64-bit parallel CRC Generation for High Speed Applications

1.P.Sushma (Student)
2.N.Sushma (Student)
3.V.Sowmya (Student)
4.B.Krishna Associate Professor
5.D.ArunKumar Assistant Professor
KITE WOMEN'S COLLEGEOF PROFESSIONAL ENGINEERING SCIENCES-Shabad, Hyderabad.

ABSTRACT:

CRC is playing a main role in the networking environment to detect the errors. . With challenging the speed of transmitting data, to synchronize with speed, it's necessary to increase speed of CRC generation.Most electrical and computer engineers are familiar with the cyclic redundancy check (CRC). Many know that it's used in communication protocols to detect bit errors, and that it's essentially a remainder of the modulo-2long division operation. Some have had closer encounters with the CRC and know that it's implemented as a linear feedback shift register (LFSR) using flip-flops and XOR gates. They likely used an online tool or an existing example to generate parallel CRC code for a design [1].

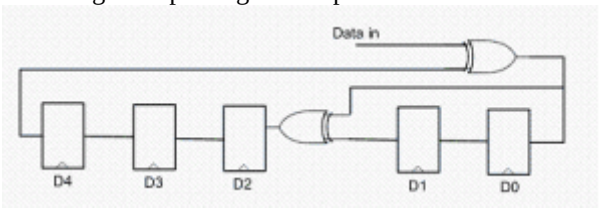


Figure1: Hardware CRC5 implementation

I.INTRODUCTION

CRC can generate in two ways:
1. Serial CRC generation
2. Parallel CRC generation

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. So, to over that problem we are moving to Parallel CRC generation. In this we are discussing about 64-bit parallel CRC generation.32-bit parallel CRC generates gigabits per second, but it will not suited for high speed applications like Ethernet [2].

II. Serial CRC

Here'd' represents the data, {P_{m-1}--------P_1, P_0} represents the generated polynomial, X is the present state and X' is the next state.

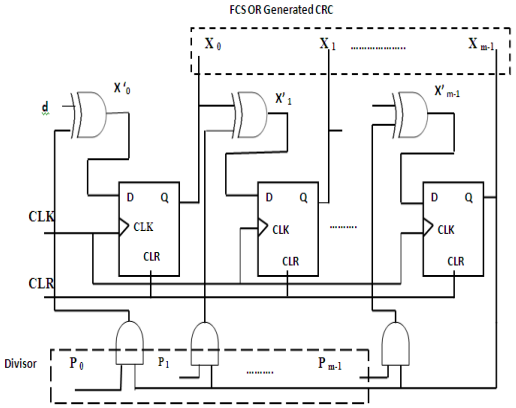


Figure2: Serial CRC generation using LFSR

Operation:

Here CRC checking is done serially. The data input will be single (binary) and every clock pulse the data input will be one. There will some delay present between the consecutive data inputs and the output will be zero if the data will be encoded with same CRC value otherwise it shows non-zero value. By this form we will conclude where the data is accurate or corrupted [3].

The data is serially processed; the polynomial is XORed with the data input, that will given to the D flip-flop (output same as the input) final CRC is generated as serially.

Working of basic LFSR architecture is expressed in terms of following equations

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

The generator polynomial for CRC-32 is as follows

G(x) = $x^{32}+ x^{28}+ x^{23}+ x^{20}+ x^{18}+ x^{15}+ x^{11}+ x^{10}+ x^{8}+ x^{6}+ x^{3}+ x^{2}+ x^{1}+ x^{0}$

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

P = {p31, p_{30}……. p0}

P= {10000010011000001000111011011011}.

III. Parallel CRC

Every modern communication protocol uses one or more error-detection algorithms. CRC is by far the most popular. CRC properties are defined by the generator polynomial length and coefficients. The protocol specification usually defines CRC in hex or polynomial notation [1].

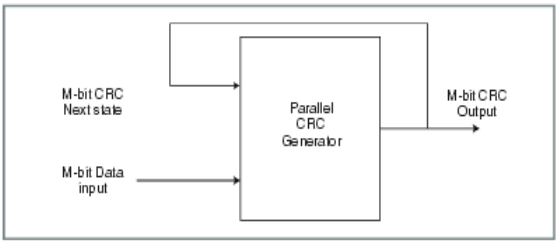


Figure3: This is a parallel CRC block. The next state CRC output is a function of the current state CRC and the data.

