w) clock cycles for 64 byte data. 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:

[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.

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Monthly Peer Reviewed Open Access International e-Journal

[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.