FPGA BASED CRC USING MULTIPLE LUT

Aseela K[1], Navya MV[2] , Lakshmi Raj V[3]

[1,2,3] *MET school of Engineering, Mala*

ABSTRACT

Error detection is an inevitable stage in data transmission. The current scenario in networking environment is high speed data transmission. It is necessary to increase speed of CRC generation to synchronize with the challenging speed of transmitting data. The main objective of this paper is to design a CRC-256 code generator for large input data blocks to increase the effectiveness of CRC codes in error detection. FPGA implementation of CRC code generator could be done in two versions. The first version of the generator designed using linear feedback shift registers is studied. Identifying the serial architecture, a recursive formula the parallel design was derived. As the second version, an accelerated CRC-256 generator designed using an acceleration technique for CRC generation. The implementation of both the CRC-256 generators was carried out in Verilog hardware description language. Simulation and functional verification Xilinx Design Suite 14.2 and Vivado is utilised. A look table-based generator is designed to compare efficiency.

Keyword: - *Cyclic Redundancy Check, Field Programmable Array, Linear Feedback Shift Register.*

1.INTRODUCTION

Cyclic redundancy check (CRC) is a powerful and most widely used error detection method being used commonly in digital data communication and storage. The check (data authentication) value is a redundancy (it expands the message without adding information) and the algorithm is based on cyclic codes hence its name CRC. The simplicity in binary hardware implementation, easy mathematical analysis, and particularly good at detecting common errors caused by noise in transmission channels makes CRCs popular.

Error detection capability of a particular CRC code in error detection hinge on two factors. First, on its generator polynomial and second on the code size. This implies that shorter the CRC code, higher is the probability that the two input bit streams will have the same CRC code. The short CRC codes can protect the input data at the cost of a minimal amount of redundant data are more suitable than a large CRC code. However, shorter CRC codes may make the receiver to accept an erroneous data stream as the correct one as an effect of the collision. Hence for larger data blocks even wider CRC codes are needed to increase the efficacy of CRCs in error detection. In storage systems applications, these improvements can provide a better quality of service and faster data transfers. This work study various acceleration techniques of CRC generation available in literature and highlight their efficacy. A fast CRC generation technique, Slice-by-N, will be used in this work to design an accelerated version of the CRC-256 generator.

A novel method which uses Cyclic Redundancy Check (CRC) for the error detection and the generated CRC error is corrected by using Hybrid Matrix Code (HMC) and they are coded in Verilog HDL and simulated using Xilinx ISE Design suite 14.2 in [1]. Serial and parallel CRC-64 implementation is done for 128 bit of input data and a compact architecture of Slicing-by-N algorithms on an FPGA, capable of reading and processing 32, 64, 128, 256 and 512 bits at a time in [2]. Bajarangbali and P.A.Anand proposed a reduced lookup table-based CRC algorithm for hardware implementation in FPGA. Instead of processing one byte of input stream at a time, the proposed implementation processes 128-bits of input data at a time and uses smaller lookup tables for CRC calculations [3]. A kind of design and implementation of a parallel CRC algorithm for FC by Verilog language from both coarse and fine dimensions is proposed in [4]. The algorithm consumes less resource of FPGA logic resources and can be easily realised.

2.CYCLIC REDUNDANCY CHECK (CRC)

Cyclic redundancy check codes (CRCs) are one of the most widely used codes for error detection during the generation, transmission, processing or storage of digital data. In simpler terms, CRC codes preserve the integrity of data during transmission, processing and when data is at rest. To enable error detection using CRCs, the input stream is divided by a standard polynomial, called the "generator polynomial" of a particular CRC standard. The remainder in this division process is called CRC code for the input data. Before transmission or storage of the data, this remainder is attached with the original data. At the receiver, the transmitted data (original data and its CRC) is again divided by the same generator polynomial. The remainder of this division should be zero if there were no errors introduced during transmission or storage. Cyclic redundancy codes are the most popular for detection of burst errors.

CRCs are used very often as they are easy to design and are capable of detecting a large class of errors. The type of errors that a particular CRC code will be able to detect depends on its generator polynomial. It is possible to detect the following type of errors with Cyclic redundancy check codes:

· All single bit or two-bit errors.
· All odd numbers of bit errors.
· All burst errors where burst size is less than or equal to the degree of the generator polynomial.

