ABSTRACT: CRC (Cyclic Redundancy Check) is an error detection method commonly used in data communication systems, computer networks and storage environments. In this method, the transmitter divides the message by an agreed upon polynomial called the generator and concatenates the calculated residue to the message. The properties of the generator determine the range of errors which are detectable in the receiver side. The division operation is currently performed using serial circuits called Linear Feedback Shift Registers especially in the Ethernet network access protocol. Developing methods for parallel computation of the residue makes CRC suitable for higher layer protocols and software applications. This paper studies a case for parallel CRC computation using special generators which have special multiples called OZO(One-Zero-One) polynomials are divisible. We first provide a systematic approach to finding such polynomials and then design and evaluate the algorithm and the hardware required to perform the parallel division.

Keywords: Cyclic Redundancy Codes, parallel CRC, OZO polynomials, digital logic, error detection, parallel, programmable.

I. INTRODUCTION

Error correction codes provides a mean to detect and correct errors introduced by the transmission channel. Two main categories of codes exist: block codes and convolution codes. They both introduce redundancy by adding parity symbols to the message data. Cyclic redundancy check (CRC) codes are the subset of the cyclic codes that are also a subset of linear block codes. Cyclic Redundancy Check (CRC) is widely used to detect errors in data communication and storage devices. CRC is a very powerful and easily implemented technique to obtain data reliability. The CRC technique is used to verify the integrity of blocks of data called Frames. In this technique, the transmitter appends an extra n bit sequence to every frame called Frame Check Sequence (FCS). FCS holds redundant information about the frame that helps the receiver detect errors in the frame. When the transmission is received or the stored data is retrieved, the CRC residue is regenerated and confirmed against the appended residue.

For high-speed data transmission, the general serial implementation cannot meet the speed requirement. Parallel processing is a very efficient way to increase the throughput rate. Although parallel processing increases the number of message bits that can be processed in one clock cycle, it can also lead to a long critical path (CP). Thus, the increase of throughput rate that is achieved by parallel processing will be reduced by the decrease of circuit speed. Another issue is the increase of hardware cost caused by parallel processing, which needs to be controlled. The parallel CRC algorithm in processes an m-bit message in (m+k)/L clock cycles, where k is the order of the generator polynomial and L is the level of parallelism. However message bits can be processed in m/L clock cycles. High speed architectures for parallel long encoders are based on the multiplication and division computations on generator polynomial are efficient in terms of speeding up the parallel linear feedback shift register (LFSR) structures. The proposed design achieves shorter critical path for parallel CRC circuits leading to high processing speed than commonly used generator polynomial. The proposed design starts from LFSR, which is generally used for serial CRC. An unfolding
However, direct application of unfolding may lead to a parallel CRC circuit with long iteration bound, which is the lowest achievable CP. Two novel look-ahead pipelining methods are developed to reduce the iteration bound of the original serial LFSR CRC structures; then, the unfolding algorithm is applied to obtain a parallel CRC structure with low iteration bound.

II. Pipelining, Unfolding and Retiming

The implementation of CRC check generation circuit can be done with the use of linear feedback circuit. The CRC architecture for generator polynomial \( G(y) = 1 + y + y^8 + y^9 \) is shown in Fig.1.

![Fig 1: Serial CRC](image)

2.1 Pipelining:

It reduces the effective critical path by introducing pipelining latches along the critical data path either to increase the clock frequency or sample speed or to reduce power consumption at the same speed. It is done using a look-ahead pipelining algorithm to reduce the iteration bound. 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 iteration bound for the circuit shown in Fig.1 is \( 2T_{XOR} \). The largest iteration bound of a general serial architecture is also \( 2T_{XOR} \). For example, the serial architectures of commonly used generator polynomials CRC-16 and CRC-12 have the iteration bound of \( 2T_{XOR} \) because they have terms \( y^{15} + y^{16} \) and \( y^{11} + y^{12} \) in their generator polynomials respectively.

In proposed look-ahead pipelining, 2-level pipelining is given by

\[
a(n+2) = a(n+1) + y(n+1) + b(n+2) \\
a(n+2) = a(n) + y(n) + b(n+1) + b(n+2) + y(n+1) \quad (2)
\]

Fig.4 shows that the loop bound for the circuit in Fig.2 has been reduced from \( 2T_{XOR} \) to \( T_{XOR} \) at the cost of two XOR gates and two flip flops. Also the loop bounds of loop1 and loop2 are \( TXOR \) and \( (5/8) TXOR \) respectively. So, the iteration bound of the two level pipelined CRC architecture is \( T_{XOR} \).

For improved look ahead pipelining consider the polynomial \( G(y) = 1 + y + y^7 + y^9 \).

2.2 Retiming:

Retiming is used to change the locations of delay elements in a circuit without affecting the input/output characteristics of the circuit. It reduces the shown in Fig. 6. The polynomial determines
Retiming has many applications in synchronous circuit design. These applications include reducing the clock period of the circuit, reducing the number of registers in the circuit, decreasing the power consumption of the circuit and logic synthesis. It can be used to increase the clock rate of a circuit by reducing the computation time of the critical path. Critical path is the path with the longest computation time among all paths that contain zero delays, and its computation time is the lower bound on the clock period of the circuit. The two factors affecting the frequency of operation is critical path and iteration bound. Retiming is done by applying the algorithm presented and using Floyd Warshall algorithm.