There different techniques are:

1. Table based algorithm
2. Fast CRC Update
3. F matrix parallel CRC generation
4. Unfolding, retiming and pipelining algorithm

1. Table based algorithm

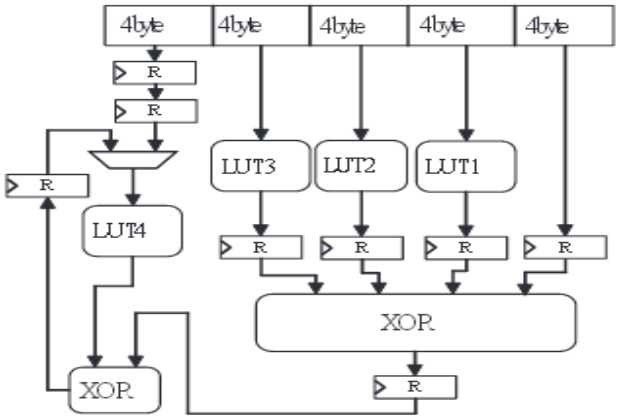


Figure4: Pipelined CRC Architecture

The pipelined architecture in Fig. 4 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 12bytes 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 fur-input XOR gate. This makes the algorithm more scalable because more blocks can be added without increasing the critical path of the pipeline. With the pre-XOR stage, the critical path is the delay of LUT4 and a two-input XOR gate and the throughput is 16 bytes per cycle [4].

Drawback: Table based architecture required pre-calculated LUT, so, it will not used for generalized CRC.

2. Fast CRC

Our fast CRC update method is extended from the parallel CRC calculation and can adapt to a number of bits processed in parallel. The method can also reduce the data traffic and power consumption of the CRC calculation unit[6].

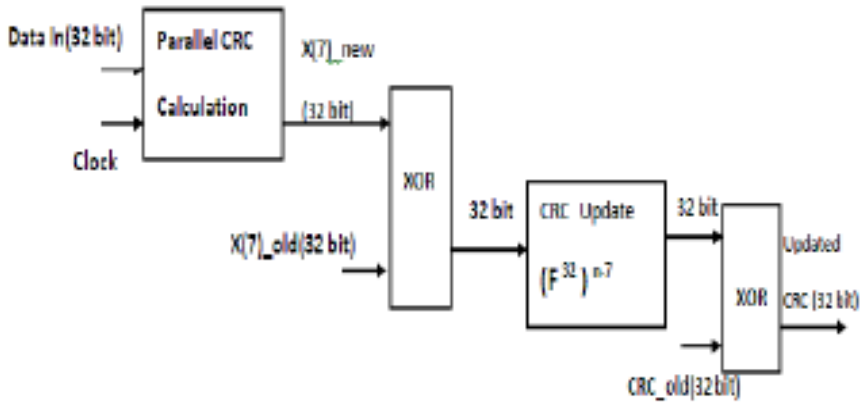


Figure5: Fast CRC update architecture

Drawback: Fast CRC update technique required buffer to store the old CRC and data.

3. Unfolding, retiming and pipelining algorithm

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 $2T_{XOR}$ [5].