2.1. CRC MATHEMATICS

The generator, input message and its CRC, all are represented as polynomials in CRC mathematics. Suppose the generator polynomial for a CRC standard of degree k be g(x) and m(x) an m-bit message. To calculate CRC of this message m(x), first m(x) is multiplied by x^k, which, in effect, shifts the original message m(x) by k bits and appends k zeros at the end of m(x). The shifted version of m(x) is then divided by generator polynomial. The division process results in a quotient q(x) and remainder r(x). The CRC of the message m(x) is this remainder itself.

The CRC calculations are done in the Galois field, i.e., GF. In GF, both addition and subtraction operations are equivalent to exclusive-or (XOR) operation, and multiplication is equivalent to logical AND operation.

The CRC calculation can be written as follows:

$$\frac{x^k m(x)}{g(x)} = q(x) \oplus \frac{r(x)}{g(x)} \quad (1)$$

As addition and subtraction in GF (2) is done by XOR, equation (1) can be written as

$$x^k m(x) \oplus r(x) = q(x) g(x) \quad (2)$$

where the degree of the remainder polynomial is always less than the degree of the generator polynomial, k. The term $x^k m(x) \oplus r(x)$ denotes the original message with its CRC, appended at the end of the message. From equation (2), it is clear that the term $x^k m(x) \oplus r(x)$ is evenly divisible by generator polynomial. This is true when no bits are changed in both m(x) and r(x). At the receiving end, the transmitted message $x^k m(x) \oplus r(x)$ is divided by g(x). If the remainder of this division is non-zero, then it indicates that message received is with errors. And if the remainder is zero then, it indicates that either the message is received without any error or some errors did occur, but remained undetected.

3. CRC ALGORITHM

Numerous cyclic check code principles are as of now being used dependent on the application necessity. Instances of some ordinarily utilized CRC principles are CRC-8, CRC-16, CRC-32, and CRC-64. These principles additionally have various variations relying upon the generator polynomials utilized for them.
LFSR based generators process the incoming message bits serially. But for current high-speed data transmission needs, LFSR based serial implementation cannot meet the speed requirements as it processes only one message bit per clock cycle. A number of algorithms have been used to accelerate the CRC code generation process. Since CRC calculations are performed in GF, the CRC circuit can be realized using shift registers and XOR gates. This implementation is popularly known as linear feedback shift register (LFSR) based CRC generator. Figure 1 shows the typical architecture of the LFSR based CRC generator with a polynomial of degree k.

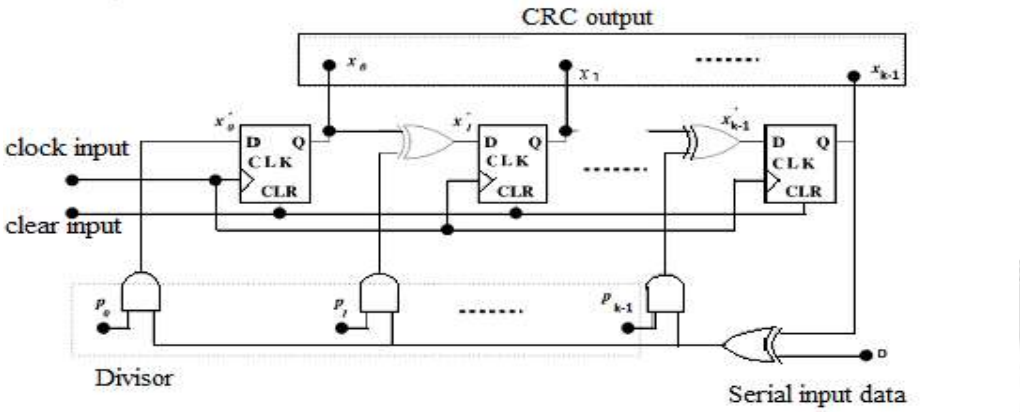

Fig-1: LFSR based CRC generator.

As LFSRs based CRC generators process only one bit of input at a time and are not suitable for the current high-speed applications, an accelerated CRC generator is implemented the CRC-256 generator using one acceleration techniques in the literature review. For the implementation of CRC-256, the first need is the generator polynomial of degree 256. Optimality of the generator polynomial is beyond the scope of this work. One such generator polynomial of degree 256 chosen to be used in current work is " $x^{256} + x^{255} + x^{31} + x^{29} + x + 1$". Processing more bits of input at a time can significantly reduce the processing time and can also increase the throughput. Look-up table based "Slice-by-N" scheme seems the most popular for designing accelerated CRC generators.
In this work, an accelerated version of CRC-256 generator is designed based on slice-by-N algorithm. The current remainder and next bits read from the input data stream, are expressed as the sum of smaller terms and the next remainder is computed by performing parallel look-ups into smaller look-up tables. Here slices of the previous remainder and input data bits work as directories into these tables.