2.3 Unfolding:

It's a transformation technique that can be applied to DSP program to create a new program describing more than one iteration of the original program. Unfolding a DSP program by an unfolding factor J creates a new program that describes J consecutive iterations of the original program. It increases the sampling rate by replicating hardware so that several inputs can be processed in parallel and several outputs can be produced at the same time. The lower bound on the iteration period of a recursive DSP program is termed as iteration bound. An implementation of the DSP program can never achieve an iteration period less than the iteration bound, even when infinite processors are made available. In some cases, the DSP program cannot be implemented with the iteration bound equal to the iteration bound without the use of unfolding. In general, for a given DFG, when unfolding algorithm is applied with unfolding factor J, the iteration bound of the resultant DFG is J times that of the original DFG.

III CRC:

The CRC is a short fixed-length datum (checksum) for an arbitrary data block. It will accompany the data and can be validated at an endpoint through recalculation. Differences between the two CRC values indicate a corruption in either the data or the received CRC itself.

The CRC calculation can be realized in hardware with an LFSR in Galois configuration as blocks. Each block contain 512 bit. So, the size and the taps of the shift register. In order to obtain the CRC, the register needs to be cleared in a first step. Then, after injecting the message and M additional zeros, the register will hold the desired CRC. A receiver can verify the received message with its appended CRC by simply applying the same procedure, with the difference that the CRC will be shifted into the circuit instead of the zeros. If the register finally equals zero, no error has been detected.

![LFSR Diagram](image)

Fig. 6. LFSR

A. CRC calculation using serial execution unit:

The input message 1024 bit is transmitted serially and it pass through CRC-32 (generator polynomial, single execution unit) and CRC is calculated.

![CRC Calculation Diagram](image)

Fig. 7. shows CRC calculation using single execution unit.

It will take more time to calculate CRC. Therefore it has high latency and low throughput. So, to overcome this problem, we are using multiple parallel execution unit.

B. CRC Calculation using multiple Parallel execution units:

High bit rate message 1024 bit data is used for transmission. During the transmission, the message 1024 bit is divided into 2, 4, 8 block. Fig. 4 shows the CRC calculation using four parallel execution units. According to these blocks got the 32 bit CRC remainder are as under.

- The message 1024 bit is divided into two
- if 32 bits are processed parallely then CRC-32
Two execution units (CRC-32 Polynomial) are connected in parallel and got the two 32-bit remainders from each one. They are xor with each other and obtained 32-bit remainder(CRC).

- The data 1024 bit distributed into four blocks. Each consist of 256 bit. Here, Four execution units are used. They are parallely connected with each other and four 32-bit remainders are obtained. They are xor with each other to get final CRC.

- 1024 bit message is divided into eight blocks. Each block contain 128 bit data. That’s why, Eight execution units are used. They are parallely connected with each other and got eight 32-bit remainders. They are xor with each other to get fast CRC. CRC Calculation with eight execution units is fast as compared to two and four execution units.

In proposed architecture w= 64 bits are parallely processed and order of generator polynomial is m= 32 as shown in fig. 9. As discussed in section 3, Fig 10. Block diagram of 64-bit parallel calculation of will be generated after $\frac{m}{w}(k+)$ cycles. If we increase number of bits to be processed parallel, number of cycles required to calculate CRC can be reduced. Proposed architecture can be realized by below equation.

$$X_{\text{temp}} = F^w \odot D(0\text{to}31) \odot D(32\text{to}63)$$
$$X' = F^w \odot X \odot X_{\text{temp}}$$

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

Fig .9 Fast CRC update architecture

In proposed architecture di 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 9 input data bits $d_0…d_{31}$ are 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 $\frac{m}{w}(k+)$ cycle, where $w=64$. 

Fig 8 shows CRC Calculation using four Parallel Execution units

**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. 9. As discussed in section 3, Fig 10. Block diagram of 64-bit parallel calculation of...
CRC-32.

V. RESULT

The simulation result for CRC using single execution unit and parallel execution units are obtained by Xilinx ISE10.1. Parallel execution units are formed by 2, 4 and 8 execution units. For 32-bit CRC, remainder is A4568879 and according to execution unit got the remainder at clock cycle. The simulation result are as follows.

a) Simulation result for single execution unit:

![Fig.11 Simulation result for single execution unit](image)

b) Simulation result for two parallel execution unit:

![Fig.12 Simulation result for two parallel execution unit](image)

c) Simulation result for four parallel execution units:

![Fig.13 Simulation result for four parallel execution unit](image)

d) Simulation result for eight parallel execution units:

![Fig.14 Simulation result for eight parallel execution units](image)

VI. CONCLUSION

The fast CRC generator using parallel multiple execution units has been designed and result are verified. The performance comparison of fast CRC using single execution unit and 2, 4, 8 execution units are obtained in term of latency and
units increases, the latency is decreases and it is very beneficial for error detection during data transmission.

REFERENCES