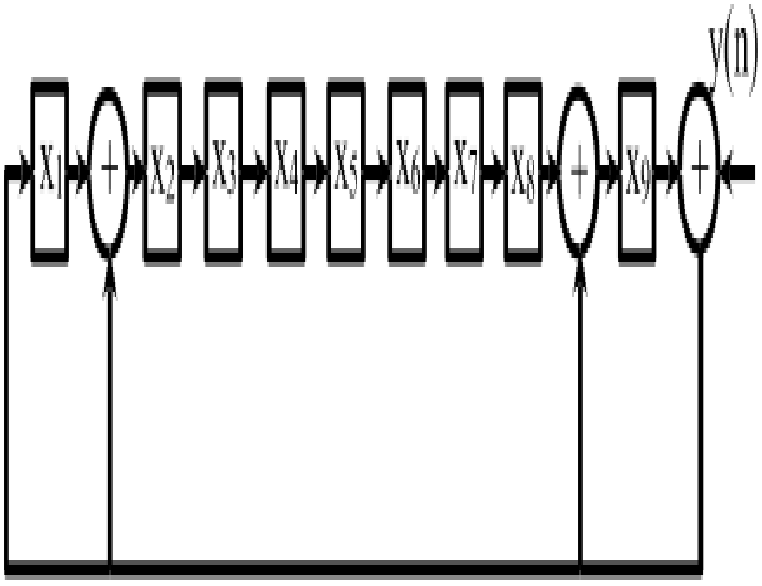


Figure6: Serial CRC

Drawback: In unfolding architecture increases the no. of iteration bound.

4. F-matrix parallel CRC generation:

F-matrix follows the algorithm as:

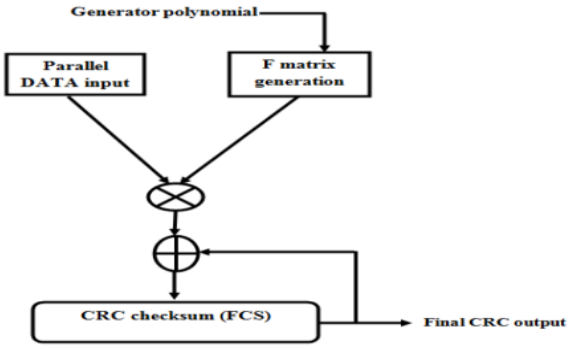


Figure7: Algorithm for F-matrix based architecture

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.

Generation of F-matrix:

F-matrix generated 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 parallely processed.

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

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

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

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 i^{th} row and j^{th} column of F^W 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 i^{th} 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.

64-bit PARALLEL ARCHITECTURE

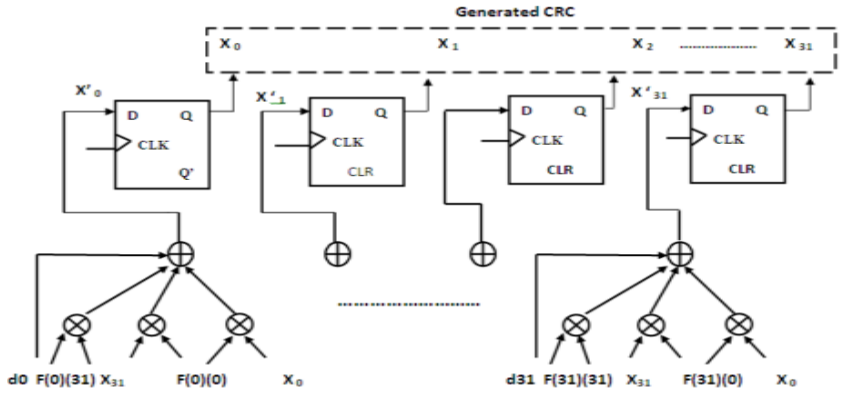


Figure8: Parallel calculation of CRC-32 for 32bit

In proposed architecture w= 64 bits are parallely processed and order of generator polynomial is m= 32 as shown in fig. 8, 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 realizedby below equation.