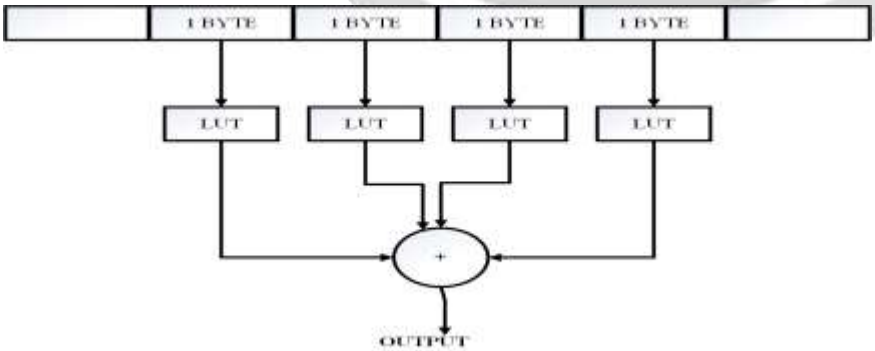

Fig-2: Zero block look up tables.

An accelerated version of CRC-256 generator is designed and implemented in Verilog based on the slice-by-N algorithm. A slice-by-32 method is employed which in each clock cycle reads 256 input data bits at a time. Here,

MSB of the input stream is assumed to come in first. The polynomial used for the CRC generation is given by the polynomial G(x)= $x^{64}+x^{52}+x^{46}+x^{44}+x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^{8}+x^{7}+x^{5}+x^{4}+x^{2}+x+1$.In Slicing by 4 algorithm, the incoming 128 bit of data is taken and it is XORed with the initial CRC value. The CRC-64 initialised by the value 0xFFFFFFFFFFFFFFFF.This modified bit stream split into four slices each 4 bytes of data. These four bytes of data divided again into four single bytes as in figure 2 and using the LFSR based method precomputed all the possible values for each byte and stored in the corresponding LUT. By XORing all the LUT values the final CRC value for each slice is obtained.

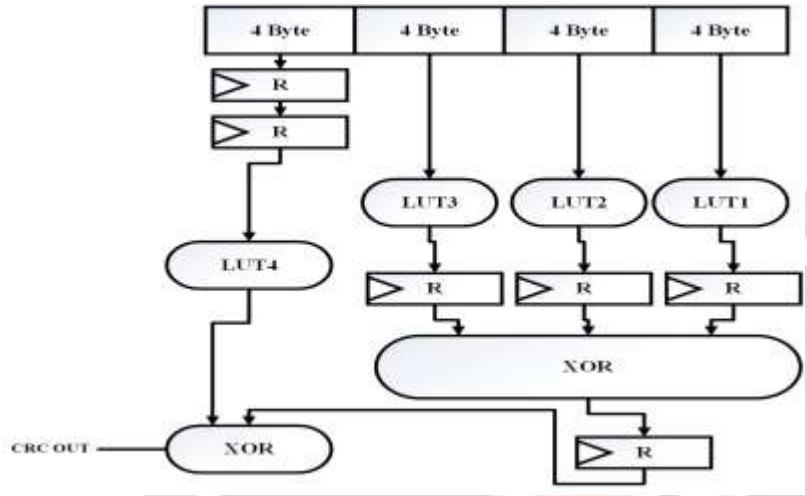

Fig-3: Parallel CRC implementation

we can design the CRC block based on pipelined architecture as the following steps:
1. The 128 bits of data is departed into four 32-bits data.
2. XOR the CRC generated by the previous with the first 32-bits data of the 128-bits data.
3. Calculate the CRC of each 32-bits data, respectively.
4. XOR the CRC calculated in step 3 so that we get the CRC of the data.

At the beginning of each k^{th} step of the algorithm, a binary number, u^{k-1} generated in $(k-1)^{th}$ step, is sliced into 33 parts. Each of the first 32 slices is 8 bit long, and the last slice is 256 bit long. The last slice is the new data read in that step, represented by Data_in. The sum of lengths of all these slices i.e., length of , is 512 bits and it remains the same for each subsequent step of the algorithm. For each of these slices, a table lookup is performed into separate tables maintained for the slices. As the number of bits read in each step is the same, the same set of look-up tables can be utilized in every step reducing the space requirement of the design.
The architecture in its design is generic as can be scaled to 64-, 128-, or 256-bits in the data path, that enables support of throughput rates up to 40 Gb/s at 256-bits. It is further noted that a physical oriented design methodology, such as a data-path compiler can be used to optimize the regular programmable cell array structure, which could significantly increase the operational frequency while maintaining a low hardware cost.

4. RESULT AND SIMULATION

We program the CRC encoding module in Verilog, designed and simulated with Vivado plan suite. Fig.5 shows the simulation results. As the paper mainly discusses the design and implementation of CRC, the figure shows the results of CRC of the data. When the input data is received by the posedge of the clock immediately the CRC can be generated. The implementation of the CRC algorithm reduces the latency from receiving data to generating its CRC.

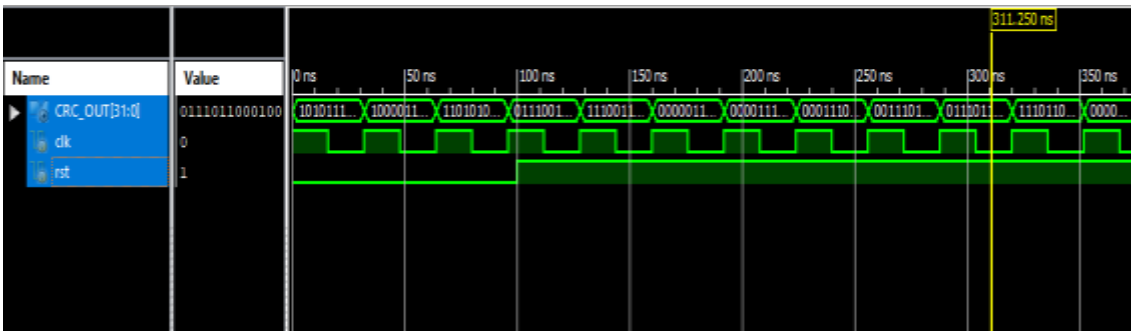

Fig-5: Simulation of CRC using LUT

5. CONCLUSION

The paper proposed an effective design of CRC-256 generator. An accelerated CRC-256 generator using slice-by-N technique was designed. The slice-by-N method was chosen for designing accelerated CRC-256 generator as it can process any number of bits at a time (even higher than the degree of generator polynomial) and requires lookup tables parallelly accessed for which FPGA being suitable platform. Also, a very high throughput maintained on FPGAs by processing a bigger number of bits at a time (increasing number of parallel lookup tables) since it only increases the number of already available used resources in FPGA. Further analysing the results, we can maintain continuous throughput growth, by increasing number of processed bits at a time thus reaching maximum throughput of 170Gbps for processing 512 bits at a time with LUT-based approach.

6. REFERENCES

[1] Amila Akagic, Hiderharu Amano, 'Performance evaluation of multiple lookup tables Algorithms for generating CRC on an FPGA' https://www.researchgate.net/publication/241186779.

[2] Neepa P. Mathew, Anith Mohan, `Matrix Code Based Error Correction for LUT Based Cyclic Redundancy Check', Global Colloquium in Recent Advancement and Effectual Researches in Engineering, Science and Technology (RAEREST), 2016.

[3] Bajarangbali D., Anand P.A.: 'Design of high-speed CRC algorithm for Ethernet on FPGA using reduced lookup table algorithm'. IEEE Annual Indian Conf. (INDICON), Bangalore, India, 2016, pp. 1–7

[4] Wu Chuxiong, Shi Haifeng, `Design and implementation of parallel CRC algorithm for fibre channel on FPGA', October 2019 www.ietdl.org

[5] Kennedy C.E., Mozaffari-Kermani, M.: 'Generalized parallel CRC computation on FPGA'. 2015 IEEE 28th Canadian Conf. on Electrical and Computer Engineering (CCECE), Halifax, Canada, 2015, pp. 107–113

[6] Kennedy C., Reyhani-Masoleh, A.: 'High-speed parallel CRC circuits. Proc. of the 42nd Annual Asilomar Conf. on Signals, Systems and Computers (ACSSC2008), Pacific Grove, CA, USA, 2008, pp. 1823–1829

[7] Sun Y., Kim, M.S.: 'A pipelined CRC calculation using lookup tables. 7th IEEE Consumer Communications and Networking Conf., Las Vegas, NV, USA, 2010, pp. 1–2