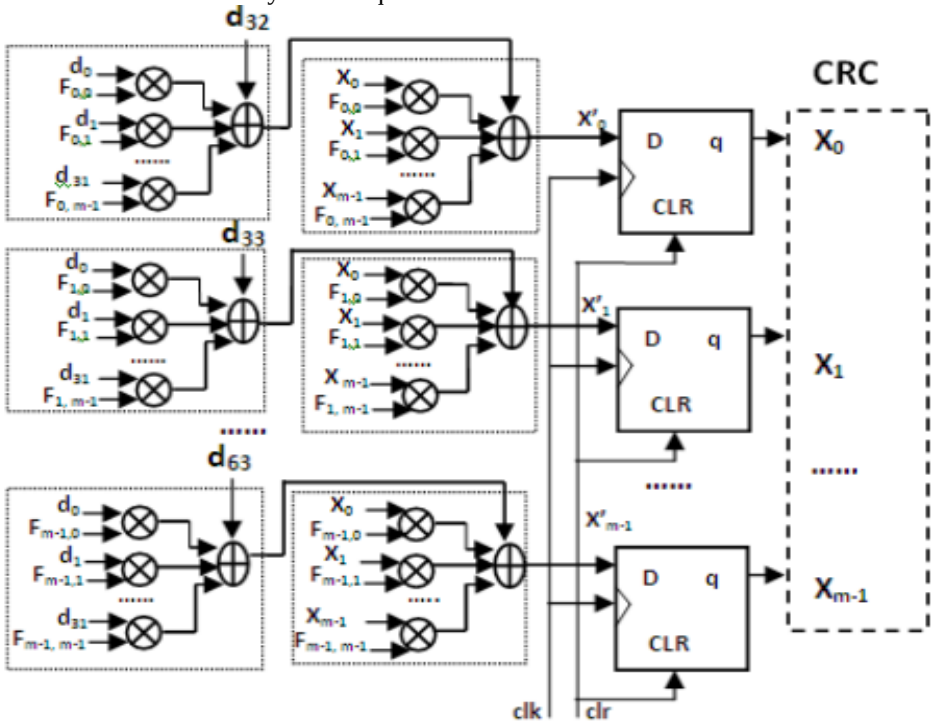
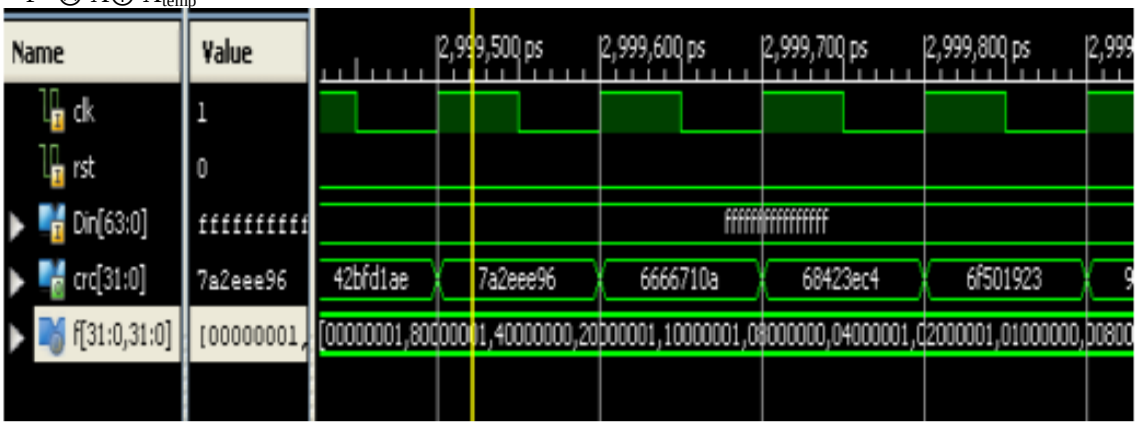


Figure: Block diagram of 64-bit parallel calculation of CRC-32.

$X_{temp} = F^{W} \otimes D\ (0to31) \oplus D(32to63) \oplus$
$X' = F^{W} \otimes X \oplus X_{temp}$



CRC-64

Figure9: Generated waveform for 64-bit parallel CRC

IV. CONCLUSION

Generally when high-speed data transmission is required serial implementation is not preferred because of slow throughput. So parallel implementation is preferred which takes less time. CRC-32 requires 17 clock cycles to transmit 64bytes of data. But CRC-64 needs 9 clock cycles to transmit the same data.

V. *REFERENCES*

[1] "A practical parallel CRC generation method" by Evgeni stavinov
[2] "Design and implementation of CRC code generation based on parallel execution method for high speed wireless LAN" by Ms.sonali, D.selokar, MR.ph.rangaree.
[3] "A high performance CRC checker for Ethernet application" by Deepti Rani Mahankudo & Suresh.
[4] "High performance table based algorithm for pipelined CRC calculation" by yan sun and Min Sik Kim.
[5] "VLSI implementation of parallel CRC using pipelining, unfolding and retiming" by Sangeeta Singh, S.Sujana, J.Babu & K.Latha.
[6] "A novel approach for parallel CRC generation for high speed application" by Hitesh H.Mathukiya & Naresh M. Patel